Coordinated Load Management in Peer-to-Peer Coupled Federated Grid Systems

Rajiv Ranjan, Aaron Harwood, and Rajkumar Buyya
GRIDS Lab and P2P Group
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{rranjan,aharwood,raj}@csse.unimelb.edu.au

***Abstract*— This paper proposes a coordinated load management protocol in Peer-to-Peer (P2P) coupled federated Grid systems. The participants in the system such as the resource providers and the consumers who belong to multiple control domains work together to enable a coordinated federation. The coordinated load management protocol embeds a logical spatial index over a Distributed Hash Table (DHT) space as regards to management of complex coordination objects. The DHT based space serves as a decentralised blackboard system, which facilitates autonomous participants with respect to coordinating their activities. We show that our coordination protocol involves low overhead with respect to number of messages generated, and has scalability properties.**

Our coordinated load management protocol is in particular applicable for resource brokering services of computational grids and PlanetLab. Resource brokering services are the main components that control the way applications are scheduled, managed and allocated in a distributed, heterogeneous and dynamic Grid computing environment. Existing Grid computing systems such as resource brokers, e-Science application work-flow schedulers operate in tandem but still lack a coordination mechanism that can lead to efficient application schedules across distributed resources. Further, lack of coordination exacerbates the utilisation of various resources (such as computing cycles and network bandwidth). The feasibility of the proposed coordinated load management protocol is studied through extensive simulations.

I. Introduction

Distributed systems including computational grids, P2P systems, Planetlab, cross-company workflows involve participants who are topologically and administratively distributed over various control domains in the Internet. Participants in these resource sharing environments can be organised based on a federated model. In a federated orgainisation [5], [10], [19], [3], [15] every autonomous provider pools his resources together for common benefit of the community. As a result, every participant gets access to much larger pool of resources. Further, federated organisation aids in coping with the bursty requests during high demand period without having the need to maintain or administer the computing, network and storage resources. Federated systems need effective resource management mechanisms that can: (i) coordinate resource usage across the system thereby leading to load balance across the resources; (ii) scale gracefully to a large number of participants; and (iii) adapt to dynamic resource and network conditions.

A. *Tragedy of Commons*

Distributed resource sharing systems such as computational grids and PlanetLab often exhibit the classical economics paradox called "tragedy of commons" during period of high demand. Study undertaken in [10] 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 distributed systems experiments. Such bursty behaviour of users often lead to sub-optimal system performance. Under 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-utilisation of particular set of nodes. Further, the users who can not successfully finish their experiments due to competing or conflicting requests in the system tend to retry their experiments which further aggravate the situation.

Another, motivating example is the way multiple Grid brokers schedule the jobs on distributed Grid resources. Majority of existing approaches to Grid scheduling are non-coordinated. Brokers such as Nimrod-G [1], Condor-G [9] perform scheduling related activities independent of the other brokers in the system. They directly submit their 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 because sub-optimal schedules are likely to occur.

B. *Centralised Approaches*

One of the possible ways to solve coordination problem among distributed brokers or users has been to host a coordinator service on a centralised machine [12], [18], [25], wherein every consumer is required to submit his demands to the coordinator (similar to Bellagio system). Similarly, resource providers update their resource usage status periodically with the coordinator. The centralised resource allocation coordinator performs system wide load-distribution primarily driven by resource demand and availability. However, this approach has several design limitations including: (i) single point of failure; (ii) lacks scalability; (iii) high network communication cost at links leading to the coordinator (i.e. network bottleneck, congestion); and (iv) computational power required to serve a large number of participants.

C. Unstructured Decentralised Approaches

The coordinated scheduling protocols adopted by NASA-Scheduler [21] and Condor-Flock P2P [7] are based on general broadcast and limited broadcast communication mechanism respectively. Similarly, scheduling coordination in Tycoon [16] is based on a decentralized and isolated auction mechanism, where a user can end up bidding across every auction in the system (broadcast messaging). OurGrid [3] system coordinates load-information among the sites based on a complete broadcast messaging approach. Specifically, OurGrid utilises JXTA search [14] primitives as regards to resource discovery and message routing. Hence, these unstructured decentralised approaches have the following serious imitations: (i) high network overhead; and (ii) scalability problems.

D. Proposed Approach: Structured Decentralised

In this work, we propose that the role of the centralised coordinator be distributed among a set of machines based on a P2P network model. New generation P2P routing substrate such as the DHTs [22], [20] can be utilised for efficient management of such decentralised coordination network. DHTs are inherently self-organising, fault-tolerant and scalable. Further, DHT services are light-weight and hence, do not warrant an expensive hardware infrastructure.

Specifically, we advocate organising resource brokers (and users in case of PlanetLab) and distributed resources based on a DHT overlay. In the proposed approach resource brokers post their resource demands by injecting a *Resource Claim* object into the DHT-based decentralised coordination space, while resource providers update the resource supply by injecting a *Resource Ticket* object (similar terminologies have been used by the Sharp [10] and Shirako [15] system). These objects are mapped to the DHT-based coordination services using a spatial hashing technique. The details on spatial hashing technique and object composition are discussed in Section V. Decentralised coordination space is managed by a software service (a component of Grid broker service) called coordination service. It undertakes activities related to decentralised load-balancing, coordination space management etc.

A coordination service on a DHT overlay is made responsible for matching the published resource tickets to subscribed resource claims such that the resource ticket issuers are not overloaded. Resource tickets and resource claims are mapped to the coordination space based on the distributed spatial hashing technique [24]. Every coordination service in the system owns a part of the coordination space governed by the overlay's hashing function (such as SHA-1). In this way, the responsibility of load-distribution and coordination is delegated to set of machines. The actual number of machines and their respective coordination load is governed by the spatial index's load-balancing capability. Note that, both resource claim and resource ticket objects have their extent in d-dimensional space.

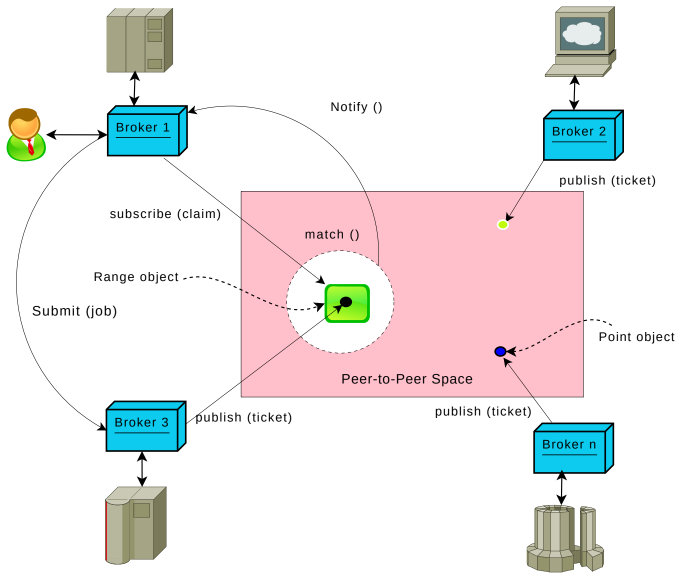


Fig. 1. Coordinated load management based on a P2P space. Users post their resource demands on the P2P space through subscribe(resource claim) primitive and resources update the resource status through publish(resource ticket) primitive. Resource claims are window objects (shown as rectangles) while resource tickets are point objects (denoted by dots). Above figure represents a P2P space in 2-dimensions.

