Dual-phase Just-in-time Workflow Scheduling in P2P Grid Systems

Sheng Di, Cho-Li Wang
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
{*sdi, clwang*}*@cs.hku.hk*

***Abstract*—This paper presents a fully decentralized just-in-time workflow scheduling method in a P2P Grid system. The proposed solution allows each peer node to autonomously dispatch inter-dependent tasks of workflows to run on geographically distributed computers. To reduce the workflow completion time and enhance the overall execution efficiency, not only does each node perform as a scheduler to distribute its tasks to execution nodes (or resource nodes), but the resource nodes will also set the execution priorities for the received tasks. By taking into account the unpredictability of tasks' finish time, we devise an efficient task scheduling heuristic, namely *dynamic shortest makespan first* (DSMF), which could be applied at both scheduling phases for determining the priority of the workflow tasks. We compare the performance of the proposed algorithm against seven other heuristics by simulation. Our algorithm achieves 20%∼60% reduction on the average completion time and 37.5%∼90% improvement on the average workflow execution efficiency over other decentralized algorithms.**

***Keywords*-P2P Grid system; just-in-time workflow scheduling; dual-phase model**

I. Introduction

Peer-to-Peer (P2P) Grid systems have been used by researchers for solving complex scientific problems by exploiting large amount of geographically dispersed resources on the Internet. By integrating P2P techniques to Grid, we are able to discover resources across multiple domains and aggregate large amount of resources in a fully decentralized manner. Such constructive synergies enable more efficient resource sharing for using idle resources at the edge of Internet. In recent years, we see many P2P Grid systems have been developed, including Condor-Flock P2P [1], P2PGrid [2], BonjourGrid [3], Alchemi [4] and Harmony [5].

Unlike independent tasks or batch-of-task (BoT) jobs, scientific workflows are usually built with complex dependencies among tasks. The delay of individual task's completion time could significantly hinder the execution progress of workflow. In the P2P Grid system, each node could be a resource consumer and also a resource provider. They usually self-organize and perform task scheduling autonomously, which is very different from traditional Grids that schedules workflow tasks by a centralized scheduler or a group of coordinated schedulers. Due to the lack of global information and centralized control, the overall execution efficiency is hard to optimize. Even worse, the conflicting scheduling decisions among uncoordinated individual schedulers may cause unpredictable delay of tasks' execution on some hotspots, consequently resulting in volatile critical paths in workflows over time. The intermittent node arrival and departure (known as *node churning problem*) post greater challenges to optimizing scheduling efficiency.

Existing workflow scheduling models mainly include *full-ahead scheduling* and *just-in-time scheduling* [6]. The first model is fully static, as it schedules the whole workflow before the actual execution starts. Most workflow scheduling algorithms (e.g., [7], [8]) follow the full-ahead scheduling model. They perform well in a more stable execution environment such as the traditional Grid systems. In comparison, the second model is more dynamic, as the task of any workflow will not be scheduled until its dependent tasks are all finished. This model could be more suitable for dynamic P2P Grid systems since the scheduler can make the scheduling decisions upon analyzing the latest progress of workflows (e.g. each workflow's changed critical path as well as critical tasks) and updated resource information.

To further improve the system-wide workflow scheduling efficiency, we propose a dual-phase just-in-time scheduling model which makes use of a more effective heuristic, namely *dynamic shortest makespan first* (DSMF). Each task will experience two scheduling phases before its final execution. The first phase is performed at the submission site. The scheduler will handle the workflow with the shortest remaining makespan first and preferentially dispatch its tasks with longer estimated execution time to the best-selected resource nodes. At the second phase, the resource nodes will further prioritize the waiting ready tasks whose workflows have the shorter remaining makespans. Such a design could distinguish different workflows based on their structures to avoid long waiting time added to the short workflows, significantly improving the average efficiency which is especially important in P2P Grid systems.

The proposed scheduling method is integrated as part of a peer-to-peer resource discovery framework to aggregate the latest resource states from other peer nodes for making scheduling decisions. Traditional solutions, such as DDWS-RL [9] relying on Globus MDS [10] or DHT-based protocol [11], suffer complex maintaining cost. We implemented a lightweight *mixed gossip protocol*, by combining *epidemic*

0190-3918/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPP.2010.31

IEEE computer society

gossip [12] and *aggregation gossip* [13] protocols, which can adapt to the dynamic environment very well.

To evaluate the proposed scheduling algorithms, we built a wide-area-network testbed based on the *Brite* tool [14] and *Waxman* model [15]. We carried out the simulation of asynchronous task execution based on the *Peersim* tool [16]. We show that our dual-phase just-in-time scheduling model with DSMF heuristic outperforms all the other just-in-time scheduling algorithms using min-min, max-min, sufferage or decentralized HEFT. It also performs better than some full-ahead scheduling algorithms (such as HEFT [7]). By DSMF, the average workflow completion time gets reduced by 20%∼60% while obtaining improvement on execution efficiency by 37.5%∼90%, which could also tolerate the dynamic environment. They show no notable performance degradation under the ratio of 20% churning nodes in every periodic scheduling interval.

The rest of the paper is organized as follows. We formally model the decentralized P2P Grid workflow scheduling problem in Section II and describe our dual-phase just-in-time workflow scheduling algorithm in Section III. In Section IV, we evaluate our algorithm. The related works are analyzed in Section V. We conclude the pros and cons of our solution and present the future work in Section VI.

II. Workflow Scheduling Model

A. Preliminaries

A workflow can be represented as a directed acyclic graph (DAG), where the vertices and the directed edges represent the tasks and task dependencies respectively. Task dependencies are formed because some tasks will generate data that serves as input to one or more subsequent tasks. In particular, if task A is dependent on task B, B is called A's precedent and A is B's successor. Let $Pre(\cdot)$ denote precedent task set and $Suc(\cdot)$ the successor task set, then $B \in Pre(A)$ and $A \in Suc(B)$. In a DAG, a task without precedent is called an *entry task*, and a task without successor is an *exit task*. For any workflow with more than one entry task, another newly added zero-cost task which connects all the original entry tasks can serve as the unique entry task of the workflow. Likewise, any workflow with many exit tasks can also be converted to the one with unique zero-cost exit task. In the rest of content, we assume every workflow has only one entry task and one exit task.

In order to complete a workflow, every task in it has to be mapped to a resource node for execution. At any time, only the tasks whose precedents are all finished can be scheduled and these unscheduled tasks are called *schedule-points* of the workflow.

