International Journal of Parallel, Emergent and Distributed Systems

ISSN: 1744-5760 (Print) 1744-5779 (Online) Journal homepage: www.tandfonline.com/journals/gpaa20

Decentralised workflow scheduling in volunteer computing systems

Toktam Ghafarian, Bahman Javadi & Rajkumar Buyya

To cite this article: Toktam Ghafarian, Bahman Javadi & Rajkumar Buyya (2015) Decentralised workflow scheduling in volunteer computing systems, International Journal of Parallel, Emergent and Distributed Systems, 30:5, 343-365, DOI: 10.1080/17445760.2014.973876

To link to this article: https://doi.org/10.1080/17445760.2014.973876

Published online: 29 Oct 2014.

International Journal of Parallel, Emergent and Distributed Systems, 2015
Vol. 30, No. 5, 343–365, http://dx.doi.org/10.1080/17445760.2014.973876

Taylor & Francis
Taylor & Francis Group

Decentralised workflow scheduling in volunteer computing systems

Toktam Ghafarian[a]*, Bahman Javadi[b] and Rajkumar Buyya[c]

[a]*Department of Computer Engineering, Khayyam University, Mashhad, Iran;* [b]*School of Computing, Engineering and Mathematics, University of Western Sydney, Sydney, Australia;* [c]*Department of Computing and Information Systems, University of Melbourne, Melbourne, Australia*

(*Received 3 October 2013; final version received 2 October 2014*)

Volunteer computing systems exploiting large amounts of geographically dispersed resources on the Internet for solving complex scientific problems. However, scheduling scientific workflows in a fully decentralised way and low overhead is a challenging task in these environments. To counter this challenge, this paper presents a fully decentralised proximity-aware workflow-scheduling policy for these environments. The proposed scheduling consists of three phases. In the first phase, each workflow application is partitioned into sub-workflows in order to minimise data dependencies among them. The second phase of the workflow-scheduling algorithm finds some resources to execute each sub-workflow. These resources are selected based on Quality of Service (QoS) constraints of the workflow, load balancing and proximity of resources. Each workflow can have QoS constraints in terms of minimum CPU speed and minimum RAM or hard disk requirements. In the third phase, sub-workflows will be executed on each resource based on local scheduling algorithm to minimise the partial makespan. The proposed scheduling policy focuses on the reduction of communication overhead to improve the performance of I/O-intensive and data-intensive workflows. Simulation results show that the proposed workflow-scheduling policy improves the average response time of scientific workflows up to 53.6% under a moderate load.

Keywords: peer-to-peer computing; volunteer computing systems; decentralised workflow scheduling; proximity-aware scheduling

1. Introduction

Volunteer computing (VC) systems provide a powerful integrated computational platform for high-throughput applications. This platform is composed of volunteers that donate idle cycles of computing resource and storage.[1] BOINC,[2] Condor,[3–5] and Aneka [6] are examples of desktop grid middleware. BOINC is a popular VC platform for many projects such as SETI@home,[7] EDGeS@Home,[8] and climateprediction.net.[9] Some examples of centralised and partly decentralised VC systems are Entropia,[10] XtremeWeb,[11] SZTAKI,[12] OurGrid,[13] ShareGrid,[14] and Condor-Flock P2P. [15] Fully decentralised peer-to-peer (P2P)-based VC systems, which introduce a self-organised and scalable platform, are becoming a promising trend. Some projects in fully decentralised VC systems are PastryGrid,[16] BonjourGrid,[17] and Self-Gridron.[18]

While the most popular application model in VC systems is Bag of Tasks (BoT), there are a number of scientific and engineering applications that are determined by a set of dependent tasks called *workflow*.[19] A scientific workflow contains a group of tasks that

*Corresponding author. Email: t.ghafarian@khayyam.ac.ir

perform a required functionality, and there is a data dependency among them. Execution of workflows on VC systems enhances the usage and scientific adoption of this platform because the group of projects in the form of BoT application is small and the scope of VC systems is extended to support richer computing models such as workflow paradigm.[20,21] On the other hand, data-intensive applications are another promising field in VC systems.[22] For instance, one of data-intensive scientific application in human genome is the Alu clustering problem.[36] Alu indicates the largest repeat families in human genome in which a small fraction of intersections are human-specific. Clustering of Alu repeats needs high capacity computational platform with a low communication overhead. CyberShake [29] is another data-intensive application that computes probabilistic seismic hazard curves for geographic sites in the Southern California region. These applications need to communicate a huge amount of data between their tasks. If these tasks are scattered on peers in P2P-based VC systems, the communication overhead reduces by transferring data directly among the peers without needing to pass through a centralised server. Therefore, proximity-aware workflow scheduling can decrease the overhead of exchanging large data-sets among the tasks in the data-intensive workflow applications.

We define proximity of resources for one workflow relative to some nodes with specific roles for this workflow. The proximate resources are the nodes that have a minimum communication overhead relative to these nodes of the workflow. The problem of scheduling workflow is locating proximate resources to execute its tasks in such a way that it satisfied the Quality of Service (QoS) constraints of this workflow and it minimises the makespan of the workflow. As we have data dependency within the tasks in the workflow, grouping the tasks into sub-workflows with minimum data dependency between these groups can improve the communication overhead. Also, selecting proximate resources that have a minimum communication overhead relative to some nodes with specific roles for this workflow can more decrease the communication overhead.

The proposed workflow-scheduling policy is designed based on our previous work, CycloidGrid.[35] CycloidGrid is an environment for resource discovery in P2P-based VC systems. There are three types of nodes in this environment. These nodes are called reporting node, host node and client node. Each reporting node keeps resource information for a group of available resources in the system. The host node has two roles in the system: it can execute allocated tasks, and it can act as a scheduler in order to find suitable resources for each workflow application. The client node submits a workflow application to execute. It sends a lookup request to a host node to schedule its workflow.

In this paper, the proposed workflow-scheduling operates in three phases. In the first phase, each workflow application is divided into sub-workflows via the K-way Fiduccia–Mattheyses algorithm (K-FM).[39,40] This algorithm is a graph-partitioning algorithm that aims to partition a graph into two or more components in order to minimise the total weight of edges whose ends lie in different partitions. Thus, in the first phase, we consider minimisation of data dependencies among workflow partitions.

The goal of the second phase is scheduling the sub-workflows on available resources in the system. The host node selects resources to run sub-workflows based on QoS constraints of the workflow application, load balancing and proximity-aware feature. Each workflow can have QoS constraints in terms of minimum CPU speed and minimum RAM or hard disk requirements. We apply a load-balancing policy to distribute workflows on available resources in the system in order to minimise the makespan of the workflow. An analytical queuing model based on parallel non-observable queues implements the load-balancing strategy.[37]

In the third phase, each sub-workflow is executed based on the local scheduling algorithm to minimise their partial makespan.[48] In summary, this paper makes the following contributions:

- Proposing a decentralised workflow-scheduling system in P2P-based VC systems which divides the workflow into sub-workflows in such a way that data dependencies among them are minimised.
- Proposing a resource discovery algorithm that considers the load-balancing policy, QoS constraints of each workflow and proximity of selected peers.
- Evaluation of the proposed workflow-scheduling system under realistic scientific workflow with different numbers of peers to show the scalability of the system.

The rest of this paper is organised as follows: Section 2 presents related work. Section 3 presents the proposed workflow-scheduling system and the analytical queuing model for implementing load balancing in the system. Section 4 describes the performance evaluation of the proposed workflow-scheduling system. Conclusion and future directions are presented in Section 5.

2. Related work

There are several studies that investigated workflow scheduling in VC systems. These studies can be divided into two categories: in the first category scheduling of workflow with QoS constraints is considered. Meanwhile, in the second category, minimisation of makespan is taken into account while QoS constraints of workflow are ignored.

Celaya and Arronategui [41] presented an approach for workflow scheduling based on partitioning a workflow into sequences of dependent tasks. A balanced tree-based overlay network is applied. Each node can have three roles: routing node that keeps the overlay structure, gathers the availability information from its sub-branch and routes each workflow request to the execution node in its sub-branch. Execution node runs sub-workflows. Submission node controls the submission of workflow and monitors it during its execution. Generation of sequences is started with the critical path of workflow, and time constraints for other sequences are calculated based on dependent sequences. Two schedulers are defined in the system as a global scheduler and a local scheduler. Global scheduler uses the routing nodes to assign generated sequences to the execution nodes based on their deadline. The local scheduler on each execution node runs queued tasks by their deadline, with the earliest first. Deadline is considered as QoS constraints of workflow. Also, the authors considered a class of workflow with negligible data transfer that does not cover all scientific workflow application. This work has considered partitioning of workflow into sub-workflows, and it benefits availability information of resource. But load balance among execution nodes is not implemented. Moreover, selection of execution nodes in the workflow-scheduling system ignores the proximity of nodes.

PastryGrid [16] presents a protocol for running a distributed application with precedence between tasks in P2P-based desktop grids. In this system, Pastry is used as a P2P overlay network. Each application has a site as a rendezvous (RDV) point to store and recover data. Each application submits to the system via a submission machine. This machine discovers the RDV point of the application by hashing the application, user names and its local date. Then, the submission machine initialises the execution phase by assigning initial tasks to the suitable resources according to the discovery protocol. Selected resources contact RDV point to get the necessary data. Each selected resource

can participate in the discovery of machines needed for the execution of task's successors. QoS constraints of request such as memory and cost are taken into account in this system, but minimisation of makespan and proximity-aware feature is ignored. In fact, the authors focus on precedence constraints in workflow-scheduling system, and minimisation of data transfer between tasks is neglected. Also, load balancing among nodes is not considered.

