Peer-to-Peer Tuple Space: A Novel Protocol for Coordinated Resource Provisioning

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

August 14, 2007

Abstract

Resource brokering services are the main components that control the way applications are scheduled, managed and allocated in a decentralised, heterogeneous and dynamic Grid computing environment. Existing Grid computing systems such as a resource broker, e-Science application work-flow scheduler operate in tandem but still lack a coordination process that can lead to efficient application schedule across distributed resources. Lack of coordination exacerbates the utilisation of various resources including computing cycles and network bandwidth.

To overcome these shortcomings, we propose a mechanism to realise a decentralised coordination process among grid application schedulers. Our dencentralised coordination process is based on a Peer-to-Peer (P2P) resource discovery system. Resource discovery system utilises the publish/subscribe model to convey coordination message among the participants. In the proposed scheme, all application schedulers and resource providers facilitate the decentralised coordination process through exchange of resource usage and application requirement information.

1 Introduction

Several research projects including Bellagio [3], Tycoon [11], NASA-Scheduler [22], OurGrid [2], Sharp [5], Condor-Flock [5] and Grid-Federation [18] have proposed federated sharing of topologically distributed networked computing resources to facilitate a cooperative and coordinated sharing environment. In a federated resource sharing environment, every participant gets access to a larger pool of resources and resource providers get economic or bartering benefits depending upon the resource leasing policy. Distributed resource sharing systems including Bellagio and Tycoon have been deployed and tested over PlanetLab environment, while the Grid-Federation, NASA-Scheduler, Condor-Flock and OurGrid are targeted towards computational grid environments.

However, the effectiveness of the federated resource sharing environments can not be optimally achieved without a proper coordination mechanism between the schedulers (resource brokers in case of grids and slice initiators in case of PlanetLab). The coordination mechanisms in NASA-Scheduler, OurGrid, and Condor-Flock P2P are based on general broadcast and limited broadcast communication mechanisms respectively. Hence, these approaches have the following limitations: (i) high network overhead; and (ii) scalability problems. Resource allocation coordination in Tycoon is based on decentralised isolated auction mechanism. Every resource owner in the system runs its own auction on behalf of his local resources. In this case, a scheduler might end-up bidding across a large number of auctions. On the other hand, resource allocation in Bellagio system is based on the bid-based proportional resource sharing model. Bids for resources are periodically cleared by a centralized auction coordinator. Clearly, the coordination mechanisms followed by Bellagio and Tycoon are neither efficient nor scalable. Sharp architecture coordinates resource allocation among various competing schedulers through pair-wise peering arrangement. For example, site A may grant

to site B a claim on its local resources in exchange for a claim that enables access to B resources. This pair-wise approach may work well for a small system size, but can prove to be serious bottleneck as the system scales.

One of the possible ways to solve this problem is to host a coordinator service on a centralised machine [10, 15, 25]. Every application scheduler 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.

Another possible way to tackle this problem is to distribute the role of the centralised coordinator among a set of machines based on a P2P network model. New generation P2P routing substrate such as DHTs [23, 20] can be utilised for efficiently managing such decentralised coordination network. DHTs have been proven to be self-organising, fault-tolerant and scalable.

We advocate organising Grid schedulers (and users in case of PlanetLab) and Grid resources based on a DHT overlay. Application schedulers post their resource demands by injecting a *Resource Claim* object into the decentralised coordination space, while resource providers update the resource supply by injecting a *Resource Ticket* object (similar terminologies have been used by Sharp 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 3. Decentralised coordination space is managed by a software service (a component of Grid peer service) called coordination service. It undertakes activities related to decentralised load-distribution, 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 distributed spatial hashing technique. 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 a set machines instead of delegating it to one. 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.

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 peer 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 in place of the 1-dimensional DHT key space. Spatial indices such as Space Filling Curves (SFC) [21], k-d Tree [8], R-Tree [13], MX-CIF quadtree [24] can be utilised for managing such complex objects over a DHT key space.

In this work, we utilise the P2P publish/subscribe based Grid resource discovery system, described in the paper [19], for managing and indexing the resource claim and resource ticket objects. Decentralised resource discovery system utilises a d-dimensional spatial index to maintain complex Grid resource look-up (resource claim) and update queries (resource ticket). More details on the spatial index can be found in the paper [19] and how we utilise it for distributed load-distribution and coordination among application schedulers can be found in Section 3.3.

The rest of paper is organised as follows: in Section 2, we present the background information on shared-spaces based coordinated communication. Section 3 discusses the P2P tuple space model that we propose in this paper. In Section 3.2, we present details on the Grid resource sharing model that is utilised as the case study for coordinated resource provisioning. In Section 4 and 5, we present the finer details on the application scheduling and resource provisioning algorithms. Section 6 presents the P2P network simulation model that we utilise for evaluating the performance of Grid resource discovery system. In Section 7, we present various experiments and discuss our results. We end this paper with concluding remarks in Section 8.

2 Background and State of the Art

2.1 Shared-space Based Coordinated Communication

The idea of implementing globally accessible "data-space" or "coordination-space" for communication between distributed services goes back to the *blackboard systems* proposed by the Artificial Intelligence research community in early 1970s. The blackboard system was utilised as a global slate by experts to collaborate on solving the difficult problems. Experts would search the blackboard for problems of their expertise and post the solutions. The idea of global slate was implemented in the systems including JavaSpaces [16], TSpaces [15] and XMLSpaces [25]. These implementations were based on the centralised CS-model which has limited scalability. Initially, the slates were utilised for coordinating parallel application execution between a cluster of computers. Traditionally, these blackboard systems supported $Read()$ and $Write()$ primitive for information coordination between services.