The DHT based P2P space performs the runtime load-balancing across the federation sites based on their current resource utilisation. The load-balancing mechanism is simple: the coordination service notifies a resource claimer to submit his jobs to a particular resource ticket issuer (resource provider) only if the contacted resource ticket issuer is not-overloaded as a result of accepting the claimer's job. Resource metrics such as the number of processors available or utilisation forms the basis for determining the load at a ticket issuer's site. In Fig. 1, brokers, on behalf of local users post the resource claims to the DHT based P2P space. Similarly, resource providers post the resource ticket objects on the space. When a claim object overlaps a ticket object, the P2P space sends notification to the respective resource claimer about the match. Following this the resource claimer can go ahead and deploy his distributed experiment or application on that resource.

E. Our Contributions

The main contributions of this paper include: (i) global coordination protocol for load-management between distributed brokers; (ii) proposal for utilising the DHT based spatial index for managing the complex coordination objects and decentralising the protocol; and (iii) extensive simulation based experiments for analysing the effectiveness of the proposed protocol. We now summarise some of our findings:

- resource claim and ticket object injection rate has significant influence on the coordination delay experienced by distributed users in the system.
- proposed coordination protocol is highly effective in curbing the number of scheduling negotiation iteration undertaken on per job basis, the redemption and notification message complexity involved with the coordination protocol is bounded by function $\Theta(1)$.
- In a federation of n heterogeneous nodes, on average the number of messages required to successfully map a job to a resource is bounded by function $\Theta(\log n)$.

II. Paper Organisation

The rest of this paper is structured as follows: Section III sets the proposed coordination protocol in context with related work. Section IV gives an overview of the Grid scheduling model that we consider in this paper. Next, Section V discusses the key elements of coordination protocol. Section VI summarises d-dimensional spatial index that forms the basis for mapping the coordination objects. In Section VII, we present the finer details on the coordinated application scheduling and resource provisioning algorithms. Message complexity analysis is presented in Section VIII. In Section IX, we present various experiments and discuss our results. We end this paper with concluding remarks in Section X.

III. Related Work

The main focus of this section is to compare the novelty of the proposed work with respect to the current state-of-the-art. Coordinated management of federated distributed computing resources has been a widely studied research problem in the recent past. Several research projects including Bellagio [4], Tycoon [16], NASA-Scheduler [21], OurGrid [3], Sharp [10], Shirako [15], and Condor-Flock [7] have proposed federated models for organising distributed compute and storage resources. These systems offer varying degree of global coordination with respect to load-management. Further based on the network and communication models these approaches have different scalabilities.

Shirako [15] system presents a mechanism for on-demand leasing of shared networked resources. Shriako's leasing architecture builds upon the Sharp framework for secure resource peering and distributed resource allocation. Participant sites delegate the resource leasing decision control to broker(s) in the system. The broker(s) implements policies for resource selection, provisioning and admission control. A resource provider in Shirako posts resource ticket with the broker, while a resource consumer requests the tickets from a broker. The broker issues a ticket for a resource type, quantity, and site location that matches the given request. However, Shirako system does not define how different brokers in the system connect with each other as the system grows to a large number of participants. This is based on the assumption that each participating site or administrative domain instantiates its own broker that controls the way local resources are made available to the outside world. In this setting, brokers should be able to perform global coordination in a scalable and efficient manner. The novel contribution of the proposed work lies in this domain i.e. efficient protocol for enforcing global coordination among distributed brokers in the system.

Bellagio [4] is a market-based resource allocation system for federated distributed computing infrastructures. Resource allocation in this system is based on bid-based proportional resource sharing model. Bids for resources are cleared by a centralized auctioneer. Effectively, the centralised auctioneer performs the role of the coordinator in the system. With the centralised auction approach the best-case communication overhead of $O(c)$ is involved if the auction is limited to c participants and $O(n)$ if not. In contrast, the proposed coordinated load-management protocol is based on a DHT based d-dimensional space that is inherently scalable, self-organising and does not suffer from a single point of failure.

Tycoon [16] is a distributed market-based resource allocation system. Application scheduling and resource allocation in Tycoon is based on decentralised isolated auction mechanism. Every resource owner in the system runs his own auction for his local resources. Furthermore, auctions are held independently, thus clearly lacking any coordination. In worst case, a scheduler can end-up bidding across all n sites in the system. Hence on per job basis a scheduler can generate $O(n)$ messages in the system. Since, Tycoon is based on distributed auction, therefore it has a best-case communication overhead of $O(n\ c)$ is based on the assumption that an auction is limited to c participants. In contrast with the proposed coordinated load-management protocol, a broker needs to undertake close to logarithmic number messages (that we show later) on per job basis.

The system [21] models a Grid broker architecture and studies three different distributed job migration algorithms. Each computational resource site has a Grid broker/superscheduler (GS) and a local scheduler (LRMS). Scheduling in the Grid environment is facilitated through coordination between site specific LRMS and the GS. System-wide load coordination algorithms such as the sender-initiated, receiver-initiated and symmetrically-initiated algorithms are based on complete broadcast messaging between participants GSes, thereby clearly incurring $O(n)$ messages on per job basis. We improve on these load coordination algorithms by reducing the number of messages close to logarithm of the number of GSes in the system.

Condor-Flock [7] presents a Grid scheduling system that consists of Internet-wide Condor work pools. It utilises the Pastry routing substrate with respect to decentralisation and scalable resource discovery. The site managers in the overlay coordinate the load-management by announcing its available resources to all the sites whose Identifiers (IDs) appear in the routing table. Hence in network of n condor sites, $O(n\ log_b\ n)$ messages are generated per resource status change on per site basis. An optimised version of this protocol proposes recursively propagating the load-information to the sites whose

IDs are indexed by contacted site's routing table. A load-information message is removed from the system based on the system-wide configured Time-to-Live (TTL) value. However, if the TTL value is sufficiently large then the number of messages generated in the system converges towards the broadcast communication.

Similarly, the load-information coordination in OurGrid [3] is also based on complete broadcast messaging. OurGrid system is implemented using the JXTA [14] P2P substrate. In contrast, our coordinated load-management protocol utilises a d-dimensional spatial index over a DHT space for deterministic lookups and coordination. This gives the system ability to produce controllable number of messages and guarantees a deterministic behaviour with respect to number of routing hops.

IV. Models

A. Grid Scheduling Model

Coordinated load management protocol derives from the *Grid-Federation* [19] resource sharing model. Grid-Federation aggregates distributed resource brokering and allocation services as part of a cooperative resource sharing environment. The Grid-Federation, $G_F = \{R_1, R_2, \ldots, R_n\}$, consists of a number of sites, n, with each site having to contribute its local resources to the federation. Every site in the federation has its own resource set R_i which contains the definition of all resources that it is willing to contribute. R_i can include information about the CPU architecture, number of processors, RAM size, secondary storage size, operating system type, etc. In this work, $R_i = (p_i, x_i, \mu_i, \o_i, \gamma_i)$, which includes the number of processors, p_i, processor architecture, x_i, their speed, μ_i, installed operating system type, $\o_i$, and underlying interconnect network bandwidth, γ_i . Refer to Table. I for the notations that we utilise in the remainder of this paper.