Rahman et al. [42] proposed a decentralised workflow-scheduling algorithm based on the cooperation of distributed workflow brokers. This study is proposed for global grid based on Grid-Federation model. This model consists of some sites. Each site shares its resource to the federation. In this system, each workflow broker posts its resource demand by injecting a resource claim object into a decentralised coordination space based on distributed hash table (DHT) in the Grid-Federation model. Each resource provider updates the information of resource by inserting a resource ticket object. These objects are mapped to the DHT-based coordination space. If a resource ticket matches with one or more resource claims, the coordinator space will send notification messages to the resource claimers. QoS constraints of request are considered in this system in the form of the number of requested processors, minimum CPU speed, OS and many other attributes. The workflow management system considers minimisation of makespan and utilises the Grid-Federation model. Proximity-aware feature is ignored in this work.

Chen and Deelman [27] proposed two methods for the integration of workflow partitioning and resource provisioning. The first method considered the resource information in the system to distribute a load in the system fairly. While the second one applied genetic algorithm for the combination of resource selection and job scheduling. They focused on reducing the makespan and resource cost, but it ignores the proximity of resources in resource-provisioning phase.

In the second category, there are a few studies that consider scheduling of workflow to minimise the makespan without the QoS constrains.

Lee et al. [43] proposed two scheduling heuristics for parallel applications with precedence constraint in VC systems. These scheduling heuristics focus on robustness as well as makespan. Their scheduling model is considered as a two-pass model. The schedule generation pass considers the generation of schedules that focuses on makespan, whereas the robustness improvement pass inspects the output of previous pass with an additional set of VC resources and looks up possible rescheduling situations to increase their robustness. In fact, in these two heuristics, a set of best resources is selected based on summation of the computation time of all tasks in the workflow. Then, this set is divided into two sets with an equal number of resources. The first set is better in terms of CPU speed. It starts the scheduling with the first set and expands it to improve the robustness with the second one. One of the advantages of this scheduling algorithm is considering robustness along with minimisation of makespan in the workflow-scheduling system. But, the network model in this work is fully connected and the bandwidth between any pair of resources is the same; therefore, the communication overhead depends only on the size of data-sets between any two tasks. The proposed algorithm is suggested in the centralised VC systems, and it is not applicable in P2P-based VC systems. Also, it considers scheduling of one workflow with a limited number of resources while ignores the load balancing and proximity-aware feature.

Di and Wang [44] proposed a dual-phase scheduling model in P2P-based Grid systems which uses a heuristic called dynamic shortest makespan first (DSMF). They used unstructured P2P overlay network based on Newscast model. In this system, each peer has two roles as a scheduler node and a resource node. A node as a scheduler role schedules each submitted workflow; meanwhile, as a resource role it runs associated tasks. If a

scheduler node receives a workflow, it will find suitable resource nodes for ready tasks. The ready tasks are the tasks that their precedence tasks are completed. The resource discovery of workflow tasks is done at the scheduler node, but data transfer between two tasks is done directly between their resource nodes. In fact, in the first phase, the scheduler node distributes workflow tasks among the available resources, whereas in the second phase each resource node will schedule the assigned tasks. The DSMF heuristic is used at both phases to determine the priority of each workflow task. In this heuristic algorithm, a task with the longest estimated remaining path's execution time towards exit node gets a higher priority to run. This algorithm uses mixed gossip protocol for getting the latest availability information of resources in the system. One of the benefits of this work is considering resource availability in the scheduling of workflow application. Also, this algorithm recognises different workflows according to its structure that helps shorter waiting time for short workflows. Proximity-aware feature and load balancing are neglected in this work.

Kalayci et al. [23] proposed a decentralised execution approach for a large-scale workflow. They partitioned a workflow into sub-workflows, and then each sub-workflow is assigned to one peer domain. If a problem that affects the response time is detected by a workflow management system on one peer domain, some tasks are selected to migrate to another peer domain. This work considers partitioning of workflow and minimisation of response time, but load balancing and proximity of resources are neglected.

Kumar et al. [26] considered a graph-partitioning algorithm to divide a workflow into sub-workflows. Multi-constraint graph-partitioning algorithm is used for workflow partitioning in order to minimise the data movement during workflow execution, and to distribute sub-workflows in the system fairly. They focused only on workflow partitioning in order to decrease the response time, while the proximity of resources is not considered.

To summarise this section, Table 1 presents a comparison among the relevant studies and the proposed scheduling system in terms of platform, QoS constraints, load balancing, proximity-aware feature and minimisation of makespan.

3. Proposed proximity-aware workflow-scheduling system

A workflow application in this paper is modelled by a directed acyclic graph (DAG) where tasks in the workflow are represented as nodes in the graph and data dependencies among the tasks are represented as the directed edges between nodes. In a workflow, if a task does not have any immediate predecessor task, it is called an *entry* task, and if a task does not have any immediate successor, it is named an *exit* task. A task can be executed if all of its immediate predecessor tasks are completed. We assume that a workflow application contains the following characteristics:

- Number of dependent tasks.
- Estimated duration of each task.
- Task dependencies in terms of the amount of data that should be transferred between two tasks.
- Minimum CPU speed and RAM or hard disk requirements are considered as QoS constraints of each workflow application.

The proposed workflow-scheduling system consists of three phases that are illustrated in Figure 1. These three phases are *partitioning phase*, *distributed resource discovery phase* and *local scheduling phase* that are executed in the different nodes in the system, as detailed in the next section.

Table 1. Comparison among previous studies and the proposed scheduling system.

Studies	Platform	QoS constraints	Load balancing	Proximity-aware feature	Minimisation of makespan
Celaya and Arronategui [41]	Balanced tree-based overlay network	Deadline	No	No	No
PastryGrid [16]	Pastry	Required memory and cost	No	No	No
Rahman et al. [42]	Grid-federation model	The number of requested processors, minimum CPU speed, OS and many other attributes	No	No	Yes
Lee et al. [43]	Centralised VC systems	No	No	No	Yes
Di et al. [44]	Unstructured P2P overlay network based on newscast model	No	No	No	Yes
Kalayci et al. [23]	Grid	No	No	No	Yes
Kumar et al. [26]	Grid	No	Yes	No	Yes
Chen et al. [27]	Grid and clouds	Deadline and cost	Yes	No	Yes
Our proposed workflow-scheduling system	Cycloid	Minimum CPU speed and RAM or hard disk requirements	Yes	Yes	Yes

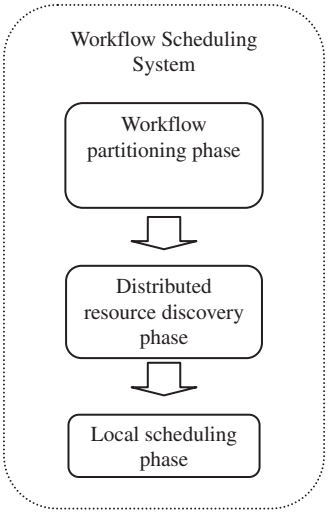

Figure 1. Workflow scheduling system of CycloidGrid.

3.1 Workflow-partitioning phase

Each workflow application is imported into the system by a client node. The first phase of scheduling (partitioning phase) is done in the client node. In fact, this phase considers workflow partitioning as a multi-way graph-partitioning problem. The multi-way graph-partitioning problem is an NP-hard problem.[45] The objective of this problem is partitioning a graph into two or more components in order to minimise the total weight of edges whose ends lie in different components.[46] The workflow-partitioning phase divides a workflow into sub-workflows with minimum data dependencies among these sub-workflows. If the total weight of edges among sub-workflows is minimised, the overall communication overhead among them is minimised as well.

The K-way Fiduccia–Mattheyses (K-FM) algorithm [39,40] is an heuristic algorithm for a multi-way graph-partitioning problem that is used in this research to partition a workflow application into sub-workflows. It is a generalised form of FM algorithm. The FM algorithm [47] is a heuristic algorithm with an iterative improvement for partitioning hyper-graphs into two partitions. It is based on partition-to-partition moves. FM starts with a random solution and improves the solution by a sequence of moves that are called passes. In the beginning of each pass, all vertices are unlocked and they can move between partitions. Each move has a gain in such a way that a positive gain decreases the solution cost while a negative gain increases it. A move with the highest gain is selected repeatedly in each pass, and the moving vertex is locked during this pass. It means that the moving vertex cannot move during this pass. The gain of adjacent vertex of the moving vertex is recomputed after each move. The process of selecting a best-gain move and updating all gain values of adjacent vertices are repeated until every vertex is locked. Then, the best solution during this pass is selected as a starting solution of the next pass. The algorithm terminates if a pass fails to improve the solution quality. Figure 2 shows an example of partitioning a sample graph with the FM algorithm. In this graph, all edges are assumed to have the same weight, equal to 1. In Figure 2(a), the graph is partitioned randomly into two parts (light nodes are in partition 1 and dark nodes are in partition 2). The net cut that is equal to sum of weights of edges across the partitions for initial partitioning

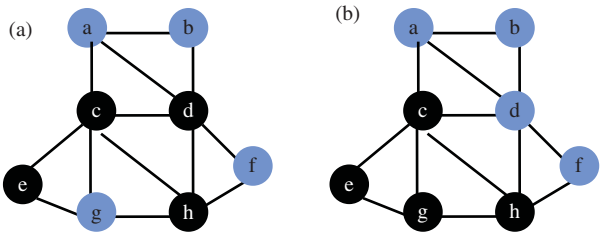

Figure 2. Example of FM partitioning of a sample graph: (a) initial partitioning and (b) after improvement by FM.

is 8. Meanwhile, after executing FM partitioning, the net cut is decreased to 4 as shown in Figure 2(b). The K-FM algorithm [39,40] as a generalised form of FM algorithm starts with an initial partitioning of the graph into two or more than two partitions. It considers all possible moves of each unlocked vertex from its home partition to any other partitions in any iteration during a pass. It selects the best of such moves. This algorithm terminates when no improvement is obtained.

