# Technical Report No. 2007-09
# Task Scheduling and Lightpath Establishment in Optical Grids

Xin Liu, Wei Wei, Chunming Qiao
Department of Computer Science and Engineering
SUNY Buffalo, NY 14260, USA
Email: xliu8@buffalo.edu

Ting Wang
NEC Laboratories America
Princeton, NJ 08540, USA
Email: ting@nec-labs.com

Weisheng Hu, Wei Guo, Min-You Wu
State Key Lab of Advanced Optical Communication Systems and Networks
SJTU, Shanghai 200030, China
Email: wshu@sjtu.edu.cn

***Abstract*—Data-intensive Grid applications require huge data transferring between multiple geographically separated computing nodes where computing tasks are executed. For a future WDM network to efficiently support this type of emerging applications, traditional approaches to establishing lightpaths between given source destination pairs are not sufficient because a computing task may be executed on any one of several computing nodes having the necessary resources. Therefore, lightpath establishment has to be considered jointly with task scheduling to achieve best performance. In this paper, we study the optimization problems of jointly scheduling both computing resources and network resources. We first present the formulation of two optimization problems with the objectives being the minimization of the completion time of a job and minimization of the resource usage/cost to satisfy a job with a deadline. When the objective is to minimize the completion time, we devise an optimal algorithm for a special type of applications. Furthermore, we propose efficient heuristics to deal with general applications with either optimization objective and demonstrate their good performances in simulation.**

## I. Introduction

We envision that a future WDM network will provide efficient support for many distributed computing applications that require both execution by multiple geographically separated computing nodes and data transferring between them. Each instance of these distributed applications, called a "job" hereafter, may be logically partitioned into multiple tasks for execution on different nodes, and some tasks may need to communicate with other tasks (e.g., exchange data, intermediate results and/or other information).

In this work, we will consider the problem of efficiently scheduling such jobs over a WDM network using wavelength routing. The edge nodes in the WDM network provide direct access to the computing nodes with various computing resources. We assume that a given task may be executed on any one of several candidate nodes having the necessary computing resources required by this task, with possibly different execution times. We refer to the problem of deciding the spatial and temporal assignment of the tasks to the computing nodes as the *task scheduling* problem, which has been an essential aspect of parallel and distributed systems or more recent works on grid computing. In task scheduling, a job is usually represented as a directed acyclic graph (DAG), where a node represents a task and an arc represents the communication between two tasks.

In a WDM network, a submitted job request can be blocked (dropped or delayed) not only due to the unavailability of the computing resources at the nodes but also due to the lack of network (wavelength) resources. This is because each lightpath will exclusively occupy a wavelength along its route. Accordingly, we will have to consider potential contention on the wavelength resources with task scheduling. We refer to the problem of efficiently allocating network resources for data transferring among tasks as the *lightpath establishment* problem.

The objective of this work is to design and evaluate approaches to the joint optimization of the task scheduling and lightpath establishment (TSLE) problem. For each accepted job request, the set of nodes chosen to execute the tasks, and the lightpaths established among them form what we call an Application Specific and Agile Private (or ASAP) network. This is the first work that considered the joint optimization of allocating both network and computing resources. One of the major contributions of this work is the formulation of two optimization problems with the objectives being the minimization of the completion time of job and minimization of the resource usage/cost to satisfy a job with a deadline. Another major contribution is to devise an optimal algorithm when the objective is to minimize the completion time for a special type of DAGs, and the final contribution is to propose efficient heuristics to deal with general DAGs with either optimization objective.

The rest of the paper is organized as follows. Section II discusses some related works. Section III describes the major challenges of solving the TSLE problem. In Section IV, we present the formal formulation of the TSLE problem. In the following two sections, we present novel algorithms to solve the TSLE problem with different objectives. The simulation results are presented in Section VII and Section VIII concludes the paper.

## II. Related Works

Several optical network architectures, control software and technologies have been proposed in [1]–[3] to efficiently support grid services. In general, these architectures can be based on either optical circuit switching (OCS) (via wavelength routing) or optical burst switching (OBS), depending on the bandwidth or delay requirement of Grid applications. In [1], an OCS based approach (or Grid-over-OCS) was proposed for applications requiring huge bandwidth for a long period. In this approach, the Grid and optical-layer resources can be managed either separately in an overlay manner or jointly by extending the optical control plane for Grid-resource provisioning. Another type of architecture to support Grid services is based on OBS (or Grid-over-OBS), which is suitable for applications having small job sizes [2], [3]. The approaches used in this study are based on OCS where we establish lightpaths between computing nodes to form an ASAP network.

A recent work [4] considered the problem of jointly scheduling computing and network resources for one DAG. The authors in [4] formulated the problem assuming that a (SONET/SDH) connection of sub-wavelength granularity can be used to satisfy the communication requirements of a pair of tasks and proposed a heuristic approach to minimize the completion time of the job.

In this work, we consider a similar but different problem where 1) lightpaths are required for communicating nodes; 2) multiple jobs (some of them may have deadlines) may arrive one after another and accordingly, the objective is to minimize the resource usage, subject to the job's deadline constraint. In addition, when the objective is to minimize the completion time of a job, we obtain an optimal solution for a pipelined DAG, and based on this, we also devise an efficient algorithm for a general DAG. As far as we know, this is the first work that addresses these optimization problems.

In retrospect, this work also differs from previous works on traditional lightpath establishment (static or dynamic) where the source and destination pairs are given [5]. Because here, the source and destination nodes depend on how tasks are assigned, so is the traffic matrix when multiple nodes (or multiple tasks) are involved. Moreover, the problem of forming an ASAP network for each job differs from virtual topology (VT) design, where a common VT is designed to optimally support traffic aggregated from several different applications, and thus only one (or very few) VT is needed at a time [6]. Only when the change of the traffic matrix exceeds a threshold is reconfiguration process triggered to form a new virtual topology [7]. However, for the envisioned applications, different ASAP networks need to be formed for different jobs. Accordingly, not only the number of concurrent ASAP networks will be much larger than that of VT, ASAP networks also come and go as jobs are admitted and finished, which has a much faster rate than that of VT reconfiguration. In addition, when jobs have deadlines, the TSLE problem is also different from the problems related to scheduled lightpath and sliding scheduled lightpath demands [8], [9].

## III. Challenges

The TSLE problem has not been addressed previously and it raises new challenges in both optical networking and grid computing research. Two major challenges are as follows.

