Future Generation Computer Systems 26 (2010) 753–768

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Cooperative and decentralized workflow scheduling in global grids

Mustafizur Rahman*, Rajiv Ranjan, Rajkumar Buyya

Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Parkville, VIC 3010, Australia

ARTICLE INFO

Article history:
Received 12 August 2008
Received in revised form
26 March 2009
Accepted 13 July 2009
Available online 24 July 2009

Keywords:
Workflow scheduling
Decentralized coordination
Peer-to-peer systems
Global grids

ABSTRACT

Existing Grid scheduling systems, such as e-Science workflow brokers operate in tandem but lack the notion of cooperation mechanism that can lead to efficient application schedules across distributed resources. Lack of coordination exacerbates the utilization of various resources including computing cycles and network bandwidth. Moreover, current brokering systems have evolved around centralized client/server or hierarchical models. The responsibilities of the key functionalities such as resource discovery are delegated to the centralized server machines. Centralized models have well-known drawbacks regarding scalability, single point of failure, and network congestion at links leading to the server.

To overcome these problems, this paper proposes a novel approach for decentralized and cooperative workflow scheduling in a dynamic and distributed Grid resource sharing environment. The participants in the system, such as the workflow brokers, resources, and users who belong to multiple control domains, work together to enable a single cooperative resource sharing environment. The proposed approach derives from a Distributed Hash Table (DHT) based *d*-dimensional logical index space with regard to resource discovery, coordination and overall system decentralization. The DHT-based *d*-dimensional index space serves as a blackboard system, where distributed participants can post and search complex coordination objects that regulate system wide scheduling decision making. With the implementation of our approach, not only the performance bottlenecks are likely to be eliminated but also efficient scheduling with enhanced scalability will be achieved. We evaluate and prove the feasibility of our approach through an extensive trace-driven simulation. In order to show the performance of the proposed approach against non-cooperative scheduling approach, we conduct experiment for different sizes of workflow. The results show that our scheduling technique can reduce the makespan up to 25% and demonstrates improved load balancing capability. We also compare the performance of the proposed approach against a centralized coordination technique and show that our approach is as efficient as the centralized technique with respect to achieving coordinated schedules.

1. Introduction

Distributed resource sharing systems such as Grids [1] and PlanetLabs [2] often exhibit the classical economics paradox called “tragedy of commons” during the period of high demand. Study undertaken in [3], confirms that PlanetLab environment often experiences the problem of flash crowd where a growing number of users simultaneously request “slices” on arbitrarily selected nodes to host their experiments on various scientific applications such as e-Science workflows. Such bursty behaviour of users often lead to poor system performance. Under the current PlanetLab resource management setting, nodes schedule the user requests locally, without any provision to discover the usage status of other nodes in the system or coordinate the local resource usage across the system. Similarly, users have no means to coordinate their resource demands with other users in the system leading to over-utilization of particular set of nodes. Further, the users who cannot successfully finish their experiments due to competing or conflicting requests in the system tend to retry their experiments which further aggravate the situation.

Workflow scheduling is a process of finding the efficient mapping of tasks in a workflow to the suitable resources so that the execution can be completed with the satisfaction of objective functions such as execution time minimization as specified by Grid users. Therefore, the efficiency of the workflow scheduling algorithm directly affects the performance of the system with respect to delivered Quality of Service (QoS), utilization, and system performance. Majority of existing approaches to workflow scheduling are non-coordinated (refer to Section 2). Workflow brokers perform scheduling related activities independent of the other brokers in the system. They directly submit their

* Corresponding author. Tel.: +61 3 8344 1355; fax: +61 3 9348 1184.
E-mail addresses: mmrahman@csse.unimelb.edu.au (M. Rahman), rranjan@csse.unimelb.edu.au (R. Ranjan), raj@csse.unimelb.edu.au (R. Buyya).

0167-739X/$ – see front matter Crown Copyright © 2009 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2009.07.002

applications to the underlying resources *without* taking into account the current load, priorities, utilization scenarios of other application level schedulers. Clearly, this leads to over-utilization or a bottleneck on some valuable resources while leaving others largely underutilized. Furthermore, these brokers do not have a coordination mechanism and this exacerbates the load sharing and utilization problems of distributed resources.

Another major drawback associated with the current workflow scheduling approaches is that the existing workflow brokers rely on the centralized or semi-centralized hierarchical resource information services such as MDS-2,3,4 [4]. Current studies have shown that [5] existing centralized model for information services do not scale well as the number of users, brokers and resource providers increase in the system. Hence in case, the centralized links leading to these services fail, then no broker in the system can undertake scheduling related activities due to the lack of up-to-date resource information.

Further, Grids are heterogeneous and dynamic environments consisting of computing, storage and network resources with different capability and availability. In [6], it is shown that the dynamic scheduling algorithms based on heuristics adapt to the changing resource conditions of Grids by performing just-in-time scheduling (generating schedules that map the tasks dynamically). But the workflow brokers, running these dynamic algorithms, still need to be coordinated in order to avoid any conflict and generate schedule globally.

To overcome the limitations of existing approaches, we propose a fully decentralized and cooperative workflow scheduling approach based on DHT-based coordination space [7,8] that provides a global virtual shared space. This space can be concurrently and associatively accessed by all participants in the system and the access is independent of the actual physical or topological proximity of the objects or hosts. New generation DHT-based routing algorithms [9,10] form the basis for organizing the coordination space. In the proposed approach, workflow brokers post their resource demands by injecting a *Resource Claim* object into the DHT-based decentralized coordination space, while resource providers update the resource information by injecting a *Resource Ticket* object.

These objects are mapped to the DHT-based coordination space using a spatial hashing technique [11], the details of which is given in Section 3.3.3. Once a resource ticket matches with one or more resource claims, the coordination space sends *notification* messages to the resource claimers such that it does not lead to the overloading of the concerned resource ticket issuer. Thus, this mechanism prevents the workflow brokers from overloading the same resource. DHT-based coordination space is managed by a software service (a component of Grid workflow broker service) called coordination service. It undertakes activities related to decentralized load balancing, coordination space management etc. With the implementation of our approach, not only the performance bottlenecks are likely to be eliminated but also efficient scheduling with enhanced scalability will be achieved. Our main **contributions** in this paper are as follows:

(i) a novel decentralized and cooperative workflow scheduling approach.
(ii) a methodology for utilizing the DHT-based *d*-dimensional spatial index for managing complex coordination objects and decentralizing the system.
(iii) a decentralized resource provisioning algorithm for mapping the tasks to the resources by the DHT-based coordination space.
(iv) a comprehensive simulation-based study of the proposed scheduling approach and analyze different performance metrics to demonstrate that our approach is scalable in terms of scheduling message complexity and makespan of the workflow.
(v) an evaluation of the proposed scheduling technique measured along two performance dimensions (load balancing and coordination efficiency) as compared to non-cooperative scheduling and centralized coordination techniques.

The rest of this paper is organized as follows. In the next section, we describe the related work that are focused on decentralized and cooperative workflow scheduling. Section 3 provides a brief description of the system models used in our scheduling approach. In Section 4, we present the details of our proposed algorithms for task scheduling, resource provisioning and coordination. Simulation setups, and the findings of the experiments performed are analyzed, compared and discussed in Section 5. Finally, we conclude the paper with the direction for future work in Section 6.

2. Related work

The main focus of this section is to compare the novelty of the proposed work with respect to the existing systems. We classify the related research in two main areas:

2.1. Scheduling infrastructure

With the increasing interest in Grid workflows, many Grid workflow systems such as Pegasus [12], Triana [13], Taverna [14], Condor DAGMan [15], Kepler [16], SwinDeW-G [17], Gridbus [18] and Askalon [19] have been developed in recent years. Among these systems, in terms of workflow scheduling infrastructure, SwinDeW-G and Triana utilize decentralized P2P-based technique. However, the P2P communication in SwinDeW-G and Triana is implemented by JXTA protocol, which uses a broadcast technique. In this work, we use a DHT (such as Chord) based P2P system for handling resource discovery and scheduling coordination. The employment of DHT gives the system the ability to perform deterministic discovery of resources and produce controllable number of messages in comparison to using JXTA.

Condor system [20] implements a matchmaking algorithm that utilizes classified advertisements (ClassAds) mechanism for mapping user's applications to suitable server machines. In this system, server machines advertise their capability and constraints to central Matchmaker by encapsulating the resource characteristics (such as CPU type and speed, memory, load) in ticket ClassAds. Accordingly, users express the resource preferences by submitting a claim ClassAds to the Matchmaker. Following that, the Matchmaker executes as generic matchmaking algorithm to create ticket to claim matches incorporating the constraints and preferences. In this paper, following the same terminologies, ticket is used for updating a Grid resources' information to the decentralized coordination space and claim is used for referring to the user's lookup requests. Additionally, our approach applies a fully decentralized and distributed algorithm for mapping tickets to claims, which guarantees high scalability and negates single point of failure in large scale federation of Grids.

2.2. Coordination mechanism

In [18], a workflow enactment engine, with a just-in-time scheduling system using tuple space, is proposed to manage the execution of scientific workflow applications. In this system, every task has its own scheduler called Task Manager (TM), which implements a scheduling algorithm and handles the processing of tasks. The TMs are controlled by a Workflow Coordinator (WC). Besides, an event-driven mechanism with subscription–notification methods supported by the tuple space model is used to control and manage scheduling activities. In this work, although the task managers are working in a distributed fashion, they communicate with each other through the tuple space which is implemented based on a client–server-based centralized technology. Further, the WC in this system does not communicate with other WCs, managing workflow applications in the Grid. In contrast to this work, we propose a completely decentralized workflow coordinator based on a scalable P2P network model.

Yao et al. [21] have extended the aforementioned architecture with an additive Reinforcement Learning Agent (RLA) to perform the Decentralized Dynamic Workflow Scheduling using Reinforcement Learning (DDWS-RL) algorithm. DDWS-RL-enabled TMs query information from the RLA and make decision on resource selection at the time of task execution. Thus, RLA is used in the tuple space to facilitate the scheduling algorithm to be more efficient. However, this approach also involves the same architecture as the other approach, stated above, which is based on centralized client–server model with respect to resource discovery and coordination space management. In contrast, with our approach there is no central component that can prove to be a bottleneck.

In [22], Chen et al. proposed a policy-based coordination mechanism to manage the services provided by the peers in the Federated Service Providing (FSP) system. Peers in FSP system share the computation resources for offering domain specific services. When a peer receives a service request that cannot be processed by itself (because either it is busy or the service type is different), it tries to find other peer in the system capable of processing the request. In order to regulate the interactions among the peers, they have proposed a recruiting protocol and a policy-driven architecture is also introduced to control the decision making process of each peer. On the other hand, all the peers in our Grid-federation system offer similar services and the coordination among the peers is managed by utilizing a DHT-based coordination service.

3. System models

3.1. Grid model

The proposed workflow scheduling algorithm utilizes the *Grid-Federation* [23] model with regard to resource organization and Grid networking. Grid-Federation aggregates distributed resource brokering and allocation services as part of a cooperative resource sharing environment. Table 1 shows the notations for Resource, Workflow and Index models.

Definition 3.1 (*Grid-Federation*). The Grid-Federation, $G_F = \{R_1, R_2, \ldots, R_n\}$, consists of a number of sites, n, with each site contributing its resource to the federation. Every site in the federation has its own resource description R_i which contains the definition of the resource that it is willing to contribute. R_i, can include information about the CPU architecture, number of processors, memory size, secondary storage size, operating system type, etc.

In this work, $R_i = (p_i, x_i, \mu_i, ø_i)$, which includes the number of processors p_i, processor architecture x_i, their speed μ_i, and installed operating system type $ø_i$.