The shared-space based coordination approach or model was proposed by Linda [10] system, which defined a centralised tuple space that provided abstraction of a shared message store for supporting generative communication. Linda defines a tuple as an ordered sequence of typed fields and a tuple space as a shared repository that includes a set of tuples which can be accessed by several distributed processes synchronously. Linda system also defines separate tuple access primitives for reading, writing and destroying. Tuples are written to the shared space through execution of $out(t)$ primitive, read using the non-destructive primitive $rd(\overline{t})$, and extracted using the destructive primitive $in(\overline{t})$.

2.2 State of the Art

In recent times, there have been proposals for organising a coordination space based on a decentralised network model, the representative systems being Lime [17], PeerWare [7], PeerSpace [4] and Comet [12]. Systems including Lime and PeerWare support a global coordination space using a distributed index called Global Virtual Data Structure (GVDS). The focus of Lime system is to provide coordination among participants in mobile environments. The global data space is built by combining the local data spaces of participating peers. The changes made in the local data space are reflected in the global data space. The data structure managed by PeerWare is organised as a graph composed of nodes and documents which are collectively referred to as items. Every peer in the system maintains a local graph structure, which are superimposed on each other to form the GVDS. The management of such a global data structure in a highly dynamic and large distributed system is not scalable.

The most related state of the art to this research is Comet System, that utilises DHTs as the basis for organising the GVDS. The advantage of utilising the DHT is that updates, inserts and deletes on the local tuples (keys) are not required to be communicated to the global tuple space. The changes to the tuple space due to these operations (insert, delete, and update) are apparently handled by the logical mapping structure that forms the basis for tuple space management. Hilbert SFC index, proposed in the work Squid [21], is utilised as the mapping structure from the logical tuple space to the Chord identifier space. In contrast, our mapping structure is based on the spatial publish/subscribe index whose details can be found in paper [19].

3 Peer-to-Peer Tuple Space Model

In this section we first describe the communication, coordination and indexing models which are utilised to facilitate the P2P tuple space. Then we look at the composition of tuples, access primitives that form the basis for coordinating the application schedules among the decentralised and distributed resource brokers.

3.1 Layered Design of the Coordination Space

Fig. 1 shows the layered design of the proposed P2P tuple space based coordination service. The OPeN architecture proposed in the work [24] is utilised as the base model in architecting and implementing the proposed service. 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 including Resource Lookup Query (RLQ) and Resource Update Query (RUQ) to the Core services layer.

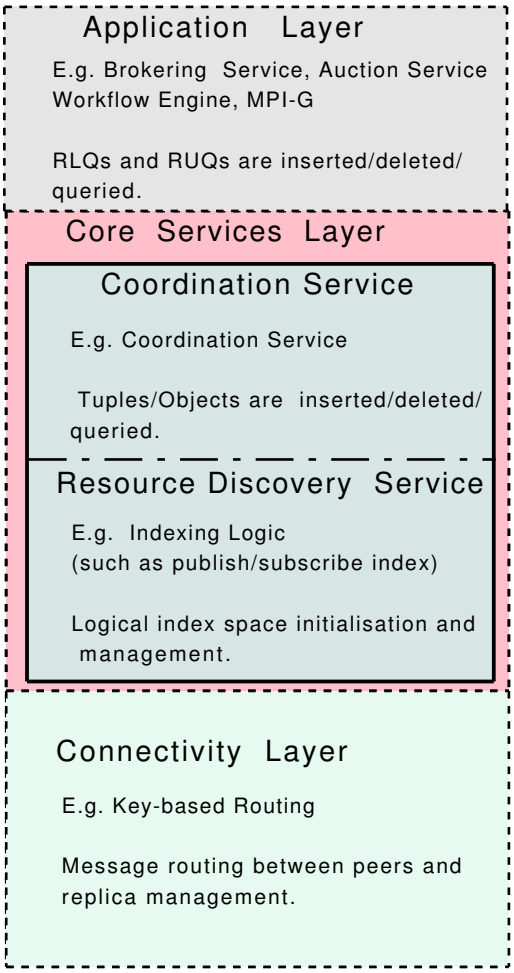


Figure 1: A schematic overview of the Coordination service architecture.

We have implemented the Coordination service as a sub-layer of the Core services layer. The Coordination service accepts the application objects such as RLQs/ RUQs. These objects are then wrapped with some additional logic to form a coordination tuple or object. The coordination logic, in this case the resource provisioning logic, are executed by the Coordination service on these tuples or objects. However, the calls between the Coordination service and Resource discovery service are done 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 details on the workings of the Resource discovery service can be found in the paper [19]. Note that, the proposed tuple space does not strictly follow the standard Linda primitive, instead it exposes the APIs such as $publish(ticket)$, $subscribe(claim)$ and $unsubscribe(claim)$ that suites the requirements of the Application layer brokering service.

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.

3.2 Grid Resource Sharing Model

In this paper, we consider a grid system model that aggregates distributed resource brokering and allocation services [18] as part of a generalised resource sharing environment, which is referred to as the *Grid-Federation*. The grid brokering model aggregates topologically and administratively separated computational grid resources such as clusters, supercomputers, and desktops. Resource brokering, indexing and allocation in the Grid-Federation is facilitated by a new Resource Management System (RMS) known as the Grid Federation Agent (GFA). More details about general Grid-Federation brokering, and the resource owner's local resource allocation services can be found in the articles [18].

In general, a GFA service requires two basic types of queries: (i) an Resource Lookup Query (RLQ), a query issued by a broker service to locate resources matching the user's application requirements; and (ii) an Resource Update Query (RUQ), is an update query sent to a resource discovery service by a Grid site owner about the underlying resource conditions. Since, a Grid resource is identified by more than one attribute, an RLQ or RUQ 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, 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*.

Recall that, compute Grid resources have two types of attributes: (i) static attributes–such as the type of operating system installed, network bandwidth (both Local Area Network (LAN) and Wide Area Network (WAN) interconnection), processor speed and storage capacity (including physical and secondary memory); and (ii) dynamic attributes–such as processor utilization, physical memory utilization, free secondary memory size, current usage price and network bandwidth utilization.