According to the K-FM algorithm, at first, the workflow is partitioned into some of sub-workflows randomly. The size of sub-workflows follows the balanced constraints.[39] Some probabilities in terms of $p_{1,}p_{2,}\ldots,p_s$ are assigned to sub-workflows in order to preserve balanced constraints. These probabilities are selected as follows [39]:

$$\sum_{i=1}^{s} p_i = 1, \quad 0 \leq p_i \leq 1, \tag{1}$$

where s is the number of sub-workflows. The size of given sub-workflow i with threshold $\theta > 0$ and m as the total number of tasks in the workflow is computed as follows [39]:

$$p_i m - \theta \leq l_i \leq p_i m + \theta. \tag{2}$$

A task can be transferred from sub-workflow i to sub-workflow j, if it preserves the balanced constraints for l_{i} and l_j. The proposed partitioning phase uses different sub-workflow probabilities, but they are close to each other.

During one pass of algorithm, tasks are moved from its home sub-workflows to the target sub-workflows that have at least one adjacent task with those tasks. The move gain of a task according to the target sub-workflow is computed as the difference in sum of the file size between this task and adjacent tasks in the target sub-workflow and sum of the file size between this task and adjacent tasks in its home sub-workflow. At the beginning of one pass, all tasks are allowed for moving between sub-workflows. The tasks are selected based on maximum move gain and they are moved to the target sub-workflows. After moving a task, that task is locked and it is not allowed moving again during this pass. Also, the move gain of its adjacent tasks is updated. The net cut is computed after each move. The net cut is equal to sum of the file size between the tasks that are placed in different sub-workflows. This procedure continues until all tasks are locked. Then, the best partitioning with a minimum net cut during this pass is selected, and it is used as the start point of next pass. The algorithm is continued until no improvement is gained.

Therefore, the partitioning phase generates sub-workflows with minimum data dependencies among them. After that, it determines the immediate predecessors and immediate successors of any task with their partitions in each sub-workflow. Each sub-workflow in this phase is called *ready sub-workflow*.

3.2 Proximity-aware workflow-scheduling system

CycloidGrid uses a decision tree (DT) in order to classify resources into clusters. Each resource in the system has attributes such as CPU speed, the amount of RAM, available hard disk size, operating system and the processor model. As shown in Figure 3, each level of DT is based on one of these attributes, and four attribute values are considered in each level. Therefore, DT classifies resources into $4^5 = 1024$ clusters. The DT is used to search a resource based on QoS constraints of request and supports keyword search of resources. There is a mapping between the DT clusters and the first 1024 clusters of CycloidGrid. The first 1024 clusters of CycloidGrid are called reporting clusters. In fact, each cluster of DT is assigned to its corresponding reporting cluster of CycloidGrid. Therefore, each reporting cluster contains information of resources whose corresponding attributes (e.g. CPU speed, RAM and hard disk) are close. The remaining clusters of CycloidGrid are called host clusters. The host nodes and client nodes are on host clusters, and the reporting nodes are on reporting clusters.

Ready sub-workflows are sent to a random active host node to schedule as shown in Figure 4. The selected host node is called an *injection* node (Step 1). The second phase (resource discovery phase) of the scheduling system is performed in this node. The injection node by using the DT as discussed before finds which reporting clusters can be useful to search according to QoS constraints of the workflow application. The injection node searches the DT level by level. Because we do not consider the processor model and operating system as QoS constraints, the injection node considers the entire category of processor model and operating system in the first two levels. If the injection node has minimum requirements for hard disk size, it considers this category on DT otherwise it

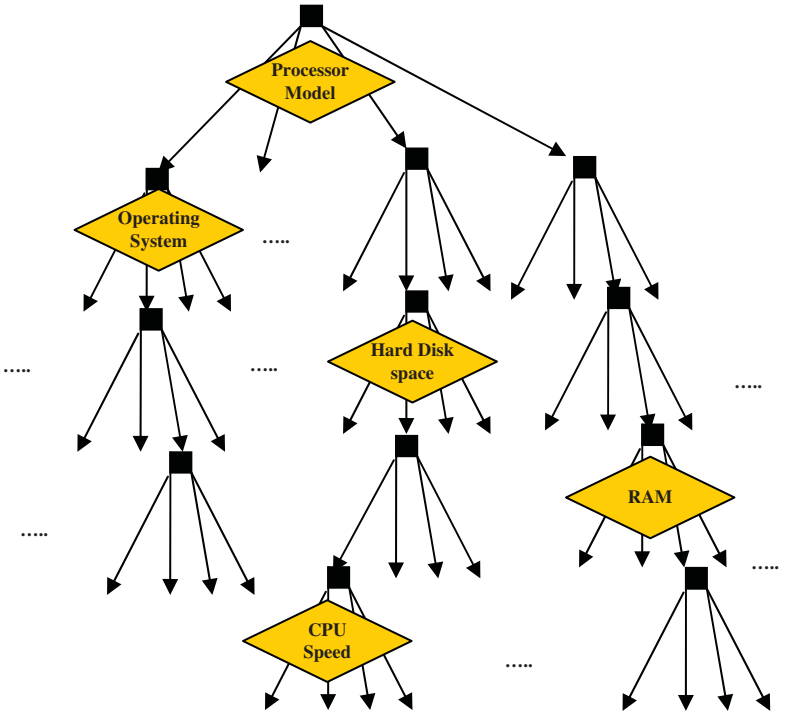

Figure 3. DT for classification of resources.

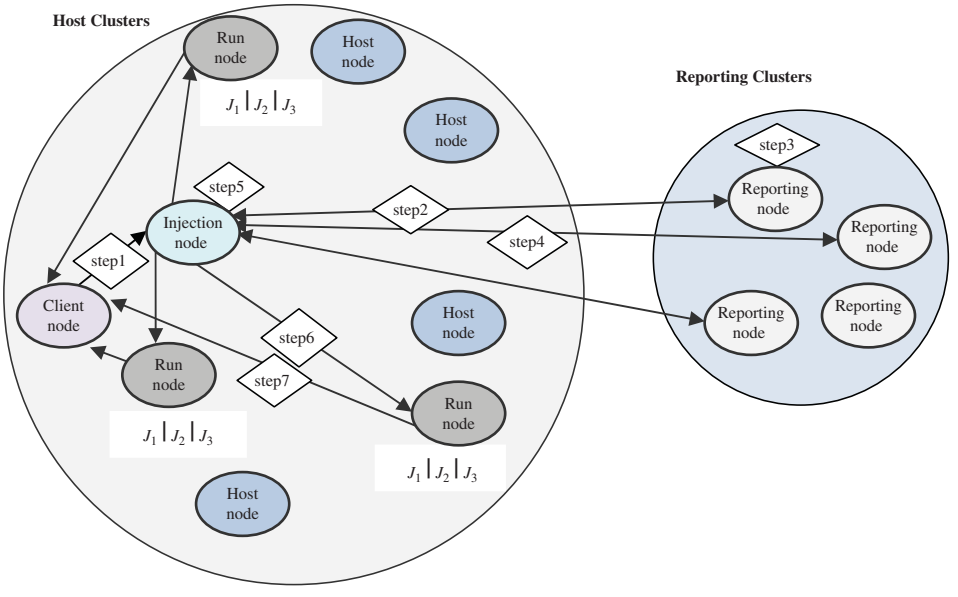

Figure 4. Resource discovery phase of workflow application.

considers all of categories. It is the same for RAM and CPU speed in the lower levels. All the categories that are selected on the leaf level are considered as selected reporting clusters. Then, the lookup request is sent to the reporting node with a minimum communication overhead relative to the injection node in every selected reporting cluster (Step 2). Each reporting node offers resources based on its load-balancing policy and QoS constraints of workflow (Step 3), and sends the address of offered resources to the injection node (Step 4). The load-balancing strategy in each reporting node is explained in Section 3.3. The injection node receives offers from every selected reporting node. It searches among the offers and selects resources with a minimum communication overhead with respect to itself and the client node of this workflow (Step 5). In fact, approximate resources relative to these two nodes are selected. Each sub-workflow is assigned to one host node in this phase. This host node is called a *run* node. Injection node sends each sub-workflow to the selected run nodes (Step 6). Finally, the run nodes put these sub-workflows in their queues and download the source code and input files of these sub-workflows from the client node (Step 7).