Resource brokering, indexing and allocation in Grid-Federation are facilitated by a Resource Management System (RMS) known as Grid-Federation Agent (GFA). Fig. 1 shows an example Grid-Federation resource sharing model consisting of Internet-wide distributed parallel resources. Every contributing site maintains its own GFA service. GFA service is composed of 3 software entities: Grid Resource Manager (GRM), Local Resource Management System (LRMS) and Distributed Information Manager (DIM) or Grid Peer.

Table 1
Notations: Resource, workflow, and index models.

Symbol	Meaning
Resource	
r	Number of resources or GFAs in the Grid system
r_i	ith Resource in the system
R_i	Configuration of the ith resource in the system.
G_i	ith GFA in the system
t_p	Number of task executions per processor
x_i	Processor architecture for resource at GFA i.
p_i	Number of processors for resource at GFA i.
ϕ_i	Operating system type for resource at GFA i.
μ_i	Processor-speed at GFA i.
Workflow	
t	Number of tasks in a workflow.
T	Total number of tasks in the system.
e	Total number of inter-task dependencies in a workflow.
l	Number of levels in a workflow.
w	Width of a fork-join workflow.
α	Number of tasks in a level that can be executed concurrently.
$W_{i,j,k}$	ith workflow from the jth user of kth GFA in the system.
$T_{x_{i,j,k}}$	xth task of the workflow, $W_{i,j,k}$.
$\mu_{x_{i,j,k}}$	Processor-speed required by the task $T_{x_{i,j,k}}$.
$p_{x_{i,j,k}}$	Number of processors required by the task $T_{x_{i,j,k}}$.
$x_{x_{i,j,k}}$	Processor architecture required by the task $T_{x_{i,j,k}}$.
$\phi_{x_{i,j,k}}$	Operating system required by the task $T_{x_{i,j,k}}$.
Index	
$r_{x_{i,j,k}}$	A claim posted for task $T_{x_{i,j,k}}$.
u_i	A ticket issued by the ith GFA/broker.
dim	Dimensionality or number of attributes in the Cartesian space.
f_{min}	Minimum division level of d-dimensional index tree.

The GRM component of GFA exports a Grid site to the federation and is responsible for coordinating federation wide application scheduling and resource allocation. The GRM is responsible for scheduling locally submitted jobs (workflows) in the federation. Further, it also manages the execution of remote jobs (workflows) in conjunction with the local resource management system. The LRMS software module can be realized using systems such as PBS [24] and SGE [25]. Additionally, LRMS performs other activities for facilitating federation wide job submission and migration process such as answering the GRM queries related to local job queue length, expected response time, and current resource utilization status.

The Grid peer module in conjunction with publish/subscribe indexing service performs tasks related to decentralized resource lookups and updates. A Grid Peer service generates two basic types of objects with respect to coordinated resource brokering: (i) a claim, is an object sent by a broker service to the P2P space for locating the resources that match the user's application requirements and (ii) a ticket, is an update object sent by a Grid site mentioning about the underlying resource conditions. Since, a Grid resource is identified by more than one attribute, a claim or ticket is always d-dimensional. Further, both of these queries can specify different kinds of constraints on the attribute values. If a query specifies a fixed value for each attribute then it is referred to as a *d-dimensional Point Query* (DPQ). However, if the query specifies a range of values for the attributes, then it is referred to as a *d-Dimensional Window Query* (DWQ) or a *d-Dimensional Range Query* (DRQ) [26]. Thus the Grid peer component of a GFA service is responsible for ticket publication, claim subscription, and overlay management processes.

3.2. Application model

In this work, we consider the Scientific workflow applications as the case study for the proposed scheduling approach. A Scientific

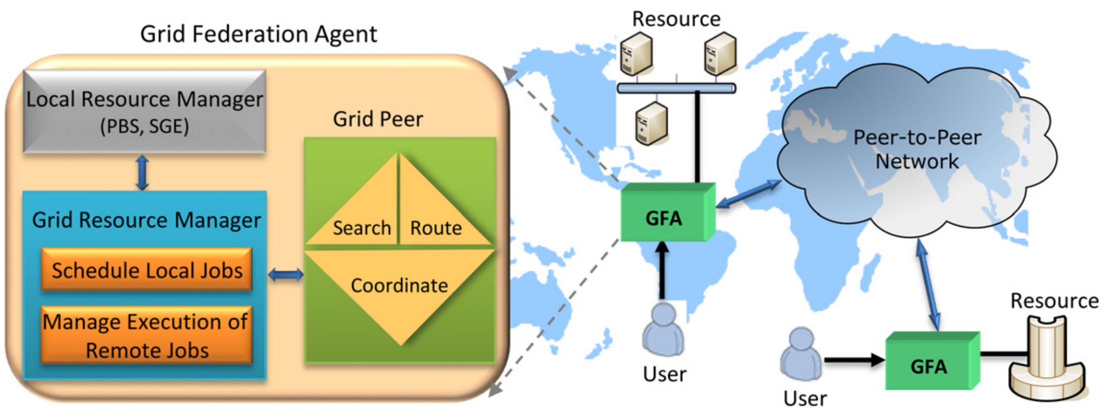


Fig. 1. Grid-Federation.

workflow application can modeled as a Directed Acyclic Graph (DAG), where the tasks in the workflow are represented as nodes in the graph and the dependencies among the tasks are represented as the directed arcs among the nodes. In general, a task in a workflow is a set of instructions that can be executed on a single processing element of a computing resource [27]. Examples of such workflow applications are [28–32].

Definition 3.2 (*Scientific Workflows*). Scientific workflows describe a series of large number of structured activities and computations that arise in scientific problem solving. Usually, scientific workflows are data or computation intensive and the activities in the workflow have data or control dependencies among them.

Example 3.1. Let $V_{i,j,k}$ be the finite set of tasks $\{T_1, T_2, \ldots, T_x, \ldots, T_y, T_m\}$ for the ith submitted workflow from the jth user of kth workflow broker (GFA) and $E_{i,j,k}$ be the set of dependencies of the form $\{T_{x_{i,j,k}}, T_{y_{i,j,k}}\}$ where, $T_{x_{i,j,k}}$ is the parent task of $T_{y_{i,j,k}}$. Thus, the ith submitted workflow from the jth user of kth workflow broker in the system can be represented as,

$$W_{i,j,k} = \{V_{i,j,k}, E_{i,j,k}\}.$$

In a workflow, we call a task that does not have any parent task, an entry task and a task that does not have any child task, an exit task. We also assume that a child task cannot be executed until all of its parent tasks are completed. At any time of scheduling, the task that has all of its parent tasks finished, is called a 'ready' task.

3.3. Coordination space model

In this section, we first describe the communication, coordination and indexing models that are utilized to facilitate the P2P coordination space. Then, we present the composition of objects, access primitives that form the basis for coordinating application schedules among distributed GFAs/brokers.

3.3.1. Layered design of the coordination service

We design and architect the coordination service based on a layered approach. The architecture consists of three layers: the *Application* layer, *Core Services* layer and *Connectivity* layer. Grid Services such as workflow brokers work at the Application layer and insert objects (claims/tickets) via the Core Services layer. In the context of a GFA, the GRM software module operates at the Application layer.

We use the Coordination service as a sub-layer of the Core Services layer. The Coordination service accepts the application objects such as claims/tickets. These objects encapsulate coordination logic, which in this case is the resource provisioning logic.

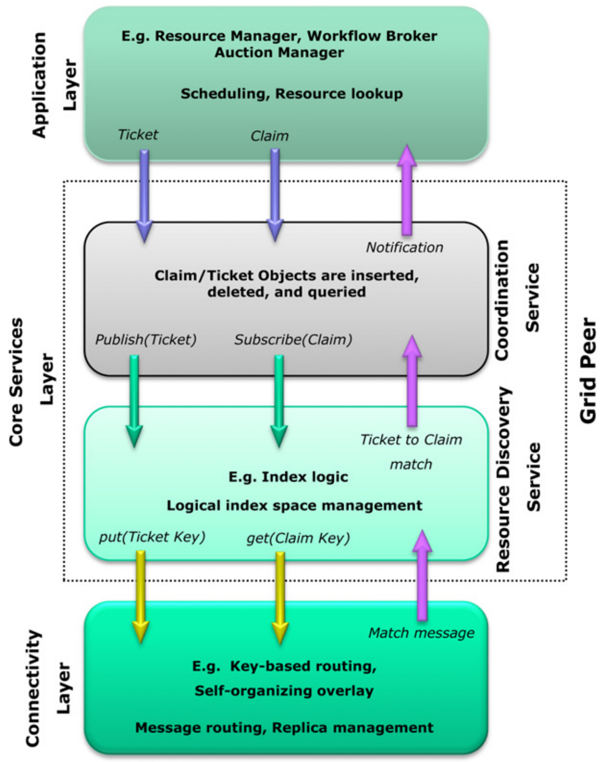

Application Layer
E.g. Resource Manager, Workflow Broker Auction Manager
Scheduling, Resource lookup
Ticket
Claim
Notification
Claim/Ticket Objects are inserted, deleted, and queried
Publish(Ticket)
Subscribe(Claim)
Coordination Service
Core Services Layer
Ticket to Claim match
E.g. Index logic
Logical index space management
put(Ticket Key)
get(Claim Key)
Resource Discovery Service
Grid Peer
Match message
E.g. Key-based routing, Self-organizing overlay
Message routing, Replica management
Connectivity Layer

Fig. 2. Layered design of the coordination model.

These objects are managed by the coordination service. The calls between the Coordination service and Resource Discovery service are made through the standard publish/subscribe technique. The Resource Discovery service is responsible for managing the logical index space and communicating with the Connectivity layer. The calls between Core Services layer and Connectivity layer are made through standard DHT primitives such as *put()*, *get()* that are defined in the P2P Common Application Programming Interface specification [33]. As shown in Fig. 2, Core Services layer is managed by the Grid Peer module of a GFA service.

The Connectivity layer is responsible for undertaking key-Based routing in the DHT space such as Chord, CAN, Pastry etc. In this paper, for simulation, we model the Chord substrate at the Connectivity layer. Chord hashes the peers and objects (such as fileIds, logical indices, etc) to the circular identifier space and the average lookup complexity is $O(\log r)$ steps with high probability. Each peer in the Chord network is required to maintain routing table state of $O(\log r)$ other peers, where r is the total number of Grid peers in the system.

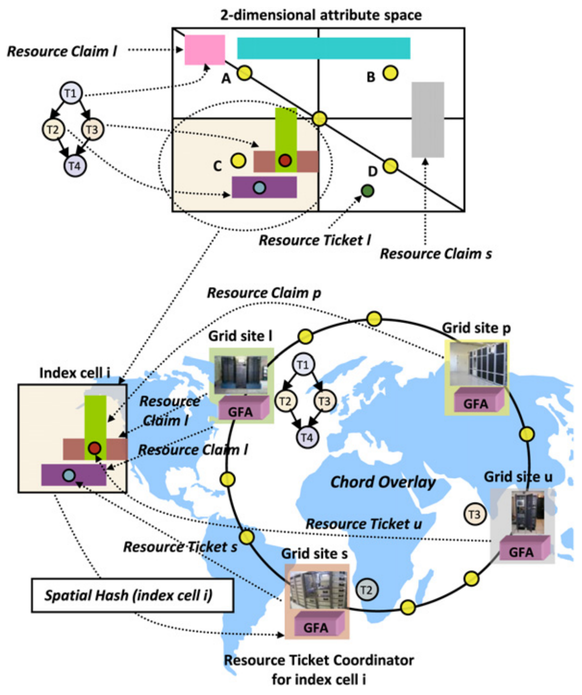


Fig. 3. Resource allocation and application scheduling coordination across Grid sites.