### A. How to measure or estimate communication cost?

In previous works on task assignment in parallel and distributed systems or more recent works on job scheduling for grid computing applications, the underlying physical network's connectivity is often taken for granted, and at best is assigned a constant "communication cost" between two nodes. However, in a WDM network supporting dynamic jobs, link usage changes dynamically, and it is possible that one cannot establish a lightpath between two communicating nodes at the time when it is needed. Besides, it is difficult to attach a price tag (the communication cost) to the use of networking resources in a way that is consistent with or comparable to the costs of using other computing resources. In contrast to the previous research, this work takes a cross-layer joint optimization approach that aims to address the fundamental interdependency between the computing resource allocation and networking resource allocation strategies within the context of supporting distributed computing applications over WDM networks.

### B. How to optimize the allocation of computing and network resources jointly?

Unlike the classic model in a parallel and distributed system, where all processors are fully connected and all communications can be performed concurrently, the edge nodes in a WDM network are connected only when lightpaths can be established between them and two lightpaths cannot use the same wavelength on a link at the same time. Thus an optimal schedule without considering the underlying limited network resources can perform badly or worse, cannot be executed due to the lack of wavelengths in a WDM network. This calls for a joint allocation of computing and network resources. However, this problem is hard to solve since a part of TSLE, the task scheduling problem is NP-hard in general [13].

## IV. Models and Formulations

In this section, we first introduce the network model that includes an optical network with attached computing nodes and the task graph to represent a job. We then formulate the TSLE problem mathematically using Integer Linear Programming (ILP).

### A. Network model

The optical network with attached computing resource nodes can be formulated as an edge-weighted undirected graph $G_n = (\mathbf{V, E, r, l, s})$. The first four elements characterize the static properties of $G_n$. More specifically, the vertex set **V** and the edge set **E** correspond to the network nodes and the links interconnecting the nodes respectively. In this study, we assume only one computing node is attached to

one edge node in the WDM network. Thus we will use the term "node" to loosely refer to either an edge node in the WDM network or the attached computing node hereafter. Each node $v \in \mathbf{V}$ provides one of $R$ categories of resources, such as computation, storage and visualization and let $\mathbf{r}_v$ denote the category of resources the node $v$ provides. There are $M_i, i = 1, 2, \ldots, R$ nodes in category $i$. Finally, $\mathbf{l}_e$ represents the physical length or cost of the edge $e \in \mathbf{E}$. In addition to the above four, we have one more element **s**, which represents the dynamic status (or the "snapshot") of the resource utilization. More specifically, $\mathbf{s} = (\mathbf{w}, \mathbf{c})$, where **w** records the earliest idle time of every wavelength on every link and **c** stores the earliest idle time of every resource node $v$.

### B. Task Graph

A *task graph* is commonly used to represent the communication requirements and precedences between tasks of a job [10]. A task graph can be formulated as a directed acyclic graph (DAG) $G_t = (\mathbf{N}, \mathbf{L}, \mathbf{r}, \mathbf{e}, \mathbf{b}, \mathbf{d})$. Each node $n \in \mathbf{N}$ represents a task and $\mathbf{r}_n$ denotes the type of the resources required by task $n$. A task $n$ with type $\mathbf{r}_n$ can be executed on any one of $M_{\mathbf{r}_n}$ resource nodes having the necessary computing resources required by this task. To illustrate the the heterogeneity of computing nodes, a $|\mathbf{N}| \times |\mathbf{V}|$ matrix $e$ is given as a part of the description of the task graph. More specifically, $\mathbf{e}_{nv}$ represents the estimated execution time of task $n \in \mathbf{N}$ when it is assigned to node $v \in \mathbf{V}$ (and if task $n$ cannot be executed on node $v$, then the corresponding execution time will be set to infinity). An arc $l_{ij} = (n_i, n_j) \in \mathbf{L}$ indicates there will be data transferring from node $n_i$ to node $n_j$ after $n_i$ has been executed and before $n_j$ could be executed. An estimated average bandwidth (in terms of the number of wavelengths, i.e., no sub-wavelength request is considered) and duration of the data transferring (or transmission delay) associated with $l_{ij}$ are given by $\mathbf{b}_{l_{ij}}$ and $\mathbf{d}_{l_{ij}}$ respectively.

### C. Problem Statement

In addition to the above models, we assume tasks of different jobs can be queued at a node. Similarly, every edge node in the optical network can store any amount of data until they are transferred to a remote node. In other words, we assume both edge node and computing node have unlimited buffer. We refer to this assumption as the queueing model in contrast to the dropping model, where a task (and the corresponding job) requiring the resources at a node has to be dropped if the node is busy running another task.

For the purpose of simplicity, we only employ *end technique* [10], which means that at time $t$, the scheduler can only schedule a job on the resources which are free in $[t, \infty]$. In other words, idle time slots between reserved time slots will not be used. Based on this assumption, we need to maintain two sets of time-stamps: $t_v^{ei}$, the earliest idle time on node $v \in \mathbf{V}$ and $t_{v_i v_j}^{ei}$, the earliest idle time on lightpath between $v_i$ and $v_j$.

Based on the above models and assumptions, for a given job we need to 1) decide an optimal assignment of each node of the task graph to a computing node and each arc of the task graph to a lightpath in the WDM network 2) decide the exact operation time of this assignment as shown in Figure 1. We refer to the assignment and the time of performing it as a schedule.

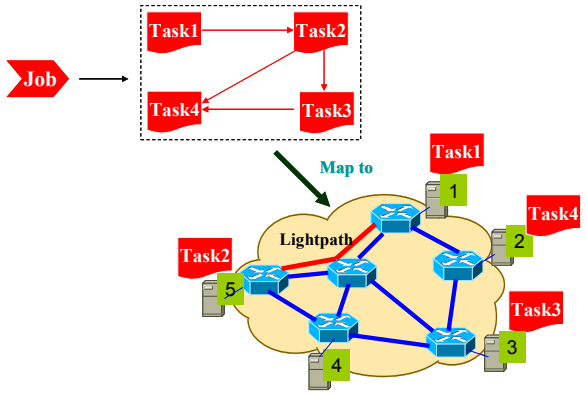


Fig. 1. An illustration of mapping a task graph onto an optical network

Once an optimal or near-optimal schedule is decided for a job, computing resources will be reserved on the assigned computing nodes according to the schedule. In addition, an ASAP network is formed by establishing $|\mathbf{L}|$ lightpaths dynamically using advanced reservation and the exact setup and release time of these lightpaths are decided by the schedule.