Therefore, proximity-aware feature is taken into account in the workflow-scheduling system in two phases: partitioning a workflow application (the first phase) and selecting approximate run nodes for running sub-workflows (the second phase).

The third phase of scheduling system as illustrated in Figure 1 targets minimisation of partial makespan of each sub-workflow. This phase is done in the run node by Algorithm 1. In this algorithm, at first tasks are prioritised by its upward rank in the main workflow. The upward rank $r(n_i)$ of task n_i is computed based on the HEFT algorithm [48] as it is computed as follows:

$$r(n_i) = w(n_i) + \max(c_{ij} + r(n_j)), \quad n_j \in succ(n_i) \quad r(n_{\text{exit}}) = w(n_{\text{exit}}) \tag{3}$$

where $w(n_i)$ is the estimated computation time of task n_i and c_{ij} is the average communication overhead between task n_i and n_j. The value of c_{ij} is computed based on the

Algorithm 1: Local scheduling algorithm at each run node

Input: sub-workflow $G' = (V', E')$, *{upward rank(v), pred(v), succ(v)} for* $\forall v \in V'$
Output: results of sub-workflow

```
1  Sort V' in descending order by upward rank value.
2  While V' is not empty do
3    If v is an entry task or run node receives all results of pred(v) then
4      run v and send the results of v to succ(v)
5      delete v from V'
6    end
7  End
```

data size between task n_i and task n_j and average network bandwidth. Also, $succ(n_i)$ is a set of immediate successors of task n_i. The upward rank of a task in each sub-workflow is initially computed by the client node and is sent to each run node along with sub-workflow. The rank of task n_i shows the length of critical path from this task to the exit task including the computation time of task n_i. If we order tasks based on decreasing order of their ranks, it provides a topological order of tasks. This topological order preserves the precedence constraints between tasks. We use a decreasing order of upward ranks in each sub-workflow to preserve precedence constraints among its tasks. Therefore, all tasks are sorted by upward ranks in the descending order at Line 1. If a task is an entry task or the results of all immediate predecessors of each task v of sub-workflow ($pred(v)$) are ready, this task will run, and its result is sent to immediate successors (lines 3 and 4). Otherwise, it will wait for the results of its predecessors. If a task is run, it is deleted from V' (Line 5). The algorithm is finished when all tasks are executed. The results of tasks are transferred between sub-workflows directly. Also, a copy of results is kept in the client node as a backup in the case of leaving one or more run nodes. If several sub-workflows are scheduled at the same time on a given node N, they are scheduled one after the other. If a sub-workflow waits for dependencies from other sub-workflow which are executed on a different node, this node switched to another sub-workflow in a circular way. But a given node N will resume the execution of a suspended sub-workflow, if this node receives the result of dependencies from other sub-workflows on other nodes.

If one or more run nodes that execute the sub-workflows of a workflow leave the system randomly, the client node of these failed sub-workflows recognises this situation. Because the client nodes send heartbeat message to all run nodes that execute their sub-workflows. In this case, the client node sends lookup requests for failed sub-workflows to another active host node in order to reschedule them. In fact, failed sub-workflows are caused to postpone the makespan of the workflow. New run nodes can download the result of dependent tasks from the client node or dependent run nodes directly.

3.3 Analytical queuing model for load balancing

In this section, an analytical model for load balancing based on routing in parallel queues is explained.[37] Then, the dispatch policy in the reporting node built upon the analytical model is discussed. In this analytical model, we assume that each reporting node consists of one or more logical clusters.[37] Therefore, each logical cluster can be considered as a server with a given service rate. The service rate of a logical cluster is assumed as the average of service rate of all resources on this cluster. Each sub-workflow is considered as a separate request in this analytical model. The analytical model aims at distributing the input rate of incoming requests among logical clusters in a fair manner. Each reporting

node acts as a router in front of logical clusters (queues) as heterogeneous multi-server parallel queues. Thus, the objective function in the reporting node i can be expressed as follows [37]:

$$T(i) = \min(\gamma_i \times \hat{\Gamma}_i), \tag{4}$$

where γ_i is the arrival rate of requests, and $\hat{\Gamma}_i$ is the average response time of incoming requests into reporting node i. Equation (4) is the minimisation of $\sum_{j=1}^{L_i}(\Lambda_j \times E(\Gamma_j))$ for logical clusters in reporting node i after load distribution. Where Λ_j is the arrival rate of requests into logical cluster j, Γ_j is the average response time of incoming requests into cluster j and L_i represents the number of logical clusters in reporting node i. By minimisation of $\sum_{j=1}^{L_i}(\Lambda_j \times E(\Gamma_j))$ with the Lagrange multiplier system, the optimal arrival rate of requests into logical cluster j after load distribution is computed as follows [37]:

$$\Lambda_j = \frac{1}{\bar{\theta}_j} - \frac{1}{\bar{\theta}_j}\sqrt{\frac{1-(\gamma_i)^2\sigma_{I_i}^2 + (C_{S_j}^2 - 1)\gamma_i\bar{\theta}_j}{1-(\gamma_i)^2\sigma_{I_i}^2 + 2\gamma_i(\bar{\theta}_j - z)}}, \tag{5}$$

where $\bar{\theta}_j$ is the average and $C_{S_j}^2$ is the squared coefficient of variance for service time of sub-workflows on cluster j. $\gamma_i\sigma_{I_i}^2$ are the arrival rate and the variance of incoming requests to the reporting node, respectively. z is a Lagrange multiplier. A numerical solution based on the bisection algorithm is used to search z in a range of its lower bound and upper bound [37]. As we discussed, each logical cluster gets a request based on optimal arrival rate Λ_j. So, the average service time of each sub-workflow in one logical cluster can be approximated as

$$\bar{\theta}_j = \bar{P}_{\mathrm{WF}}\bar{T}_{\mathrm{WF}}\frac{E_m}{\bar{E}_j} \quad 1 \le j \le L_i. \tag{6}$$

In Equation (6), the service time of each sub-workflow can be approximated as the product of average size of each sub-workflow ($\bar{P}_{\mathrm{WF}}$) and approximation of average execution time for any task in the sub-workflow ($\bar{T}_{\mathrm{WF}}$). This value is scaled by the division of maximum average processing speed of all logical clusters in each reporting node by average processing speed of cluster j.

We consider a dispatch policy called Bernoulli between logical clusters with a very limited overhead.[37] Every time a request is sent to the reporting node to select one resource offer, Bernoulli dispatch policy is run. It determines the sequence of choosing logical cluster in such a way that each logical cluster gives requests proportional to its optimal arrival rate. In this dispatch policy, the selection of logical cluster is random, but each logical cluster gives a request based on its corresponding routing probability. The routing probability of one logical cluster in the reporting node i can be computed as $\rho_j = \Lambda_j/\gamma_i$. After the selection of a logical cluster, resources are examined based on QoS constraints of request in a round-robin manner. The first resource that satisfied the QoS constraint is selected on this logical cluster. If no resource satisfies QoS requirements, another logical cluster is selected and this procedure continues until a resource is selected.

4. Performance evaluation

In order to evaluate the performance of the proposed workflow-scheduling system, we use CycloidGrid simulator [35] as a discrete event simulator. The physical network in CycloidGrid is emulated using the Brite topology generator.[24] In this topology

generator, the nodes are scatted by heavy tailed node placement, and they are interconnected by Waxman model [24] that uses the Waxman probability model for interconnecting two nodes based on Euclidean distance between these two nodes and maximum distance of any two nodes in the system. Also the bandwidth between two nodes varies from 10 to 100 Mb/s with uniform distribution.[25] As each node in the system is a volunteer resource in VC systems, we use Xtremlab trace [28] to assign attributes to the output resources of topology generator. This trace is the output of the resource-monitoring project of BOINC database. BOINC is the largest distributed computing project on VC systems. Xtremlab periodically measures the availability of CPU, memory and disk space in a trace file.