The whole procedure of P2P workflow scheduling is illustrated in Fig. 1. Whenever one workflow is submitted to a node (a.k.a. *home node* or *scheduler node*) (step 1 in Fig. 1), the initial schedule-point is set as its entry task (step 2). Whenever a task is allowed to run (i.e. all its precedents are completed), the home node performs a selection policy (a.k.a. workflow scheduling algorithm) to determine which resource node should execute the task (step 2 & 6). Every task scheduled will be added to the resource node's *ready set* and wait for the execution (step 3 & 7). Meanwhile, all dependent data of the ready task should be transmitted from the nodes on which its precedents were computed to the selected resource node (step 8). Once some tasks are finished at the resource nodes, only tasks in the ready set which have received data from all precedents will be selected for execution (step 4 & 9).

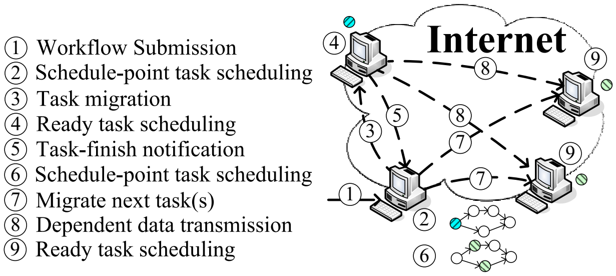


Figure 1. Use-case of Workflow Scheduling

B. Problem Formulation

Without loss of generality, suppose there are n peer nodes in the whole system, denoted as p_i (i=1,2,...,n), each serving as both scheduler node and resource node. The computing capacity of each node p_i is denoted by c_i (evaluated by the number of instructions processed per second). At each node p_i (i=1,2,...,n), we assume w_i workflows are submitted, which are denoted as f_{ij}, where j=1,2,...,w_i. We also define the *task set* of the workflow f_{ij} to be $T(f_{ij})$, whose kth task is denoted as $t_k^{(ij)}$, where k=1,2,...,$|T(f_{ij})|$. Each task's load (denoted as $l_k^{(ij)}$) is evaluated by the number of its instructions. On each node p_i, a *resource set* (denoted as *RSS*(p_i)) is used to store the aggregated resource states and *ready set* (denoted as *RDS*(p_i)) is used to keep the ready tasks. The summation of loads of all ready tasks (including running tasks) at the resource node p_r (r=1,2,...,n) refers to the node's total load (denoted by l_r).

Let *ct*(f_{ij}) denote the *real completion time* of workflow f_{ij}. *ct*(f_{ij}) is also known as response time, which is counted from the start of its entry task until f_{ij}'s exit task completes. To evaluate the execution efficiency of the workflow f_{ij}, we compare its real completion time with the *expected finish-time* $eft(f_{ij})$. We define the efficiency of the workflow f_{ij} (denoted by $e(f_{ij})$) as Equation (1).

$$\begin{aligned} e(f_{ij}) &= \tfrac{eft(f_{ij})}{ct(f_{ij})}, where \\ eft(f_{ij}) &= \sum_{t_k^{*(ij)} \in T(f_{ij})} (eet(t_k^{*(ij)}) + ett(t_k^{*(ij)})) \end{aligned} \qquad (1)$$

The critical path from a workflow's entry task to its exit task determines the expected finish-time $eft(f_{ij})$. We call the tasks along the critical path *critical workflow tasks*, each denoted as $t_k^{*(ij)}$. Let *eet*($t_k^{*(ij)}$) and $ett(t_k^*(ij))$ denote the expected execution time of task $t_k^{*(ij)}$ and the expected data aggregation time for task $t_k^{*(ij)}$ to collect data from its precedents. The expected finish-time of the workflow f_{ij} is equal to the sum of $eet(t_k^{*(ij)})$ and $ett(t_k^{*(ij)})$ of all of its critical tasks. Because the expected finish-time is used as a baseline for comparing with the actual performance, both *eet*($t_k^{*(ij)}$) and $ett(t_k^{*(ij)})$ are estimated using the system-wide average node capacity and average network bandwidth.

Our objective is to reduce the average workflow completion time (ACT) and to enhance the average efficiency (AE) based on a fully decentralized workflow scheduling algorithm. The two objective functions are defined as follows.

$$ACT = \frac{1}{\sum_{i=1}^{n} w_i} \sum\nolimits_{i=1}^{n} \sum\nolimits_{j=1}^{w_i} ct(f_{ij}) \qquad (2)$$

$$AE = \frac{1}{\sum_{i=1}^{n} w_i} \sum\nolimits_{i=1}^{n} \sum\nolimits_{j=1}^{w_i} e(f_{ij}) \qquad (3)$$

III. Dual-phase Just-in-time Workflow Scheduling

A. Overview

The whole scheduling process will be conducted in two phases. At the first scheduling phase, all home nodes (a.k.a. scheduler nodes) will dispatch their workflow tasks to resource nodes and add the tasks to the resource nodes' ready set. At the second phase, the resource nodes will further schedule the tasks waiting in their ready set for execution. The proposed solution makes use of a mixed gossip protocol to aggregate the latest resource states from other peer nodes for determining the site(s) of executing its schedule-point tasks once they become available for execution. A *dynamic shortest makespan first* (DSMF) heuristic is applied at both phases for determining the priority of workflow tasks. (to be discussed in Section III.C)

Fig. 2 shows the basic idea of the dual-phase scheduling method. At each home node, there could be many workflows submitted. Based on DSMF, at the scheduler nodes, the schedule-point tasks from the workflows that have shorter makespans will be preferentially processed, with the aim to minimize the average waiting time of workflows.

As execution proceeds, more schedule-point tasks become activated at the home node. Within one workflow, the estimated remaining path's execution time (i.e., RPM to be defined later) of each schedule-point task is computed. The schedule-point task that has the longest remaining path's execution time toward the exit task will be scheduled with higher priority. Each home node will dispatch their schedule-point tasks to the resource nodes which can finish their execution the earliest based on the latest resource states aggregated from other peer nodes. The scheduling policy is similar to the static HEFT algorithm [7], yet in our approach, the schedule-point task's finish-time can be dynamically estimated based on the real-time state of resource node's load and the network status. This achieves more proper resource allocation and higher adaptability. At the second phase, upon the completion of a task, the resource node will select another task from its ready set such that the task's workflow has the shortest remaining makespan for execution, following the DSMF guideline. The details are discussed in Section III.D.