The following notations will be used in the mathematical formulas which describe the TSLE problem.

- $\Gamma$: the assignment function from **N** to **V**. $\Gamma(n) = v$ indicates that the schedule assigns task $n$ to be executed on node $v$.
- $t_{nv}^s, t_{nv}^f$: the starting time and finish time of task $n$ scheduled on node $v = \Gamma(n)$. When $v$ is not specific, we just use the shorter term $t_n^s, t_n^f$ to represent the starting time and finish time of task $n$. Obviously, we have $t_{nv}^f = t_{nv}^s + e_{nv}$.
- $t_{l_{ij}}^s, t_{l_{ij}}^f$: the starting time and finish time of the communication associated with $l_{ij}$. Obviously, we have $t_{l_{ij}}^f = t_{l_{ij}}^s + \mathbf{d}_{l_{ij}}$.

### D. Constraints

The following two constraints (computing resource constraint and precedence constraint) are adopted from [10] for task scheduling.

- **computing resource constraint:**

$$\Gamma(n) = v \;\Rightarrow\; t_{nv}^s \geq \mathbf{c}_v, \; n \in \mathbf{N} \tag{1}$$

$$\Gamma(n_i) = \Gamma(n_j) = v \;\Rightarrow\; \begin{cases} t_{n_i v}^f \leq t_{n_j v}^s \text{ or} \\ t_{n_j v}^f \leq t_{n_i v}^s \end{cases}, n_i, n_j \in \mathbf{N} \tag{2}$$

Constraint (1) ensures a task can only be scheduled on a node after its initial earliest idle time. Constraint (2) ensures that if any two tasks are scheduled on a node, their execution time intervals must be disjoint.

- **precedence constraint:**

$$t^s_{n_j} \geq \max_{l_{ij} \in \mathbf{L}} t^f_{l_{ij}},\ n_i, n_j \in \mathbf{N} \tag{3}$$

$$t^s_{l_{ij}} \geq t^f_{n_i}, n_i \in \mathbf{N},\ l_{ij} \in \mathbf{L} \tag{4}$$

where $l_{ij} = (n_i, n_j)$ represents an arc from $n_i$ to $n_j$. Constraint (3) ensures a task must wait for all the required data from its predecessors to start execution. The righthand side formula is defined to be the data ready time $t^{\text{drt}}_{n_j} = \max_{l_{ij} \in \mathbf{L}} t^f_{l_{ij}}$. Constraint (4) reflects the fact that the data transferring from $n_i$ must start after the execution of $n_i$ has finished.

As mentioned before, the network resources are limited and a job can be blocked due to the lack of wavelengths. Thus in addition to the classic task scheduling constraints, a feasible schedule is subject to the following constraints related to wavelength resources.

- **network resource constraint:**
  For any arc $l \in \mathbf{L}$ scheduled on the lightpath $p$ and for any wavelength $k$ used on link $e$ along $p$, we have

$$t^s_l \geq \mathbf{w}_{e_k} \tag{5}$$

  where $e_k$ represents the $k$th wavelength on link $e$ and recall that $\mathbf{w}_{e_k}$ denotes the initial earliest idle time of wavelength $e_k$.
  For any two arcs $l, l' \in \mathbf{L}$ scheduled on the lightpaths $p, p'$ respectively, if $p$ and $p'$ share an identical wavelength on a common physical link, then

$$t^f_l \leq t^s_{l'} \text{ or } t^f_{l'} \leq t^s_l \tag{6}$$

  These two constraints are similar in form to the computing resource constraint but more complicated because an arc is scheduled on a lightpath, which can use many different wavelengths on many links whereas a task is only scheduled on one computing node. Note that the contention on wavelength resources will delay the data ready time and thus influence task scheduling.

Since we only have a limited number of wavelengths available ($W$ per link), the following capacity constraint must hold,

- **capacity constraint:**
  Recall that a subset of arcs $\mathbf{L}' \subseteq \mathbf{L}$ corresponds to a set of lightpaths $\mathbf{P}$ after assignment. If the paths in $\mathbf{P}$ cause contention in both spatial domain and temporal domain, i.e., they share a physical link and their busy times $(t^s_{l_{ij}}, t^f_{l_{ij}}), l_{ij} \in L'$ share a common time interval, then we have

$$\sum_{l \in \mathbf{L}'} \mathbf{b}_l \leq W \tag{7}$$

Finally, in some situation, we may require the job to be executed within certain time,

- (optional) **deadline constraint:**

$$\max_{n \in \mathbf{N}} t^f_n \leq D \tag{8}$$

### *E. Objectives*

Based the given models and assumptions, possible objectives could be,

- minimize the completion time of a given job (without deadline constraint):

$$\text{minimize } \max_{n \in \mathbf{N}} t^f_n \tag{9}$$

The completion time is also called schedule length or makespan. This objective is commonly adopted in the context of parallel systems and grid computing. Another objective is to minimize execution and communication cost when there is a deadline for the given job. This objective is important from the view of either economics or resource utilization. The user sometimes wants to minimize the cost of leasing an ASAP network to execute a job as long as it's finished on time. In addition, service provider prefers to spend as little resources as possible on executing a job such that the network can accommodate more jobs or other types of traffic. More specifically, the second objective is:

- minimize the cost (with deadline constraint):

$$\text{minimize } C_c \sum_{l \in \mathbf{L}} \mathbf{b}_l \mathbf{d}_l |p_l| + C_e \sum_{n \in \mathbf{N}} \mathbf{e}_{nv} \tag{10}$$

where $|p_l|$ is the physical length of the lightpath on which $l$ is scheduled. In this study, we assume the cost of leasing a lightpath is a linear function of bandwidth $\mathbf{b}_l$, duration $\mathbf{d}_l$ and path length. Similarly the cost of leasing a computing node is proportional to the execution time. $C_c$ and $C_e$ are two constants representing the communication unit cost and execution unit cost respectively.

## V. Minimize the Completion Time

In this section, we focus on the TSLE problem when the objective is to minimize the completion time. Hereafter, we refer to the TSLE problem of minimizing the completion time as the *min-time* problem. Accordingly, we have *min-time* solutions and *min-time* algorithms. First, an algorithm which yields optimal solution is presented for a special case, where the task graph is in the form of a pipeline. Second, we employ this algorithm to improve an existing heuristic approach to solve the general case.