3.3 Coordination Tuples/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

Every GFA in the federation publishes its resource ticket with the local Coordination service. A resource ticket object U_i consists of a resource description R_i, for a cluster i. A R_i can include information about the CPU architecture, number of processors, RAM size, secondary storage size, operating system type, resource usage cost etc. In this work $R_i = (p_i, x_i, \mu_i, \emptyset_i, \rho_i, c_i)$, which includes the number of processors, p_i, processor architecture, x_i, their speed, μ_i, their utilization, ρ_i, installed operating system type, $\emptyset_i$, and a cost c_i for using that resource. A site owner charges c_i per unit time or per unit of million instructions (MI) executed, e.g. per 1000 MI. The ticket publication process can be based on time intervals or resource load triggers. Recall from the paper [19] that a resource ticket object has similar semantics to the RUQ object.

Resource Ticket: Total-Processors= 100 && Processor-Arch= Pentium && Processor-Speed= 2 GHz && Operating-System = Linux && Utilization=0.80 && Acess-Cost=1 Dollar/min.

Resource Claim

A resource claim object encapsulates the resource configuration needs of a user's job. In this work, we focus on the job types whose needs 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. An user job in the Grid-Federation system is written as $J_{i,j,k}$, to represent the i-th job from the j-th user of the k-th resource. A job consists of the number of processors required, $p_{i,j,k}$, processor architecture, $x_{i,j,k}$, the job length, $l_{i,j,k}$ (in terms of instructions), the budget, $b_{i,j,k}$, the deadline or maximum delay, $d_{i,j,k}$ and operating system required, $\emptyset_{i,jk}$. A GFA aggregates these application characteristics including $p_{i,j,k}$, $x_{i,j,k}$, $\emptyset_{i,j,k}$ with constraint on maximum speed, cost and resource

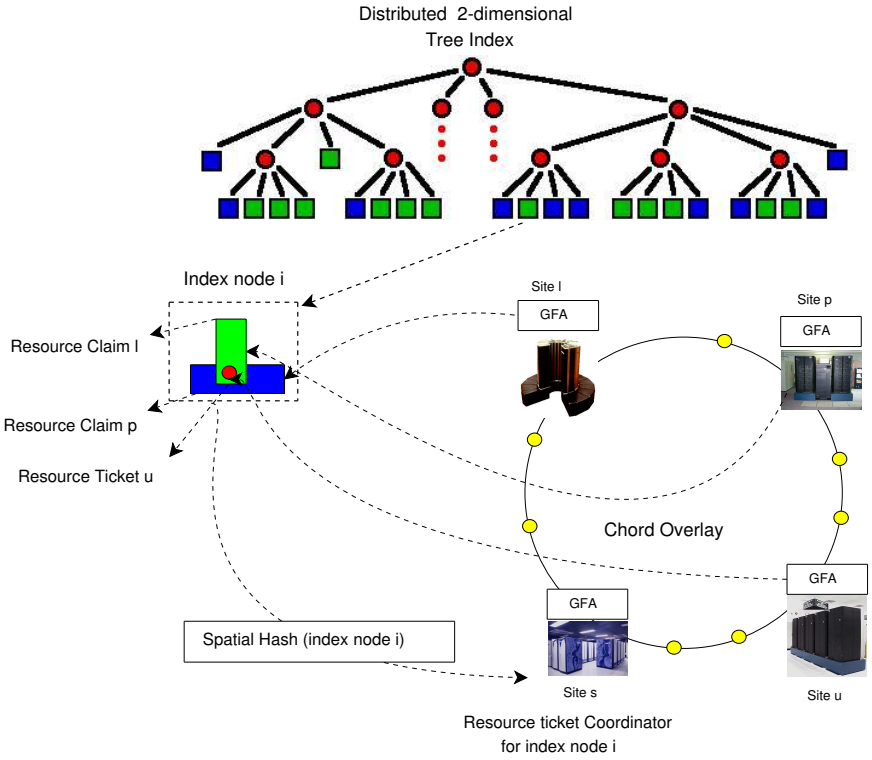


Figure 2: Resource allocation and application scheduling coordination across Grid sites.

utilization into a resource claim object, $r_{i,j,k}$. Recall from the paper [19] that a resource claim object has similar semantics as an RLQ object and is d-dimensional in composition.

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 && 0 Dollar/min* $\leq$ *Access-Cost* $\leq$ *5 Dollar/min .*

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). The details on index cell mapping to the coordination services is described in the paper [19]. In Fig. 2, 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. 2) which falls under the region of 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.

In case 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. Once a scheduler receives notification that its resource claim has matched with an advertised resource ticket, the scheduler undertakes a Service Level Agreement (SLA) contract negotiation with the ticket issuer site. In case agreement is reached, the scheduler can go ahead and deploy its application/experiment. The GFAs have to reply as soon as the SLA enquiry arrives. In other words, we set the SLA timeout interval

