# P2P Based Service Provisioning on Distributed Resources

Sujoy Mistry[1], Dibyanshu Jaiswal[2], Arijit Mukherjee[2] and Nandini Mukherjee[3]
[1]School Of Mobile Computing and Communication, Jadavpur University, Kolkata- 700032, India
[2]TCS Research and Innovation, Tata Consultancy Services, Kolkata-700091, India
[3]Department Of Computer Science and Engineering, Jadavpur University,Kolkata-700032, India

Dynamic or demand-driven service deployment in a Grid environment is an important issue considering the varying nature of demand. Most distributed frameworks either offer static service deployment which results in resource allocation problems, or, are job-based where for each invocation, the job along with the data has to be transferred for remote execution resulting in increased communication cost. An alternative approach is dynamic demand-driven provisioning of services as proposed in earlier literature, but the proposed methods fail to account for the volatility of resources in a Grid environment. In this paper, we propose a unique peer-to-peer based approach for dynamic service provisioning which incorporates a Bit-Torrent like protocol for provisioning the service on a remote node. Being built around a P2P model, the proposed framework caters to resource volatility and also incurs lower provisioning cost.

Keywords: Grid Computing, SOA, dynamic deployment, Bit-Torrent.

## 1. INTRODUCTION

During the recent years, several architectural styles, like client-server model, 3-Tier architecture, n-Tier architecture and peer-to-peer computing have emerged to support computational models. Alongside, Grid Foster [2002] (and recently Cloud) evolved with an aim at resource sharing and problem solving in dynamic and controlled environment with sets of resource providers and resource consumers. The size of resource providers in such distributed systems vary from a few nodes to a large number of heterogeneous nodes distributed over geographic boundaries to form *virtual organizations*. In many small organizations with limited computational resources, significant cost saving can be achieved by opting for third party computational resources instead of investing in dedicated resources. Hence computationally expensive jobs are executed on dynamically acquired third party resources, which could be leased from a large organization. Such a distributed computing model has led distributed computing to a next level where distributed applications and platforms are comprised of heterogeneous resources and services.

Due to change in architecture, the applications have to be re-designed to be composed of small software components, communicating and executing among the different nodes over the network, to achieve a desired goal. These software components have been called *web services* or simply *services*. To make such services readily available to the users, some mechanisms for service discovery and deployment over the Internet on top of physically distributed set of resources have become necessary. The service discovery and deployment need to match the service requirements as well as user needs and therefore have become a major challenge these days.

Traditionally, web services are hosted on fixed web servers and services are registered and made available to serve requests from the service consumers. In such situations, efficient service provisioning entirely depends on the capability of the web server and consumer requests can be satisfied only if the web server has sufficient resources to do so (that is web server is not overloaded). On the other hand, if the services are not utilized at some point of time, web

servers remain under utilized. In order to overcome these bottlenecks, dynamic web service provisioning has been proposed by the researchers Watson et al. [2006], Mukherjee and Watson [2012]. A three-tier architecture is proposed in these research works, where web service providers are separated from the computational resource providers. Computational resource providers are termed as *Host Providers*. However, in most of these works, service registration is centralized and often a client-server model is used which actually adds more bottleneck than actually reducing it.

In order to overcome the above problem, we propose a more distributed framework with an essence of sharing of data and computational resources (also called nodes) by collaboration and communication among each other. The framework is based on a P2P system and unlike the previous research works, it uses a decentralized service registration, resource discovery (here *Host Providers* are the resources on which services are deployed dynamically) and dynamic service deployment over a P2P network. Over time, P2P-based systems for services discovery and deployment have evolved utilizing concepts like DHT Mondejar et al. [2006], SOAP Curbera et al. [2002], UDDI Walsh [2002] etc. But dynamic service discovery and deployment over a geographically distributed area and resource volatility have not been considered in many. This work presents the concepts of demand-driven deployment of services, and the implementation of a non-centralized service registry which has been carried out in a distributed environment. Decentralized service registry and resource discovery resolves the scalability issue for handling a large number of consumer requests. Since a P2P system is devoid of any centralized resources and is adaptable to ad-hoc nature of volatile resources it can overcome the bottlenecks of centralized systems Tan [2009].

Thus, the novelty of this work is the implementation of a framework which enables dynamic web service provisioning in a P2P-based distributed environment. This paper first introduces a formal description of the proposed framework and provides a functional overview of it.

One major contribution of this work is the use of decentralized service registry for efficient catering of consumer requests and deploying requested services in timely manner. The paper describes the service registration and service discovery mechanisms used in the framework and also presents a comparative study of different strategies for scheduling the consumer requests to the *Host Providers* on the basis of information gathered from them.