### *A. Optimal Solution for a Pipelined DAG*

If the task graph is a in the form of a pipeline, i.e., $\mathbf{L} = \{(n_i, n_{i+1}) | 1 \leq i \leq |\mathbf{N}| - 1\}$, an algorithm extended from [11] can be used to obtain minimum completion time in polynomial time. The authors in [11] considered the task assignment problem (without scheduling) in the context of parallel and distributed computing and suggested a transformation from a pipelined task graph to an extended graph $G_e$. More specifically, we define the following transformations:

- A node $n \in \mathbf{N}$ is mapped to $M_{\mathbf{r}_n}$ nodes in $G_e$, each of which corresponds to an assignment of $n$ to a computing node $v \in \mathbf{V}$.

- An arc $l_{ij} = (n_i, n_j) \in \mathbf{L}$ is mapped to $M_{\mathbf{r}_{n_i}} \times M_{\mathbf{r}_{n_j}}$ arcs in $G_e$, each of which corresponds to one of possible connections from $\Gamma_{n_i}$ to $\Gamma_{n_j}$.
- A dummy source node $s$ and a dummy destination node $t$ are added in $G_e$. $s$ is connected to all the nodes transformed from source node $n_1$ and all the nodes transformed from sink node $n_{|\mathbf{N}|}$ are connected to $t$.

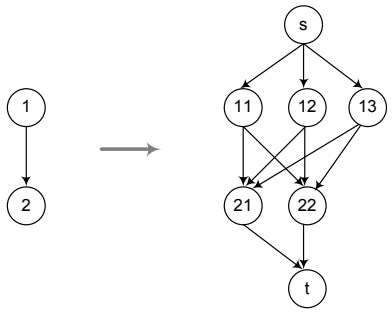


Fig. 2. An illustration of transforming a two-node task graph

Figure 2 shows an example of the transformation from a simple two-node graph, where $M_{\mathbf{r}_{n_1}} = 3$ and $M_{\mathbf{r}_{n_2}} = 2$. The basic idea is that by assigning execution times and lightpath durations as weights to nodes and arcs respectively, one can use Dijkstra's algorithm to find a shortest path from $s$ to $t$, which corresponds to an optimal assignment with minimum completion time. However, in the context of scheduling, this algorithm cannot be used directly because the availability of both computing and networking resource have to be considered to generate a feasible schedule. Thus we need to modify the "Relax" procedure in a standard Dijkstra's algorithm [12] as shown below. We also refer to the Dijkstra's algorithm which uses the following "Relax" procedure as Algorithm 1.

**Algorithm 1** Relax($u', v'$)

1: $u', v' \in G_e$ correspond to $n_i, n_j \in \mathbf{N}$ and $v_i, v_j \in \mathbf{V}$ respectively.
2: **if** $v_i = v_j$ **then**
3: $\quad t^{ei}_{v_j} \leftarrow t^{f}_{u'} + e_{n_j v_j}$
4: **else**
5: $\quad t^{ei}_{v_j} \leftarrow \max(\max(t^{f}_{u'}, t^{ei}_{v_i v_j}) + \mathbf{d}_{l_{ij}}, t^{ei}_{v_j}) + e_{n_j v_j}$
6: **end if**
7: **if** $t^{f}_{v'} > t^{ei}_{v_j}$ **then**
8: $\quad t^{f}_{v'} \leftarrow t^{ei}_{v_j}$
9: **end if**

In this excerpt of the modified Dijkstra's algorithm, let $n_i, n_j \in \mathbf{N}$ be the tasks and $v_i, v_j \in \mathbf{V}$ be the computing nodes corresponding to $u', v' \in G_e$ respectively. $u'$ is a node in $G_e$ whose shortest distance (which is defined to be the finish time of $n_i$ if scheduled on $v_i$) from source $s$ has been found and recorded and $v'$ is adjacent to $u'$. This procedure is used to find out if the shortest distance of $v'$ can be shortened by going through $u'$.

If $n_i$ and $n_j$ are scheduled on the same node, then there is no transmission delay. In this case, task $n_j$ can be executed immediately after task $n_i$ is finished. On the other hand, if task $n_i, n_j$ are scheduled on different nodes, the data can only be transmitted after task $n_i$ is finished and the lightpath between $v_i$ and $v_j$ has sufficient bandwidth. After all the data arrive at $v_j$, only when it's free can node $v_j$ start to execute $n_j$. If the new path through $u'$ is shorter than the recorded one, the value of shortest distance $t^{f}_{v'}$ is updated.

The initial value of earliest idle time $t^{ei}_{v}$ of a node $v \in \mathbf{V}$ is given as a part of the description of the optical network, i.e., initially $t^{ei}_{v} = \mathbf{c}_v$. Its value is updated in each iteration of "Relax" operation as shown above. Similarly, the earliest idle time $t^{ei}_{v_i v_j}$ of a path between $v_i$ and $v_j$ can be decided by simply searching the maximum earliest time over all links along the path when sufficient bandwidth $\mathbf{b}_l$ is available.

Algorithm 1 is optimal only if fix routing is assumed, i.e., the shortest path is used between any two nodes in $\mathbf{V}$ [5]. This is because a path from $s$ to $t$ in $G_e$ corresponds to a possible assignment and vice versa. Thus the shortest path returned from Dijkstra's algorithm corresponds to the optimal schedule. If alternate routing is used, one can enumerate all the possible paths between $v_i$ and $v_j$ and select the one with the earliest idle time in line 5. If adaptive routing is allowed (with much higher complexity of course), an algorithm proposed in [4] can be used to find a path with earliest idle time. Algorithm 1 does not guarantee optimality if either alternate routing or adaptive routing is used but it's still effective. In this paper, we assume the number of nodes in each category of resources are the same, which is denoted by $\overline{M}$. Thus there are $O(\overline{M}|\mathbf{N}|)$ nodes and $O(\overline{M}^2|\mathbf{L}|)$ arcs in $G_p$ and the time complexity of this algorithm is $O(\overline{M}|\mathbf{N}|\log(\overline{M}|\mathbf{N}|) + \overline{M}^2|\mathbf{L}|)$ according to [12].

### *B. A Heuristic for a General DAG*

The above Algorithm 1 is optimal only when the task graph is a pipeline, i.e., tasks are executed sequentially and thus no contention will occur when allocating computing resources and network resources. For a general DAG, where the contention from parallel executions happens commonly in both computing nodes and underlying networks, Algorithm 1 can not be employed directly to solve the TSLE problem.