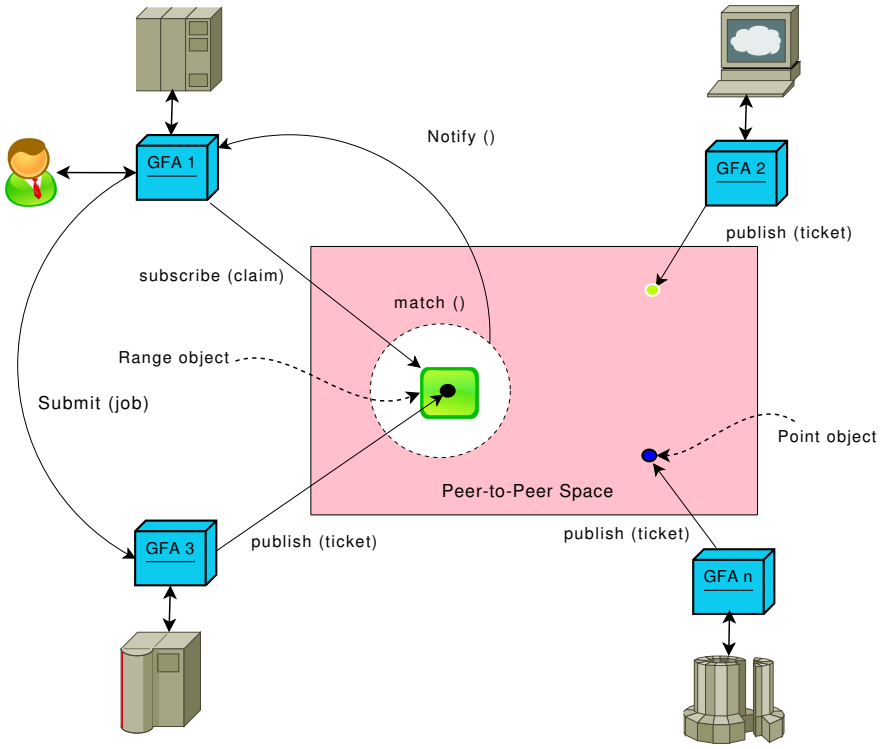


Figure 3: Scheduling and resource provisioning coordination through P2P tuple space.

as 0. We do this in order to study the effectiveness of coordination space with respect to decentralised load-balancing. As excessive timeout interval can lead to deadlock kind of situation in the system, with coordination service sending the notifications while the ticket issuer not accepting SLA contracts. In future we intend to study how does varying degree of SLA timeouts can affect the system performance in terms of load-balancing and provider's economic benefit.

4 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 the arrival of a job:

1. When a job arrives at a GFA, the GFA compiles a resource ticket object for that job. It then posts this resource ticket object with the P2P tuple space though the Core services layer. The complete pseudo code for this process is shown in the Fig. 1. In Fig. 3 GFA 1 is posting a resource claim on behalf of its local user.

2. When a GFA receives a notification for resource ticket and resource claim match from the P2P coordination space, then it undertakes SLA-based negotiation with the ticket issuer GFA. After successful notification, the coordination service unsubscribes the resource claim for that job from the tuple space. In Fig. 3 the match event occurs and GFA 1 is notified that it can place the job with GFA 3. Following this, GFA 1 undertakes SLA negotiation with GFA 3, which is accepted and, finally GFA 1 migrates the locally submitted job to the GFA 3.

3. If SLA negotiation is successful then the GFA sends the job to the remote GFA, otherwise it again posts the resource claim object for that particular job to the coordination space.

```
PROCEDURE: GFA_SCHEDULING
begin
    begin
        Sub-Procedure: Event_User_Job_Submit (Job  J_{i,j,k})
        encapsulate the claim object r_{i,j,k} for job J_{i,j,k}
        call Post_Resource_Claim (r_{i,j,k}).
    end
    begin
        Sub-Procedure: Post_Resource_Claim (Claim r_{i,j,k})
        call subscribe (r_{i,j,k}).
    end
    begin
        Sub-Procedure: Event_Resource_Status_Changed(Resource R_i)
        encapsulate the ticket object U_i for resource R_i
        call publish (U_i).
    end
    begin
        Sub-Procedure: Event_Coordinator_Reply (GFA  gindex_i)
        call SLA_Bid (J_{i,j,k}, gindex_i).
    end
    begin
        Sub-Procedure: SLA_Bid (Job  J_{i,j,k}, GFA  gindex_i)
        Send SLA bid for job J_{i,j,k} to the decentralised coordinator adviced GFA gindex_i.
    end
    begin
        Sub-Procedure: Event_SLA_Bid_Reply (J_{i,j,k})
        if (SLA Contract Accepted) then
            Send the job J_{i,j,k} to accepting GFA.
        end
        else
            call SLA_Bid_Timeout (J_{i,j,k}).
        end
    end
    begin
        Sub-Procedure: SLA_Bid_Timeout(J_{i,j,k})
        call Post_Resource_Claim (r_{i,j,k}) .
    end
end
```

Algorithm 1: SLA-based GFA application scheduling algorithm.

5 Distributed Resource Provisioning Coordination Algorithm

In this section we present the details on the decentralised resource provisioning algorithm which is undertaken by the coordination services across the P2P tuple 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 Fig. 2.

2. When a resource ticket object arrives at a coordination service, the coordination service calls the auxiliary procedure match(ticket) (as shown in Fig. 2) to gather the list of resource claims that overlaps 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).

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 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 tuple space either periodically or whenever the resource condition changes such as a job completion event happens.

6 Simulation Model

In this section, we present simulation model for evaluating the performance of our P2P tuple space with respect to coordinated resource provisioning. The proposed model is applicable to large networks of the scale of the Internet. The simulation model considers the message queuing and processing delays at the intermediate peers in the network. In a centralised system, the index look-up latency is essentially zero, assuming the computation delay due to processing of local indices is negligible. For the P2P system, assuming negligible computation delay for index processing logic at intermediate peers, the time to complete an RLQ or RUQ is time for the query to reach all the cells (including both parent and child cells) that intersect with the query region.

In our message queueing model, a Grid peer node (through its Chord routing service) is connected to an outgoing message queue and an incoming link from the Internet (as shown in Fig. 4). The network messages delivered through the incoming link (effectively coming from other Grid peers in the overlay) are processed as soon as they arrive. Further, the Chord routing service receives messages from the local publish/subscribe index service. 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 [1] queue steady state probabilities.