3.3.2. Coordination objects

This section gives details about the resource claim and ticket objects that form the basis for enabling decentralized coordination mechanism among the brokers/GFAs in the Grid-Federation system. These coordination objects include Resource Claim and Resource Ticket.

Every GFA in the federation posts its resource ticket through the local Coordination service. A resource ticket object u_i or update query consists of a resource description R_i, for a resource i.

Example 3.2 (*Resource Ticket*). Total-Processors = 100 && Processor-Arch = Pentium && Processor-Speed = 2 GHz && Operating System = Linux && Utilization = 0.80.

A resource claim or lookup object encapsulates the resource configuration requirements of a task in the workflow submitted by the users. In this work, we focus on the workflows for which the requirements are confined to a computational Grid or PlanetLab resources. Users submit their workflow application's resource requirements to the local GFA. The corresponding GFA service is responsible for searching the suitable resources in the federated system. A GFA aggregates the characteristics of a task including the number of processors, processor architecture, and installed operating system with constraint on maximum speed, and resource utilization into a resource claim object, $r_{i,j,k}$.

Example 3.3 (*Resource claim*). Total-Processors $\geq$70 && Processor-Arch = Pentium && 2 GHz $\leq$ Processor-Speed $\leq$ 5GHz && Operating System = Solaris && 0.0 $\leq$ Utilization $\leq$ 0.90.

The GRM module of a GFA passes the resource ticket and claim object to the coordination service operating at the Core services layer. Recall that, the Core Services layer is managed by the Grid peer module, which interacts with the DHT-based overlay network. The operation between Grid peer module and DHT-based overlay is transparent to the GRM. In other words, a GRM is not aware of how the Grid peer module is routing, searching, and matching the objects in the system. It is the responsibility of a Grid peer module to implement specific communication and data

Table 2
Claims stored with the coordination service at time τ.

Time	Claim ID	$\mu_{x_{i,j,k}}$	$p_{x_{i,j,k}}$	$x_{x_{i,j,k}}$	$\phi_{x_{i,j,k}}$	Rank
200	Claim 1	>800	1	Intel	Linux	0.2
350	Claim 2	>1200	1	Intel	Linux	0.3
500	Claim 3	>700	1	Sparc	Solaris	0.1
700	Claim 4	>1500	1	Intel	Windows XP	0.4

Table 3
Ticket published to the coordination service at time τ.

Time	GFA ID	μ_i	p_i	p_i avail	x_i	ϕ_i
900	GFA-8	1400	3	2	Intel	Linux

organization methods, which can provide desired functionality to the Application layer services such as a GRM. In order to efficiently route, search, and match the d-dimensional claim and ticket objects, the Grid peer module embeds a logical spatial data-structure over the DHT overlay space. More details on this spatial data-structure is given in Section 3.3.3.

The resource ticket and claim objects are spatially hashed to an index cell i in the d-dimensional coordination space. Similarly, the coordination services of the resource sites in the Grid network hash themselves into this coordination space using the overlay hashing function (SHA-1 in the case of Chord and Pastry). In Fig. 3, resource claim objects issued by site p and l are mapped to the index cell i and these claim objects are currently hashed to the site s. Thus site s is responsible for coordinating the resource sharing among all the resource claims that are mapped to the cell i. Subsequently, site u issues a resource ticket (shown as small dark circles in Fig. 3) that falls under a region of the space currently required by users at site p and l. In this case, the coordination service of site s has to decide, which of the sites (i.e. either l or p or both) will be allowed to claim the ticket issued by site u. This load-distribution decision is based on the fact that it should not lead to over-provisioning of resources at site u.

In Table 2, we show an example list of claim objects that are stored with a coordination service at time $t = 900$ s. Essentially, the claims in the list arrived at a time $\leq$900 and are waiting for a suitable ticket object that can meet its resource configuration requirements. While Table 3 depicts the list of ticket objects that have arrived at $t = 900$ s. Following the ticket arrival event, the coordination service undertakes a procedure that divides this ticket object among the list of claims. Based on the resource attribute specification and availability, only Claim 1 and Claim 2 matches the ticket's resource configuration. As specified in the ticket object, there are currently 2 processors are available with the GFA 8, which is equal to the sum of processors required by Claim 1 and 2 (i.e., 2). Hence in this case the coordination service, based on the priority of the task (rank), first notifies the GFA that has posted Claim 2 and follows it with the GFA responsible for Claim 1. However, Claim 3 and 4 have to wait for the arrival of tickets that can match their required resource configuration.

Once a resource ticket matches with one or more resource claims, a coordination service sends *notification* messages to the resource claimers such that it does not lead to the overloading of the concerned resource ticket issuer. Thus, this mechanism prevents the workflow brokers from overloading the same resource. As a result, the problem of conflicting schedules is overcome.

This paper focuses on scheduling of workflow application, which consists of a collection of tasks. Our approach supports allocation of different tasks in a workflow across multiple sites in the Grid-Federation (as shown in Fig. 4), if the total number of processors needed for executing all the tasks in a workflow are not available within a single Grid site. As discussed earlier,

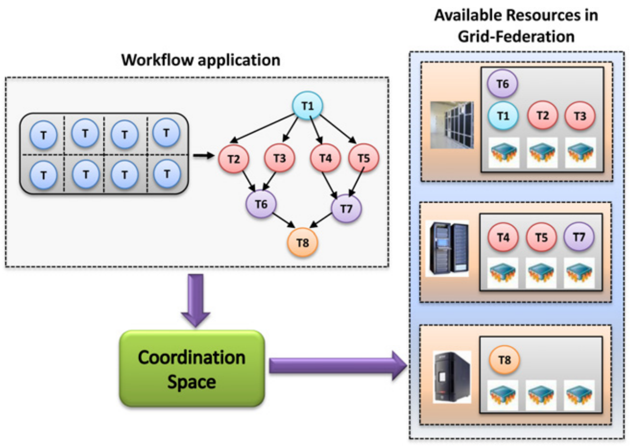


Fig. 4. Multi-Site allocation of workflow tasks.

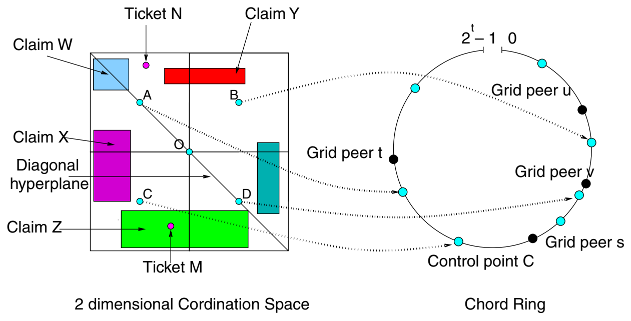


Fig. 5. Spatial resource claims $\{W, X, Y, Z\}$, cell control points $\{A, B, C, D\}$, point resource tickets $\{M, N\}$ and some of the spacial hashings (dotted lines) to the Chord, i.e., the d-dimensional coordinate values of a cell's control point is used as the DHT key and hashed on to the Chord ring. For this figure, $f_{\min} = 2$, $\dim = 2$.

in our application model, each task needs availability of only one processor within a Grid site (refer to Section 3.2). Thus the resource claim object for a task encapsulates request for a single processor, i.e. the requirement of the number of processors available is 1 (see Table 2). In case, at any given instance of time, if no resource ticket is able to offer single processor as requested by a resource claim object then the claim object is stored in the coordination spaced until one of the Grid site publishes a resource ticket offering one available processor. Grid sites publish resource tickets after a certain interval of time, which we model as the exponential time distribution in our experiments.

In addition, our cooperative and decentralized scheduling approach can also support mapping of Message Passing Interface (MPI) type of applications [34] that require multiple processors based on parallelism and granularity of the application. These processors can be acquired from a single Grid site or multiple Grid sites depending on resource availability across the environment. A resource claim object for a MPI application can encapsulate request for multiple processors, which can be matched against one resource ticket (single-site allocation) or multiple resource tickets (multi-site allocation). Hence, if the resource claim cannot be matched to any resource ticket as its resource requirement, such as the number of processors is too large to be satisfied by one Grid site then multiple tickets can be aggregated to meet the processor requirement, requested by the claim object. However, multi-site scheduling of a MPI application that has run time communication dependency requires handling of additional complexities related to co-allocation, and synchronization [34]. Supporting different aspects of scheduling for this kind of applications in decentralized and distributed multi-site environments is beyond the scope of this work as we focus on workflow applications in this paper.

3.3.3. *D-dimensional coordination object mapping and routing*

1-dimensional hashing, provided by current implementation of DHTs are insufficient to manage complex objects such as resource tickets and claims. DHTs generally hash a given unique value/identifier (e.g. a file name) to a 1-dimensional DHT key space and hence they cannot support mapping and lookups for complex objects. Management of these objects whose extents lie in the d-dimensional space requires embedding of a logical index structure over the 1-dimensional DHT key space [35].

We now describe the features of the P2P-based spatial index that we utilize for mapping the d-dimensional claim and ticket objects over the DHT space. Providing the background and details on this topic is beyond the scope of this paper; here we only give a high level view. The spatial index that we consider in this work, assigns regions of space to the Grid peers in the Grid-Federation system. If a Grid peer is assigned a region of d-dimensional space, then it is responsible for handling query computation associated with the claim and ticket objects that intersect this region, as well as storing the objects that are associated with the region. Fig. 5 depicts a 2-dimensional Grid resource attribute space for mapping claim and ticket objects. The attribute space has a grid-like structure due to its recursive division process. The index cells, resulted from this process, remain constant throughout the life of the d-dimensional attribute space and serve as the entry points for subsequent mapping of claim and ticket objects. The number of index cells produced at the minimum division level, $f_{\min}$ is always equal to $(f_{\min})^{\dim}$, where dim is the dimensionality of the Cartesian space. These index cells are called base index cells and they are computed when the Grid peers bootstrap to the coordination network. Finer details on recursive sub-division technique can be found in [11]. Every Grid peer in the network has the basic information about the Cartesian space coordinate values, dimensions and minimum division level.

Every cell at the $f_{\min}$ level is uniquely identified by its centroid, termed as a *control point*. Fig. 5 depicts four control points A, B, C and D. A DHT hashing method such as the Chord method is utilized to hash these control points. So the responsibility for managing an index cell is associated with a Grid peer in the system. In Fig. 5, control point C is hashed to the Grid peer t, which is responsible for managing all claim and ticket objects that are stored with that control point (Claim X, Z and Ticket M).

For mapping claim objects, the process of mapping index cells to the Grid peers depends on whether it is a DPQ or DRQ. For a DPQ type query, the mapping is simple since every point is mapped to only one cell in the Cartesian space. For a DRQ type query, mapping is not always singular because a range lookup can cross more than one cell. To avoid mapping a DRQ to all the cells that it crosses (which can create many unnecessary duplicates), a mapping strategy based on diagonal hyperplane [36] of the Cartesian space is utilized. This mapping involves feeding a DRQ candidate index cell as an input into a mapping function, F_{map}. This function returns the IDs of index cells to which the given DRQ should be mapped. Spatial hashing is performed on these IDs (which returns keys for Chord space) to identify the current Grid peers responsible for managing the given keys. A Grid peer service uses the index cell(s) currently assigned to it and a set of known base index cells obtained at initialization as the candidate index cells.

Similarly, the mapping process of a ticket also involves the identification of the cell in the Cartesian space. A ticket is always associated with a region [36] and all cells that fall fully or partially within that region will be selected to receive the corresponding ticket. The calculation of the region is based upon the diagonal hyperplane of the Cartesian space.

Table 4
Notations: Scheduling and network models.