The general scheduling problem of minimizing schedule length is NP-hard [13]. The best known solution of this scheduling problem assuming a general DAG in the context of parallel and distributed computing is a heuristic, namely list scheduling [14], where the tasks are ordered according to some priority scheme and then scheduled one by one to achieve the objective greedily. The downside of list scheduling is obviously that it doesn't guarantee optimal solution. In fact, the performance of list scheduling on a pipelined DAG would be much worse than that of Algorithm 1 as shown in the simulation studies.

The above discussions thus motivate us to embed Algorithm 1 into a list heuristic approach. Because many practical task graphs have subgraphs in the form of a pipeline, we could take

advantage of Algorithm 1 to achieve a better performance than a common list scheduling.

**Algorithm 2** A list scheduling heuristic to minimize the completion time

1: Short-circuit all pipelines and obtain a new task graph $G_p$.
2: Compute bottom levels and sort tasks $n \in \mathbf{N}$ into a list $\mathbf{Q}$.
3: **for** each $n_j \in \mathbf{Q}$ in the decreasing order of the bottom levels **do**
4: **for** each $v_j$ such that $v_j$ can execute $n_j$ and let $\Gamma(n_j) = v_j$ **do**
5: **for** each $n_i$ such that $(n_i, n_j) \in G_p$ in the decreasing order of the bottom levels **do**
6: Find the finish times of all the pipelines or normal arcs between $n_i$ and $n_j$
7: **end for**
8: $t_{n_j}^{\text{drt}} \leftarrow \max_{l_{ij}=(n_i,n_j)\in\mathbf{L}} t_{l_{ij}}^f$
9: $t_{n_j v_j}^f \leftarrow \max(t_{v_j}^{ei}, t_{n_j}^{drt}) + e_{n_j v_j}$
10: **end for**
11: $\Gamma(n_j) \leftarrow \text{argmin}_{v_j} t_{n_j v_j}^f$
12: $t_{n_j}^f \leftarrow \min_{v_j} t_{n_j v_j}^f$
13: $t_{\Gamma(n_j)}^{ei} \leftarrow t_{n_j}^f$
14: update the earliest idle time of all the used wavelengths along the paths between $\Gamma(n_j)$ and its predecessors.
15: **end for**

Algorithm 2 is a formal implementation of the above idea. First, all the pipelines in $G_t$ are recognized and short-circuited. The new graph $G_p$ formed by this operation is identical to $G_t$ except that every pipeline in $G_t$ is now an arc in $G_p$. This can be done by merging each node which has one parent and one child with its incident nodes iteratively. Second, as shown in [15], the priority scheme based on the *bottom level* achieves the best performance among other priority schemes. The bottom level, which is recursively defined below, represents the length (in terms of the sum of execution and communication time) of the longest path leaving the node,

$$bl(n_i) = \overline{e}_{n_i} + \max_{l_{ij}=(n_i,n_j)\in\mathbf{L}} (bl(n_j) + \mathbf{d}_{l_{ij}}) \qquad (11)$$

where $\overline{e}_{n_i}$ denote the average execution time of task $n_i$.

The rationale behind this scheme is that by allowing the task with higher bottom level to be executed earlier, the long path behind it can be finished earlier. We use the bottom level as the priority scheme in line 2. Note that an arc in $G_p$ corresponds to a pipeline $pl$ in $G_t$, thus its cost is computed as $\sum_{n_i \in pl} \overline{e_{n_i}} + \sum_{l_{ij} \in pl} \mathbf{d}_{l_{ij}}$. Next, we schedule tasks one by one according to the priority scheme.

Because the availability of network resources has to be considered, the scheduling order of arcs between $n_j$ and its predecessors is also important since different order can yield different resource allocation and accordingly different data ready time $t_{n_j}^{\text{drt}}$. We still use the decreasing order of the bottom levels. In $G_p$, an arc could be a pipeline or a normal arc in $G_t$. For a pipeline $pl_{ij}$ between $n_i$ and $n_j$, the modified Dijkstra's algorithm is used and the its finish time is recorded in line 6. For a normal arc $(n_i, n_j) \in \mathbf{L}$, the modified "Relax" procedure is used to update the arc finish time $t_{l_{ij}}^f$.

After enumerating all possible candidate nodes, task $n_j$ is scheduled on the node with earliest finish time and accordingly, the earliest idle time on the computing node and all the used wavelengths along the lightpaths connecting $\Gamma(n_j)$ and its predecessors are updated.

We will refer to a general list scheduling heuristic, i.e., without short-circuiting pipelines in line 1, as the *min-time* algorithm. The time complexity of finding the finish time of an arc is $O(|\mathbf{E}|W \log W)$ because we first need to sort the earliest idle time of wavelengths on each link along $p_l$ to find the earliest idle time of each link and then obtain the earliest idle time of path $p_l$. Accordingly, the time complexity of the *min-time* algorithm is $O(\overline{M}(|\mathbf{N}| + |\mathbf{L}||\mathbf{E}|W \log W))$ according to [10]. In the presence of pipelines, the time complexity of list heuristic depends on the exact number of pipelines and the number of nodes along each pipeline, thus a general analysis of the time complexity is omitted here.

## VI. Minimize the Cost with Deadline Constraint

In this section, we focus on the problem of scheduling a job constrained by a deadline with a different objective, which is to minimize the execution cost and communication cost. This problem will be called *min-cost* problem to differentiate it from the *min-time* problem. We aim to solve it by giving an efficient *min-cost* heuristic.

**Algorithm 3** A list scheduling heuristic to minimize the cost with deadline constraint

1: **for** each task $n_j$ of remaining $d_p$ tasks in $\mathbf{Q}$ in the decreasing order of the bottom levels **do**
2: **for** each $v_j$ such that $v_j$ can execute $n_j$ and let $\Gamma(n_j) = v_j$ **do**
3: Find the finish time of task $n_j$ if $\Gamma(n_j) = v_j$.
4: Find the introduced cost $C_c \cdot (\sum_{n_i} \mathbf{b}_{l_{ij}} \mathbf{d}_{l_{ij}} |p_{l_{ij}}|) + C_e \cdot e_{n_j v_j}$.
5: **end for**
6: Assign $n_j$ to $v_j$ which achieves the minimum cost in this iteration and update the total cost.
7: Update the earliest idle time of nodes and wavelengths.
8: **end for**

First of all, the feasibility issue of the *min-cost* problem can be solved based on a *min-time* solution. If the minimum completion time $F$ of a given job returned by an optimal *min-time* solution is larger than the given deadline $D$, then there is no feasible solution. Otherwise, the optimal *min-time* solution is also a feasible solution to the *min-cost* problem. Note that the *min-time* algorithm is a heuristic, which does not guarantee the optimal solution. Thus the feasibility verification based on it is an approximation.