Our simulation model considers an interconnection network of n Grid peers whose overlay topology can be considered as a graph in which each peer maintains connection to a $O(\log\ n)$ other Grid peers (i.e. the Chord overlay graph). As shown in Fig. 4, every Grid peer is connected to a broker service that initiates lookup and update queries on behalf of the users and site owner. We denote the rates for RLQ and RUQ by λ_l^{in} and λ_u^{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 RLQs and RUQs that map to the control area currently owned by the Grid peer, and the notification messages arriving from the the network.

```
1.1  PROCEDURE: Resource_Provision
1.2  begin
1.3      list ← φ
1.4      begin
1.5          Sub-Procedure: Event_Resource_Claim_Submit (Claim  r_{i,j,k})
1.6          list ← list ∪ r_{i,j,k}.
1.7      end
1.8      begin
1.9          Sub-Procedure: Match (Ticket  U_i)
1.10         list_m ← φ
1.11         set index = 0
1.12         while ( list[index] ≠ null ) do
1.13             if ( Overlap (list[index], U_i) ) then
1.14                 list_m ← list_m ∪ list[index]
1.15             end
1.16             else
1.17                 continue
1.18             end
1.19             reset index = index + 1
1.20         end
1.21         return list_m .
1.22     end
1.23     begin
1.24         Sub-Procedure: Overlap (Claim  r_{i,j,k}, Ticket  U_i)
1.25         if (r_{i,j,k} ∩ U_i ≠ null ) then
1.26             return true.
1.27         end
1.28         else
1.29             return false.
1.30         end
1.31     end
1.32     begin
1.33         Sub-Procedure: Event_Resource_Ticket_Submit (U_i)
1.34         call Load_Dist(U_i, Match(U_i)).
1.35     end
1.36     begin
1.37         Sub-Procedure: Load_Dist (U_i, list_m)
1.38         set index = 0
1.39         while (R_i is not over-provisioned) do
1.40             send notification match event to resource claimer: list_m [index]
1.41             remove(list_m [index])
1.42             reset index = index + 1.
1.43         end
1.44     end
1.45 end
```

Algorithm 2: Resource provisioning algorithm for coordination service.

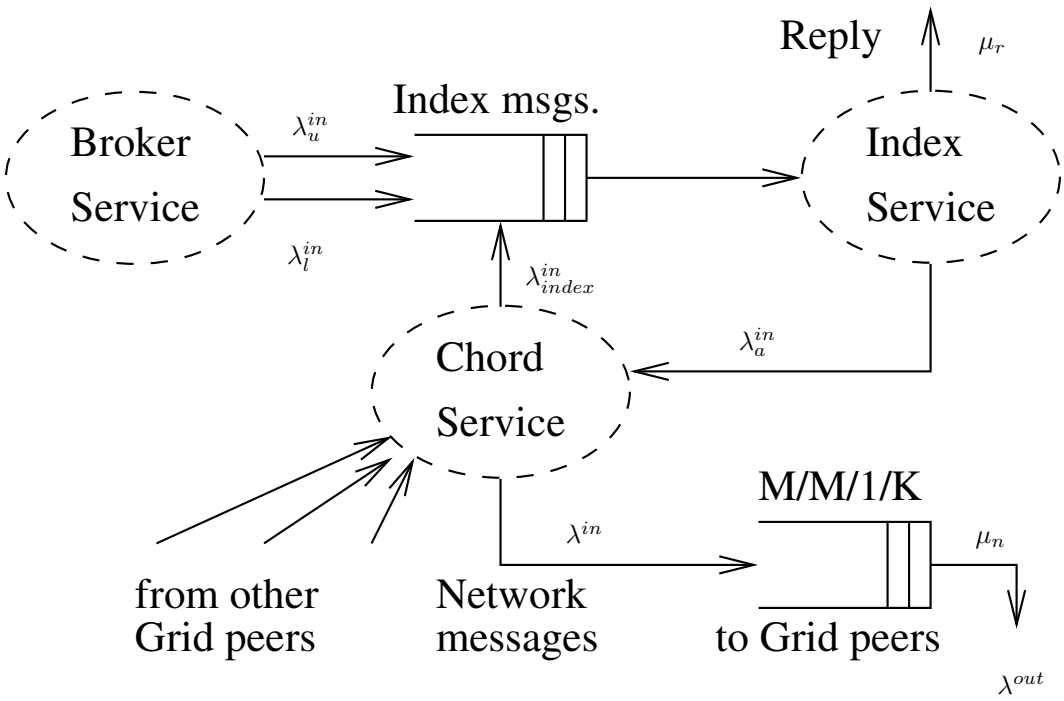


Figure 4: Network message queueing model at a Grid peer i.

7 Performance Evaluation

In this section, we validate the proposed P2P tuple space-based coordinated resource provisioning model through the trace-based simulation. The simulated environment models the Grid-Federation resource sharing environment presented in the paper [18].

7.1 Experimental Setup

We start by describing the test environment setup.

7.1.1 Broker Network Simulation:

Our simulation infrastructure includes two discrete event simulators namely *GridSim* [6], and *PlanetSim* [9]. We model the resource brokering service i.e. a GFA inside the GridSim that injects resource claims and resource tickets on behalf of both, the users and the resource providers respectively. Every GFA connects to Core services layer which also has implementations for Coordination service and publish/subscribe Index service as sub-layers. At the Connectivity layer we utilised the Chord implementation provided with the PlanetSim.

Experiment configuration:

- Network configuration: The experiments ran Chord overlay with 32 bit configuration i.e. number of bits utilised to generate node and key ids. The network size n was fixed at 100 GFA/broker nodes. The network queue message processing rate, μ, at a Grid peer was fixed at 500 messages per second. The message queue size, K, was fixed at 10^4.
- Resource claim and resource ticket injection rate: The GFAs inject resource claim and resource ticket objects based on the exponential inter-arrival time distribution. The value for resource claim inter-

arrival delay ($\frac{1}{\lambda_l^{in}}$) is distributed over the interval $[5, 60]$ in step of 5 secs. While the inter-arrival delay ($\frac{1}{\lambda_u^{in}}$) of resource claim object was fixed to 30 secs. The inter-arrival delay in claim/ticket injection is considered same for all GFAs/brokers in the system. The spatial extent of both resource claims and resource ticket objects lies in a 5-dimensional attribute space. The attribute dimension includes the number of processors, p_i, resource access cost, c_i, processor speed, m_i, processor architecture, x_i, and operating system type, ϕ_i. The distributions for these resource dimensions have been obtained from the Top 500 supercomputer list[1].