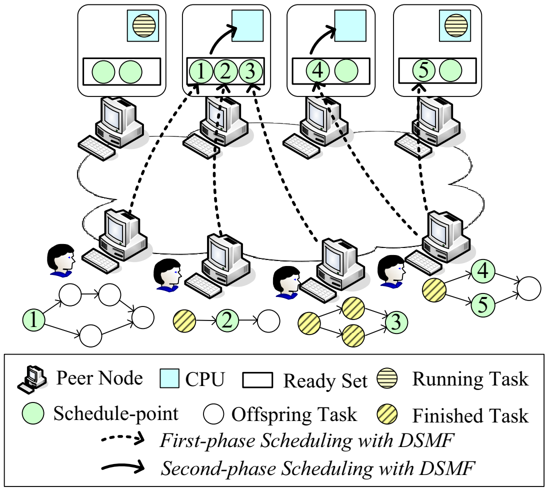


Figure 2. Dual-Phase Workflow Scheduling Model

B. Mixed Gossip Protocol

The proposed scheduling method is integrated as part of a peer-to-peer resource discovery framework for supporting the dynamic task scheduling work. Each node needs to aggregate the latest resource information to facilitate the subsequent task scheduling process. This information could be split into two categories, state information and statistics. We devise a mixed gossip protocol, which integrates the *epidemic gossip protocol* and *aggregation gossip protocol*, to synthetically aggregate the two types of information on each node required by the task scheduling.

The epidemic gossip protocol is responsible for collecting the updated state information, such as the nodes' states. Each node periodically pushes the messages about its known state, including its latest total load (l_i), its capacity (c_i) and those of other nodes collected, to only a few neighbors. The neighbors of node p_i are randomly selected at every propagation cycle based on the Newscast model [13] and the fan-out degree (i.e., the number of neighbors) is limited to $\log_2(n)$ to avoid excessive network traffic. Each node p_i will maintain a resource node set ($RSS(p_i)$) containing these messages while the space complexity $|RSS(p_i)|$ on average per node is $O(\log_2(n))$ (to be shown in Section IV).

The estimation of network status can be conducted using landmark based mechanism [17]: each node monitors its network links connecting to $\log_2(n)$ landmarks, and duly propagates the list containing its network conditions to other nodes via epidemic gossip protocol. Thereafter, the global network conditions can be estimated at every node.

The aggregation gossip protocol is responsible for aggregating global statistics, including average node capacity, average bandwidth, and make them known to every node. Particularly, these statistics will be used for the estimation of $eft(f_{ij})$. For each metric, each node just continually computes a new average using its updated value and the means calculated by neighbors, then push it back to neighbors.

Both epidemic gossip protocol and aggregation gossip protocol have been proven very efficient - low cost and exponential converging speed are obtained [12], [13].

C. Dynamic Shortest Makespan First (DSMF) Heuristic

DSMF first handles the workflow with the shortest remaining makespan at any time and priorities its tasks in the scheduling procedure. We will describe how to estimate makespan for each workflow in this section, and present details with pseudo-codes in next section.

We denote $ST(t_k^{(ij)},p_h)$ and $FT(t_k^{(ij)},\ p_h)$ to be the start time and completion time of task $t_k^{(ij)}$ if it is to be executed at node p_h. They can be estimated based on Equation (5) and Equation (6) before the tasks are actually allocated to resource nodes. In Equation (5), $R(t_k^{(ij)},p_h)$ stands for the queuing delay that $t_k^{(ij)}$ has to wait before p_h becomes available for executing this task, while $et(t_k^{(ij)},p_h)$ represents $t_k^{(ij)}$'s execution time on p_h. $R(t_k^{(ij)},p_h)$ can be conservatively estimated via the total load (i.e. l_h, related to the running task and waiting tasks) of p_h and its capacity (c_h). The $cost(t_{k'}^{(ij)},t_k^{(ij)})$ denotes the time of aggregating dependent data from $t_k^{(ij)}$'s precedent $t_{k'}^{(ij)}$. Since the data transmissions toward the execution nodes could be performed concurrently on the network, the slowest one will determine task's final dependent transmission delay, a.k.a. longest transmission delay ($LTD(t_k^{(ij)})$) as defined in Equation (4).

$$LTD(t_k^{(ij)}) = \max_{t_{k'}^{(ij)} \in Pre(t_k^{(ij)})} (FT(t_{k'}^{(ij)},p_{h'}) + cost(t_{k'}^{(ij)},t_k^{(ij)})) \tag{4}$$

$$ST(t_k^{(ij)},p_h) = \max(\ R(t_k^{(ij)},p_h), LTD(t_k^{(ij)})) \tag{5}$$

$$FT(t_k^{(ij)},p_h) = ST(t_k^{(ij)},p_h) + et(t_k^{(ij)},p_h) \tag{6}$$

In general, the data transmission time could be estimated as $datasize(t_{k'}^{(ij)},t_k^{(ij)})/bandwidth(p_{h'},p_h)$, where $p_{h'}$ refers to the node completing the precedent task $t_{k'}^{(ij)}$. Then, the start time $ST(t_k^{(ij)},p_h)$ could be determined by either the longest transmission delay or the longest queuing delay on the target node, because such two delays could overlap in time.

Apparently, it is impossible to precisely predict the remaining execution time of the workflow f_{ij} as the actual locations for performing the schedule-point task's subsequent tasks have not yet been decided. In particular, $et(t_k^{(ij)},p_h)$, $cost(t_{k'}^{(ij)},t_k^{(ij)})$, and $R(t_k^{(ij)},p_h)$ of all the offspring tasks remain unknown. Therefore, we propose a special metric *rest path makespan* (*RPM*), denoted by $RPM(t_k^{(ij)})$, to estimate the longest execution time along the paths from the schedule-point $t_k^{(ij)}$ to f_{ij}'s exit task.