In many cases, a service deployment operation may require additional data to be accompanied which increases the service package size, and hence incurs a considerable deployment cost. In this context, fast P2P file sharing techniques (such as BitTorrent protocol) can act as a value addition to the framework. Use of such protocol increases the speed of the download, decreases the possibility of download failure and increases the availability of the files and services. The other significant contribution of the work is incorporation of BitTorrent protocol in the P2P-based framework by making use of such a file sharing technique to download service packages from the existing deployments to cater to new deployments.

Rest of the paper is organized as follows. Section 2 presents the related work. A formal description and a functional overview of the proposed architecture is given in Section 3. The architecture of the extended framework with chord decentralized registry features is given in Section 4 and Section 5 discusses how dynamic deployment of web services manages to cope up with the P2P technology for this architecture. The implementation and experimental results for dynamic deployment are presented in Section 6 and Section 7 respectively. Finally Section 8 concludes with a discussion of future scope of the proposed architecture.

## 2. RELATED WORK

Grid Computing has made it possible for users to execute computationally expensive applications on dynamically acquired distributed resources. Users are allowed to combine data and analysis components distributed over the globe to build new complex applications. Virtual Organizations built over Grids allow collaborative sharing of computational and data resources over a wide area

network. To support this, several tools, such as Globus Foster [2005], Condor Tannenbaum et al. [2010] and SunGridEngine Gentzsch [2001] allow the construction of distributed applications over grid-based resources. Researchers have introduced many different architectural styles and standards, such as SOA, Web Services, REST Feng et al. [2009] and more recently Cloud Computing over the past decade to ease the cost of discovery, deployment and maintenance, with varying degrees of success.

Integrating Peer-to-Peer (P2P) based systems with frameworks based on Service Oriented Architecture (SOA) and Web Services Newcomer and Lomow [2005] is emerging as a powerful technology for different industry standard applications, specifically in the context of Grid computing. Among different architectural styles of Service Orientation, the Web Service model is a popular form. Large-scale Grid Computing environments use different standard mechanisms like the Open Grid Services Architecture (OGSA) Foster et al. [2001] and the Web Service Resource Framework (WSRF) for creating Virtual Organizations (VO) Foster [2002] meant for secure resource sharing among several users. Our interest grows from the point of view of trying to take advantage from the dynamic P2P network within a WS-based Grid Computing framework. Universal Description Discovery and Integration (UDDI) was originally proposed by the W3C Brooks [2010] as a standard for Web Service publishing and discovery where services will be advertised by the publishers to facilitate discovery and consumption by service users. But the static UDDI model has certain limitations related to service metadata and dynamic resources. Several recent frameworks for service discovery and deployment based on P2P technology can support scalability, load balancing and fault-tolerance. The P2P systems like CAN Ratnasamy et al. [2001], Chord Stoica et al. [2003], Pastry Rowstron and Druschel [2001], Tapestry Rowstron and Druschel [2001] use distributed hash tables (DHT) as their basic component to create the Peer-to-peer network.

Dynamic deployment of services is considered with utmost importance in Grid frameworks to allow services to be deployed on the fly on available resources. This can be compared to job-oriented frameworks as in Condor, where jobs are submitted to a Condor master, which schedules the actual execution on one or more suitable resources. One advantage of dynamic service deployment over a job-based framework is that once the service is deployed, the deployed cost can be shared over many invocations of the service till the service is explicitly removed, whereas, in case of jobs, once the execution is over, it is removed from the Condor queue, and each subsequent execution requires the execution code and data to be resubmitted to the cluster.

In this context, certain frameworks such as DynaSOAr Watson et al. [2006], WSPeer Harrison and Taylor [2005], HAND Qi et al. [2007] provide some good solutions to handle dynamic deployments of services. Specifically DynaSOAr provides a framework for Dynamic Web Service provisioning in Internet. DynaSOAr is a service based approach to grid computing where instead of jobs, services are hosted and deployed dynamically on available resources, if no existing deployments exist. It involves a Web Service Provider who offers services to the consumer, and deploys them dynamically on Host Providers. A host provider offers resources to the services. The major advantage of this framework lies in the reusability of the services to serve subsequent consumer requests via single deployment and has been successfully used in Mukherjee and Watson [2006] and later more comprehensively in Mukherjee and Watson [2012]. However, the framework evolves around a static centralized UDDI registry and hence is unable to adapt to a volatile grid framework where the resources are not constant. Further, a centralized registry gives rise to bottlenecks while dealing with large service deployments and handling huge number of service requests.

The other two frameworks, i.e. both WSPeer and HAND support dynamic web service deployment. However, these frameworks have differences in their implementation. In case of WSPeer, one implementation is based on UDDI which uses a centralized registry similar to DynaSOAr. The other implementation is P2PS-based Wang [2003], which forms a tree of interfaces where peers are communicating via abstract channels called pipe. This architecture basically facilitates

service requests mainly on the basis of direct communication between the peers via pipes, ignoring the service deployment. On the other hand, HAND uses GTV4 for dynamic service provisioning, where HAND-C provides container-level dynamic deployment, i.e. during dynamic deployment the whole container is redeployed. Alternatively, HAND-S provides service deployment, where instead of whole container only the required service needs to be deployed.