Symbol	Meaning
Scheduling	
$\overline{CD}$	Average coordination delay per task in the system.
$\overline{RT}$	Average response time per task.
$\overline{MS}$	Average makespan per workflow.
NT	Total number of notifications for all tasks.
HP	Average number of routing hops per claim/ticket.
CL	Total number of claims in the system.
$E(T_{x_{i,j,k}})$	Execution time for task $T_{x_{i,j,k}}$.
$D(T_{(xy)_{i,j,k}})$	Data transfer time from task $T_{x_{i,j,k}}$ to $T_{y_{i,j,k}}$.
$Rank(T_{x_{i,j,k}})$	Rank value of task $T_{x_{i,j,k}}$.
Network	
λ^{in}	Total incoming rate a Chord service from the network queue.
λ^{out}	Outgoing claim/ticket rate at a network queue.
μ_n	Average network queue service rate at a Grid peer.
μ_r	Average query reply rate for index service at GFA.
λ_t^{in}	Incoming ticket rate at a index service.
λ_c^{in}	Incoming claim rate at a index service.
λ_a^{in}	Incoming query rate at a DHT routing service from the local index service.
λ_{index}^{in}	Incoming index query rate at a index service from its local DHT routing service.
K	Network queue size.

4. Proposed algorithms

In this section, we provide the description of the algorithms that have been developed for: (i) task scheduling; (ii) resource provisioning; and (iii) resource coordination. In Table 4, we show the model parameters related to scheduling and peer-to-peer coordination network.

4.1. Scheduling and provisioning algorithm

4.1.1. Task scheduling

Algorithm 1 TASK SCHEDULING AT GFA

```
1: PROCEDURE: Initialize Task Priority
2: Input: Workflow $W_{ijk}$
3: begin
4:     for all $t \in TaskList$ of workflow $W_{ijk}$
5:         Calculate execution time for $t$ according to (1)
6:     end for
7:     for all $e \in EdgeList$ of workflow $W_{ijk}$
8:         Calculate data transfer time for $e$ according to (2)
9:     end for
10:    Run BFS following reverse task dependency and calculate
       rank value for each task according to (3) (4)
11: end
12: PROCEDURE: Event User Workflow Submit
13: Input: Workflow $W_{ijk}$
14: begin
15:     Initialize Task Priority ($W_{ijk}$)
16:     Generate Ready TaskList for $W_{ijk}$
17:     Submit Ready tasks for execution
18: end
19: PROCEDURE: Event Task Finish Notification
20: Input: Task $T_{x_{ijk}}$
21: begin
22:     Update dependency list of each task in TaskList
23:     Generate Ready TaskList for $W_{ijk}$
24:     Submit Ready tasks for execution
25: end
```

Here, we discuss about the task scheduling algorithm (refer to Algorithm 1) that is undertaken by a GFA in the Grid-Federation system on the arrival of a job or workflow. When a user submits a workflow application, $W_{i,j,k}$ to a GFA, the GFA calculates the priority of each task (line 12–15). Earliest Finish Time (EFT) heuristic is used to calculate task priorities. This heuristic first computes the execution time for each task and communication time between resources of two successive tasks in the workflow (line 1–9). Let $|T_{x_{i,j,k}}|$, be the size of task $T_{x_{i,j,k}}$, submitted to GFA_i and R_i be the local resource with the processing power, $|R_i|$. Thus the execution time of the task is defined as,

$$E(T_{x_{i,j,k}}) = \frac{|T_{x_{i,j,k}}|}{|R_i|}. \tag{1}$$

Let $\overline{T}_{(xy)_{i,j,k}}$, be the size of data to be transferred between task $T_{x_{i,j,k}}$ and $T_{y_{i,j,k}}$, submitted to GFA_i and R_i be the local resource with the data processing capacity, $\overline{R}_i$. Thus the execution time of the task is defined as,

$$D(T_{xy_{i,j,k}}) = \frac{\overline{T}_{xy_{i,j,k}}}{\overline{R}_i}. \tag{2}$$

$E(T_{x_{i,j,k}})$ and $D(T_{(xy)_{i,j,k}})$ are used to calculate the rank of a task. For an exit task, the rank value is,

$$rank(T_{x_{i,j,k}}) = E(T_{x_{i,j,k}}). \tag{3}$$

Now, the rank values of other tasks in the workflow can be computed recursively based on Eqs. (1)–(3) and is represented as,

$$rank(T_{x_{i,j,k}}) = \max_{T_{y_{i,j,k}} \in succ(T_{y_{i,j,k}})} (D(T_{(xy)_{i,j,k}})) + rank(T_{y_{i,j,k}}) + E(T_{x_{i,j,k}}). \tag{4}$$

Since a workflow is represented as a DAG, the rank values of the tasks are calculated (line 10) by traversing the task graph in a Breadth First Search (BFS) manner in the reverse direction of task dependencies (i.e. starting from the exit tasks).

Once the rank values are calculated, the GFA generates the 'ready' tasks in the *TaskList* based on the dependency of each task and put them into the *Ready TaskList* (line 16). Finally, the GFA submits the 'ready' tasks for execution (line 17). Further, when the GFA receives a notification message stating task, $T_{x_{i,j,k}}$ has finished execution, it first updates the dependency lists of the tasks that are dependent on $T_{x_{i,j,k}}$ (line 19–22); then it computes the 'ready' tasks at that moment and submits them for execution (line 23–24).

4.1.2. Resource provisioning

The details of the decentralized resource provisioning algorithm (refer to Algorithm 2) that is undertaken by the P2P coordination space is presented here. When a resource claim object, $r_{x_{ijk}}$ arrives at the coordination service, it is added to the existing claim list, *ClaimList* by the coordination service (line 16–20). When a resource ticket object, u_i arrives at the coordination service, the list of resource claims that overlap or match with the submitted resource ticket object in the d-dimensional space is computed (line 21–25). The overlap signifies that the task associated with the given claim object can be executed on the ticket issuer's resource subject to its availability.

In order to get the matches, the coordination service first, sorts the claim objects in the *ClaimList* in descending order according to their rank value (line 4–5); then from this list, the number of claims that overlap with the ticket are selected to the *ClaimList*$_m$ (line 6–13). From the *ClaimList*$_m$, the resource claimers are selected one by one based on their rank value (higher rank first) and the resource claimers are notified about the resource ticket match until the ticket issuer is not over-provisioned (line 25–30).

The coordination procedure can utilize the dynamic resource parameters such as the number of available processors, queue length etc. as the over-provision indicator. These over-provision indicators are encapsulated with the resource ticket object by the GFAs.

4.2. Coordination algorithm

This section provides the description of the algorithm (refer to Algorithm 3) for coordinating the interactions among different entities in the system. When a GFA, G_j intends to submit a task, $T_{x_{ijk}}$ for execution, it compiles a resource claim object, $r_{x_{ijk}}$ for the task and posts it to the P2P coordination space (line 1–6). On the other hand, the GFA, G_i compiles the resource ticket object, u_i for resource, R_i and publishes it to the P2P coordination space periodically depending on the ticket injection rate (refer to Section 5.2.3). Besides, whenever the resource condition changes such as a task completion event happens, the GFA also posts a ticket object for the corresponding resource immediately (line 7–12).

Once there is a match between a ticket object, u_i and a claim object, $r_{x_{ijk}}$, the coordination service sends a ticket redemption request for task, $T_{x_{ijk}}$ to ticket issuer GFA, G_i(line 13–17). The request message contains the information that task, $T_{x_{ijk}}$ is submitted by GFA, G_j. After notifying the resource claimer GFA, the coordination service unsubscribes the resource claim for that task from the P2P coordination space.

When the ticket issuer GFA, G_i receives the notification of match from the coordination service, it sends a *Reply* to the resource claimer GFA, G_j. If the task can be executed in the local resource at that time, it sends a positive reply; otherwise it sends a negative reply to the claimer GFA (line 18–26). If the *Reply* is 'accept', then the claimer GFA, G_j transfers the locally submitted task, $T_{x_{ijk}}$ to the ticket issuer GFA and unsubscribes the claim object from the coordination space to remove duplicates (line 27–32). However, if the ticket issuer GFA fails to grant access due to local resource sharing policy (i.e. *Reply* is 'reject'), then the claimer GFA reposts the resource claim object for that task to the coordination space for future notifications (line 33–34). The interaction between different entities in the system is depicted in Fig. 6.

An example scenario of the process of task scheduling, resource provisioning and coordination is illustrated in Fig. 7.

Algorithm 2 Resource Provisioning at Coordination Space

```
1:  PROCEDURE: Match
2:  Input: Ticket $u_i$ from Resource $R_i$
3:  begin
4:      Obtain rank value of each task in the ClaimList
5:      Sort ClaimList in descending order of task's rank value
6:      $index \leftarrow 0$
7:      $ClaimList_m \leftarrow \Phi$
8:      while $ClaimList[\text{index}] \neq null$ do
9:          $r_{x_{ijk}} \leftarrow ClaimList[\text{index}]$
10:         if $r_{x_{ijk}} \cap u_i \neq null$ then
11:             $ClaimList_m \leftarrow ClaimList_m \cup r_{x_{ijk}}$
12:         end if
13:         $index \leftarrow index + 1$
14:     end
15:     return $ClaimList_m$
16: end
17: PROCEDURE: Event Resource Claim Submit
18: Input: Claim $r_{x_{ijk}}$
19: begin
20:     $ClaimList \leftarrow ClaimList \cup r_{x_{ijk}}$
21: end
22: PROCEDURE: Event Resource Ticket Submit
23: Input: Ticket $u_i$ from Resource $R_i$
24: begin
25:     $ClaimList_m \leftarrow \text{Match}(u_i)$
26:     $index \leftarrow 0$
27:     while $R_i$ is not over-provisioned do
28:         Send notification of match event to resource claimer
    $ClaimList_m[\text{index}]$
29:         Remove $ClaimList_m[\text{index}]$
30:         $index \leftarrow index + 1$
31:     end
32: end
```

4.3. Time complexity

Let us consider r number of Grid resources. Thus there are r number of GFAs in the system. Let, each user associated with a GFA submits a workflow consisting of t number of tasks and e number of dependencies among the tasks. Then the complexity of calculating execution times of the tasks in the workflow is $O(t)$ and data transfer times for the dependencies in the workflow is $O(e)$. Further, the complexity of running BFS to compute the rank values of all the tasks is $O(e + t)$ and if an adjacency list is used to handle the dependencies, then the complexity of generating 'ready' tasks is $O(e)$. Therefore, the overall time complexity of Algorithm 1 is $O(e + t)$.