Secondly, the *min-time* algorithm can be modified to obtain the minimum cost as shown in Algorithm 3. This algorithm is similar to the *min-time* algorithm except that in addition to finish times, we also obtain and record the communication cost

of arcs ($C_c \cdot \sum_{n_i} \mathbf{b}_{l_{ij}} \mathbf{d}_{l_{ij}} |p_{l_{ij}}|$) and the execution cost of $n_j$ on $v_j$ ($C_e \mathbf{e}_{n_j v_j}$). Because the objective of Algorithm 3 is not to minimize completion time but to minimize cost, it is possible that a job can not be scheduled within its deadline even though a feasible solution exists. In this case, the algorithm reports a failed scheduling. As Algorithm 3 is not sufficient to solve the *min-cost* problem because it does not consider the deadline, we devise another heuristic as described next.

### A. A Heuristic based on Perturbation

This heuristic is based on the idea of perturbation. In Algorithm 4, we try to obtain the minimum cost by perturbing the schedule of some tasks from a *min-time* solution. More specifically, the output of this algorithm is a hybrid schedule, where the first $k$ tasks in the list $\mathbf{Q}$ follow the *min-time* schedule and the remaining $|\mathbf{N}| - k$ (which is referred to as the *perturbation degree* $d_p$ hereafter) tasks are scheduled by a *min-cost* algorithm, such as Algorithm 3. Such hybrid schedule with perturbation degree $d_p$ is also called a $d_p$-schedule.

**Algorithm 4** A hybrid scheduling algorithm to minimize the cost with deadline constraint

1: Invoke the *min-time* algorithm to get the minimum completion time $F$, priority list $\mathbf{Q}$ and schedule $S$.
2: **if** $F > D$ **then**
3: then there is no feasible solution. stop.
4: **else**
5: Use binary search recursively to find the largest perturbation degree $d_p$ such that the completion time of either $d_p$-schedule or either $(d_p + 1)$-schedule is smaller than $D$.
6: Use upward and downward linear search around $d_p$ with the radius $\log |\mathbf{N}|$ and obtain the cost and the completion time of the $d_p$-schedule.
7: **end if**
8: Return the schedule with the minimum cost provided it finishes before deadline.

Note that in line 5 and 6, given a $d_p$, we use the *min-time* algorithm to schedule first $|\mathbf{N}| - d_p$ tasks in $\mathbf{Q}$ and Algorithm 3 to schedule the remaining $d_p$ tasks.

Intuitively, as $d_p$ increases, a $d_p$-schedule will yield a longer completion time but with smaller cost. Thus if ideally, the completion time is a strictly increasing function of $d_p$ and the cost is a strictly decreasing function of $d_p$, we can just use a binary search procedure to locate a suitable perturbation degree $d_p$, such that the corresponding $d_p$-schedule finishes within the deadline $D$ but the $(d_p+1)$-schedule does not. The cost of this schedule should be a good approximation of the minimum cost we are looking for. However, as shown in the simulation studies, the completion time turns out to be a very complicated function of $d_p$ even though in the long run, it will generally increase as $d_p$ increases. As an approximation, we use multiple iterations of binary search to find the largest perturbation degree $d_p$ such that the completion time of either $d_p$-schedule or either $(d_p + 1)$-schedule is smaller than $D$. Because every binary search will generate a suitable $d_p$, we can then search from $d_p$ to $|\mathbf{N}|$ to find a larger one. Based on the observation that the cost is generally smaller with a larger $d_p$, the largest $d_p$ found in line 5 of Algorithm 4 will be treated as a good starting point for a local linear search with a $\log |\mathbf{N}|$ radius. Finally, the schedule with the minimum cost and without exceeding the deadline is returned. The performance of this heuristic is studied in Section VII.

## VII. Simulation Studies

In this section, we have implemented above proposed algorithms on task scheduling and lightpath establishment over a WDM network (100 wavelengths per link) with a NSFNET topology (16 nodes) and each edge node is connected to one computing node. For simplicity, fixed routing is used. The performance of these algorithms is evaluated and compared below.

### A. Minimize the Completion Time

In order to study the performance of proposed Algorithm 2, we compare it with a general list scheduling approach described in [4] and [10], which has the same structure as Algorithm 2 except that no pipelines are short-circuited and the order of enumerating the predecessors of a task (see line 5 in Algorithm 2) is different. In [4], the predecessors of a task are sorted in an increasing order of their earliest starting times, which is also suggested in [10].

In addition to pipelined DAGs, we also compare the above algorithms over randomly generated DAGs. To generate DAGs randomly, three main parameter are used: the number of nodes ($|\mathbf{N}|$), the average in-degree and out-degree (assumed to be identical) of a node $\overline{\delta}$ and the radio of average communication time (or lightpath duration) to average execution time $r_{ce} = \overline{d}/\overline{e}$. More specifically, after $|\mathbf{N}|$ is decided, $\overline{\delta}|\mathbf{N}|$ arcs are added into the task graph one by one provided that the task graph is connected and acyclic. The endpoints and the direction of each arc are chosen randomly from all possible combinations with the same probability. The execution time of tasks on computing nodes follows Gaussian distribution with the average $\overline{e}$ and the transmission time also follows Gaussian distribution with the average $\overline{d}$. In both cases, we assume the variance is twice the average.

In the following, we refer to our proposed Algorithm 2 as the "New" one and the general list scheduling in [4] and [10] as the "Classic" one. We compare them in term of schedule length over the following three types of DAGs.

- Pipeline: $|\mathbf{N}| \in \{20, 40, 60, 80, 100\}, \overline{\delta} \approx 1, r_{ce} = 1, 10$.

We can see from Figure 3 that the proposed algorithm generally yields a shorter schedule length than the "classic" algorithm does (approximately $10\%$ shorter when $r_{ce} = 10$). When the number of tasks or $r_{ce}$ increases, the difference between them is bigger. This shows that the advantage of the proposed Algorithm 2 becomes more significant when the size of the pipelined DAG is larger and the communication time are longer.

- Parallel workflow: $|\mathbf{N}| = 15, \overline{\delta} \approx 1, r_{ce} = 1$ or $10$.