The performance metric related to the performance of workflow application is average makespan (AMN) of workflow. The makespan of each workflow application is equal to the maximum finish time of the exit tasks in the workflow after scheduling of a workflow is completed. So, the AMN of given R workflows are defined as follows:

$$\text{AMN} = \frac{\sum_{f=1}^{R} M_f}{R}, \tag{7}$$

where M_f is the makespan of a given workflow f and is defined as follows:

$$M_f = \text{MAX}(\text{FT}_j) \quad \text{for } j \in \text{exit tasks}, \tag{8}$$

where FT_j is the finish time of task j. As each workflow application is partitioned into sub-workflows (partitions), the finish time of each task can be computed based on its partition number (i) and its run node (m). $\text{FT}_j^{i^m}$ is the finish time of task j when it runs within the partition i on run node m:

$$\text{FT}_j^{i^m} = \text{MAX}\left[\left(A_i^m + \sum_{k \in \text{precedent}(j)} r_k\right), \text{MAX}\left(\text{FT}_{pred(j)} + \bar{L}_{pred(j)}^{j}\right)\right] + r_j, \tag{9}$$

where A_i^m is the time when a run node m is ready for running the partition i. Waiting time for task j is the maximum of two components: A_i^m plus computation time of all precedent tasks in its partition, and the maximum finish time of its immediate predecessors (parents) plus communication delay for sending results between them ($\bar{L}_{pred(j)}^{j}$). Finish time of task j is equal to the waiting time plus its computation time (r_j). A_i^m is computed as follows:

$$A_i^m = T_{m\,\text{comm}}^{i} + \text{MAX}(T_{\text{Q}}^m, \bar{L}_{\text{C}}^m), \tag{10}$$

where $T_{m\,\text{comm}}^{i}$ is the communication delay for sending partition i to run node m. T_{Q}^m is the waiting time of queue on run node m and $\bar{L}_{\text{C}}^m$ is the communication delay for sending input files of partition i from client node to run node m. We used the maximum operator between T_{Q}^m and $\bar{L}_{\text{C}}^m$ because they are done in parallel. $T_{m\,\text{comm}}^{i}$ based on Section 3.2 is equal to summation of $\bar{L}_{\text{C}}^I$ (communication delay between the client node and the injection node), $\bar{L}_I^{\text{R}}$ (the maximum communication delay between the injection node and each of selected reporting nodes, and the communication between injection node and reporting nodes are parallel, therefore the maximum operator is used) and $\bar{L}_I^m$ (communication delay between

injection node I and run node m):

$$T^{i}_{m_{\text{comm}}} = \bar{L}^{I}_{\text{C}} + \text{MAX}(\bar{L}^{\text{R}}_{I}) + \bar{L}^{m}_{I}. \tag{11}$$

The communication delay between two peers is computed by an analytical model based on queuing theory and can be approximated as follows [35]:

$$\bar{L} = 2S_{\text{C}} + \frac{\sigma^2_{I_{\text{C}}}\lambda^2_{\text{C}}}{2((S_{\text{C}})^{-1} - \lambda_{\text{C}})} + \sum_{i=s+1}^{d} S_{\text{C}_i}, \tag{12}$$

where

$$S_{\text{C}} = 0.5\alpha_{\text{net}} + F\beta_{\text{net}}, \tag{13}$$

in which S_{C} is a service time of each connection, and α_{net} and β_{net} are the network latency and the inverse of bandwidth along the link between two adjacent peers, respectively. F is considered as the flow size that is transmitted between two peers. $\sigma^2_{I_{\text{C}}}$ and λ_{C} are the variance and the inter-arrival rate of traffic at source peer's queue, respectively. The last term of Equation (12) equals to a summation of S_{C} along the route between source and destination peer. The numbering starts from the peer next to the source node and goes up to reach the destination peer.

4.1 Workload model

The proposed workflow-scheduling system is evaluated for four different scientific workflow applications. Figure 5 presents these workflows with a relatively small number of tasks. These workflows are an astronomy application (Montage),[30] seismology application (CyberShake) [29] and two bioinformatics applications (Epigenomics [31] and Sipht [33]). They were chosen because they show a wide range of application domains and a variety of resource requirements.[32] Montage is an astronomy application that is used to construct large image mosaics of the sky. CyberShake is a seismology application that calculates probabilistic seismic hazard curves for geographic sites in the Southern California region. Sipht is a bioinformatics project that is conducting a wide search for small untranslated RNAs that regulate several processes such as secretion or virulence in bacteria. Epigenomics workflow is a data-processing pipeline that automates the execution of the various genome-sequencing operations. Four scientific workflow applications with 30 tasks are generated by the workflow generator developed by Bharathi et al. [31,34] Inter-arrival time of each workflow follows the Weibull distribution based on Grid workload archive.[50] This distribution with its parameters is listed in Table 2. Each workflow application may have QoS constraints in terms of minimum CPU speed and minimum RAM or hard disk requirements. These QoS constraints are different from one workflow to another.

We generate the workload for 1 day where 2.5 h is considered as a warm-up phase to avoid bias before the system reaches steady state. Each experiment is performed on each of these workloads separately. For the sake of accuracy, each experiment is carried out several times by using different workloads and average results are reported. In all of reported results, coefficient of variance is less than 0.01 (CV $<$ 0.01). The number of resources is equal to 1000 and 3000 peers with heterogeneous computing speeds.

In order to generate different workloads, we modified the parameters one at a time. Thus, to change the inter-arrival time, we modified the first parameter of Weibull

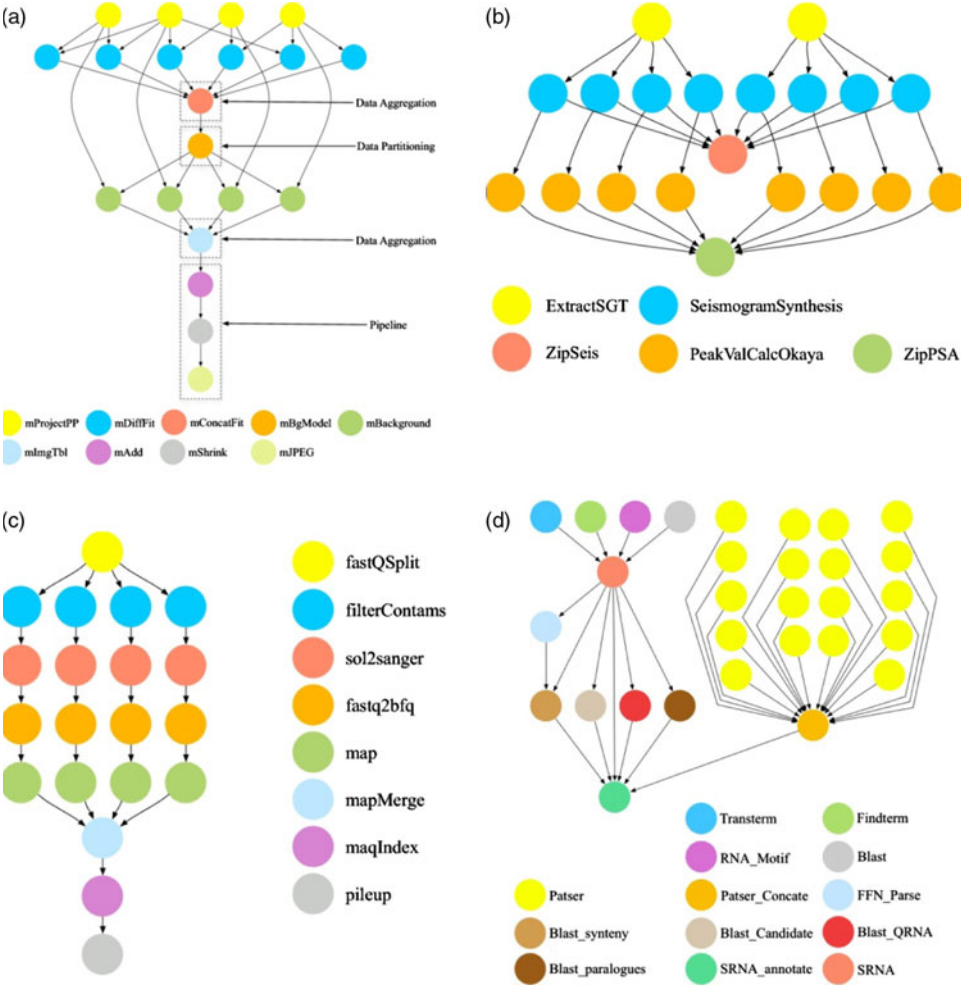

Figure 5. Examples of scientific workflows: (a) Montage workflow, (b) CyberShake workflow, (c) Epigenomics workflow and (d) Sipht workflow.[34]

distribution (the scale parameter α) as shown in Table 2. Therefore, the number of workflows increases from 8000 (i.e. $\alpha = 11$) to 14,000 (i.e. $\alpha = 7$). The average inter-arrival time (τ) of peer churn modelled using a Poisson distribution [38] is presented in Table 2. It varies from 2.13 to 13.33 min. However, between 10% and 70% ($\tau = 13.33 \text{and} \tau = 2.13$) of peers leave the system, while some nodes join the system.

Table 2. Input parameters for the workload model.

Parameters	Distribution/value
Workflow inter-arrival time	Weibull ($7 \leq \alpha \leq 11, \beta = 4.25$)
Inter-arrival time of peer churn	Poisson ($2.13 \leq \tau \leq 13.33$)
Internet inter-arrival time	Weibull ($\alpha = 0.06, \beta = 0.15$)
Internet flow size	Pareto ($\alpha = 3, \beta = 1.05$)

We consider that the background traffic of Internet follows a Weibull distribution [49] as shown in Table 2. Also Internet flow size follows a Pareto distribution.[49] The mean of Pareto distribution is considered as a flow size for Internet traffic.

4.2 Baseline policies

We evaluate the proposed policy (*FMProx*) against three other policies as follows:

- *RandomProx*: In this strategy, each reporting node in the system selects resources with a Bernoulli strategy (Section 3.3) based on QoS constraints of each workflow application. Each workflow application is partitioned randomly, but run nodes with minimum communication delay are selected for running sub-workflows.
- *RandomNoProx*: This policy is the same as *RandomProx*, but run nodes for running each sub-workflow are selected randomly among suggested resources by reporting nodes (stage 5 of Section 3.2).
- *RRNoProx*: In this policy, each reporting node in the system selects resources with a round-robin strategy. If a resource does not satisfy the QoS constraints of workflow, the next resource is examined in a deterministic sequence. Each workflow application is partitioned randomly and run nodes are selected randomly among suggested resources by reporting nodes (stage 5 of Section 3.2).