Resource brokering, indexing and allocation in the Grid-Federation is facilitated by a Resource Management System (RMS) known as the Grid Federation Agent (GFA). Fig. 2 shows an example Grid-Federation resource sharing model consisting of Internet-wide distributed parallel resources. Every contributing site maintains its own GFA service to keep control of its resources, manage local jobs and allocate processors to jobs that are migrated from remote sites in the federation. The GFA service is composed of three software entities including a Grid Resource Manager (GRM), Local Resource Management System (LRMS) and Distributed Information Manager (DIM) or Grid Peer.

The GRM component of a 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 the locally submitted jobs in the federation. Further, it also manages the execution of remote jobs in conjunction with the local resource management system. The LRMS software module can be realised using systems such as PBS [6], SGE [13]. Additionally, LRMS implements the following methods for facilitating federation wide job submission and migration process: answering the GRM queries related to local job queue length, expected response time and current resource utilization status.

TABLE I

NOTATIONS.

Symbol	Meaning
Resource	
n	number of Grid Federation Agents (GFAs) in the Grid network
R_i	configuration of the i-th resource in the system.
ρ_i	resource utilisation for resource at GFA i.
x_i	processor architecture for resource at GFA i.
p_i	number of processors for reosurce at GFA i.
ϕ_i	operating system type for resource at GFA i.
μ_i	processor speed at GFA i.
Job	
$J_{i,j,k}$	i-th job from the j-th user of k-th GFA.
$p_{i,j,k}$	number of processor required by $J_{i,j,k}$.
$\phi_{i,j,k}$	operating system type required by $J_{i,j,k}$.
$T(J_{i,j,k}, R_k)$	time function (expected response time for $J_{i,j,k}$ at resource k).
Index	
$r_{i,j,k}$	a claim posted for job $J_{i,j,k}$.
U_i	a ticket issued by the i-th GFA/broker.
dim	dimensionality or number of attributes in the Cartesian space.
f_{min}	minimum division level of d-dimensional index tree.
$gindex_i$	object encapsulating details on a GFA's IP address, service port number etc.
M_c, M_t	random variables denoting number of of messages generated in mapping a claim and ticket object.
T_c, T_t	random variables denoting number of disjoint query paths undertaken in mapping a claim and ticket object.
Network	
λ^{in}	total incoming claim/ticket arrival rate at a network queue i.
λ^{out}	outgoing claim/ticket rate at a network queue i.
μ_{n_i}	average network queue service rate at a Grid peer i.
μ_r	average query reply rate for index service at GFA i.
λ_t^{in}	incoming ticket rate at a application service i.
λ_c^{in}	incoming claim rate at a application service i.
λ_a^{in}	incoming query rate at a DHT routing service i from the local application service.
λ_{index}^{in}	incoming index query rate at a application service i from its local DHT routing service.
K	network queue size .

The Grid peer module in conjunction with indexing service performs tasks related to decentralised resource lookups and updates. A Grid Peer service generates two basic types of objects with respect to coordinated grid brokering: (i) a claim, a object sent by a GFA service to the P2P space for locating the resources matching a user's application requirements; and (ii) a ticket, is an update object sent by a Grid site owner 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,

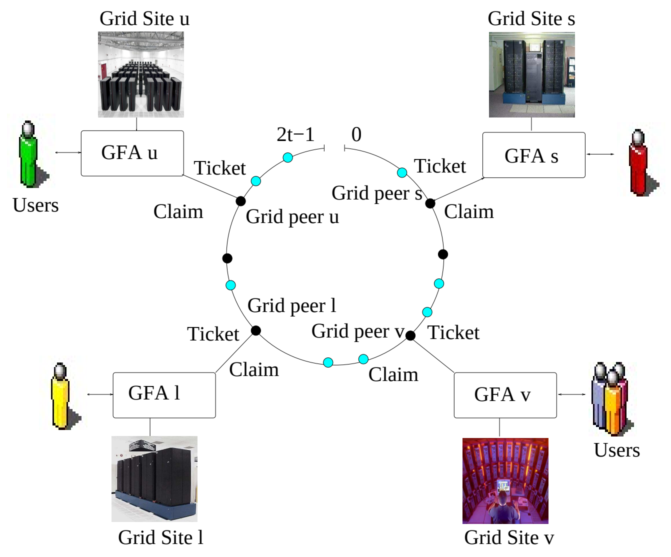


Fig. 2. GFAs and Grid sites with their Grid peer service and some of the hashings to the Chord ring. Dark dots are the Grid peers that are currently part of Chord based Grid network. Light dots are the ticket/claim object posted by Grid sites and GFA service.

in case the query specifies a range of values for attributes, then it is referred to as a d-*dimensional Window Query* (DWQ) or a d-*dimensional Range Query* (DRQ). In database literature, a DWQ or an DRQ is also referred to as a *spatial range query*. Grid peer component of GFA service is responsible for distributed ticket publication, claim subscription and overlay management processes.

B. Job Model

In order to study the effectiveness of the proposed protocol with respect to load management, we consider coordinated scheduling of synchronous parallel applications on a distributed system of parallel resources (such as cluster or supercomputer). A job consists of the number of processors required, $p_{i,j,k}$, the job length, $l_{i,j,k}$ (in terms of instructions), and the communication overhead, $\alpha_{i,j,k}$. We write $J_{i,j,k}$ to represent the i-th job from the j-th user of the k-th site.

To capture the nature of parallel execution with message passing overhead involved in the parallel application, we considered a part of total execution time as the communication overhead and remaining as the computational time. We consider the network communication overhead $\alpha_{i,j,k}$ for a parallel job $J_{i,j,k}$ to be randomly distributed over the processes. In other words, we don't consider the case e.g. when a parallel program written for a hypercube is mapped to a mesh architecture. We assume that the communication overhead parameter $\alpha_{i,j,k}$ would scale the same way over all the clusters depending on γ_i. The total data transfer involved during a parallel job execution is given by,

$$\Gamma(J_{i,j,k}, R_k) = \alpha_{i,j,k} \times \gamma_k.$$

The time for job $J_{i,j,k} = (p_{i,j,k}, l_{i,j,k}, \alpha_{i,j,k})$ to execute on a parallel resource R_m is,

$$T(J_{i,j,k}, R_m) = \frac{l_{i,j,k}}{\mu_m \, p_{i,j,k}} + \frac{\Gamma(J_{i,j,k}, R_k)}{\gamma_m}.$$

V. Coordination Protocol

We start this section with the description on the communication, coordination and indexing models that are utilised to facilitate the P2P coordination space. Thereafter we look at the composition of objects, access primitives that form the basis for coordinating the application schedules among the distributed GFAs/brokers.

A. Coordination Objects

This section gives details about the resource claim and ticket objects that form the basis for enabling decentralised coordination mechanism among the brokers/GFAs in a Grid system. These coordination objects include:- Resource Claim and Resource Ticket. We start with the description of the components that form the part of a Grid-Federation resource ticket object.