Algorithm 3 Resource Coordination

```
1:  PROCEDURE: Event Task Submit
2:  Input: Task $T_{ijk}$
3:  begin
4:      Encapsulate claim object $r_{x_{ijk}}$ for task $T_{x_{ijk}}$
5:      Subscribe $r_{x_{ijk}}$ to coordination space
6:  end
7:  PROCEDURE: Event Resource Status Changed
8:  Input: Resource $R_i$
9:  begin
10:     Encapsulate ticket object $u_i$ for resource $R_i$
11:     Publish $u_i$ to coordination space
12: end
13: PROCEDURE: Event Ticket Redemption Request
14: Input: Task $T_{x_{ijk}}$, GFA $G_i$
15: begin
16:     Send ticket redemption Request for task $T_{x_{ijk}}$ to
         GFA $G_i$; Request message implies $G_j$
17: end
18: PROCEDURE: Event Ticket Redemption Reply
19: Input: Task $T_{x_{ijk}}$, GFA $G_j$
20: begin
21:     if $T_{x_{ijk}}$ can be executed in local resource then
22:         Send Reply "accept" to GFA $G_j$
23:     else
24:         Send Reply "reject" to GFA $G_j$
25:     endif
26: end
27: PROCEDURE: Event Ticket Redemption Reply Action
28: Input: Task $T_{x_{ijk}}$, GFA $G_i$, Reply
29: begin
30:     if Reply is "accept" then
31:         Send $T_{x_{ijk}}$ to accepting GFA $G_i$ for execution
32:         Unsubscribes the claim object for $T_{x_{ijk}}$
33:     else
34:         Subscribe claim object $r_{x_{ijk}}$ to coordination space
35:     endif
36: end
```

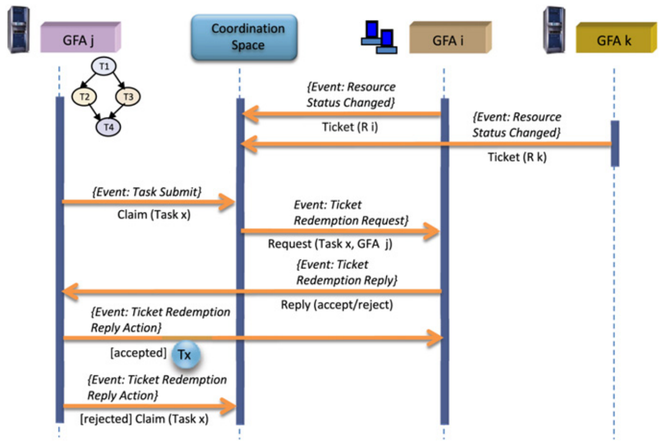

GFA j
Coordination Space
GFA i
GFA k
T1
T2
T3
T4
{Event: Resource Status Changed}
Ticket (R i)
{Event: Resource Status Changed}
Ticket (R k)
{Event: Task Submit}
Claim (Task x)
Event: Ticket Redemption Request}
Request (Task x, GFA j)
{Event: Ticket Redemption Reply}
Reply (accept/reject)
{Event: Ticket Redemption Reply Action}
[accepted]
Tx
{Event: Ticket Redemption Reply Action}
[rejected] Claim (Task x)

Fig. 6. Interaction between various entities in the system during resource coordination.

In worst case, *ClaimList* in the Algorithm 2 can contain $r.t$ number of tasks. So the complexity of sorting the *ClaimList* is $O((r.t)\log(r.t))$ and finding out the total number of matches is $O(r.t)$. Calculating the number of resource claimers to be notified about the matches also requires $O(r.t)$ steps in worst case. Thus the overall complexity of Resource Provisioning algorithm is $O((r.t)\log(r.t))$.

The time complexity of resource coordination algorithm is $O(1)$.

5. Performance evaluation

5.1. Network model

The network model, considered for the simulation study is an interconnected network of r Grid peers, where a Grid peer node (through its Chord routing service) is connected to an network message queue and an incoming link from the Internet (as shown in Fig. 8). The network message queue accepts the following two types of incoming messages from: (i) other Grid peers on the Chord overlay; and (ii) the local Chord service. The Chord service receives messages from local publish/subscribe index service (at rate λ_a^{in}) and the network message queue (at rate λ^{in}). These messages are processed as soon as they arrive at the Chord routing service. After processing, Chord routing service queues the messages in the local network message queue at a rate, λ^{out}. We denote the message processing rate of network message queue by μ_n. Basically, this queue models the network latencies that a message encounters as it is transferred from one Chord routing service to another on the Grid-Federation overlay. The distributions for the delays (including queuing and processing) encountered in a network message queue are given by the M/M/1/K queue steady state probabilities.

We denote the rates for claim and ticket object by λ_c^{in} and λ_t^{in} respectively. The queries are directly sent to the local index service which first processes them and then forwards them to the local Chord routing service. Although, we consider a message queue for the index service but we do not take into account the queuing and processing delays as it is in microseconds. Index service also receives messages from the Chord routing service at a rate λ_{index}^{in}. The index messages include the claims and tickets that map to the control area currently owned by the Grid peer, and the notification messages arriving from the network. The local index service sends message to its Chord service at a rate, λ_a^{in}. The index service sends notification messages (claim-ticket match message) to its broker service at a rate, μ_r.

5.2. Simulation setup

Our simulation infrastructure is created by combining two discrete event simulators namely *GridSim* [37], and *PlanetSim* [38]. GridSim offers a concrete base framework for simulation of different kinds of heterogeneous resources, services and application types. PlanetSim is an event-based overlay network simulator that can simulate both unstructured and structured overlays.

5.2.1. Workload configuration

We implement a workflow generator that creates various formats of weighted pseudo-application workflows. The following input parameters are used to create a workflow.

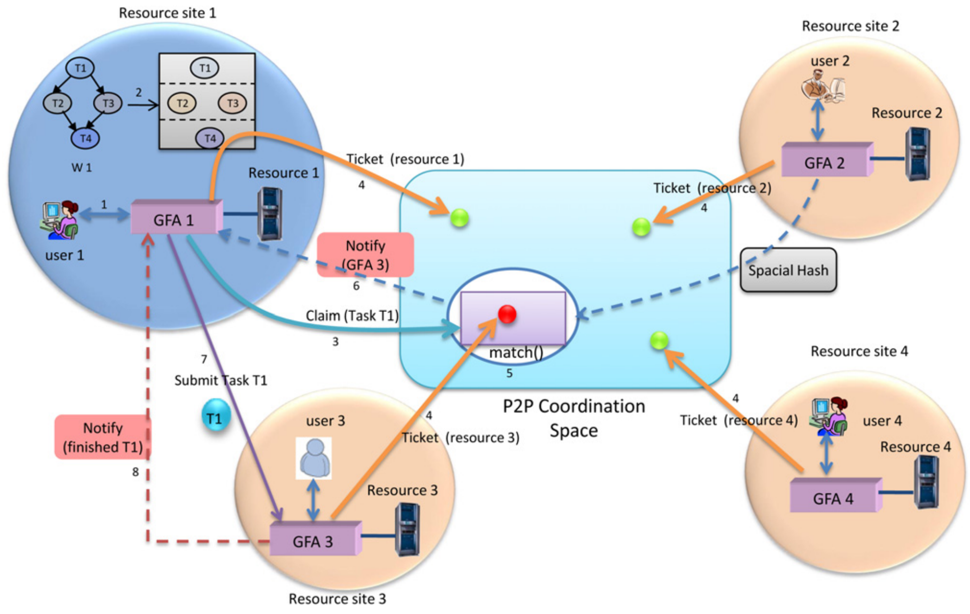


Fig. 7. Example of the process of task scheduling, resource provisioning, and coordination: (1) *user*1 submits the workflow, *W*1 to the local broker, *GFA*1; (2) *GFA*1 ranks the tasks in the workflow using EFT heuristic; (3) *GFA*1 picks the ready task with higher rank (*T*1) and posts or subscribes a resource claim to the coordination space; (4) all the *GFA*s in the system sends or publish the resource ticket or Resource Update Query (RUQ) to the P2P coordination space; (5) in the P2P coordination space, resource ticket of *GFA*3 matches with the resource claim of *GFA*1; (6) *GFA*1 receives the notification message of resource matching (*GFA*3) from the coordination space; (7) *GFA*1 submits task *T*1 to *GFA*3; (8) *T*1 is executed in *Resource*3 and *GFA*3 notifies *GFA*1 of task completion; steps 4–8 continues until all the tasks in *W*1 have been finished execution.

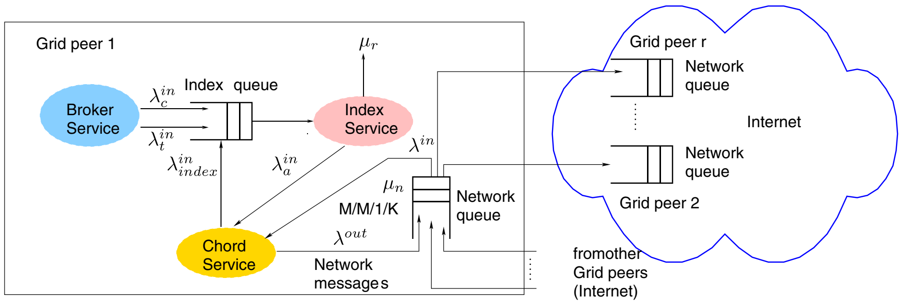


Fig. 8. Network message queuing model at a Grid peer.

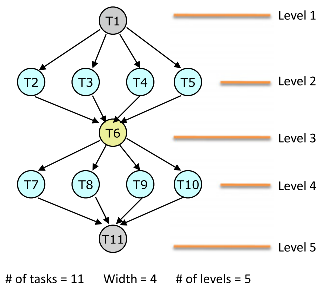


Fig. 9. A fork-join workflow.

- t, the total number of tasks in the workflow.
- l, the total number of levels in the workflow. l represents the ratio of the total number of tasks to the width (i.e. maximum number of tasks in a level). Hence, width: $w = \frac{t-1-\frac{l-1}{2}}{\frac{l-1}{2}}$.

In this study, we consider fork-join workflow (refer to Fig. 9) and an example of such workflow is WIEN2K [28], which is a quantum chemistry application developed at Vienna University of Technology. In this kind of workflow, forks of tasks are created and then joined, such that there can be only one entry task and one exit task. But the number of tasks at each level depends on total number of tasks and the width of that level, w. We vary the number of tasks in a workflow over the interval [50, 500] and the size of each task is randomly generated form a uniform distribution between 50000 MI (Million Instruction) to 500000 MI. Further, we assume that workflows are computation intensive. Thus, the data dependency among the tasks in the workflow is negligible.

5.2.2. *Network configuration*

The experiments run a Chord overlay with 32 bit configuration, i.e. number of bits utilized to generate node and key ids. The GFA/broker network size r is fixed to 100. Further, network queue message processing rate, μ_n, is fixed at 4000 messages per second and message queue size, K is fixed at 10^4.

5.2.3. *Resource claim and ticket injection rate*

The GFAs inject the ticket objects based on the exponential inter-arrival time distribution. The injection rate (i.e. RUQ rate), λ_t^{in}, for the resource tickets is distributed over the interval [100, 300] in step of 100 s. Note that, the inter-arrival delay between injecting the ticket objects is modeled to be the same for all the GFAs in the system. At the beginning of the simulation, the resource claims for the entry tasks of all the workflows in the system are injected. Subsequently, when these tasks finish, then the resource claims for the successive tasks in the workflow are posted. This process is repeated until all the tasks in the workflow are successfully completed. Spatial extent of both resource claims and ticket objects lie in a 4-dimensional attribute space. These attribute dimensions include the number of processors, p_i, their speed, m_i, their architecture, x_i, and operating system type, ϕ_i. The distribution for these resource dimensions is generated by utilizing the configuration of resources that are deployed in the various Grids including NorduGrid, AuverGrid, Grid5000, NaregiGrid, and SHARCNET.[1]

5.2.4. *Spatial index configuration*

In this simulation, the $f_{\min}$ of logical d-dimensional spatial index is set to 4. The index space resembles a Grid-like structure, where each index cell is randomly hashed to a Grid peer based on its control point value. With dim $=$ 4, total of 256 index cells were produced at the $f_{\min}$ level. Hence in a network that consisted of 100 GFAs, on an average the responsibility of managing 2.5 index cells were assigned to each GFA.

5.2.5. *Resource load indicator*

The GFAs/brokers encode the metric *"number of available processors"* at time τ with the resource ticket object, u_i. A coordination service utilizes this metric as the indicator for the current load on a resource, R_i. In other words, a coordination service stops sending the notifications as the number of processors available with a ticket issuer approaches *zero*.