These policies differ in three ways. The first one is the workflow-partitioning algorithm. FMProx uses the K-FM algorithm for partitioning the workflow into sub-workflows, whereas the other policies partition a workflow into sub-workflows randomly. The second one is the load-balancing policy. FMProx, RandomProx and RandomNoProx use a Bernoulli dispatch policy based on the optimal arrival rate computed in Section 3.3, whereas RRNoProx uses a round-robin policy within logical clusters. The third one is proximity-aware feature. FMProx and RandomProx select proximate resources to execute sub-workflows that have a minimum communication overhead relative to the client node and injection node of the workflow, whereas RandomNoProx and RRNoProx select resources randomly.

4.3 Simulation results

The simulation results for AMN versus arrival rate are shown in the followingfigures for different policies. In Figures 6–9, we assume that the system is relatively static and no peer joins or leaves during the experiment. Meanwhile, Figures 10 and 11 show the evaluation of the performance of the system under dynamic environments.

Depicted in Figure 6 are results for the Montage workflow with 1000 peers (a) and 3000 peers (b). As we expected, by increasing the arrival rate, AMN dramatically increases too. FMProx and RandomProx strategies can control AMN by considering the proximity-awareness when choosing target resources. Meanwhile, RandomNoProx and RRNoProx approach the saturation point exponentially. FMProx surpasses RandomProx, RandomNoProx and RRNoProx with improvement factors of 14%, 55% and 54%, respectively, in Figure 6(a). The improvement of FMProx with respect to these policies is 15%, 38% and 40%, respectively, in Figure 6(b). Montage is an I/O-intensive workflow with a few CPU-intensive jobs, so proximity-awareness has more impact on reducing AMN of proximity-aware methods such as FMProx and RandomProx compared with RandomNoProx and RRNoProx. However, this feature influences the better performance of FMProx compared with RandomProx in this workflow.

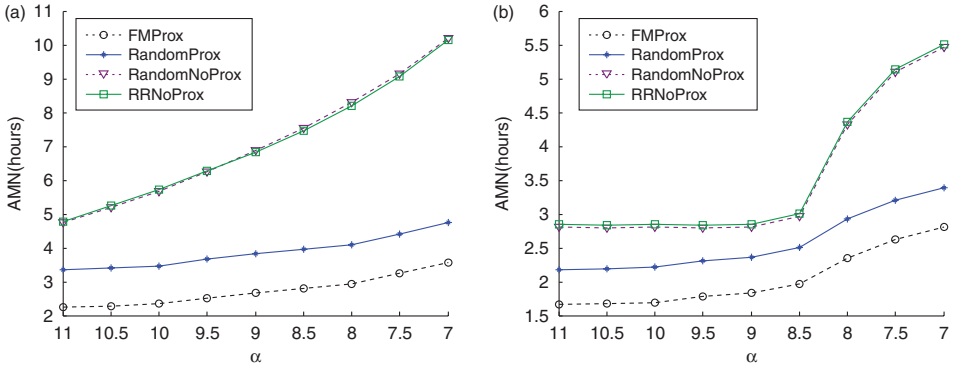

Figure 6. Average response time resulting from different policies for the Montage workflow by modifying the α parameter in the inter-arrival time: (a) with 1000 peer and (b) with 3000 peer.

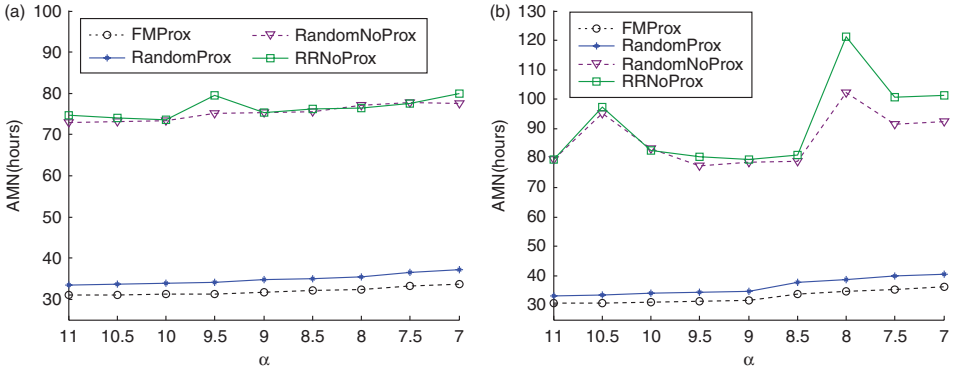

Figure 7. Average response time resulting from different policies for the CyberShake workflow by modifying the α parameter in the inter-arrival time: (a) with 1000 peer and (b) with 3000 peer.

Figure 7 shows the experimental results for the CyberShake workflow with 1000 peers (a) and 3000 peers (b). The improvement factor of FMProx with respect to RandomProx is 6% and 4% in Figure 7(a) and (b), respectively. However, it is 88% and 59% with respect

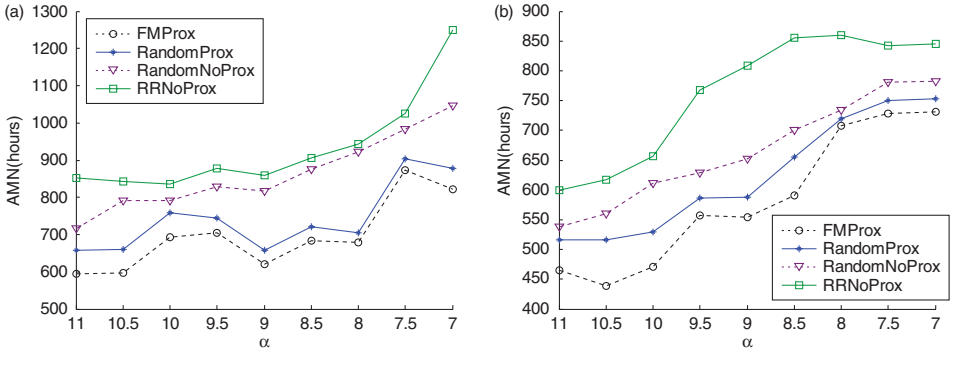

Figure 8. Average response time resulting from different policies for the Epigenomics workflow by modifying the α parameter in the inter-arrival time: (a) with 1000 peer and (b) with 3000 peer.

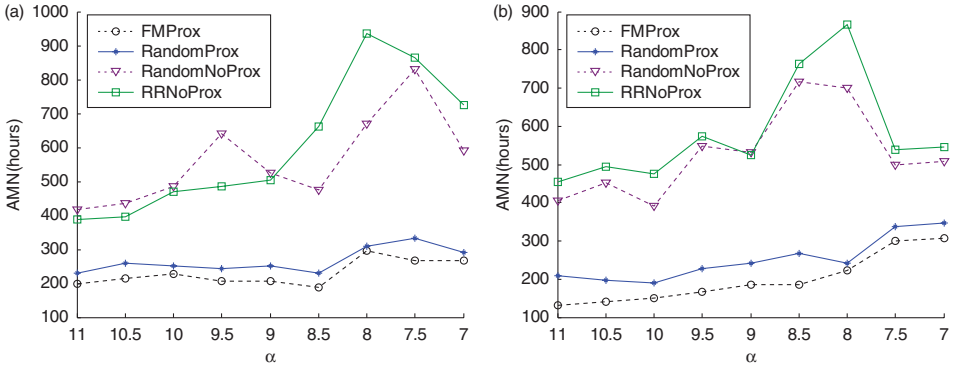

Figure 9. Average response time resulting from different policies for the Sipht workflow by modifying the α parameter in the inter-arrival time: (a) with 1000 peer and (b) with 3000 peer.

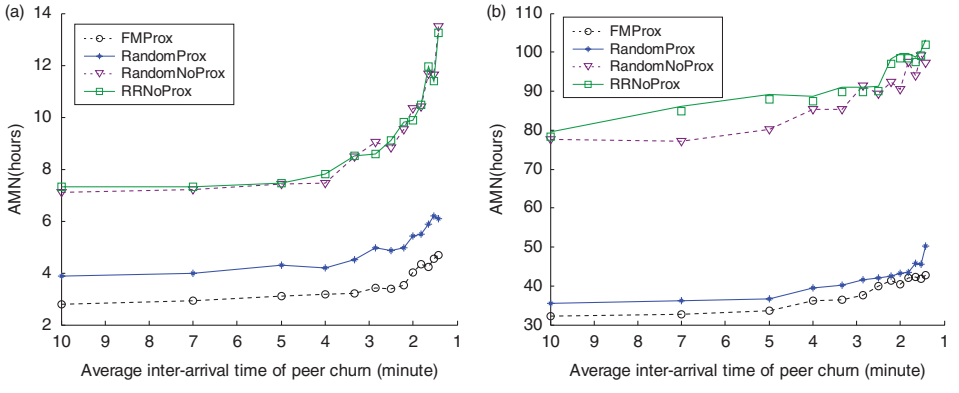

Figure 10. Average response time resulting from different policies for 1000 peers, the simulations are carried out by modifying the average inter-arrival time of peer churn. (a) Montage workflow and (b) CyberShake workflow.