Resource Ticket

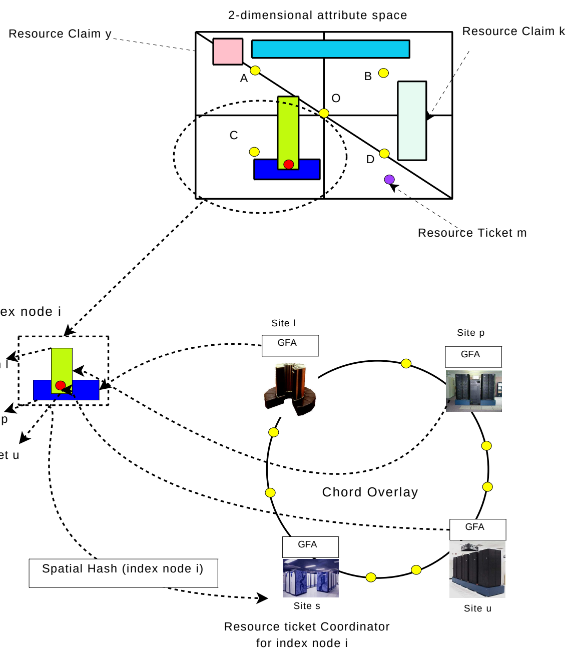


Fig. 3. Resource allocation and application scheduling coordination across Grid sites.

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

Resource Ticket: Total-Processors= 100 && Processor-Arch= Pentium && Processor-Speed= 2 GHz && Operating-

System = Linux && Utilization=0.80.

Resource Claim

A resource claim or look-up object encapsulates the resource configuration needs of a user's job. In this work, we focus on the job types the needs of which are confined to computational grid or PlanetLab resources. Users submit their application's resource requirements to the local GFA. The GFA service is responsible for searching the resources in the federated system. A GFA aggregates the characteristics of a job including $p_{i,j,k}$, $x_{i,j,k}$, $\o_{i,j,k}$ with constraint on maximum speed, and resource utilization into a resource claim object, $r_{i,j,k}$.

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 resource ticket and claim objects are spatially hashed to an index cell i in the d-dimensional coordination space. Similarly, coordination services in the Grid network hash themselves into the space using the overlay hashing function (SHA-1 in case of Chord and Pastry). In Fig. 3, resource claim objects issued by site p and l are mapped to the index cell i, are currently hashed to the site s. In this case, 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 dot in the Fig. 3) that falls under a region of the space currently required by users at site p and l. In this case, the coordinator service of site s has to decide which of the sites (i.e. either l or p or both) 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.

TABLE II
CLAIM LIST.

Time	Claim ID	$\mu_{i,j,k}$	$p_{i,j,k}$	$x_{i,j,k}$	$\phi_{i,j,k}$
200	GFA2U1J2	$>$ 800	50	Intel	Linux
350	GFA5U3J5	$>$ 1200	20	Intel	Linux
500	GFA6U10J13	$>$ 700	10	Sparc	Solaris
700	GFA10U3J5	$>$ 1500	1	Intel	Windows XP

TABLE III
TICKET LIST.

Time	GFA ID	μ_i	p_i	p_i avail	x_i	ϕ_i
900	GFA-8	1400	80	75	Intel	Linux

In Table II, we show an example list of claim objects that are stored with a coordination service at time $t = 900$ $secs$. 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 III depicts the list of ticket objects that have arrived at $t = 900$ $secs$. 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 only Claim 1 and Claim 2 matches the ticket's resource configuration. As specified in the ticket object there are currently 75 processors available with the GFA 8, which is less than the sum of processors required by Claim 1 and 2 (i.e., 70). Hence in this case, the coordination service based on a First-Come-First-Server (FCFS) queue processing scheme, first notifies the GFA that has posted Claim 1 and follows it with the GFA responsible for Claim 2. However, the Claims 3 and 4 has 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, then a coordinator 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 resource brokers from overloading the same resource. In case of PlanetLab environment, it can prevent the users from instantiating *slivers* on the same set of nodes. The coordination service notifies a claimer for resources by issuing a soft state pass that is redeemable for a lease at the ticket issuer GFA in the system. The soft state pass specifies the resource type R_i for which access should be granted over a duration (expected running time of an application). GFAs on behalf of locate site issue tickets for resources and post to the coordination space. Our coordination protocol can leverage the Sharp framework [10] with respect to secure resource exchanges. In Sharp all exchanges are digitally signed, and the GFAs/brokers endorse the public keys of the GFAs in the system.

VI. 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 those objects whose extents lie in d-dimensional space warrants embedding a logical index structure over the 1-dimensional DHT key space.

We now describe the features of the P2P-based spatial index that we utilise for mapping the d-dimensional claim and ticket objects over the DHT space. Providing background work and details on this topic is beyond the scope of this paper; here we only give a high level picture. 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 that region, and for storing the objects that are associated with the region. Fig. 4 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 a entry points for subsequent mapping of claim and ticket objects. The number of index cells produced at the minimum division level is always equal to $(f_{min})^{dim}$, where dim is the dimensionality of the Cartesian space. More finer details on recursive sub-division technique can be found in the article [24]. Every Grid peer in the network has the basic information about the Cartesian space coordinate values, dimensions and minimum division level.

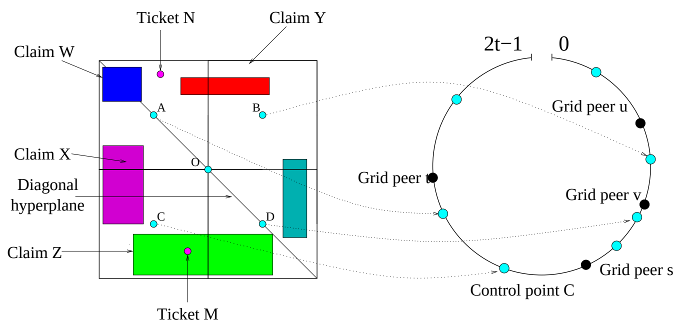


Fig. 4. Spatial resource claims $\{W,X,Y,Z\}$, cell control points, point resource tickets $\{M\}$ and some of the hashings to the Chord, i.e., the d-dimensional coordinate values of a cell's control point is used as the key and hashed onto the Chord. Dark dots are the grid peers that are currently part of the network. Light dots are the control points hashed on the Chord. For this figure, f_{min} = 2, dim=2.

Every cell at the f_{min} level is uniquely identified by its centroid, termed a *control point*. Fig. 4 depicts four control points A, B, C and D. DHT hashing method such as the Chord method is utilised to hash these control points so the responsibility for managing a index cell is associated with a Grid peer in the system. In Fig. 4, control point B is hashed to the Grid peer s, which is responsible for managing all claim and ticket object that are stored with that control point. For mapping claim objects, the search strategy depends whether it is a DPQ or DRQ. For a DPQ type claim, the mapping is straight forward since every point is mapped to only one cell in the Cartesian space. For a DRQ type claim, mapping is not always singular because a range look-up can cross more than one cell. To avoid mapping a range claim to all the cells that it crosses (which can create many unnecessary duplicates) a mapping strategy based on diagonal hyperplane of the Cartesian space is utilised. This mapping involves feeding a claim candidate index cells as inputs into a mapping function, F_{map}. This function returns the IDs of index cells to which given claim 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 use the index cell(s) currently assigned to it and a set of known base index cells obtained at initialisation as the candidate index cells.

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

VII. ALGORITHMS

A. *Distributed Application Scheduling Algorithm*

In this section we provide detailed descriptions of the scheduling algorithm that is undertaken by a GFA in the Grid-Federation system following arrival of a job:

1. When a job $J_{i,j,k,}$ arrives at a GFA, the GFA compiles a resource ticket object for that job. It then posts this resource ticket object with the P2P space though the Core services layer. In Fig. 1 GFA 1 is posting a resource claim on behalf of its local user.