5.3. *Results and observations*

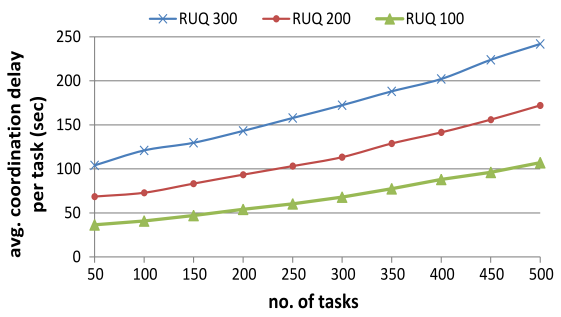
In our simulation, we vary the resource ticket update (RUQ) inter-arrival delay over the interval [100, 300] in steps of 100 s and the size of the workflow from 50 to 500 tasks. The graphs in Figs. 10 and 11 show the performance of the proposed scheduling algorithm in terms of scheduling and coordination perspective, respectively.

5.3.1. *Scheduling perspective*

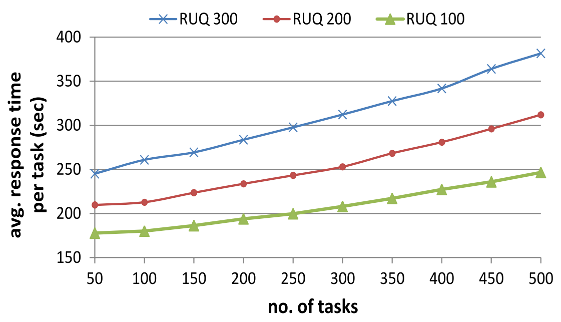
As a measurement of the scheduling performance, we use the following metrics namely, *average makespan*, *average coordination delay*, *average response time*, and *average number of notifications*.

The metric coordination delay sums up the latencies for: (i) resource claim to reach the index cell, (ii) waiting time till a resource ticket matches with the claim, and (iii) notification delay from coordination service to the relevant GFA. CPU time for a task is defined as the time, a task takes to actually execute on a processor. Response time for a task is the delay between the submission time and the arrival time of execution output. Effectively, the response time includes the latencies for coordination and the CPU time. Makespan is measured as the response time of a whole workflow, which equals the difference between the submission time of the entry task in the workflow and the output arrival time of the exit task in that workflow. Note that, these measurements (except

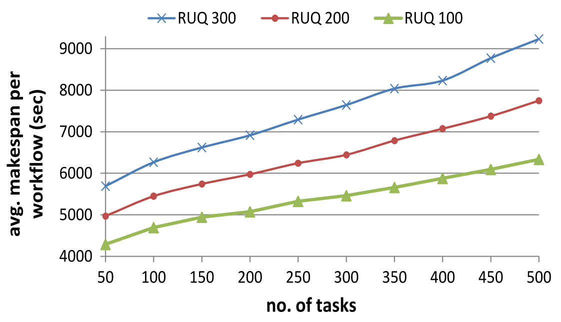
[1] http://gwa.ewi.tudelft.nl/.



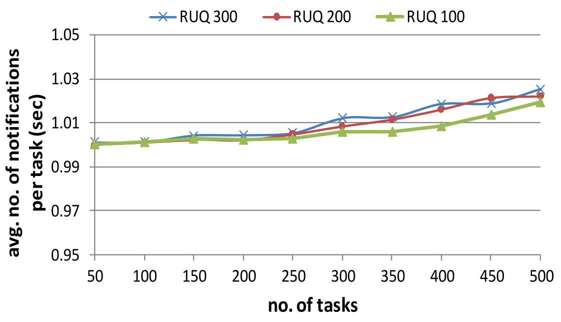
(a) Number of tasks vs. Coordination delay.



(b) Number of tasks vs. Response time.

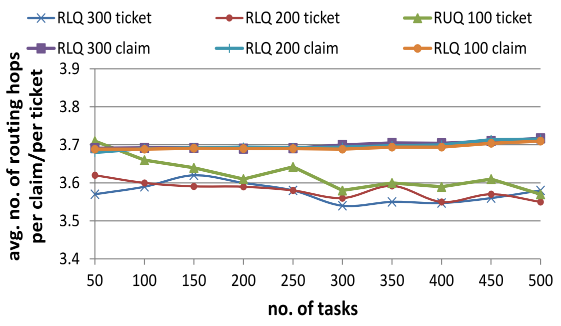


(c) Number of tasks vs. Makespan.

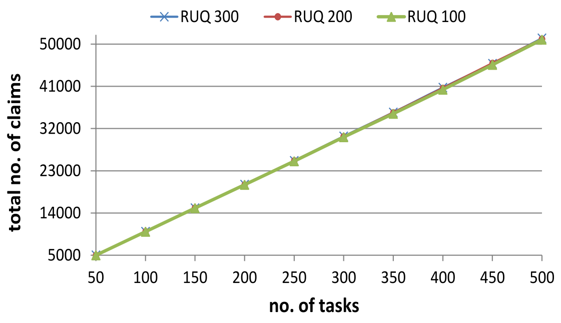


(d) Number of tasks vs. Number of notifications.

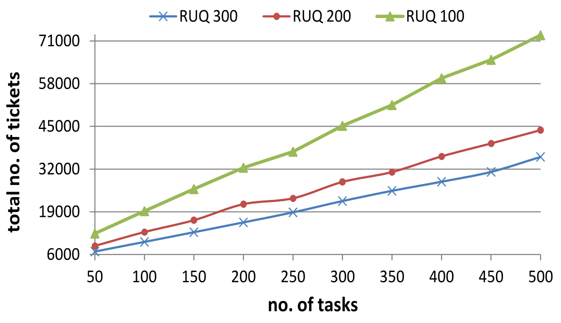
Fig. 10. Effect of workflow size and resource information update interval on different performance metrics (scheduling perspective).



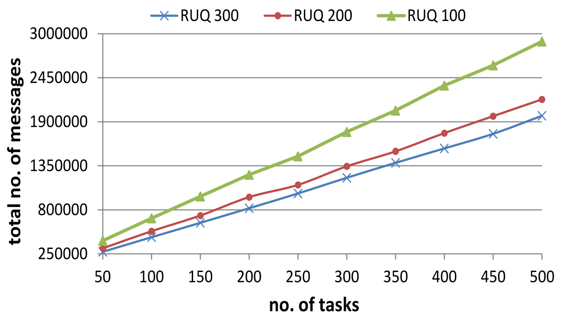
(a) Number of tasks vs. Number of hops.



(b) Number of tasks vs. Number of claims.



(c) Number of tasks vs. Number of tickets.



(d) Number of tasks vs. Number of messages.

Fig. 11. Effect of workflow size and resource information update interval on different performance metrics (coordination perspective).

makespan) are collected by averaging the values obtained for each task in the system. The measurement of makespan is taken by averaging over all the workflows in the system.

Fig. 10(a) presents the results of average coordination delay for a task with respect to the increase of the number of tasks in a workflow for different inter-arrival delays of resource information update (ticket posting frequency). The results show that at higher inter-arrival delay of tickets, the tasks in a workflow experience increased coordination delay. This happens due to the reason that in this case, the resource claim objects of the corresponding tasks have to wait for longer period of time before they are hit by ticket objects. As the task processing time (CPU time) is not affected by the ticket posting frequency, the average response time for a task shows (refer to Fig. 10(b)) the similar trend as coordination delay with the changes of resource information update delay. However, when the number of tasks in the workflow increases, the resource claim to ticket ratio in the system also increases. This leads to increased coordination delay for each task due to longer waiting period. Therefore, the response time of a task in the workflow is also increased, while the size of workflow increases.

The average makespan of the workflows also shows (see Fig. 10(c)) similar growth over the number of tasks and ticket inter-arrival delay as reflected in coordination delay or response time. Thus in our proposed scheduling environment, if the resources update their availability information frequently, then the workflows submitted by the users will be completed early.

The proposed scheduling approach is also highly successful in reducing the number of negotiations undertaken for the successful submission of a task (see Fig. 10(d)). In a centralized scheduling technique, it requires few negotiation iterations to successfully submit a task, while in this case, the average number of negotiations (notification messages) per task is about one.

Discussion: Let us consider, r number of Grid resources. Therefore, there are r number of GFAs in the system. Now if each user associated with the GFA submits a workflow that consists of t number of tasks, then the total number of tasks executed in the system are,

$$T = r.t. \tag{5}$$

Let, CD_{ij} is the coordination delay of ith task, submitted by jth GFA in the system. Then the average coordination delay per task is,

$$\overline{CD} = \frac{\sum\limits_{\substack{1\le i\le r\\1\le j\le t}} CD_{ij}}{r.t}. \tag{6}$$

The latency for the claim object of a task, to reach the index cell (refer to Section 3.3.3) and the notification delay for that task are negligible. Thus the coordination delay of a task mainly depends on the waiting time (in the *ClaimList* queue) for the claim object of that task to be matched with a ticket object. In worst case, the length of the *ClaimList* queue equals the total number of tasks in the system. Therefore, for a fixed resource information update interval (RUQ), the waiting time of a task depends on T and we can write,

$$\begin{aligned}\overline{CD} &= O(r.t)\\ &= O(t).\end{aligned} \tag{7}$$

In Fig. 10(a), for a particular RUQ, it is also evident that the average coordination delay of a task in the system is bounded by the number of tasks in the workflow.

The response time of a task is the summation of coordination delay and execution time of that task. Thus, the average response time of a task is,

$$\begin{aligned}\overline{RT} &= \overline{CD} + Avg.\ CPU\ time\\ &= \overline{CD} + \frac{Avg.\ task\ size}{Avg.\ processing\ power\ of\ resources}\\ &= \overline{CD} + c_1 \quad ; c_1 \text{ is a constant}\\ &= O(\overline{CD})\\ &= O(t); \quad \text{as } \overline{CD} = O(t).\end{aligned} \tag{8}$$

Therefore, the average response time of a task in the system is also linearly dependent on the number of tasks in that workflow, which is shown in Fig. 10(b).

The average makespan of a fork-join workflow (see Fig. 9) consisting of t tasks and l levels, can be represented as,

$$\overline{MS} = l.\alpha.\overline{RT} \tag{9}$$

where, the value of α depends on the number of tasks in a level (known as width, w), which can be concurrently executed depending on the availability of resources and

$$\alpha = \begin{cases} 1 & \text{in best case}\\ w/2 & \text{in average cases}\\ w & \text{in worst case.}\end{cases}$$

As $\alpha = O(t)$, $\overline{RT} = O(t)$ and in our experiments, $l = 10$, so we can write,

$$\overline{MS} = O(t). \tag{10}$$

Therefore, the average makespan of a fork-join workflow in the system linearly depends on the number of tasks in that workflow, which is also depicted in Fig. 10(c).

Moreover, for most of the tasks, the GFAs/brokers in the system receive 1 notification message from the coordination service. Thus total number of notifications for all the tasks in the system are,

$$NT = r.t + \beta; \quad \beta \text{ is a constant and } \beta \le t.$$

Average number of notifications per task,

$$\begin{aligned}\overline{NT} &= \frac{r.t}{r.t} + \frac{\beta}{r.t}\\ &= 1 + c_3; \quad c_3 \text{ is a constant}\\ &= O(1).\end{aligned} \tag{11}$$

This suggests that the negotiation or notification complexity involved with this scheduling technique is $O(1)$, which we can also see in Fig. 10(d).

5.3.2. Coordination perspective

Here, we analyze the performance overhead of the DHT-based coordination space with regard to facilitating coordinated scheduling among the distributed GFAs. For that, we measure the following metrics: (i) *number of routing hops*, undertaken per task to map claim and ticket objects to index cells; (ii) *total number of tickets or claims*, produced in the system; and (iii) *total number of messages*, generated for the successfully mapping the coordination objects (tickets and claims) and receiving notifications.