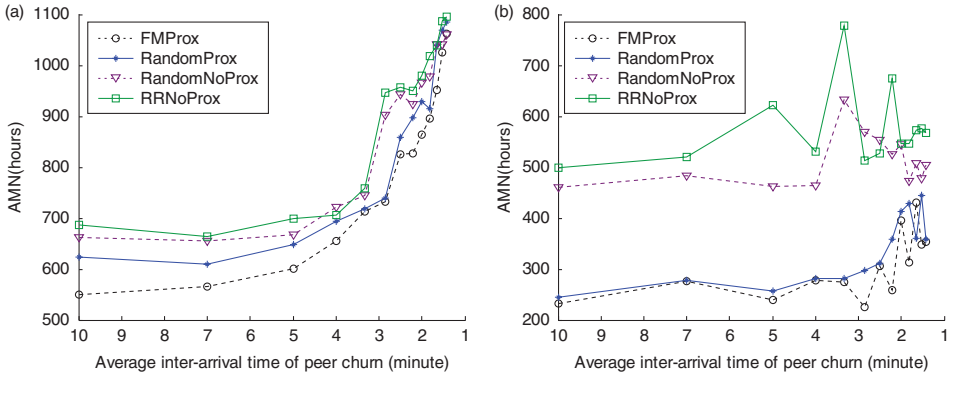

Figure 11. Average response time resulting from different policies for 1000 peers, the simulations are carried out by modifying the average inter-arrival time of peer churn. (a) Epigenomics workflow and (b) Sipht workflow.

to RandomNoProx and 90% and 63% compared with RRNoProx in Figure 7(a) and (b), respectively. Unlike the Montage workflow, CyberShake is a data-intensive workflow with large input/output files, so improvement of FMProx with respect to non-proximity-aware policies (RandomNoProx and RRNoProx) is more considerable compared with other workflows. Because FMProx selects run nodes that have a minimum communication overhead relative to the injection node and the client node, transferring of input/output files has smaller communication delay compared with non-proximity-aware policies. So non-proximity-aware policies (RRNoProx and RandomNoProx) arrive at saturation point sooner at point 8 of Figure 7(b). Especially in this figure, the diversity of resources is higher compared with Figure 7(a), and the number of resources is three times larger. Therefore, selecting a random resource by these two policies for transferring large input/output files by increasing the number of workflows in the system has caused these two policies approach the saturation point sooner, and after saturation point, the behaviour of these policies is not predictable and has rise and fall.

Figure 8 presents the experimental results for the Epigenomics workflow with different numbers of peers: 1000 peers (a) and 3000 peers (b). The improvement factor of FMProx with respect to RandomProx is 7% and 10% in Figure 8(a) and (b), respectively. However, it is 25% and 20% with respect to RandomNoProx and 35% and 44% compared with RRNoProx in Figure 8(a) and (b), respectively. Unlike the Montage workflow, Epigenomics is a CPU-intensive workflow with fewer fan-in jobs, so reduction in AMN of FMProx compared with RandomProx is smaller than the Montage workflow. Furthermore, the difference between proximity-aware policies (FMProx and RandomProx) and other polices is smaller than the Montage and CyberShake workflows.

Figure 9 represents the experimental results for the Sipht workflow. In this workflow, FMProx reduces AMN by 4% compared with RandomProx and 44% and 49% with respect to RandomNoProx and RRNoProx in Figure 9(a). Also, this reduction is 7%, 46% and 54% in Figure 9(b) with respect to these policies. In Figure 9(a), when the number of workflows is small, RRNoProx performs better; however, it changes the behaviour and performs worse than RandomNoProx by increasing the number of workflows. The Sipht workflow has a single task type (Blast) that accounts for most of its runtime (about 74%). Findterm and Blast_QRNA also contribute a significant amount of the runtime. Meanwhile, other tasks have a very short runtime. Therefore, there are many tasks with a short runtime and a few tasks with a long runtime in this workflow. If the runtime of tasks is small, the load-balancing strategy has little influence on the system performance especially with a small number of workflows. Thus, RRNoProx performs better up to $\alpha < 9$. In fact, the influence of the load-balancing policy is more significant with long runtime tasks. While the system approaches the saturation point (number of workflows is increased), the load-balancing strategy shows its effect and RandomNoProx performs better than RRNOProx. Also RRNoProx arrives at saturation sooner at point 8 of Figure 9 (a) and (b). Because in this point, the number of workflows in the system is high, on the other hand Sipht has a small number of tasks (Blast, Findterm and Blast_QRNA) with a large runtime; therefore, the influence of load-balancing policy is more important with a large number of workflows. However, RRNoProx that follows a round-robin policy cannot control balancing a load in the system by increasing the number of workflows, and system approaches saturation point sooner.

Sipht similar to Epigenomics is primarily a CPU-bound workflow. Thus, the most of jobs have high CPU utilisation and relatively low I/O. The reduction in AMN by proximity-aware policies in this workflow is less than the Montage and CyberShake workflows, but more than Epigenomics.

The simulation results show that the proposed scheduling policy performs better for I/O-intensive with more fan-in tasks compared with CPU-intensive workflows. Also, it has a good performance for data-intensive workflows with large input/output files. Proximity-aware policies significantly decreases AMN of data-intensive workflows compared with I/O-intensive or CPU-intensive workflow. The communication overhead for send or receive input/output files in the data-intensive workflow is large, so choosing approximate resources with a minimum communication overhead relative to the injection node and the client node has a great impact on AMN than other workflows. The impact of the partitioning algorithm has more effect on I/O-intensive workflows. Because such workflows have significant data dependencies between tasks, grouping tasks with more data dependencies in the same partition decreases AMN more than other workflows.

Figures 10 and 11 present AMN of different policies for 1000 peers under dynamic environments for different workflow applications. The average inter-arrival time of peer churn differs from 2.13 to 13.33 min. However, the percentage of leaving peers from the system varies from 10% ($\tau = 13.33$ min) to 70% ($\tau = 2.13$ min) of all peers (with a step of 5%) while some nodes join the system. In these figures, the inter-arrival time of workflow application is kept in the medium size ($\alpha = 9$). As illustrated in these figures, FMProx has a stable behaviour when the average inter-arrival time is decreased to 3.26 min with 45% of peers leaving the system for the Montage workflow. It is almost the same for the CyberShake workflow. It has a similar behaviour till 35% of peers leave the system, after that it starts to increase. For the Epigenomics workflow, AMN is approximately invariant till 20% of peers leave the system, after that it has an increscent trend. Also, in the Sipht workflow, it is almost stable until 30% of peers leave, and then it starts to oscillate. It is concluded that if a workflow is CPU-intensive such as Epigenomics and Sipht workflows, leaving of resources postpones more the overall AMN of workflows. Because when a peer leaves the system, all of assigned sub-workflows on the leaving node are reassigned to another peer. In the CPU-intensive workflow application, the number of uncompleted tasks is more because they require more CPU time and a smaller number of them are finished before leaving a resource. Thus, an increscent peer churn has more influence on Epigenomics and Sipht than other workflows. So these workflows, especially Sipht, with few numbers of tasks and a huge amount of execution time has a larger number of incomplete tasks with an increscent peer churn, and it arrives at saturation sooner at the point of 30% of peer leave and has an unpredictable behaviour after that.

FMProx works well under a moderate churn for data-intensive or I/O-intensive workflows such as CyberShake and Montage. The behaviour of RandomProx is almost the same as FMProx under different workflow applications. But, RandomNoProx and RRNoProx have a little different action. The behaviour of these policies is stable under the lower churn rate. They are in the steady state till 25% of peer churn in the Montage workflow, 30% of peer churn in the CyberShake workflow, 20% in the Epigenomics workflow and 15–25% in the Sipht workflow. After that, these policies approach oscillation or saturation point exponentially.

5. Conclusions

In this paper, we proposed a proximity-aware and decentralised workflow-scheduling system in P2P-based VC systems. Each workflow application is considered as a DAG and it can have QoS constraints such as minimum CPU speed and minimum RAM or hard disk requirements. The proposed scheduling system consists of three phases. In the first phase, each workflow application is partitioned with the multi-way Fiduccia–Mattheyses

algorithm into sub-workflows with respect to minimisation of data dependencies among them. The second phase of workflow-scheduling system finds resources with respect to QoS constraints of workflow, load balancing and proximity of resources. In the third phase, sub-workflows run on each resource based on local scheduling algorithm. We compared the performance of the proposed workflow-scheduling system with three other baseline policies. The results of the experiments indicated that FMProx significantly decreases AMN of the system with improvement factors of 8.3%, 46.8% and 53.6% on average with respect to RandomProx, RandomNoProx and RRNoProx, respectively. The proposed scheduling policy performs better for data-intensive and I/O-intensive workflows. Because the emphasis of the proposed policy is on the reduction of communication overhead, it decreases AMN of data-intensive workflows by reducing the overhead of transferring input/output files with the selection of approximate resources with a minimum communication overhead relative to the injection node and the client node of the workflow. Also, it lessens the communication delay in I/O-intensive workflows by grouping tasks with more data dependencies in the same sub-workflow.