2. When a GFA receives a notification (soft state pass) for resource ticket and resource claim match from the P2P coordination space, then it contacts the ticket issuer GFA for redeeming the ticket. After notifying the claimer GFA, the coordination service unsubscribes the resource claim for that job from the space. In Fig. 1, the match event occurs and GFA 1 is notified that it can redeem the matched ticket with GFA 3. Once GFA 3 agrees to grant access to its local resources then GFA 1 transfers the locally submitted job to the GFA 3. Further, GFA 1 unsubscribes the claim object from the space to remove duplicates.

3. However, if the ticket issuer GFA fails to grant access due to local resource sharing policy then the claimer GFA re-posts the resource claim for $J_{i,j,k}$ with P2P space for future notifications.

B. *Decentralised Resource Provisioning Algorithm*

In this section, we present the details on the decentralised resource provisioning algorithm that is undertaken by the coordination services across the P2P space.

1. When a resource claim object arrives at a coordination service for future consideration, the coordination service queues it in the existing claim list as shown in the Algorithm. 1.

2. When a resource ticket object arrives at a coordination service, the coordination service calls the auxiliary procedure match(ticket) (as shown in Algorithm. 1) to gather the list of resource claims, which overlap with the submitted resource ticket object in the d-dimensional space. This initial resource claim match list is passed to another auxiliary procedure Load_Dist(matchList, ticket).

```
0.1  PROCEDURE: Resource_Provision
0.2  begin
0.3      $list \leftarrow \phi$
0.4      begin
0.5          Claim_Arrival (Claim $r_{i,j,k}$)
0.6          $list \leftarrow list \cup r_{i,j,k}$.
0.7      end
0.8      begin
0.9          Match (Ticket $U_i$)
0.10         $list_m \leftarrow \phi$
0.11         set $index = 0$
0.12         while ( $list[index] \neq null$ ) do
0.13             if ( Overlap ($list[index]$, $U_i$) ) then
0.14                 $list_m \leftarrow list_m \cup list[index]$
0.15             end
0.16             else
0.17                 continue
0.18             end
0.19             reset $index$ = $index$ + 1
0.20         end
0.21         return $list_m$ .
0.22     end
0.23     begin
0.24         Overlap (Claim $r_{i,j,k}$, Ticket $U_i$)
0.25         if ($r_{i,j,k} \cap U_i \neq null$ ) then
0.26             return true.
0.27         end
0.28         else
0.29             return false.
0.30         end
0.31     end
0.32     begin
0.33         Ticket_Arrival ($U_i$)
0.34         call Load_Dist($U_i$, Match($U_i$)).
0.35     end
0.36     begin
0.37         Load_Dist ($U_i$, $list_m$)
0.38         set $index$ = 0
0.39         while ($R_i$ is not over-provisioned) do
0.40             send notification match event to resource claimer:
                 $list_m$ [$index$]
0.41             remove($list_m$ [$index$])
0.42             reset $index$ = $index$ + 1.
0.43         end
0.44     end
0.45 end
     algocf
```

Algorithm 1: Decentralised Resource provisioning algorithm for coordination service.

3. The Load_dist() procedure notifies the resource claimers about the resource ticket match until the ticket issuer is not over-provisioned. The Load_Dist() procedure can utilise the resource parameters such as the number of available processors, threshold queue length etc as the over-provision indicator. And these over-provision indicators are encapsulated with the resource ticket object by the GFAs. The GFAs can post the resource ticket object to the P2P space either periodically or whenever the resource condition changes such as a job completion event happens.

C. Layered Design of the Coordination Protocol

Fig. 5 shows the layered design of the proposed coordination protocol. The OPeN architecture proposed in the work [23] is utilised as the base model in architecting and implementing the proposed protocol. The OPeN architecture consists of three layers: the *Application* layer, *Core Services* layer and *Connectivity* layer. Grid Services such as resource brokers work at Application layer and insert objects to the Core services layer.

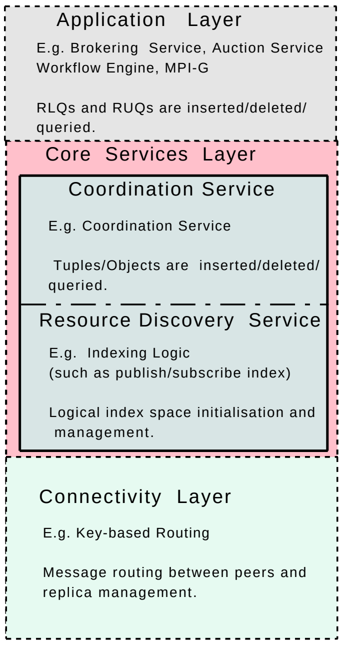

Application Layer
E.g. Brokering Service, Auction Service
Workflow Engine, MPI-G
RLQs and RUQs are inserted/deleted/
queried.
Core Services Layer
Coordination Service
E.g. Coordination Service
Tuples/Objects are inserted/deleted/
queried.
Resource Discovery Service
E.g. Indexing Logic
(such as publish/subscribe index)
Logical index space initialisation and
management.
Connectivity Layer
E.g. Key-based Routing
Message routing between peers and
replica management.

Fig. 5. Layered design of the coordination protocol.

We have implemented 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. These objects are managed by coordination service. The calls between the Coordination service and Resource discovery service are made through the standard publish/subscribe way. The Resource discovery service is responsible for managing the logical index space and communicating with the Connectivity layer.

The Connectivity layer is responsible for undertaking key-Based routing in the DHT space such as Chord, CAN, Pastry etc. The actual implementation protocol at this layer does not directly affect the operations of the Core services layer. In principle, any DHT implementation at this layer could perform the desired task. However, in this paper the simulation models 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 guarantees that an object in the network can be located in $\Theta(\log\ n)$ steps with high probability. Each peer in the Chord network is required to maintain the routing state of only $\Theta(\log\ n)$ other peers, where n is the total network size.

VIII. Complexity Analysis

Definition 1 In a federation of n heterogeneous nodes, on average a job $J_{i,j,k}$ requires $\Theta(\log n)$ messages to be sent in the network in order to locate a node that can successfully complete the job without being overloaded.

Scheduling a job $J_{i,j,k}$ in the coordinated federation involves the following steps: (i) posting the resource claim object to DHT based tuple space; (ii) receiving the notification message from the tuple space when a resource ticket object hits the claim object; (iii) contacting the GFA (site authority) about the claim-ticket match, the contacted GFA performs certain checks such as security, resource availability. Hence, the total number of messages produced in successfully allocating a job to a resource is summation of the number of messages produced in these three steps.

We denote the number of messages generated in mapping a resource claim by a random variable M_c. The distribution of M_c is function of the problem parameters including query size, dimensionality of search space, query rate, division threshold and data distribution. Note that, the derivation presented in this paper assumes that the Chord method is used for delegation of service messages in the network. Essentially, a control point at the f_{min} level of the logical d-dimensional Cartesian space can be reached in $\Theta(\log n)$ routing hops using the Chord method. Based on the above discussion, in order to compute the worst case message lookup and routing complexity one additional random variable T_c is considered. T_c denotes the number of disjoint query path undertaken in mapping a claim object. Note that, with this spatial index a resource claim can be utmost mapped to 2 index cells. And every disjoint path will undertake $E[T_c] \times (log_2\ n)$ routing hops with high probability. Hence, the expected value of M_c is given by:

$$E[M_c] = E[T_c] \times (log_2\ n),$$

substituting $E[T_c]$ with the value 2 and adding 2 for messages involved with sending notification and negotiating SLAs (steps 2 and 3),

$$E[M_c] = 2 \times (\log_2 n) + 2,$$

ignoring the constant terms in the above equation we get,

$$E[M_c] = \Theta(\log n). \qquad (1)$$

Lemma 1 In a federation of n GFAs/brokers, each broker having m jobs to schedule then total scheduling messages generated in the system is bounded by the function $\Theta(m \times n \times \log n)$.