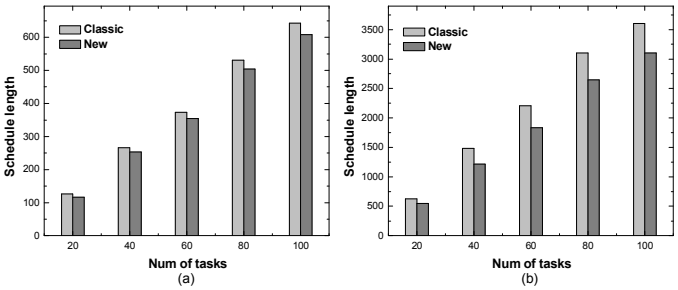


Fig. 3. Comparison of Algorithm 2 and "classic" list scheduling in terms of schedule length on pipelined DAGs when (a)$r_{ce} = 1$, (b)$r_{ce} = 10$

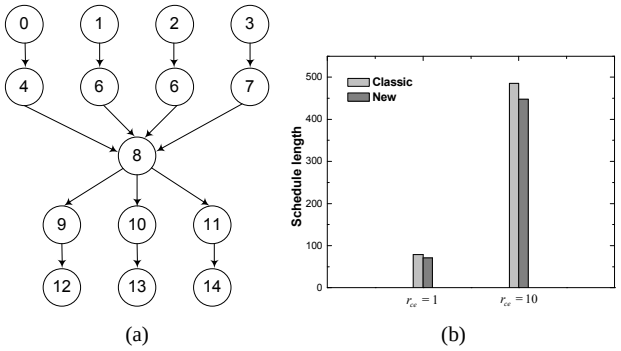


Fig. 4. Comparison of Algorithm 2 and "classic" list scheduling in terms of schedule length on a specific DAG. (a) A specific DAG for a parallel workflow (b) Comparison results when $r_{ce} = 1$ and $r_{ce} = 10$

In addition to the above pipelined DAGs, the subgraphs of many other DAGs can be pipelines, as shown in Figure 4(a) (adopted from [16]). In this type of DAGs, the results from Figure 4(b) show that the proposed algorithm also reduces the schedule length significantly.

- Random DAG:$|\mathbf{N}| \in \{20, 40, 60, 80, 100\}, \overline{\delta} = 2, r_{ce} =$ 1 or 10.

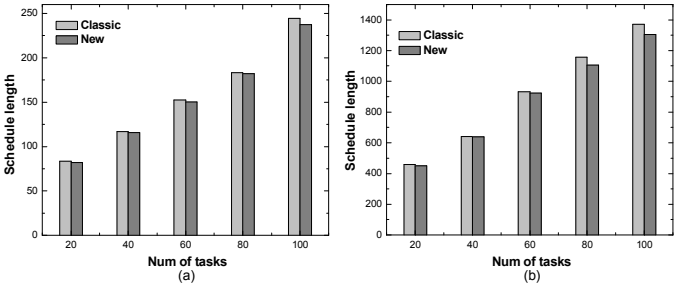


Fig. 5. Comparison of Algorithm 2 and "classic" list scheduling in terms of schedule length on random DAGs when (a)$r_{ce} = 1$, (b)$r_{ce} = 10$

Finally, we investigate the performance of Algorithm 2 on random DAGs. A randomly generated DAG can have pipelines as subgraphs. Thus in this general case, the proposed Algorithm 2 still have advantages but not as significant as in previous two cases (see Figure 5). When $\overline{\delta}$ increases, the probability that a randomly generated DAG has pipelines as subgraphs will become very small and thus Algorithm 2 will behaves the same as the "classic" list scheduling.

### *B. Minimize the Cost with Deadline Constraint*

In the above discussions, the objective is to minimize the completion time of one job without considering other jobs or other types of traffic. However, in the presence of other traffic, the execution of one job could be significantly delayed if the network is congested and most of the wavelengths are already reserved for other traffic. This is shown in Figure 6, where we assume that only a limited number of wavelengths are available for a job or in other words, the earliest idle time of other wavelengths are extremely large. We can see that schedule length (or job completion time) can be extremely long with a smaller $W$. This is because when many requests compete for a wavelength, they have to be scheduled one by one, which delays the total completion time.

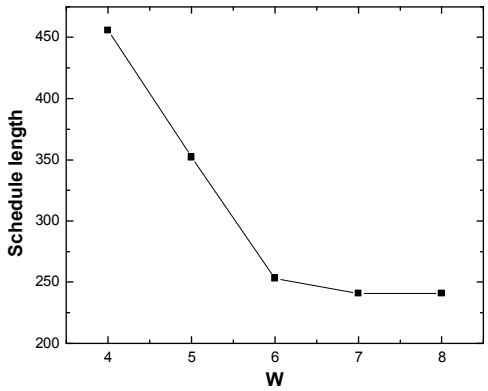


Fig. 6. The relation between schedule length and the number of available wavelengths

To effectively deal with such situations, we use Algorithm 3 and Algorithm 4 to minimize the resource usage/cost with deadline constraint. As mentioned before, the schedule length obtained from Algorithm 2 is not a strictly increasing function of perturbation degree $d_p$ as shown in Figure 7(a). However, based on the observation that the schedule length roughly increases with $d_p$ and only a few $d_p$-schedules achieve the same completion time, we can use binary search multiple times to find a suitable $d_p$ as described in Algorithm 4. We can also see from Figure 7(b) that the cost roughly decreases with $d_p$.

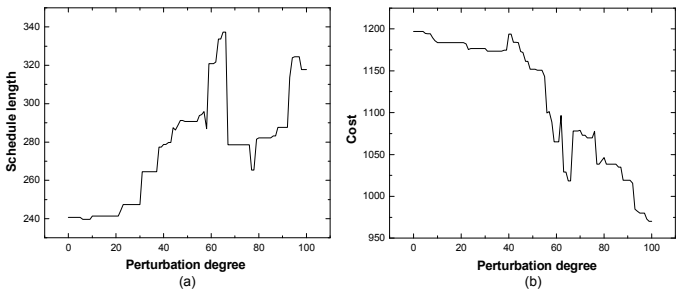


Fig. 7. The relation between (a) schedule length, (b)cost and the perturbation degree

Figure 8 shows the schedule length and the cost with different deadlines assuming a random DAG ($|\mathbf{N}| = 100$, $\delta = 3$,

$r_{ce} = 1$). In addition, we define the ratio of communication cost to execution cost to be $C_{ce} = \frac{C_c \overline{\mathbf{bd}|p|}}{C_e \overline{e}}$, where $\overline{|p|}$ is the average number of links along a lightpath. In this study, we let $C_{ce} = 1$ and then decide $C_c$ and $C_e$ accordingly. As we can see from Figure 8(b), the cost decreases when a