By expanding the definition of $FT(t_k^{(ij)},p_h)$ based an Equation (6), we derive a new recursive function:

$$FT(t_k^{(ij)},p_h) = \max(R(t_k^{(ij)},p_h), LTD(t_k^{(ij)})) + et(t_k^{(ij)},p_h) \tag{7}$$

Thus, to predict the remaining execution time of the workflow f_{ij}, we can recursively expand Equation (7) from its exit task backward (or upward) to all schedule-points to deduce their *rest path makespans* (*RPM*). Given a workflow task $t_k^{(ij)}$, the $RPM(t_k^{(ij)})$ consists of two parts: (1) $t_k^{(ij)}$'s execution time and transmission/waiting delay on resource node p_h and (2) the execution time of $t_k^{(ij)}$'s offspring tasks. The transmission/waiting delay could be computed based on the resource node states (i.e. l_h and c_h) collected by the *epidemic gossip protocol*, while each of its offspring tasks t_o will be computed based on the expected execution time ($eet(t_o)$) and expected data transmission time ($ett(t_o)$). Both $eet(t_o)$ and $ett(t_o)$ are approximated using the system-wide average node capacity and average network bandwidth which are dynamically collected by the *aggregation gossip protocol*. Thus, each workflow's remaining makespan can be expressed as Equation (8), where $spset(f_{ij})$ denotes f_{ij}'s current schedule-point set.

$$ms(f_{ij}) = \max_{t_k^{(ij)} \in spset(f_{ij})} (RPM(t_k^{(ij)})) \tag{8}$$

D. Dual-Phase Just-in-Time Scheduling Algorithm

Algorithm 1 shows the pseudo-code which autonomously runs on each scheduler node (denoted by p_s) at the first scheduling phase, while Algorithm 2 shows that of the second-phase scheduling on each resource node (denoted by p_r). As every node in the P2P Grid system could serve as both the scheduler node and resource node, the two algorithms are actually running concurrently at each node.

The design of Algorithm 1 aims at reducing the waiting time of workflows and also tries to minimize each workflow's completion time, synthetically improving the execution efficiency. We will select the workflow with the smallest $ms(f_{ij})$ to schedule first at each home node, similar to the shortest job first scheduling policy which can minimize the overall waiting time. In Line 2∼7, we first calculate the *RPM* of schedule-point tasks for each submitted workflow based

Algorithm 1 Workflow Scheduling on Scheduler Node

```
Notice: We show the code executed on scheduler node $p_s$
Input: workflows ($f_{sj}$, $1 \le j \le w_s$; resource node set ($RSS(p_s)$);
Output: Schedule tasks from $T(p_s)$ to the nodes selected from $RSS(p_s)$.
 1: while (TRUE) do
 2:   for (each workflow $f_{sj}$, $1 \le j \le w_s$) do
 3:     for (each schedule-point task $t_k^{(sj)}$ in $spset(f_{sj})$) do
 4:       Calculate $RPM(t_k^{(sj)})$ based on Equation (7);
 5:     end for
 6:     Compute workflow $f_{sj}$'s makespan $ms(f_{sj})$ via Equation (8);
 7:   end for
 8:   Sort workflows at $p_s$ in ascending order w.r.t. their remaining
      makespans;
 9:   if ($RSS(p_s) \neq \emptyset$)) then
10:     for (each $f_{sj}$ in the sorted list) do
11:       Sort $spset(f_{sj})$ in descending order w.r.t. RPM;
12:       for (each task $\tau \in$ the sorted $spset(f_{sj})$) do
13:         Select resource node $p_r$ from $RSS(p_s)$ via Formula (9);
14:         Migrate $\tau$ from $p_s$ to $p_r$ (kept in ready set $RDS(p_r)$);
15:         Update $p_r$'s state record in $RSS(p_s)$;
16:       end for
17:     end for
18:   end if
19:   Sleep and wait for the next scheduling cycle;
20: end while
```

on Equation (7), then we estimate the remaining makespan of each workflow based on Equation (8). These workflows are sorted in ascending order (line 8) with respect to their remaining makespans ($ms(f_{sj})$), and are handled one by one in this order at line 9∼18. Among the schedule-point tasks of a specific workflow, the one with the largest $RPM(t_k^{(ij)})$ will be handled with the highest priority. Therefore, we sort its schedule-point tasks in descending order with respect to their *RPM* (Line 11), then schedule them in order. Both the calculation of *RPM* and $ms(f_{ij})$ make use of the system-wide average node capacity and average network bandwidth periodically collected by the *aggregation gossip protocol* to reflect the most updated system status.

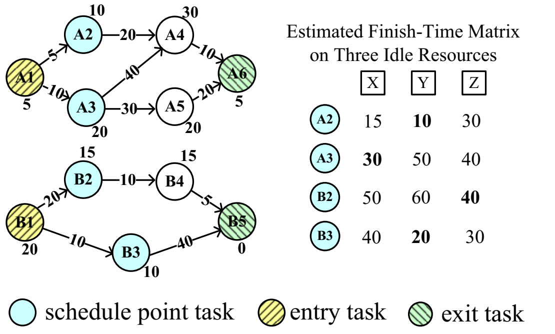
The selection of target node for completing task τ is based on Formula (9) below. We choose a resource node p_r from $RSS(p_s)$, which can achieve its earliest finish time.

$$target\ node = \arg \min_{p_h \in RSS(p_s)} (FT(\tau, p_h)) \qquad (9)$$

According to Equation (6), the finish time of task τ performed at resource node p_r (i.e., $FT(\tau, p_r)$) could be affected by the task waiting time $R(\tau, p_r)$ (i.e., $\frac{l_h}{c_h}$), dependent data aggregation time, and τ's execution time at p_r. Intuitively, we will select a less loaded or more powerful node for executing the task. In addition we consider the node locality issue for minimizing the network delay during data aggregation based on an unified equation, i.e., $FT(\tau, p_r)$. As the resource node is selected, the task τ will be migrated to the node together with its rest path makespan and its workflow's makespan.

Indeed, such a "finish-earliest" guideline has been used commonly by traditional Grids (e.g., the static HEFT algorithm). But it may not be suitable for rather dynamic P2P Grid systems which adopted a fully decentralized scheduling method. Improper task scheduling may cause contention on resources, which could severely delay tasks' finish times on some hotspot nodes. The node churning problems further add challenges on the accuracy of the estimation of finish time used to guide the scheduling process. Our just-in-time scheduling approach proactively estimates workflows' changeable remaining makespans which can adapt to such a volatile environment well. Based on each node's limited space complexity ($O(\log_2(n))$) of the randomly aggregated $RSS(p_s)$, the selection of candidate resource nodes can effectively mitigate the execution hotspot problem.