As part of future work, we will consider other partitioning algorithms to improve the workflow-partitioning phase. Another interesting extension is using Cloud resources in some of the peers. Some workflow applications have QoS requirements such as deadline that could not be satisfied by the available resources of the VC systems, thus Cloud resources can be used in order to satisfy the QoS requirements of these applications. Also, the performance improvement of the proposed workflow-scheduling system for CPU-intensive workflows is left for future work.

Acknowledgement

The authors thank Rodrigo N. Calheiros for useful discussions.

References

[1] Anderson DP, Reed K. Celebrating diversity in volunteer computing. HICSS-42 Conference; Waikoloa, Big Island, HI; 2009.
[2] Anderson DP. BOINC: a system for public resource computing and storage. GRID Workshop; Pittsburgh, PA; 2004.
[3] Epema DHJ, Livny M, Dantzig RV, Evers X, Pruyne J. A worldwide flock of condors: load sharing among workstation clusters. Future Gener Comput Syst. 1996;12:53–65.
[4] Litzkow MJ, Livny M, Mutka MW. Condor – a hunter of idle workstations. ICDCS Conference; Amsterdam, The Netherlands; 1998.
[5] Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the condor experience. Concurrency Comput Pract Exp. 2005;17:323–356.
[6] Chu X, Nadiminti K, Jin C, Venugopal S, Buyya R. Aneka: next-generation enterprise grid platform for e-science and e-business applications. E-SCIENCE Conference; Bangalore, India; 2007.
[7] Anderson DP, Cobb J, Korpela E, Lebofsky M, Werthimer D. SETI@home: an experiment in public-resource computing. Commun ACM. 2002;45:56–61.
[8] EDGeS@Home project. Available from: http://home.edges-grid.eu/
[9] Christensen C, Aina T, Stainforth D. The challenge of volunteer computing with lengthy climate model simulation. E-SCIENCE Conference; Melbourne, VIC; 2005.
[10] Chien A, Calder B, Elbert S, Bhatia K. Entropia: architecture and performance of an enterprise desktop grid system. J Parallel Distrib Comput. 2003;63:597–610.
[11] Cappello F, Djilali S, Fedak G, Herault T, Magniette F, Neri V, Lodygensky O. Computing on large scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. Future Gener Comput Syst. 2005;21:417–437.

[12] Kacsuk P, Kovacs J, Farkas Z, Marosi CA, Gombas G, Balaton Z. SZTAKI desktop grid (SZDG): a flexible and scalable desktop grid system. J Grid Comput. 2009;7:439–461.
[13] Cirne W, Brasileiro F, Andrade N, Costa LB, Andrade A, Novaes R, Mowbray M. Labs of the World, Unite!!! J Grid Comput. 2006;4:225–246.
[14] Anglano C, Canonico M, Guazzone M. The ShareGrid peer-to-peer desktop grid: infrastructure, applications, and performance evaluation. J Grid Comput. 2010;8:543–570.
[15] Butt AR, Zhang R, Hu CY. A self-organizing flock of condors. J Parallel Distrib Comput. 2006;66:145–161.
[16] Abbes H, Cerin C, Jemni M. A decentralized and fault-tolerant desktop grid system for distributed applications. Concurrency Comput Pract Exp. 2010;22:261–277.
[17] Abbes H, Cerin C, Jemni M. Bonjourgrid: orchestration of multi-instances of grid middlewares on institutional desktop grids. IPDPS Conference; Rome, Italy; 2009.
[18] Byun E, Kim H, Choi S, Lee S, Han YS, Gil JM, Jung SY. Self-gridron: reliable, autonomous, and fully decentralized desktop grid computing system based on neural overlay network. PDPTA Conference; Las Vegas, NV; 2008.
[19] Taylor IJ, Deelman E, Gannon DB, Shields M. Workflows for e-Science: scientific workflows for grids. Berlin: Springer; 2007.
[20] Kacsuk P. How to make BOINC-based desktop grids even more popular. IPDPSW Conference; Anchorage, AK; 2011.
[21] Anderson DP. Volunteer computing the ultimate cloud. ACM Crossroad. 2010;16:7–10.
[22] Fedak G. Recent advances and research challenges in desktop grid and volunteer computing. In: Desprez F, Getov V, Priol T, Yahyapour R, editors. Grids, P2P and services computing. Berlin: Springer; 2010. p. 171–185.
[23] Kalayci S, Dasgupta G, Fong L, Ezenwoye O, Sadjadi S. Distributed and adaptive execution of condor DAGman workflows. SEKE Conference; Redwood City, CA; 2010.
[24] Medina A, Lakhina A, Matta I, Byers J. BRITE: an approach to universal topology generation. MASCOTS Conference; Cincinnati, OH; 2001.
[25] Elwaer A, Harrison A, Kelley I, Taylor I. Attic: a case study for distributing data in BOINC projects; IPDPS Conference; Anchorage, AK; 2011.
[26] Kumar S, Das S, Biswas R. Graph partitioning for parallel applications in heterogeneous grid environments. IPDPS Conference; Fort Lauderdale, FL; 2002.
[27] Chen W, Deelman E. Integration of workflow partitioning and resource provisioning. CCGrid Conference; Ottawa, Canada; 2012.
[28] Malecot P, Kondo D, Fedak G. XtremLab: a platform for observation and characterization Grids of PCs on the Internet. RenPar Conference; French; 2006.
[29] Deelman E, Callaghan S, Field E, Francoeur H, Graves R, Gupta V, Jordan TH, Kesselman C, Maechling P, Mehta G, Okaya D, Vahi K, Zhao L. Managing large-scale workflow execution from resource provisioning to provenance tracking: the CyberShake example. E-SCIENCE Conference; Amsterdam, The Netherlands; 2006.
[30] Berriman GB, Deelman E, Good JC, Jacob JC, Katz DS, Kesselman C, Laity AC, Prince TA, Singh G, Su MH. Montage: a grid-enabled engine for delivering custom science grade mosaics on demand. IS&T/SPIE Conference; San Jose, CA; 2004.
[31] Bharathi S, Chervenak A, Deelman E, Mehta G, Su MH, Vahi K. Characterization of Scientific Workflows. WORKS Conference; Austin, TX; 2008.
[32] Juve G, Deelman E, Vahi K, Mehta G, Berriman B, Berman BP, Maechling P. Scientific workfow applications on Amazon EC2. E-Science Conference; Oxford, UK; 2009.
[33] Livny J, Teonadi H, Livny M, Waldor MK. High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs. PLoS ONE. 2008;3:e3197. Available from: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2527527/
[34] Workflow Generator. Available from: http://pegasus.isi.edu/workflow_gallery/index.php
[35] Ghafarian T, Deldari H, Javadi B, Yaghmaee MH, Buyya R. CycloidGrid: a proximity-aware P2P-based resource discovery architecture in volunteer computing systems. Future Gener Comput Syst. 2013;29:1583–1595.
[36] Fox G, Qiu X, Beason S, Choi J, Ekanayake J, Gunarathne T, Rho M, Tang H, Devadasan N, Liu G. Biomedical case studies in data intensive computing. In: Jaatun MG, Zhao G, Rong C, editors. Cloud computing. Berlin: Springer; 2009. p. 2–18.
[37] Ghafarian T, Deldari H, Javadi B, Buyya R. A proximity-aware load balancing in peer-to-peer-based volunteer computing systems. J Supercomput. 2013;65:797–822.

[38] Kim JS, Nam B, Keleher P, Marsh M, Bhattacharjee B, Sussman A. Trade-offs in matchmaking job and balancing load for distributed desktop grids. Future Gener Comput Syst. 2008;24:415–424.
[39] Sanchis LA. Multiple-way network partitioning. IEEE Trans Comput. 1989;38:62–81.
[40] Sanchis LA. Multiple-way network partitioning with different cost functions. IEEE Trans Comput. 1993;42:1500–1504.
[41] Celaya J, Arronategui U. Distributed scheduler of workflows with deadlines in a P2P desktop grid. PDP Conference; Pisa, Italy; 2010.
[42] Rahman M, Ranjan R, Buyya R. Cooperative and decentralized workflow scheduling in global grids. Future Gener Comput Syst. 2010;26:753–768.
[43] Lee YC, Zomaya AY, Siegel HJ. Robust task scheduling for volunteer computing systems. J Supercomput. 2010;53:163–181.
[44] Di S, Wang CL. Dual-phase Just-in-time workflow scheduling in P2P Grid systems. ICPP Conference; San Diego, CA; 2010.
[45] Garey MS, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. New York, NY: W.H. Freeman; 1979.
[46] Aggarwal CC, Wang H. A survey of clustering algorithms for graph data. In: Aggarwal CC, Wang H, editors. Managing and mining graph data. Berlin: Springer; 2010. p. 275–301.
[47] Fiduccia CM, Mattheyses RM. A linear time heuristic for improving network partitions. DAC Conference; Las Vegas, NV; 1982.
[48] Topcuouglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst. 2002;13:260–274.
[49] Basher N, Mahanti A, Williamson C, Arlitt M. A comparative analysis of web and peer-to-peer traffic. WWW Conference; Beijing, China; 2008.
[50] Iosup A, Sonmez O, Anoep S, Epema D. The performance of bags-of-tasks in large-scale distributed systems. HPDC Conference; Boston, MA; 2008.