Note that, in our simulation we did not utilize resource utilization, ρ_i , as the GFA's load indicator. Instead GFAs encode the metric *"number of available processors"* at time t with the resource ticket object U_i. Specifically, the information on the number of available processor is updated inside the $gindex_i$ object and sent to the coordination service along with ticket object U_i. The coordination service utilizes this metric as the indicator for the current load on the resource R_i. In other words, the coordinator service would stop sending the notifications as the number of processors available with a ticket issuer reaches *zero*.

- Publish/subscribe index configuration: The minimum division, f_{min}, of logical d-dimensional publish/subscribe index was set to 3, while the maximum height of the index tree, f_{max}, was also limited to 3. This means we basically do not allow the partitioning of the P2P tuples space beyond f_{min} level. In this case, a cell at a minimum division level does not undergo any further division. Hence, no resource claim or resource ticket object is stored beyond the f_{min} level. 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. The publish/subscribe Cartesian space had 6 dimensions including number of processors, p_i, resource access cost, c_i, processor speed, m_i, processor architecture, x_i, and operating system type, ϕ_i. Hence, this configuration resulted into 243 (3^5) Grid index cells at the f_{min} level. On an average, 2 index cells are hashed to a Grid peer in a network comprising of 100 Grid sites.

Indexed data distribution: We generated a resource type distribution using the resource configuration obtained from the Top 500 Supercomputer list. We utilised the resource attributes including processor architecture, its number, its speed, and installed an operating system from the Supercomputer list. The value for c_i was fabricated. The values for c_i were uniformly distributed over the interval $[0, 10]$.

Workload configuration: We generated the workload distributions across GFAs based on the model given in the paper [14]. 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.

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. More details on how the execution time for jobs are computed on space shared resource facilities can be found in the paper [18].

7.2 Effect of Job Inter-Arrival Delay: Lightly-Constrained Workloads

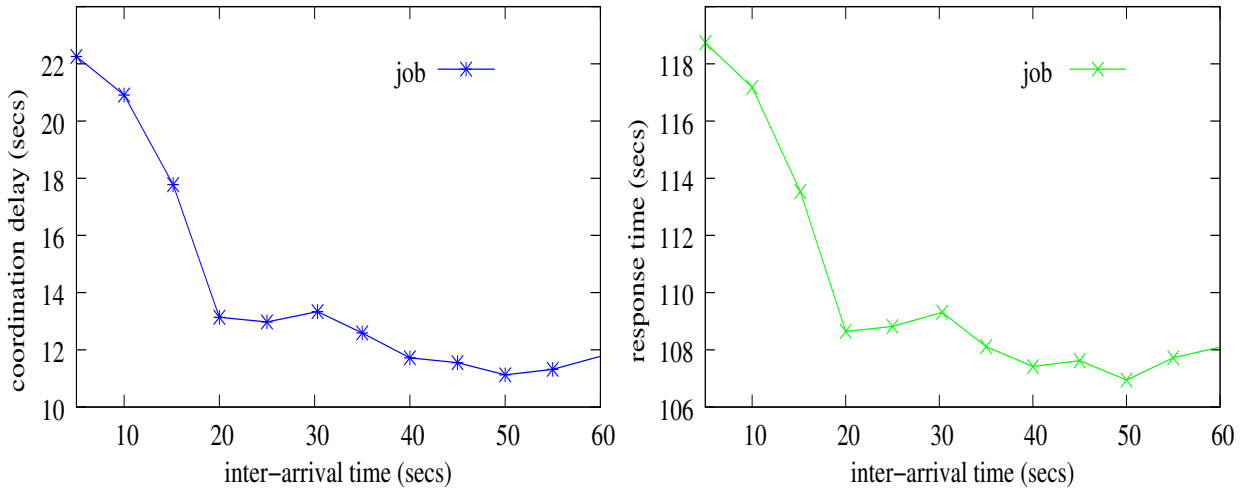
The first set of experiments measured the performance of P2P tuple space in coordinating resource provisioning with respect to the following metrics: average coordination delay, average response time and average

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

processing time for jobs. Further, it also quantifies the details about the job migration statistics in the system i.e. number of jobs executed locally and number jobs executed remotely. In this experiment, the resource claim injection rate is varied from 12 to 1 per minute while the resource ticket injection rate is fixed to 2 per minute. This experiment simulates a lightly-constrained workload or job characteristic. In other words, on an average the simulated jobs did not require large number of processors for execution. For this experiment, the job characteristics were generated by configuring the minimum and maximum processor per job as 2 and 2^6 respectively in the workload model.

Fig. 5 and Fig. 6 show the measurement for parameters coordination delay, response time, processing time and job migration. 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 the 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.

Fig. 5(a) depicts results for the average coordination delay in seconds with increasing job inter-arrival delay. With increase in average job 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 leads to an overall decrease in the coordination delay across the system. The effect of this can also be seen in the response time metric for the jobs (refer to Fig. 5(b)), which is also seen to improve with increase in inter-arrival delays.