This lemma directly follows from Definition 1. Since a job in the system requires $\Theta(\log n)$ messages to be undertaken before it can be successfully allocated to resource, therefore computing the scheduling message complexity for m jobs distributed over n GFAs/brokers is straightforward.

Definition 2 In a federation of n heterogeneous resources if GFAs/brokers posts p resource ticket object over a time period t, then the average-case message complexity involved with mapping these tickets to Grid peers in the network is bound by the function $\Theta(E[T_t] \times p \times \log\ n)$.

The proof for this definition directly follows from Definition 1 and Lemma 1. The procedure for mapping the ticket object to the Grid peers is similar to the one followed for a claim object. A ticket object is always associated with an event region, and all index cells that fall fully and partially within the even region will be selected to receive the corresponding ticket object. The number of disjoint query path taken to map a ticket object is denoted by random variable T_t with mean $E[T_t]$.

IX. Performance Evaluation

In this section, we validate the effectiveness of the proposed coordination protocol through the trace driven simulation.

A. Simulation Model

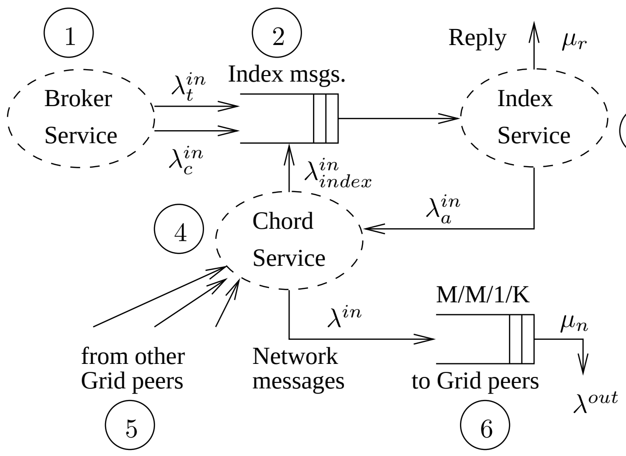


Fig. 6. Network message queueing model at a Grid peer i.

In our message queueing model, a Grid peer node (through its Chord routing service) is connected to an outgoing message queue (see label 6 in Fig. 6) and an incoming link from the Internet (see label 5 Fig. 6). The network messages that are delivered through the incoming link (effectively coming from other Grid peers in the overlay) are processed as soon as they arrive. If the destination of the incoming messages is the local index service then they are put in the index message queue (see label 2 in Fig. 6). Else the messages are routed through the outgoing queue to the next successor Chord service, which is more closer to the destination in the Chord key space. Further, the Chord routing service (see label 4 in Fig. 6) receives messages from the local index service (see label 3 in Fig. 6). Similarly, these messages are processed as soon as they arrive at the Chord routing service. After processing, Chord routing service queues the message in the local outgoing queue. Basically, this queue models the network latencies that a message encounters as it is transferred from one Chord routing service to another on the overlay. Once a message leaves an outgoing queue it is directly delivered to a Chord routing service through the incoming link. The distributions for the delays (including queueing and processing) encountered in an outgoing queue are given by the M/M/1/K [2] queue steady state probabilities.

B. *Experimental Setup*

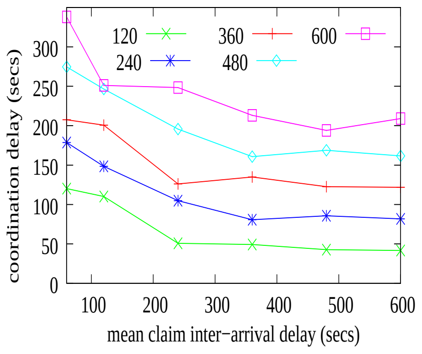
We start by describing the test environment setup.

Experiment configuration:

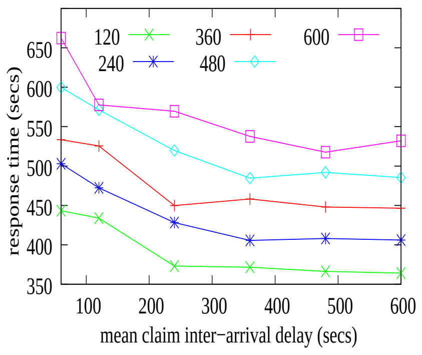
- Simulators: Our simulation infrastructure was modeled by combining two discrete event simulators namely *GridSim* [8], and *PlanetSim* [11]. GridSim offers a concrete base framework for simulation of different kinds of heterogeneous resources, services and application types. PlanetSim is an event-based overlay network simulator. It can simulate both unstructured and structured overlays.

- Network configuration: The experiments ran a Chord overlay with 32 bit configuration i.e. number of bits utilised to generate node and key ids. The GFA/broker network size n was fixed to 100. Next, network queue message processing rate, μ, was fixed at 500 messages per second. And message queue size, K, was fixed at 10^4.
- Resource claim and resource ticket injection rate: The GFAs injected resource claim and ticket objects based on the exponential inter-arrival time distribution. The value for resource claim inter-arrival delay, $\frac{1}{\lambda_c^{in}}$, was distributed over $[60, 600]$ in steps of 60 seconds. While the GFAs injected resource ticket objects with mean delay, $\frac{1}{\lambda_t^{in}}$, distributed over $[120, 600]$ in steps of 120 seconds. Note that, mean inter-arrival delay for injecting the claim/ticket object was modeled to be the same for all the GFAs/brokers in the system. The GFAs in the system posted ticket objects in the system untill all the jobs are successfully executed and job's output returned to their respective GFA. Spatial extent of both resource claim and ticket objects lied in a 4-dimensional attribute space, i.e., $dim = 4$. 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 distributions for these resource dimensions were obtained from the Top 500 supercomputer list[1].
- Resource load indicator: The GFAs/brokers encode the metric *"number of available processors"* at time t 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 coordinator service will stop sending the notifications as the number of processors available with a ticket issuer approaches *zero*.
- Spatial index configuration: f_{min} of the logical d-dimensional spatial index was 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.

[1]Top 500 Supercomputer List, http://www.top500.org/



(a) mean claim inter-arrival delay (secs) vs average coordination delay (secs).

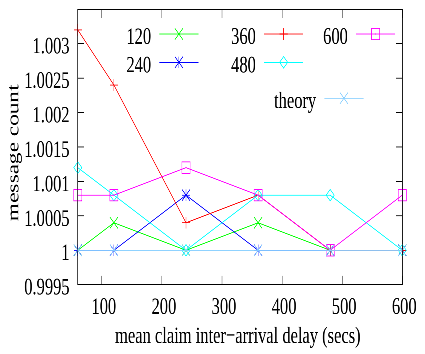


(b) average claim inter-arrival delay (secs) vs average response time (secs).