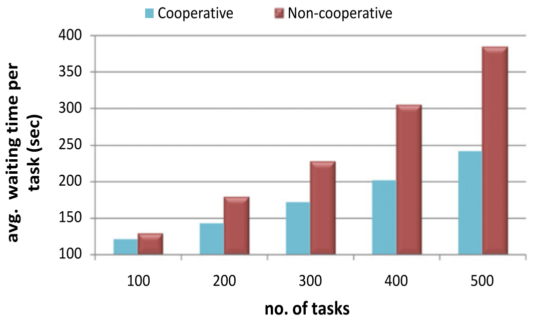
Fig. 11(a) shows the number of routing hops, undertaken at different ticket injection rates for ticket and claim object across the GFAs in the system for different sizes of workflow. From the figure, it is evident that the number of routing hops is not changed significantly with the increase of the ticket injection rate or the size of the workflow. Here, the average number of routing hops for mapping ticket or claim objects is around 3.62.

However, the number of claims, generated during the simulation, remains the same with the variation of ticket inter-arrival delay (refer to Fig. 11(b)). But it shows a linear growth over the increase in the number of tasks in the workflow.

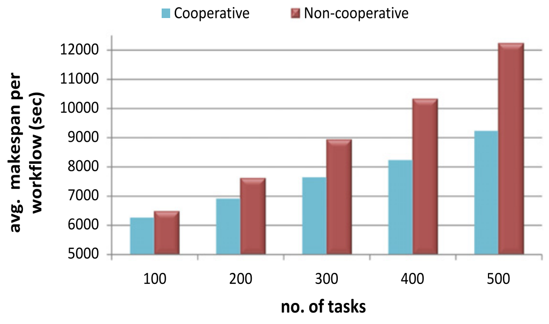
In Fig. 11(c) and (d), we show the message overhead, involved with the inter-arrival delay of ticket objects. Fig. 11(c) depicts the total number of ticket objects, posted by all GFAs in the system with respect to increasing workflow size and ticket inter-arrival delay. In Fig. 11(d), we can see that as ticket inter-arrival delay and size of the workflow increase, the number of messages generated during simulation period increases. For instance, when ticket inter-arrival delay is 100 s and each workflow consists of 500 tasks, 72754 tickets as well as 2904113 messages are generated in the system. Thus if the GFAs publish tickets at relatively faster rate, the message overhead of the system increases substantially. But in this case, average coordination delays per task also decreases moderately (see Fig. 10(a)). Therefore, the ticket inter-arrival delay should be chosen in such a way that a balance between coordination delay and message overhead can exist in the system. In addition, from Fig. 11(d), it is evident that our system is scalable since the total number of messages is increased linearly with respect to the number of tasks in the workflow.

Discussion: In our experiment, the average number of routing hops per claim or per ticket is,

$$\begin{aligned}HP &= 3.62; \quad \text{refer to Fig. 11(a)}\\ &= 6.64 - 3.02\\ &= \log 100 - c_4; \quad c_4 \text{ is a constant}\\ &= O(\log r); \quad \text{here, } r = 100.\end{aligned} \tag{12}$$



(a) Task waiting time vs. Number of tasks.



(b) Makespan vs. Number of tasks.

Fig. 12. Performance comparison of cooperative approach against non-cooperative approach with varying workflow size.

This shows that the number of routing hops for mapping claim/ticket object is as expected in a Chord-based routing space, i.e. bounded by the function $O(\log r)$.

The GFAs in the system post one claim object for each task in the workflow. Thus the total number of claim objects generated in the system is,

$$\begin{aligned} CL &= r.t \\ &= O(t). \end{aligned} \tag{13}$$

Fig. 11(b) shows that the total number of claims generated in the system is linearly dependent on the number of tasks in the workflow.

According to [8], in a federation of r heterogeneous resources, on average, a task requires HP (see Eq. (12)) messages to be sent in the network in order to map a task to a resource. Hence given r workflows each having t tasks, the total number of messages generated to map all tasks to resources is bounded by $O((r.t)\log r)$. Further if GFAs posts $\frac{1}{\lambda_t^{in}}$ tickets over a time period, then the average-case message complexity involved with mapping the ticket objects to Grid peers in the federation is bounded by the function $O(E[query\ paths].\frac{1}{\lambda_t^{in}}.\log r)$, where $E[query\ paths]$ denotes the mean number of disjoint query paths taken to map a ticket object to a Grid peer. Thus, the total number of messages generated in the system as a result of successfully scheduling all tasks of workflows in the system is,

$$O((r.t)\log r) + O\left(E[query\ paths].\frac{1}{\lambda_t^{in}}.\log r\right).$$

5.4. Comparison

5.4.1. Cooperative vs. non-cooperative load balancing

In this section, we present the details and simulation results related to how our proposed decentralized and cooperative scheduling approach is effective in solving load balancing problem in mapping workflow tasks to distributed Grid resources. The effectiveness of the proposed approach as regards to load balancing is proven by conducting a comparative evaluation against a traditional non-cooperative workflow task mapping approach. In non-cooperative approach, workflow brokers operate in an independent and non-cooperative manner by submitting all the matched tasks to the first matched Grid resource, suggested by the decentralized resource discovery service in response to a resource lookup request. Further, unlike the proposed approach, the non-cooperative scheduler does not take into account the dynamic resource conditions (utilization, queue length) and priorities of other workflow brokers in the environment.

To show the performance gain achieved by applying proposed approach for mapping workflow tasks in the Grid environment, we conduct experiments by varying the size of the workflow from 100 to 500 tasks and fixing the RUQ inter-arrival delay at 300 s. As a measurement of the load balancing efficiency, we use the following metrics: (i) *average makespan per workflow* (see Section 5.3.1), an efficient workflow task mapping approach should be able to minimize the makespan for workflow applications across the environment; (ii) *average task waiting time per task* in the system; and (iii) *number of task executions per processor* for different Grid resources.

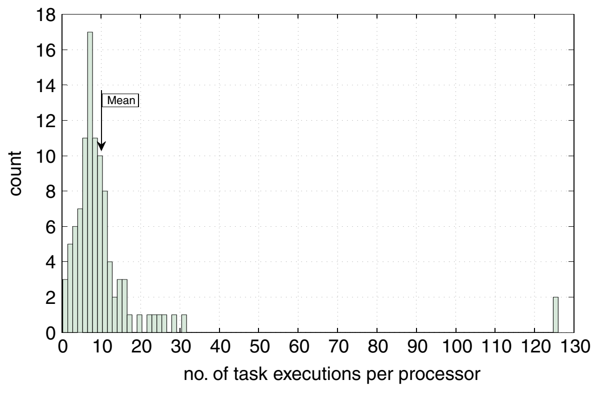
The metric, task waiting time is measured as the total queuing time of a single task in the system, which is equal to the difference between the submission time of a task and its execution starting time on a Grid resource. The number of task executions per processor, t_p, indicates how the task processing load is balanced across Grid resources in the environment. t_p, for a particular Grid Resource, r_i is calculated as,

$$t_p(r_i) = \frac{total\ number\ of\ tasks\ executed\ by\ r_i}{p_i}.$$

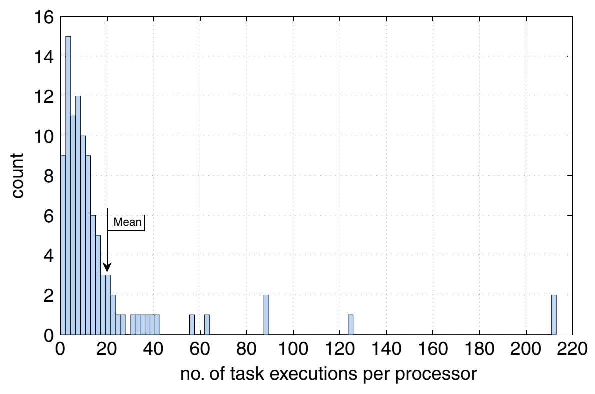
Fig. 12(a) presents the results related to the average waiting time for a task with increase in workflow size for both the cooperative and non-cooperative scheduling approaches. The results show that if workflow brokers undertake cooperative scheduling, the tasks across the Grid environment experience relatively smaller waiting time as compared to non-cooperative scheduling. Furthermore, with the increase in workflow size (number of tasks) across the system, the performance gain achieved by applying the proposed approach is more evident. For instance, when the number of tasks in a workflow is 100, the difference between the task waiting times for cooperative and non-cooperative approaches is 7 s, whereas the difference increases to 142 s when each workflow consists of 500 tasks. Accordingly, the average makespan of the workflow also shows similar improvement for cooperative scheduling approach in comparison to non-cooperative scheduling (refer to Fig. 12(b)). For example, when workflow size is 100, makespan is reduced by 5%, whereas makespan is decreased by 25% when the workflow consists of 500 tasks.

As the non-cooperative scheduling technique applied by the workflow brokers does not take into account the resource behaviors (utilization, queue length) that dynamically change across the environment, resources are often overloaded and the amount of time, a task has to wait in a queue before being assigned to a processor increases. This situation is further aggravated as the workflow size scales. However, the proposed approach is able to dynamically monitor status of the resources and coordinated the distribution of load across available resources uniformly in the Grid environment, thereby minimizing task waiting time and makespan.

In Fig. 13 we show histograms of the distribution of the number of task executions per processor for different Grid resources. The



(a) Cooperative scheduling (workflow size = 500).



(b) Non-cooperative scheduling (workflow size = 500).

Fig. 13. Distribution of the number of tasks (per processor) executed by Grid resources.

Table 5
Summary statistics of the number of tasks (per processor) executed by Grid resources for different workflow size.

Tasks	Approach	Min	Q_1	Median	Q_3	Max
100	Cooperative	0.11	0.90	2.07	3.70	26.75
100	Non-cooperative	0.09	1.06	1.80	3.53	45.75
500	Cooperative	0.25	5.65	7.97	10.78	126.00
500	Non-cooperative	0.12	4.70	9.06	15.95	213.00

distribution for non-cooperative approach shows more variability as it is more spread out and the values differ more from the center (mean value) as it is skewed. This happens because some resources are allocated more tasks to execute, while other resources are underutilized. On the contrary, the distribution for cooperative scheduling approach is bell shaped and more densely clustered around the mean. Therefore, from Fig. 13, it is evident that the load balancing technique employed by the cooperative scheduler is effective in terms of uniformly distributing workflow tasks across the resources in the Grid environment. The summary statistics of these two distributions for different workflow sizes are presented in Table 5.

5.4.2. *Decentralized vs. centralized coordination*

In order to compare the effectiveness of our proposed decentralized and cooperative scheduling technique, we compare it against the scheduling technique with centralized coordination service. The simulation system, developed for evaluating a centrally coordinated Grid-Federation for this study assumes that the coordination service is hosted on a machine, which is known to all the workflow brokers (GFAs) in the system. In this setup, every workflow broker is required to submit its resource claim objects to the centralized coordination service. Similarly, all resources in the Grid-Federation periodically update their usage information to the coordination service.

In the case of decentralized setting (i.e. P2P spatial index), a claim object can be mapped to more than one control points based on its spatial extent or query window size. If the control points are assigned to different Grid peers in the network then the problem of duplicate or conflicting notification can arise for a given claim (i.e. a claim is hit by more than one ticket object at more than one Grid peer at the same time). This problem is quite evident in the case of decentralized spatial index [39]. On the other hand, under the centralized setting, the coordination space (see Fig. 5) is hosted and managed by a single machine in the system and all the control points, claims and tickets are stored with it. Therefore, the problem of duplicate or conflicting notifications does not occur here and the notification complexity

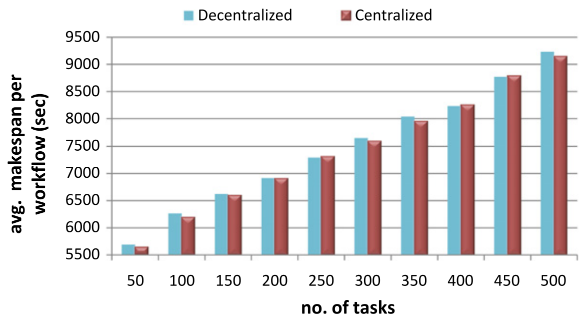


Fig. 14. Makespan for decentralized and centralized coordination service.

involved with the centralized approach is bounded by the function $O(1)$. In addition, the centralized coordination service generates schedule that can efficiently distribute the workload among all the Grid peers as it receives all the claims and tickets in the system.