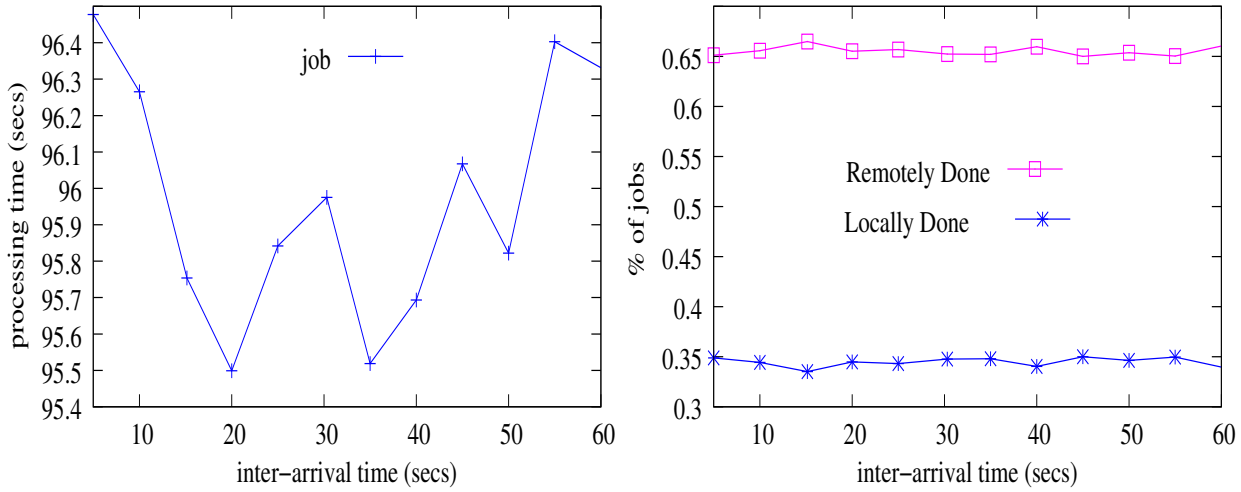


(a) average job inter-arrival delay (secs) vs average coordination delay (secs).

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

Figure 5: Simulation: Effect of job inter-arrival delay: lightly-constrained.

Fig. 6(a) depicts results for the average job processing delay in secs with increasing job inter-arrival delay. As expected, the processing delays do not change significantly with increase in the inter-arrival delay. This is due to the availability of resources with similar or near similar processing capabilities in the Top 500 list. Hence, allocation of jobs to any of the resource does not have significant effect on the overall processing time. Further, the job-migration statistics also showed negligible or very little change with increasing job inter-arrival delays (refer to Fig. 6(b)). At every step, approximately 65% of jobs were executed remotely while remaining executed at the originating site itself.



(a) average job inter-arrival delay (secs) vs processing time (secs).

(b) average job inter-arrival delay (secs) vs % of jobs.

Figure 6: Simulation: Effect of job inter-arrival delay: lightly-constrained.

7.3 Effect of Job Inter-Arrival Delay: Heavily-Constrained Workloads

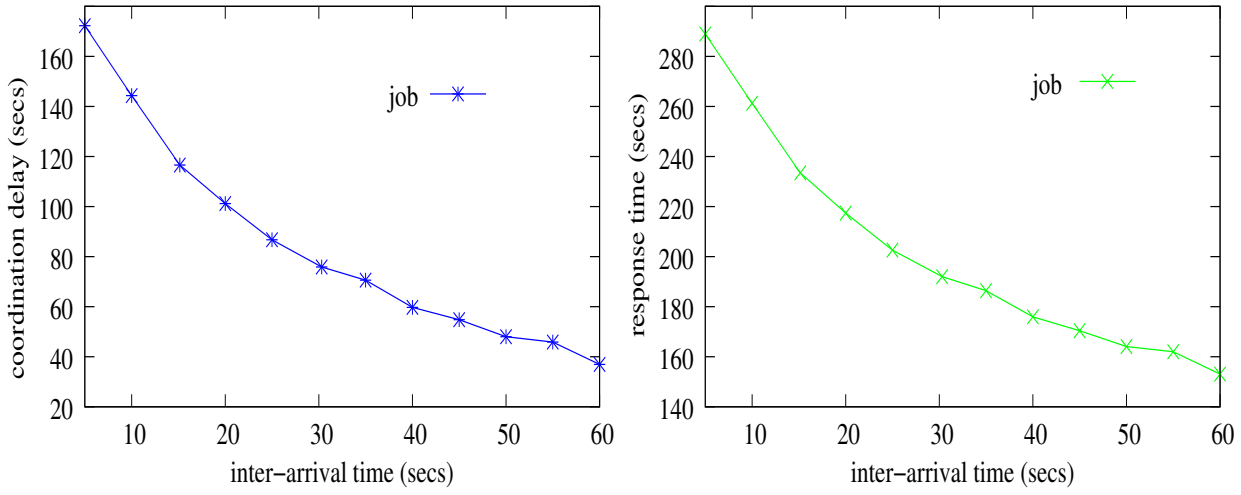
This experiment simulates the performance of P2P tuple space in coordinating resource provisioning for highly-constrained workload or job characteristic. The heavily-constrained workloads on an average require relatively larger number of processors on per job-basis as compared to the lightly-constrained ones. For this experiment, the job characteristics were generated by configuring the minimum and maximum processor per job as 2^6 and 2^8 respectively in the workload model. Other simulation configurations stay the same as described for the previous experiment.

Fig. 7(a) depicts results for the average coordination delay in secs with increasing job inter-arrival delay. With increase in average job inter-arrival delay, we observed noticeable decrease in the average coordination delay. At inter-arrival delay of 5 secs, on the average job experienced a coordination delay of about 172 secs (refer to Fig. 7(a)). And at inter-arrival delay of 50 secs, the coordination delay decreased to 45 secs. The results show that at higher inter-arrival delays, resource claim objects experience less network traffic and competing requests. Although, we saw the same trend in case of lightly-constrained jobs as well, the decrease in case of heavily-constrained jobs is more significant (about 73%). The chief reason behind this being higher degree of competition between resource claim requests, as on the average they required larger number of processors for execution. The effect of diminishing coordination delay can be seen in the response time metric for the jobs as well (refer to Fig. 7(b)), which is also seen to improve with increase in inter-arrival delays.

Similar to lightly-constrained case, we observed that the processing delays (refer to Fig. 8(a)) does not change significantly with increase in inter-arrival delay. Further, the job-migration statistics also showed negligible or very little change with increasing job inter-arrival delays (refer to Fig. 8(b)). At every step, approximately 62% of jobs were executed remotely while remaining executed at the originating site itself.