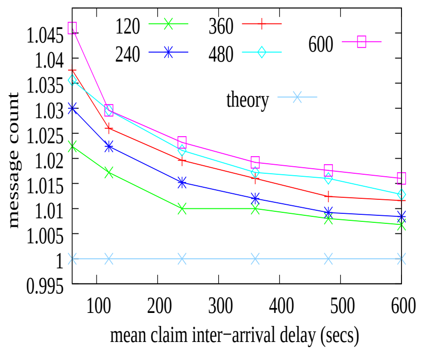
Fig. 7. Scheduling perspective

- Workload configuration: We generated the workload distributions across GFAs based on the model given in the paper [17]. The workload model generates the job-mixes having the details on their run times, sizes, and inter-arrival times. This model is statistically derived from existing workload traces and incorporates correlations between job run times and job sizes and daytime cycles in job inter-arrival times. The model calculates for each job its arrival time using 2-gamma distributions, number of nodes using a two-stage-uniform distribution, and run time using the number of nodes and hyper-gamma distribution. The job characteristics were generated by configuring the minimum and maximum processor per job as 2 and 2^6 respectively in the workload model. Mostly we utilised the default parameters already given by the model except for the number of processors/machines. The processor count for a resource was fed to the workload model based on the resource configuration obtained from the Top 500 list. The simulation environment models 25 jobs at each GFA, and since there are 100 GFAs therefore total number of jobs in the system accounts to 2500. Also note that, we simulated the supercomputing resources in space shared processor allocation mode.

C. Results and Observations



(a) mean claim inter-arrival delay (secs) vs message count



(b) mean claim inter-arrival delay (secs) vs message count

Fig. 8. Scheduling perspective

1) Scheduling Perspective: We measured the performance of coordination protocol with respect to the following scheduling metrics: average coordination delay, average response time, and average scheduling messages.

Fig. 8 shows the measurement for parameters coordination delay, response time, processing time, job migration, scheduling, and notification iterations. 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. Processing time for a job is defined as the time a job takes to actually execute on a processor or set of processors. Average response time for a job is the delay between the submission and arrival of execution output. Effectively, the response time includes the latencies for coordination and processing delays. Note that, these measurements were collected by averaging the values obtained for each job in the system. These results presented here were averaged for all the jobs across every GFA in the system.

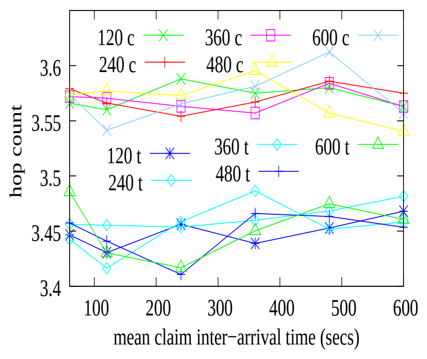
Fig. 7(a) depicts results for average coordination delay in seconds with increasing claim mean inter-arrival delay, $\frac{1}{\lambda_c^{in}}$. With increase in claim's mean inter-arrival delay, we observed decrease in the average coordination delay. The results show that at higher inter-arrival delays, resource claim objects experience less network traffic and competing requests. Thus, this lead to an overall decrease in the coordination delay across the system (see Fig. 7(a)). An other interesting thing to note here is the variation of coordination delay as the resource ticket's mean inter-arrival delay increase. We varied the $\frac{1}{\lambda_t^{in}}$ over the interval $[120, 600]$ in steps of 120. As seen in the Fig. 7(a), the increase in ticket's mean inter-arrival delay worsens the coordination delay. The chief reason for this being that claims had to wait for longer period of time before they were hit by a ticket object. The effect of this can also be seen in the response time metric for the jobs (see Fig. 7(b)), which is also seen to worsen with increase in ticket's mean inter-arrival delay.

The proposed coordination protocol was highly successful in curbing the number of scheduling negotiation (redemption) iteration undertaken on per job basis (see Fig. 8(a)). Next, on the average GFAs/brokers receive close to 1 coordination notification on per job basis Fig. 8(b)). This suggests that the redemption and notification complexity involved with the coordination protocol is $\Theta(1)$. In Fig. 8(a) and 8(b), we also plot the function $\Theta(1)$ with label "theory".

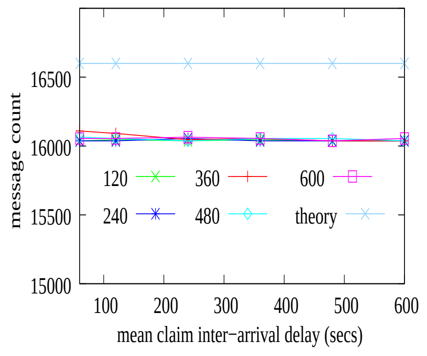
2) Coordination Space Perspective: Here we analyse the performance overhead of the DHT-based space as regards to facilitating coordinated scheduling among distributed GFAs. In particular we measured the following metrics: (i) messages produced mapping claim and ticket objects, (ii) number of routing hops undertaken to map claim and ticket objects to index cells and (iii) total network latency observed as claim and ticket objects are routed over the Chord space.

Fig. 9(a) shows the number of routing hops undertaken at different claim and ticket injection rate across the GFAs in the system. Suffices *"t"* and *"c"* are used in as the label in Fig. 9(a) to represent the values for ticket and claim object respectively. Recall that that values shown here has been averaged for all the jobs over all the GFAs in the system. As expected with increase in the injection rates of the objects, the number of routing hops did not change significantly. We observed that the average routing hops for mapping ticket objects was around 3.6, while for claim objects it was around 3.4. This shows that routing hops for mapping claim/ticket object is as expected in a Chord based routing space, i.e., bounded by the function $\Theta(\log\ n)$. As $\log_2\ (100) = 6.64$, it can easily shown that $c_1 \times 3.6 \leq 6.64 \leq c_2 \times 3.6$, where c_1 and c_2 are constants.

In Fig. 9(b), we show the total number of messages produced as a result of successfully mapping and executing all the jobs in the system. The measurement shown in this figure includes both mapping and scheduling messages. Mapping messages for a job equals to the number of messages that are required to map a claim object for that job to the coordination space. According to Section VIII, in a federation of n heterogeneous resources or nodes, on average a job requires $\Theta(\log n)$ messages to be sent in the network in order to locate a node that can successfully complete the job without being overloaded. In our simulation, the total number of messages produced for all the jobs (refer to Fig. 9(b))

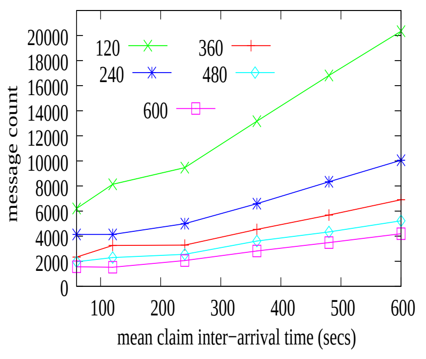


(a) average claim inter-arrival delay (secs) vs average routing hops.

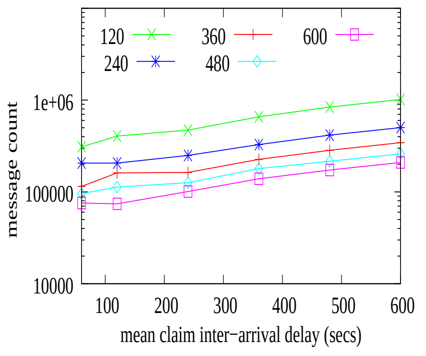


(b) average claim inter-arrival delay (secs) vs message count.