We give an example to illustrate our algorithm in Fig. 3: Suppose there are two workflows submitted to node p_s, whose DAGs are shown in Fig. 3. The numerical values marked on the vertices and edges represent the estimated execution time and data transmission time predicted based on average node capacity and average bandwidth. The tasks to be scheduled are A2, A3, B2, and B3 and there are three resources (denoted by X, Y, and Z) known by the node p_s. The matrix represents the estimated finish-time for each task to run on the three resource nodes. In terms of Equation (5) and Equation (6), we can compute RPM(A2)=80, RPM(A3)=115, RPM(B2)=65, and RPM(B3)=60. Therefore, the makespans of the two workflows are 115 and 65 respectively. According to DSMF, the scheduling order is thus B2, B3, A3, A2. In comparison, the min-min and max-min algorithms will respectively select A2 and B2 first, and then respectively select the next earliest-finish and latest-finish task after updating the state of resources (node Y and node Z), and so on. The HEFT algorithm will choose A3, A2, B2, and B3 one by one, due to their decreasing order of RPM.



Estimated Finish-Time Matrix on Three Idle Resources

	X	Y	Z
A2	15	**10**	30
A3	**30**	50	40
B2	50	60	**40**
B3	40	**20**	30

Figure 3. A Use-case with Two Workflows on a Scheduler Node

During the second phase, the waiting tasks in each resource node's ready set will be chosen for execution according to Algorithm 2, which applies the same heuristic. Note that the tasks in the ready set are likely to correspond to different workflows from different home nodes, thus the task to be executed is supposed to be the one whose workflow has the shortest remaining makespan (i.e. Formula (10), where

f_k refers to the task τ's corresponding workflow).

$$target\ task = \arg \min_{\tau \in RDS(p_r) \& \tau \in T(f_k)} (ms(f_k)) \quad (10)$$

Algorithm 2 Ready Task Scheduling on Resource Node

Notice: We show the code executed on resource node p_r

Input: $RDS(p_r)$, the rest path makespan (RPM) of every task in $RDS(p_r)$ and the makespans of their corresponding workflows

Output: Assign local computing resource to one task from p_r's ready set.

```
Sort tasks in $RDS(p_r)$ in ascending order of the makespans of their workflows;
From $RDS(p_r)$, Select the tasks based on Formula (10);
if (the number of tasks selected $> 1$) then
    Choose the task with the longest RPM;
end if
Assign the local resource (CPU) to the selected task;
Waiting for the task-finish signal;
```

E. Time Complexity of DSMF

The time complexity of Algorithm 1 is analyzed as follows. From Line 2∼7, since Line 3∼5 could be performed by calculating the rest path makespans of offspring tasks reversely rooted from the exit task to each schedule-point task, each edge of the DAG just needs to be checked (or passed) at most once. Thus, the time complexity of Line 3∼5 is $O(\theta(f_{sj}))$, where $\theta(f_{sj})$ denotes the number of edges in the DAG of workflow f_{sj}. As the time complexity of Line 6 is $|spset(f_{sj})|$, the complexity of Line 2∼7 is $O(\sum_{j=1}^{w_s} \theta(f_{sj}))$ = $O(w_s \cdot \alpha)$, where α = $\max_{j=1,2,\ldots,w_s}(\theta(f_{ij}))$.

The complexity of Line 8 is $O(w_s \cdot \log(w_s))$. The time complexity of Line 11 is $O(|spset(f_{sj})| \cdot \log(|spset(f_{sj})|))$. Accordingly, the complexity of Line 9∼18 could be estimated as $O(w_s \cdot \max(\beta, \gamma))$, where β = $\max_{j=1,2,\ldots,w_s}(|spset(f_{sj})|)$, and γ = $\beta \cdot |RSS(p_s)|$.

As a result, time complexity of Algorithm 1 is $O(w_s \cdot \max(\alpha, \gamma, \log(w_s)))$. The complexity should be satisfied in that $|RSS(p_s)|$ is $O(\log(n))$ per node. The time complexity of Algorithm 2 is $O(|RDS(p_r)| \cdot \log(|RDS(p_r)|))$.

IV. Performance Evaluation

A. Experimental Setting

We carry out the simulation via the *Peersim* tool [16] over an emulated Internet environment constructed by *Brite* tool [14] and *Waxman* model [15]. Each node receives one or more workflows initially. For each workflow, the fan-out degree of each task ranges from one to five. The total experimental time is 36 hours. The scheduler is activated every 15 minutes. Each node forwards the state information to its neighbors every gossip cycle (five minutes) with message's TTL (the max number of hops) set as four. Each node maintains connections to $\log_2(n)$ neighbors, but will be reselected at the end of each gossip cycle.

In our design, each message carries about 80 bytes data payload and 20 bytes header information (about 100 bytes in total). Suppose system scales up to 10^6 nodes, then each node needs to communicate with other 20 (=$\log_2(10^6)$) neighbors. So the average amount of network traffic generated at each node is 20×100 *bytes*=$2K$ *bytes* per gossip cycle. The overhead is insignificant as compared to the task migration cost and dependent data transmission overhead. We assume each node owns unique CPU, which is non-sharable and non-preemptive. That is, only one task can be executed at any time. Other settings are shown in Table I.

Table I
Experimental Setting

Parameter	Value
# of nodes	200 ∼ 2000
# of tasks per workflow	2 ∼ 30
the computing amount per task	100 ∼ 10000 *MI*
image size per task	10 ∼ 100 *Mb*
dependent data size to be transmitted	100 ∼ 10000 *Mb*
network bandwidth	0.1 ∼ 10 Mb/s
heterogeneous node's capacity	1,2,4,8,or 16 *MIPS*
Communication-to-Computation Ratio (CCR)	0.16∼16

We compare the performance of the dual-phase scheduling algorithm (denoted as DSMF) against seven other scheduling algorithms. Among them, *Heterogeneous Earliest Finish Time* (HEFT) [7] and the self-implemented *Shortest Makespan First* (SMF) adopt static full-ahead scheduling model. HEFT is a popular list scheduling algorithm for heterogeneous system. It uses a recursive procedure to compute the rank for each task, which is similar to the way we compute RPM. SMF gives higher priority to the workflows with shorter makespans. For tasks within a workflow, SMF preferentially schedules the tasks with longer RPMs. Since the scheduling work of the two algorithms is centrally performed before the execution starts, the resource nodes will just execute the ready tasks via the FCFS policy. The two algorithms are selected as the base for comparison.

We also compare our DSMF heuristic with five other classical heuristics used in decentralized scheduling algorithms, namely min-min, max-min, sufferage, decentralized HEFT (DHEFT), and dynamic shortest deadline first (DSDF). We implemented all of them in our dual-phase scheduling framework, and make use of the mixed gossip protocol for the aggregation of resource information. The decentralized versions of min-min, max-min, and sufferage algorithms are modified from the work described in [18]. Basically we apply min-min, max-min, and sufferage heuristics in the first phase scheduling, while in the second phase scheduling (i.e. ready set scheduling) we adopt *shortest task first* (STF), *longest task first* (LTF), and *largest sufferage first* (LSF) respectively. We made these modifications since the original version of min-min, max-min, and sufferage algorithms in [18] did not carry the second phase scheduling.