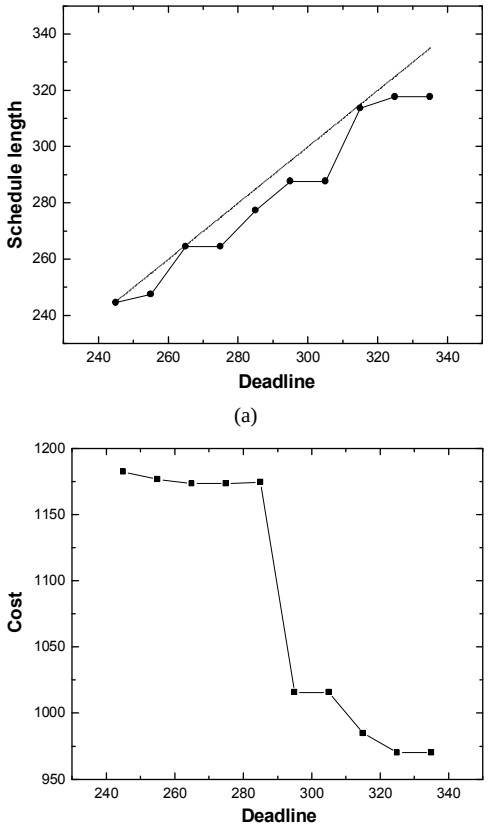


Fig. 8. The (a) schedule length and (b)cost obtained from Algorithm 4 with different deadlines

larger deadline is given and the actual schedule lengths are all below the deadline (the dash line in Figure 8(a)). The minimum completion time is around $240$ time unit and thus no feasible solutions exist if the deadline is lower than this value. The minimum cost corresponds to the longest schedule length in this figure since in each iteration of scheduling tasks, we minimize the introduced cost instead of the finish time as long as the deadline is met. Since this is the first time such a problem is studied, there is no existing methods to compare our approach with. Nonetheless, we believe that these results provide useful insights and lay down basis for further studies.

## VIII. Conclusion

We have defined a joint optimization problem called task scheduling and lightpath establishment (TSLE) in the context of providing efficient support for emerging distributed computing applications in a future wavelength routed WDM network using the concept of Application Specific and Agile Private (ASAP) networking. We have formulated the TSLE problem using ILP and proposed Algorithm 2 to minimize the completion time of a job. Our simulation results have shown that it performs better than a traditional list scheduling algorithm. In addition, the proposed Algorithm 4 is the first attempt to minimize the cost with a deadline constraint in optical grids.

This work opens up many research issues and we are currently investigating the potential benefits of using optical burst switching (OBS) or polymorphous OBS (POBS) [17] to form ASAP networks, over optical circuit switching. In addition, ongoing and future works will also investigate 1) the performance of other types of approaches, such as duplicated heuristics [18] and clustering heuristic [19] applied in optical grids and 2) new scheduling algorithms under various QoS constraints.

## References

[1] D. Simeonidou, et al., *Dynamic optical-network architectures and technologies for existing and emerging grid services*, Journal of Lightwave Technology, 23(10), pp. 3347–3357, 2005.
[2] M. De Leenheer, et al., *A view on enabling-consumer oriented grids through optical burst switching*, IEEE Communications Magazine, 44(3), pp. 124–131, 2006.
[3] G. Zervas, et al., *A Fully Functional Application-aware Optical Burst Switched Network Test-bed*, OFC/NFOEC 2007, OWC2.
[4] Y. Wang, et al., *Joint scheduling for optical grid applicaitons*, Journal of Optical Networking, 6(3), pp. 304-318, 2007.
[5] Hui Zang, Jason P. Jue, and Biswanath Mukherjee, *A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks*, SPIE Optical Networks Magazine, vol. 1, no. 1, Jan. 2000.
[6] Rudra Dutta and George Rouskas, *Survey of Virtual Topology Design Algorithms for Wavelength Routed Networks*, Optical Networks (SPIE) Vol 1, No 1, pp. 73-89, January 2000.
[7] Dhritiman Banerjee and Biswanath Mukherjee, *Wavelenth-Routed Optical Networks: Linear Formulation, Resource Budgeting Tradeoffs, and a Reconfiguration Study*, IEEE/ACM Transactions on Networking, 8(5), pp. 598–607, 2000.
[8] J. Kuri, et al., *Routing and wavelength assignment of scheduled lightpath demands*, IEEE JSAC, 21, 8, 2003, pp. 1231-1240.
[9] B. Wang, et al., *On service provisioning under a scheduled traffic model in reconfigurable WDM optical networks*, Broadband Networks, 13-22, 2005
[10] O. Sinnen and L.A. Sousa, *Communication contention in task scheduling*, IEEE Transactions on Parallel and Distributed Systems, 16(6), pp. 503–515, 2005.
[11] S.H. Bokhari, *A Shortest Tree Algorithm for Optimal Assignment Across Space and Time in a Distributed Processor System*, SE-7(6), pp. 583–589, 1981.
[12] H. Thomas, et al., *Introduction to Algorithms*, The MIT Press, 2001.
[13] J.D. Ullman, *NP-Complete Scheduling Problems*, J. Computing System Science, vol. 10, pp. 384-393, 1975.
[14] T.L. Adam, K.M. Chandy, and J.R. Dickson, *A Comparison of List Schedules for Parallel Processing Systems*, Comm. ACM, vol. 17, pp. 685-689, 1974.
[15] O. Sinnen and L. Sousa, *List Scheduling: Extension for Contention Awareness and Evaluation of Node Priorities for Heterogeneous Cluster Architectures*, Parallel Computing, vol. 30, no. 1, pp. 81-101, Jan. 2004.
[16] Y. Zhao, et al., *Grid Middleware Services for Virtual Data Discovery, Composition, and Integration*, In 2nd Workshop on Middleware for Grid Computing, October 18, 2004, Toronto, Canada.
[17] X. Liu, et al., *Approaches to support various types of traffic in WDM networks*, OFC/NFOEC 2007, OThQ4.
[18] S. Darbha and D.P. Agrawal, *Optimal Scheduling Algorithm for Distributed Memory Machines*, in IEEE Trans. on Parallel and Distributed Systems, vol. 9, no. 1, pp. 87-95, January 1998.
[19] A. Geras, *A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors*, in J. of Parallel and Distributed Computing, Vol. 16, No.4, pp.276-291, 1992.