8 Conclusion

In this paper, we described a P2P tuple space framework for efficiently coordinating resource provisioning in a federated Grid system such as the Grid-Federation. The proposed coordination space built upon the resource discovery system presented in the paper [19]. The simulation based study shows that heavily-constrained workloads can experience significant coordination delays due to the competing requests in the



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

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

Figure 7: Simulation: Effect of job inter-arrival delay: heavily-constrained.

system. However, the same is not true when the workloads are lightly-constrained i.e. the resource claim requests for lesser number of processors.

One limitation with our approach is that the current publish/subscribe 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

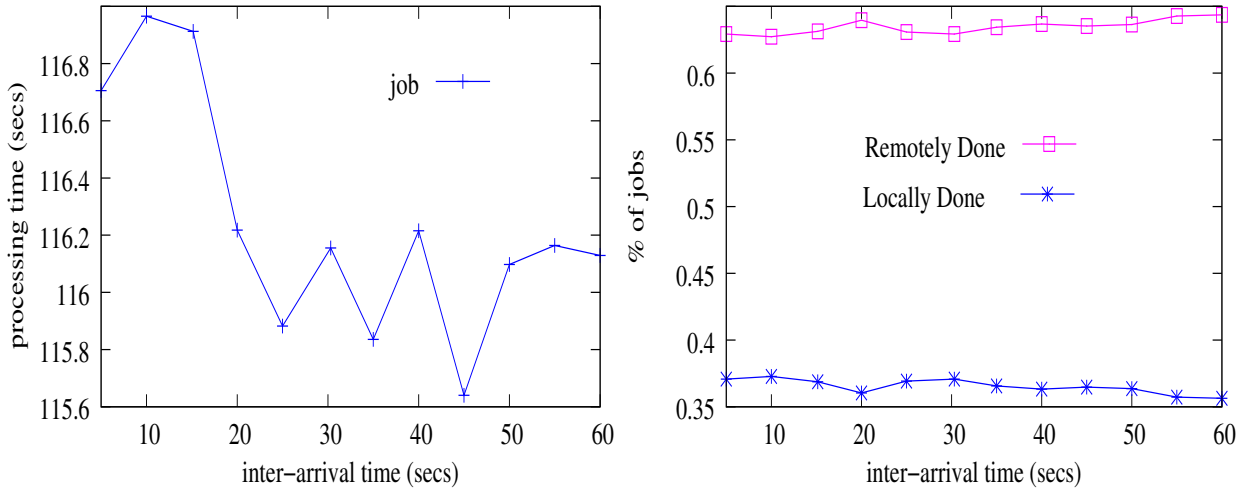
[1] A. O. Allen. *Probability, Statistics and Queuing Theory with computer science applications.* Academic Press, INC., 1978.

[2] 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.

[3] 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 Ondemand IT InfraStructure, Boston, MA, October*, 2004.

[4] N. Busi, C. Manfredini, A. Montresor, and G. Zavattaro. Peerspaces: data-driven coordination in peer-to-peer networks. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing, Melbourne, Florida*, pages 380–386. ACM Press, New York, NY, USA, 2003.

[5] 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.



(a) average job inter-arrival delay (secs) vs processing time (secs).

(b) average job inter-arrival delay (secs) vs % of jobs.

Figure 8: Simulation: Effect of job inter-arrival delay: heavily-constrained.

[6] R. Buyya and M. Murched. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Journal of Concurrency and Computation: Practice and Experience;14(13-15), Pages:1175-1220*, 2002.

[7] G. Cugola and G. Picco. PeerWare: Core middleware support for peer-to-peer and mobile systems. *Technical Report, Politecnico Di Milano*, 2001.

[8] P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule them all: multi-dimensional queries in p2p systems. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, Paris, France*, pages 19–24. ACM Press, New York, NY, USA, 2004.

[9] 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, 2005.

[10] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems, ACM Press, New York, NY, USA*, 7(1):80–112, 1985.

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

[12] Z. Li and M. Parashar. Comet: A scalable coordination space for decentralized distributed environments. In *HOT-P2P '05: Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 104–112. IEEE Computer Society, Los Alamitos, CA, USA, 2005.

[13] B. Liu, W. Lee, and D. L. Lee. Supporting complex multi-dimensional queries in p2p systems. In *ICDCS'05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, Columbus, OH, USA*, pages 155– 164. IEEE Computer Society, Los Alamitos, CA, USA, 2005.

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

[15] 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.

[16] S. MicroSystems. Javaspace specification 2.0, http://www.sun.com/software/jini/specs/js2_0.pdf. *Technical Report*, 2003.

[17] A. L. Murphy, G. P. Picco, and G. Roman. LIME: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering Methodology, ACM Press, New York, NY, USA*, 15(3):279–328, 2006.

[18] R. Ranjan, R. Buyya, and A. Harwood. A case for cooperative and incentive based coupling of distributed clusters. In *Cluster'05: Proceedings of the 7th IEEE International Conference on Cluster Computing, Boston, MA, USA*. IEEE Computer Society, Los Alamitos, CA, USA, 2005.

[19] R. Ranjan, L. Chan, A. Harwood, R. Buyya, and S. Karunasekera. A scalable, robust, and decentralised resource discovery service for large scale federated grids. Technical Report GRIDS-TR-2007-6, Grids Laboratory, CSSE Department, The University of Melbourne, Australia, 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] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *HPDC'12: In the Twelfth International Symposium on High Performance Distributed Computing, June.* IEEE Computer Society, Los Alamitos, CA, USA, 2003.

[22] 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*, page 44. IEEE Computer Society, Los Alamitos, CA, USA, 2003.

[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: 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.

[24] E. Tanin, A. Harwood, and H. Samet. A distributed quadtree index for peer-to-peer settings,. In *ICDE'05: Proceedings of the International Conference on Data Engineering*, pages 254–255, 2005.

[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.