Lastly, the decentralized HEFT (DHEFT) applies a longest RPM first policy at both scheduling phases, while DSDF schedules tasks with the shortest *deadlines* (defined as the difference between its rest path makespan and its workflow's makespan) to run first at both phases.

B. Experimental Result

We arrange the evaluation under both static environment (i.e., without churning nodes) and dynamic environment (i.e. nodes may arbitrarily join/leave the system).

Fig. 4 reports the system throughput of the eight algorithms running in a static environment. In this experiment, there are 1000 nodes, each receiving three workflows and $l_k^{(ij)}$=100∼10000 million instructions (MI) and $datasize(t_k^{(ij)})$=10∼1000. The CCR is about 0.16. Without further note, following tests will also use the above setting.

We observe that both HEFT and DHEFT achieve the lowest system throughput in the beginning stage (i.e., during the first 12 hours) though it could firstly complete all the workflows. This is because HEFT adopts the rather static approach with global information but overlook the different makespans among workflows, which may probably cause long waiting times to the workflows with short makespans.

SMF performs the best as it preferentially processes the workflows with shorter makespans, which will lead to much higher system throughput especially during the beginning period. The proposed DSMF heuristic can achieve very good system throughput, only second to SMF.

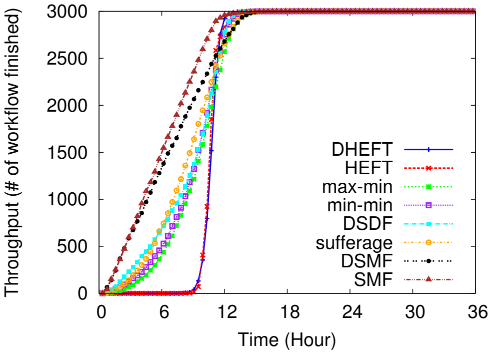


Figure 4. Throughput of Workflows in Static P2P Grid System

Fig. 5 and Fig. 6 present the average completion time (i.e. Formula (2)) and average execution efficiency (i.e. Formula (3)) of workflows under the same static setting. The converged average finish-time of min-min, max-min, sufferage, and decentralized HEFT are 31977, 33495, 30321, and 30728 respectively, compared to the 32874, 33746, 32781, and 32636 in their original versions using FCFS on the second-phase scheduling at resource nodes. Thus, FCFS is not suggested to take over the ready task scheduling work. Two figures show the full-ahead SMF still keeps the best performance, i.e. the lowest average workflow finish-time and highest average efficiency. The outcome of DSMF approaches SMF and outperforms the other decentralized algorithms and even the full-ahead HEFT with 20%∼60% reduction on average completion time and 37.5%∼90% improvement on average efficiency.

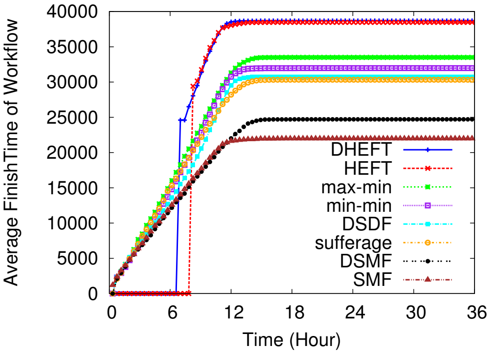


Figure 5. Average Finish-time of Workflows in Static P2P Grid System

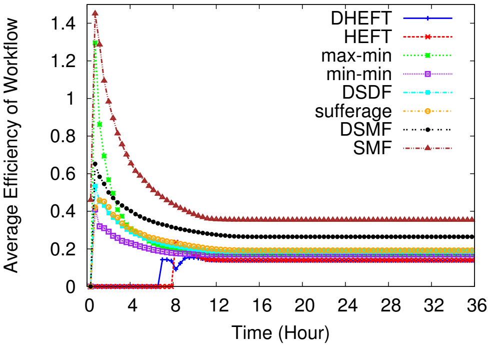


Figure 6. Average Efficiency of Workflows in Static P2P Grid System

Through our experiments, we also noted that the performance would be influenced to different extents of resource competition. We define *load factor* as the average number of submitted workflows per node. The increase of load factor implies the whole system is under a higher workload, thus higher resource competition. Fig. 7 and Fig. 8 compare the average finish-time and average execution efficiency when we scale the load factor from 1 to 8. We can observe that our DSMF adapts better to the situation when resource competition is getting more serious (i.e., load factor = 6,7,8), owing to its dynamic analysis on the remaining makespan for each workflow. DSMF outperforms the rest of algorithms on the average finish-time, when load factor is 7 and 8.

Fig. 9 and Fig. 10 study the performances affected by CCR by using different combinations of average task loads and data sizes. We could estimate that the CCRs in the four cases are respectively 1.6, 0.16, 1.6, and 16. Through these two figures, we found SMF could still achieve good results in most cases, while our DSMF remains the winner among all decentralized algorithms with different CCRs.

We further study the effectiveness of the mixed gossip protocol built on the P2P network. Fig. 11 (a) shows the