The objective of the discussion of this section is to show that (i) our decentralized and cooperative scheduling technique is as efficient as the centralized coordination technique at solving the duplicate or conflicting notification problem; and (ii) the final makespan achieved per workflow as a result of decentralized coordination service is comparable to the centralized approach.

As shown in the previous section (refer to Section 5.3.1), the simulation results for our proposed approach show that the notification complexity is bounded by the function $O(1)$ (similar to the centralized approach). This is achieved due to unsubscribing the claim object of a task from the coordination space as soon as the match notification arrives. Moreover, in Fig. 14, we show the average makespan, achieved per workflow in the case of decentralized and centralized approaches for different sizes of workflow. From the figure, it is evident that our proposed decentralized technique is also successful in generating schedules, which are as efficient as the centralized approach. This proves that our decentralized coordination service is able to efficiently schedule the workload among the Grid peers in the system.

6. Conclusion and future work

In this paper, we have presented a decentralized and cooperative scheduling technique for workflow applications in Peer-to-Peer Grids. Using simulation, we have measured the performance of our scheduling approach with respect to both scheduling and coordination perspective and analyzed different performance metrics according to the obtained results. The results show that our approach is scalable in terms of scheduling message complexity and makespan. As we leverage the DHT-based coordination space for our scheduling, it can also avoid the limitation of single point of failure regarding resource information service in centralized scheduling techniques.

In order to show the performance of the proposed approach against non-cooperative scheduling approach, we have conducted experiment for different sizes of workflow and the results show that our scheduling technique can reduce the makespan up to 25% by decreasing the task waiting time up to 37% and is able to balance the load among the available Grid resources significantly.

We have also compared the effectiveness of our proposed scheduling technique against the scheduling technique with centralized coordination service. According to the results obtained, our decentralized and cooperative scheduling technique is as efficient as the centralized approach at generating $O(1)$ notifications per task and the makespan achieved per workflow by our approach is also comparable to the centralized approach.

In future, we intend to investigate the incorporation of different optimizations into our proposed algorithm such as spatial hashing of all the tasks in a workflow within a single claim object. This hashing is based on the assumption that the tasks have homogeneous requirements with regard to resource configuration. This approach would be significantly helpful in curbing the number of messages, produced as a result of mapping individual claim objects for different tasks in a workflow. Further, we intend to address the resource failure and fault tolerance issues into our scheduling technique in our future work.

Acknowledgement

This work is partially supported by Australian Research Council (ARC) Discovery Project grant.

References

[1] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1998.

[2] L. Paterson, T. Roscoe, The design principles of planetlab, Operating Systems Review 40 (1) (2006) 11–16.

[3] Y. Fu, J. Chase, B. Chun, S. Schwab, A. Vahdat, SHARP: An architecture for secure resource peering, in: Proceedings of the 9th ACM symposium on Operating systems principles, NY, USA, 2003.

[4] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, A directory service for configuring high-performance distributed computations, in: Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, Portland, USA, 1997.

[5] X. Zhang, J.L. Freschl, J.M. Schopf, A performance study of monitoring and information services for distributed systems, in: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, Seattle, USA, 2003.

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

[7] Z. Li, M. Parashar, Comet: A scalable coordination space for decentralized distributed environments, in: Proceedings of the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems, Sun Diego, USA, 2005.

[8] R. Ranjan, A. Harwood, R. Buyya, Coordinated load management in peer-to-peer coupled federated grid systems, Technical Report, GRIDS-TR-2008-2, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, 2008.

[9] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, in: Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, San Diego, USA, 2001.

[10] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, 2001.

[11] E. Tanin, A. Harwood, H. Samet, Using a distributed quadtree index in peer-to-peer networks, VLDB Journal 16 (2) (2007) 165–178.

[12] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.H. Su, K. Vahi, M. Livny, Pegasus: Mapping scientific workflow onto the grid, in: Proceedings of the Across Grids Conference, Cyprus, 2004.

[13] I. Taylor, M. Shields, I. Wang, Resource management of triana p2p services, Grid Resource Management, Netherlands, June 2003.

[14] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, P. Li, Taverna: A tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20 (17) (2004) 3045–3054.

[15] M. Litzkow, M. Livny, M. Mutka, Condor-a hunter of idle workstations, in: Proceedings of the 8th International Conference of Distributed Computing Systems, IEEE CS Press, USA, 1988.

[16] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A Lee, J. Tao, Y. Zhao, Scientific workflow management and the kepler system, Concurrency and Computation: Practice and Experience (2005) Special Issue on Scientific Workflows.

[17] Y. Yang, J. Chen, J. Lignier, H. Jin, Peer-to-peer based grid workflow runtime environment of swindew-g, in: Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing, India, December 2007.

[18] J. Yu, R. Buyya, A novel architecture for realizing grid workflow using tuple spaces, in: Proceedings of the 5th IEEE/ACM Workshop on Grid Computing, IEEE CS Press, USA, 2004.

[19] T. Fahringer, et al., Askalon: A tool set for cluster and grid computing, Concurrency and Computation: Practice and Experience 17 (2005) 143–169.

[20] R. Raman, M. Livny, M. Solomon, Resource management through multilateral matchmaking, in: Proceedings of the 9th IEEE International Symposium on High-Performance Distributed Computing, USA, August 2000.

[21] J. Yao, C.K. Tham, K.Y. Ng, Decentralized dynamic workflow scheduling for grid computing using reinforcement learning, in: Proceedings of the14th IEEE International Conference on Networks, Singapore, 2006.

[22] G. Chen, C.P. Low, Z. Yang, Coordinated service provisioning in peer-to-peer environments, IEEE Transactions on Parallel and Distributed Systems 19 (4) (2008).

[23] R. Ranjan, A. Harwood, R. Buyya, A case for cooperative and incentive based coupling of distributed clusters, Future Generation Computing Systems (ISSN: 0167-739X) 24 (4) (2008) 280–295.

[24] B. Bode, D. Halstead, R. Kendall, D. Jackson, PBS: The portable batch scheduler and the maui scheduler on linux clusters. in: Proceedings of the 4th Linux Showcase and Conference, Atlanta, USA, 2000.

[25] W. Gentzsch, Sun grid engine: Towards creating a compute power grid, in: Proceedings of 1st IEEE International Symposium on Cluster Computing and the Grid, Brisbane, Australia, 2001.

[26] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley Publishing Company, 1989.

[27] E. Byun, Y. Kee, E. Deelman, K. Vahi, G. Mehta, J. Kim, Estimating Resource Needs for Time-Constrained Workflows, in: Proceedings of 4th IEEE International Conference on eScience, USA, December 2008.

[28] P. Blaha, K. Schwarz, G.K.H. Madsen, D. Kvasnicka, J. Luitz, Wien2k — An augmented plane wave plus local orbitals program for calculating crystal properties, Technical report, Vienna University of Technology, Austria, 2001.

[29] D. Theiner, P. Rutschmann, An inverse modelling approach for the estimation of hydrological model parameters, in: Journal of Hydroinformatics, IWA Publishing, 2005.

[30] F. Schuller, J. Qin, Towards a Workflow Model for Meteorological Simulations on the AustrianGrid, in: Proceedings of 1st Austrian Grid Symposium, Schloss Hagenberg, Austria, December 2005.

[31] L. Clementi, et al. Services oriented architecture for managing workflows of avian flu grid, in: Proceedings of 4th IEEE International Conference on eScience, USA, December 2008.

[32] L. Ramakrishnan, M. Reed, J. Tilson, D. Reed, Grid Portals for Bioinformatics, Renaissance Computing Institute, University of North Carolina, USA.

[33] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, I. Stoica, Towards a common api for structured peer-to-peer overlays, in: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03), CA, USA, 2003.

[34] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message Passing Interface, MIT press, 1999.

[35] R. Ranjan, A. Harwood, R. Buyya, Peer-to-peer resource discovery in global grids: A tutorial, IEEE Communications Surveys and Tutorials 10 (2) (2008) 6–33. Second Quarter.

[36] A. Gupta, O.D. Sahin, D. Agrawal, A. El Abbadi, Meghdoot: Content-based publish/subscribe over p2p networks, in: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, Toronto, Canada, 2004.

[37] R. Buyya, M. Murshed, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency and Computation: Practice and Experience 14 (13–15) (2002) 1175–1220.

[38] P. Garca, C. Pairot, R. Mondjar, J. Pujol, H. Tejedor, R. Rallo, Planetsim: A new overlay network simulation framework, in: Proceedings of Software Engineering and Middleware, Linz, Austria, 2004.

[39] L. Chan, S. Karunasekera, Designing configurable publish-subscribe scheme for decentralised overlay networks, in: Proceedings of the 21st IEEE International Conference on Advanced Information Networking and Applications, Canada, 2007.

Mustafizur Rahman received BSc. in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh in 2004. He worked as a lecturer in the Department of Computer Science and Engineering at BUET. Currently, he is with GRIDS Lab and a PhD candidate in the Department of Computer Science and Software Engineering at the University of Melbourne, Melbourne, Australia. His research interests include scientific workflow management, scheduling in Grids and P2P systems, and autonomic systems. He is a student member of IEEE and IEEE Computer Society.

Dr. Rajiv Ranjan received BE in Computer Engineering from Gujarat University, Gujarat, India in 2002 and Doctor of Philosophy (PhD) from the University of Melbourne, Melbourne, Australia in 2007. He worked as a lecturer in the Department of Computer Engineering at Gujarat University. Currently, he is working as a post-doctoral research fellow in the Department of Computer Science and Software Engineering at the University of Melbourne. His research interests lie in the algorithmic aspects of resource allocation and resource discovery in decentralized Grid and Peer-to-Peer computing systems. He has served as a reviewer for journals including Future Generation Computer Systems, Journal of Parallel and Distributed Computing, IEEE Internet Computing, and IEEE Transactions on Computer Systems.

Dr. Rajkumar Buyya Rajkumar Buyya is an Associate Professor of Computer Science and Software Engineering; and Director of the Grid Computing and Distributed Systems (GRIDS) Laboratory at the University of Melbourne, Australia. He received BE and ME in Computer Science and Engineering from Mysore and Bangalore Universities in 1992 and 1995 respectively; and Doctor of Philosophy (PhD) from Monash University, Melbourne, Australia in 2002. He has authored over 220 publications and three books. The books on emerging topics that Dr. Buyya edited include, High Performance Cluster Computing (Prentice Hall, USA, 1999) and Market-Oriented Grid and Utility Computing (Wiley, 2008). Dr. Buyya has contributed to the creation of high performance computing and communication system software for Indian PARAM supercomputers. He received "Research Excellence Award" from the University of Melbourne for productive and quality research in computer science and software engineering in 2005. The Journal of Information and Software Technology in Jan 2007 issue, based on an analysis of ISI citations, ranked Dr. Buyya's work (published in Software: Practice and Experience Journal in 2002) as one among the "Top 20 cited Software Engineering Articles in 1986-2005". He received the Chris Wallace Award for Outstanding Research Contribution 2008 from the Computing Research and Education Association of Australasia, CORE, which is an association of university departments of computer science in Australia and New Zealand. Dr. Buyya served as the first elected Chair of the IEEE Technical Committee on Scalable Computing (TCSC) during 2005–2007. In recognition of these dedicated services to computing community over a decade, President of the IEEE Computer Society, USA presented Dr. Buyya a "Distinguished Service Award" in 2008.