Fig. 9. Coordination space perspective



(a) mean claim inter-arrival delay (secs) vs message count



(b) average claim inter-arrival delay (secs) vs message count.

Fig. 10. Coordination space perspective

satisfies the theoretical bound on message generation, i.e., $\Theta(m \times n \times \log n)$. This function is ploted with label "theory" in Fig. 9(b)

In Fig. 10(a) and 10(b), we show the message overhead involved with ticket objects. Fig. 10(a) depicts the total number of ticket object posted by all GFAs in the system with increasing claim inter-arrival delay ($\frac{1}{\lambda_c^{in}}$). In Fig. 10(b), we can see that as claim inter-arrival delay increases the number of claim objects issued during the simulation period increases. For $\frac{1}{\lambda_c^{in}} = 100\ seconds$ and $\frac{1}{\lambda_t^{in}} = 120\ seconds$, total 6000 claim object were produced in order to successfully schedule all the jobs in the federation. While for or $\frac{1}{d\lambda_c^{in}} = 600\ seconds$ and with same ticket inter-arrival delay, the GFAs issued 20000 ticket objects. This shows that if jobs are injected at slow rate into the system and the GFAs publish tickets as relatively faster rate, then it leads to generation of large number ticket objects in the system. However, if the GFAs inject ticket at slower rate, i.e., $\frac{1}{\lambda_t^{in}} = 600\ seconds$, then a relatively lesser number of ticket objects are required to be issued in order to satisfy all the jobs in the system. But in this case average coordination delays on per jobs basis increases substantially (see Fig. 7(a)). Fig. 10(b) depicts the total number of messages produced in mapping the ticket objects to Grid peers with varying ticket and claim inter-arrival delays.

X. Conclusion

We presented a DHT based coordination protocol for efficiently managing the load in federated Grid computing systems. A d-dimensional spatial index formed the basis for distributing claim and ticket objects over a structured Grid broker overlay. Our simulation showed that: (i) resource claim and ticket object injection rate has significant influence on the coordination delay experienced by distributed users in the system; (ii) proposed coordination protocol is highly effective in curbing the number of scheduling negotiation iteration required to be undertaken on per job basis, the redemption and notification message complexity is bounded by the function $\Theta(1)$; and (iii) on average the number of messages required to successfully map a job to a resource is bounded by the function $\Theta(log\ n)$.

One limitation with our approach is that the current index can map a resource claim object to at most 2 index cells. In some cases this can lead to generation of unwanted notification messages in the system and may be to an extent sub-optimal load-balancing as well. In our future work, we are going to address this issue by constraining the mapping of a resource claim object to an index cell. Another way to tackle this problem is to make the peers currently managing the same resource claim object communicate with each other before sending the notifications.

References

[1] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems (FGCS) Journal, Volume 18, Issue 8, pages 1061-1074, Elsevier Science, The Netherlands, October*, 2002.

[2] A. O. Allen. *Probability, Statistics and Queuing Theory with computer science applications*. Academic Press, INC., 1978.

[3] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. OurGrid: An approach to easily assemble grids with equitable resource sharing. In *JSSPP'03: Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*. LNCS, Springer, Berlin/Heidelberg, Germany, 2003.

[4] A. Auyoung, B. Chun, A. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *OASIS '04: 1st Workshop on Operating System and Architectural Support for the On-demand IT Infrastructure, Boston, MA, October*, 2004.

[5] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-based load management in federated distributed systems. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 15–15. USENIX Association, Berkeley, CA, USA, 2004.

[6] B. Bode, D. Halstead, R. Kendall, and D. Jackson. PBS: The portable batch scheduler and the maui scheduler on linux clusters. *Proceedings of the 4th Linux Showcase and Conference, Atlanta, GA, USENIX Press, Berkley, CA, October*, 2000.

[7] A. Raza Butt, R. Zhang, and Y. C. Hu. A self-organizng flock of condors. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, Los Alamitos, CA, USA, 2003.

[8] R. Buyya and 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 (CCPE), Volume 14, Issue 13-15, pages 1175-1220, Wiley Press*, 2002.

[9] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01), 2001*, pages 237 – 246. IEEE Computer Society, Los Alamitos, CA, USA, 2001.

[10] Yun Fu, Jeffrey Chase, Brent Chun, Stephen Schwab, and Amin Vahdat. SHARP: an architecture for secure resource peering. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing, NY, USA*, pages 133–148. ACM Press, New York, NY, USA, 2003.

[11] P. Garca, C. Pairot, R. Mondjar, J. Pujol, H. Tejedor, and R. Rallo. Planetsim: A new overlay network simulation framework. In *Software Engineering and Middleware, SEM 2004, Linz, Austria*, pages 123–137. Lecture Notes in Computer Science (LNCS), Springer, Germany, 2005.

[12] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems, Volume 7, Issue 1, pages 80–112, ACM Press, New York, NY, USA*, 1985.

[13] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *CCGRID'01: 1st IEEE International Symposium on Cluster Computing and the Grid, Brisbane, Australia*, page 35. IEEE Computer Society, Los Alamitos, CA, USA, 2001.

[14] L. Gong. JXTA: a network programming environment. *IEEE Internet Computing, Volume 05, Issue 3, pages 88–95, IEEE Computer Society, Los Alamitos, CA, USA*, 2001.

[15] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing networked resources with brokered leases. In *2006 USENIX Annual Technical Conference, Boston, MA, USA*, pages 199–212.

[16] K. Lai, B. A. Huberman, and L. Fine. Tycoon: A distributed market-based resource allocation system. *Technical Report, HP Labs*, 2004.

[17] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing, Volume 63, Issue 11, pages 1105–1122, Academic Press, Inc., Orlando, FL, USA*, 2003.

[18] S. W. McLaughry and P. Wycko. T spaces: The next wave. In *HICSS '99: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8*, page 8037. IEEE Computer Society, Los Alamitos, CA, USA, 1999.

[19] R. Ranjan, A. Harwood, and R. Buyya. A case for cooperative and incentive based coupling of distributed clusters. *Future Generation Computer Systems, In Press, Accepted Manuscript, Elsevier Science, The Netherlands*, Available online 15 June 2007.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware'01: Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–359. SpringerLink, Heidelberg, Germany, 2001.

[21] H. Shan, L. Oliker, and R. Biswas. Job superscheduler architecture and performance in computational grid environments. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, pages 44–51. IEEE Computer Society, Los Alamitos, CA, USA, 2003.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, California, USA*, pages 149–160. ACM Press, New York, NY, USA, 2001.

[23] E. Tanin, A. Harwood, and H. Samet. A distributed quad-tree index for peer-to-peer settings,. In *ICDE'05: Proceedings of the International Conference on Data Engineering*, pages 254–255. IEEE Computer Society, Los Alamitos, CA, USA, 2005.

[24] E. Tanin, A. Harwood, and H. Samet. Using a distributed quadtree index in peer-to-peer networks. *VLDB Journal, Volume 16, Issue 2, pages 165–178*, 2007.

[25] R. Tolksdorf and D. Glaubitz. Coordinating web-based systems with documents in xmlspaces. In *CooplS '01: Proceedings of the 9th International Conference on Cooperative Information Systems*, pages 356–370. Springer-Verlag, London, UK, 2001.