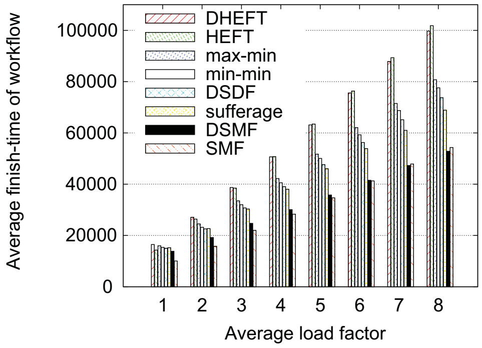


Figure 7. Average Finish-Time of Workflows under Different Load Factor

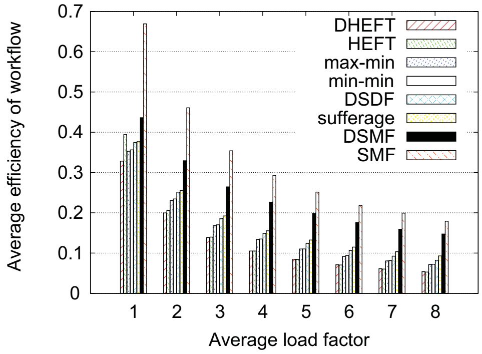


Figure 8. Average Efficiency of Workflows under Different Load Factor

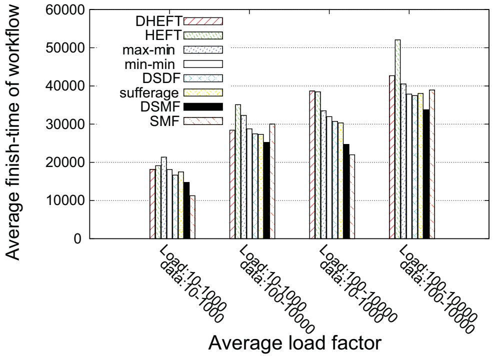


Figure 9. Average Finish-Time of Workflows under Different CCRs

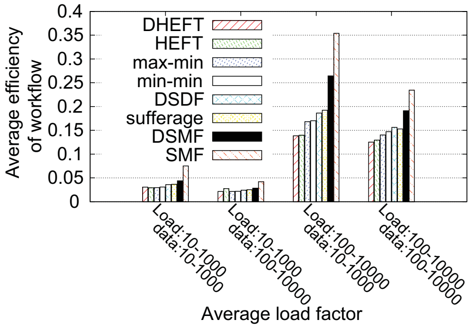
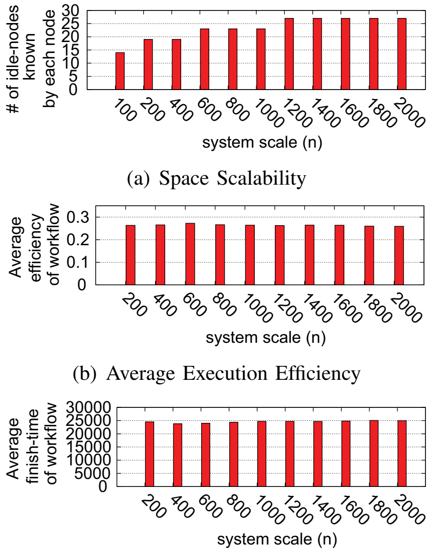


Figure 10. Average Efficiency of Workflows under Different CCRs



(a) Space Scalability

(b) Average Execution Efficiency

(c) Average Finish-Time

Figure 11. System Scalability of DSMF

average number of resource nodes known by each node via the mixed gossip protocol, as we scale the system size. We could see that the number of "acquaintance nodes" are bounded by a small number (less than 30), even we extend the system size to 2000. This can greatly reduce the time complexity on each scheduler node when selecting the best resource node for each task, in addition to the benefit of saving memory space. Fig. 11 (b) and (c) report the average execution efficiency and finish-time of workflows with increasing scale. Our DSMF can consistently achieve quite stable results due to its fully decentralized design.

We also evaluated the performance of DSMF over a dynamic environment with churning nodes. We define the *dynamic factor (df)* to be the ratio of the number of churning nodes (i.e. joined/disconnected nodes) and that of the total number of nodes in every task scheduling interval. For example, if *df*=0.1 and the system scale is 1000 nodes, there will be 100 new nodes joined and 100 old nodes disconnected within every task scheduling interval. We just consider the dynamic cases where the churning nodes are not home nodes, because checkpointing-based solution is beyond the scope of this research. In this experiment, 500 out of 1000 nodes (which serve as both scheduler nodes and resource nodes) will permanently stay in the system, while the rest 500 nodes are allowed to dynamically join/leave based on different dynamic factors.

Fig. 12, Fig. 13 and Fig. 14 report the throughput, average finish-time and execution efficiency of workflows in the dynamic environment. We observe that the throughput of DSMF could be distinctly lower with the increasing value of *dynamic factor (df)*, but each successfully finished workflow keeps relatively stable finish-time and efficiency when $df \leq$

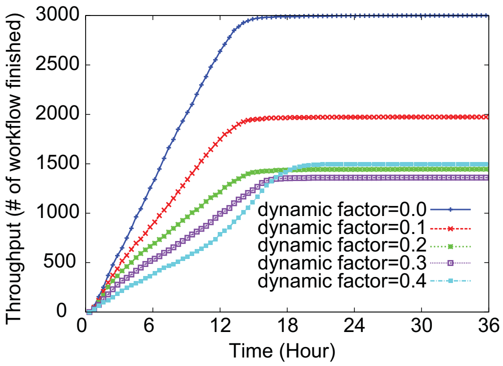


Figure 12. Throughput of DSMF in Dynamic Environment

0.2. The degraded throughput is mainly induced by the large-load tasks which cannot be finished quickly. This issue can be solved by automatically rescheduling the failed tasks at the scheduler nodes, which will be left to our future work.

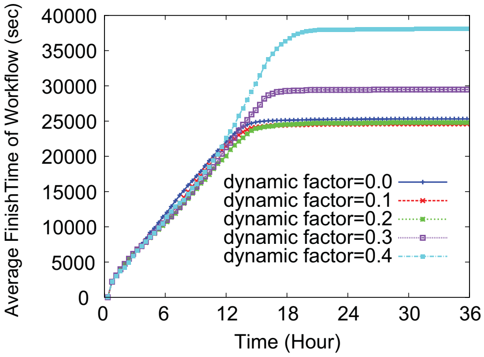


Figure 13. Average Finish-Time of DSMF in Dynamic Environment

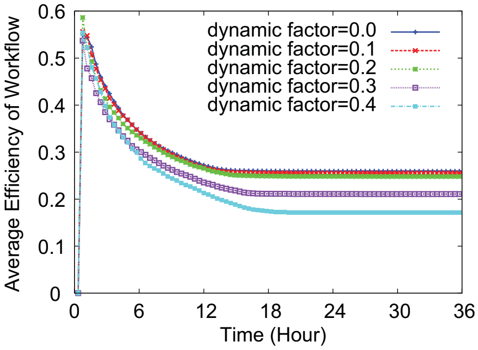


Figure 14. Average Efficiency of DSMF in Dynamic Environment

V. Related Work

There are several well-known decentralized workflow scheduling algorithms, including SwinDeW [19], DDWS-RL [9] and R. Ranjan's work [11].

The basic idea of SwinDew is to classify widely distributed web services into multiple virtual groups which have reciprocal dependencies and execute them one by one. However, it never considered how to optimize the workflow's execution efficiency by analyzing changeable makespans of workflows. SwinDeW-S [20] and SwinDew-G [21] are two extended versions of SwinDeW.

In DDWS-RL, every agent makes scheduling decisions by leveraging a machine learning strategy, namely reinforcement learning. Such a method is more suitable for the traditional grid systems rather than P2P Grid systems, in that it adopts a centralized resource discovery method (i.e., client-server or hierarchical model).

R. Ranjan et al. [11] proposed a fully-decentralized cooperative workflow scheduling based on DHT [22], [23]. Their work focuses on the scheduling coordination. There are two drawbacks: (1) DHT needs to be carefully maintained if there are many frequently churning nodes; (2) The workflow scheduling adopts a simple First-Come-First-Serve (FCFS) ticket-matching policy to assign resources, which is likely to suffer inferior workflow execution efficiency.

Indeed, more heuristics are based on centralized collection of global information. Two popular list scheduling algorithms, *Heterogeneous Earliest Finish Time* (HEFT) and *Critical-Path-on-a-Processor* (CPOP) were studied in [7]. Such algorithms adopted a two-step scheduling approach. Before the entry task's execution, every task will be set a upward (downward) rank estimated as the sum of communication time and execution time routing from this task to the exit task (entry task), and then be scheduled in order of decreasing ranks. Luiz et al. [24] improved HEFT's average workflow execution time by up to 20% via the lookahead method by estimating each task's upward ranks as well as those of its children. Obviously, all of them are unsuitable for the autonomous workflow scheduling in dynamic P2P Grid systems, because of their full-ahead model.

Another popular algorithm *Dynamic Critical Path for Grid* (DCP-G) [8] also takes critical path into account. They consider the locality issue to optimize the selection of critical path, which performs well under full-ahead model. This algorithm, however, is not suitable for dynamic P2P Grid systems because the full-ahead model is not adaptive to the delay of tasks caused by the dynamic environment.

VI. Conclusion and Future Work

This paper proposes an autonomic workflow scheduling strategy especially suitable for the dynamic P2P Grid systems. This algorithm is self-organized by each individual node and designed based on the dynamic estimation for the priority of tasks of given workflows. Through our designed mixed gossip protocol, the algorithm has high robustness and adaptability in resource discovery. With the dynamic estimation of workflow's changeable makespans and volatile critical tasks, it may effectively mitigate the efficiency loss caused by the control/data dependencies. We finally prove via simulation that DSMF outperforms other decentralized algorithms in different cases, such as different load factor, CCR, etc. Our algorithm also shows satisfactory average efficiency under dynamic situation. The throughput under our approach might somehow be degraded in dynamic environments. This issue can be solved by rescheduling the failed tasks, and this study will be our future work.

VII. Acknowledgments

This research is supported by Hong Kong RGC grant HKU 7179/09E and China 863 grant 2006AA01A111.

References

[1] A. R. Butt, R. Zhang, and C. Y. Hu, "A self-organizing flock of condors," *Journal of Parallel and Distributed Computing*, vol. 66, no. 1, pp. 145–161, January 2006.

[2] J. Cao, F. B. Liu, and C. Z. Xu, "P2pgrid: integrating p2p networks into the grid environment: Research articles," vol. 19, no. 7. Chichester, UK: John Wiley and Sons Ltd., 2007, pp. 1023–1046.

[3] H. Abbes, C. Cerin, and M. Jemni, "Bonjourgrid: Orchestration of multi-instances of grid middlewares on institutional desktop grids," in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–8.

[4] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Alchemi: A .net-based grid computing framework and its integration into global grids," Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Tech. Rep., December 2003.

[5] V. K. Naik, S. Sivasubramanian, D. Bantz, and S. Krishnan, "Harmony: a desktop grid for delivering enterprise computations," in *4th International Workshop on Grid Comp*, 2003.

[6] M. Wieczorek, R. Prodan, A. Hoheisel, M. Wieczorek, R. Prodan, and A. Hoheisel, "Taxonomies of the multi-criteria grid workflow scheduling problem," Tech. Rep., 2007.

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

[8] M. Rahman, S. Venugopal, and R. Buyya, "A dynamic critical path algorithm for scheduling scientific workflow applications on global grids," in *In Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing*, 2007.

[9] J. Yao, C.-K. Tham, and K.-Y. Ng, "Decentralized dynamic workflow scheduling for grid computing using reinforcement learning," in *14th IEEE International Conference on Networks*, vol. 1, February 2007, pp. 1–6.

[10] "Monitoring and discovery service: http://www.globus.org/toolkit/mds/."

[11] R. Ranjan, M. Rahman, and R. Buyya, "A decentralized and cooperative workflow scheduling algorithm," in *8th IEEE International Symposium on Cluster Computing and the Grid*, May 2008, pp. 1–8.

[12] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: design, analysis and applications," in *INFOCOM'05: 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, 2005, pp. 1653–1664.

[13] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219–252, August 2005.

[14] "Brite topology generator: http://cs-pub.bu.edu/brite/."

[15] M. Naldi, "Connectivity of waxman topology models," *Computer Communications*, vol. 29, no. 1, pp. 24–31, December 2005.

[16] "Peersim simulator: http://peersim.sourceforge.net."

[17] B. Maniymaran and M. Maheswaran, "Bandwidth landmarking: A scalable bandwidth prediction mechanism for distributed systems," in *GLOBECOM 2007: IEEE Global Telecommunications Conference*. IEEE, November 2007, pp. 248–252.

[18] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *HCW '99: Proceedings of the Eighth Heterogeneous Computing Workshop*. Washington, DC, USA: IEEE Computer Society, 1999, p. 30.

[19] J. Yan, Y. Yang, and G. K. Raikundalia, "Swindewła p2p-based decentralized workflow management system," *IEEE Transactions on Systems, Man, and CyberneticsłPart A: Systems and Humans*, vol. 36, no. 5, pp. 922–935, 2006.

[20] J. Shen, J. Yan, and Y. Yang, "Swindew-s: extending p2p workflow systems for adaptive composite web services," *Software Engineering Conference, 2006. Australian*, pp. 9–17, April 2006.

[21] Y. Yang, K. Liu, J. Chen, J. Lignier, and H. Jin, "Peer-to-peer based grid workflow runtime environment of swindew-g," *International Conference on e-Science and Grid Computing*, vol. 0, pp. 51–58, 2007.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 31, no. 4. New York, NY, USA: ACM, October 2001, pp. 161–172.

[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: the SIGCOMM conference on Applications, tech., arch., and protocols for compu. commu.* ACM, 2001, pp. 149–160.

[24] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, "Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *PDP '10: Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 27–34.