Although these frameworks support dynamic web service provisioning but none of this framework offers full dynamism over a volatile set of resources.

The advantages of Peer-to-peer networks have often been tried to be leveraged in Grid and distributed computing. In Verbeke et al. [2002], the "JNGI" framework was introduced for large-scale distributed computation and was based on the hybrid P2P network JXTA. The model proposed in this comprised of separate levels of peer groups, such as monitors, task dispatchers and workers, which has limited similarity with the work proposed in this paper. CompuP2P Gupta et al. [2006] was a highly appreciated research work and proposed a marketplace for resource sharing and computation using P2P as the backbone. However, it was more centred around a marketplace for computational cycles and thus differs from the current proposal. In a similar manner, CoDiP2P Castellà et al. [2009], talks about computational resource sharing over a P2P network, rather than demand-driven service discovery and provisioning. GlobalStat Wu and Du [2005] proposes an approach for statistical calculation within heterogeneous nodes using a semi-P2P structured designed to achieve efficient load-balancing and avoiding performance bottlenecks. DuDE Danielis et al. [2015], on the other hand, distributes the analysis of log files across a distributed system using the concepts of P2P systems.

Unlike the above research works, the current work introduces a *demand-driven and dynamic* P2P-based service provisioning framework with distributed registry and distributed functioning of the Web Service Providers and Host Providers. The work is further improved with the use of BitTorrent protocol which implements a file sharing technique to download service packages from the existing deployments to cater to new deployments.

## 3. PROPOSED FRAMEWORK

In this section a framework for dynamic service provisioning is proposed. The proposed framework acts as the basis of a service-oriented system using P2P as its communication backbone, thus allowing more flexibility and dynamism when compared with previous approaches used for dynamic service deployment in distributed environments.

One of the key features of this architecture is complete segregation of provider of services and provider of resources. Thus, providers of resources (platforms for service execution), i.e. the Host Providers (HPs) are placed in a different layer as compared with the Web Service Providers (WSPs), who provide services to the consumers and take care of all the collaboration with hosts. Consumers are placed in the third layer. As shown in Figure 1, the three layers are described below:

In this three-layer architecture all the nodes act as peers to each other providing P2P based service publication, discovery, deployment and management. Resource discovery and allocation are done in a heterogeneous environment as per resource availability and required performance of the web service. The major goal of this framework that differentiates it from other existing frameworks are as follows-

- ✓ *decentralizing the service registry in a structured manner*
- ✓ *making the registry adaptable with volatile set of resources*
- ✓ *decentralizing the service deployment/execution in the environment*
- ✓ *sharing the deployment cost of a given service incurred by a single WSP among all WSPs.*
- ✓ *making the registry scalable, having definite time bounds for a service query*

The framework provides a new platform for dynamic web service provisioning. One of the main features of this framework is the implicit demand driven nature, i.e. services are deployed

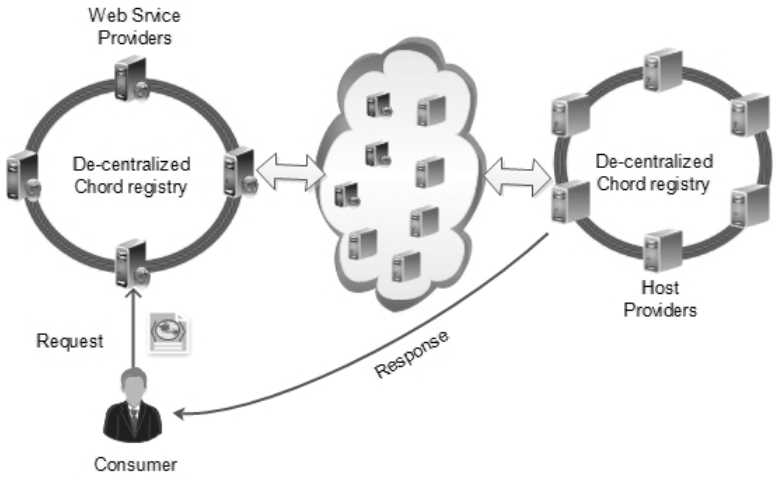


Figure 1. Basic Architecture

on a suitable host exposed by a host/resource provider only when they are required. Similar to a typical web service scenario, when a consumer makes a request for using a certain service, it sends a request in the form of a SOAP message to one of the endpoints exposed by the service provider, which contains service request. This SOAP message may be extended to contain other consumer requirements as well - such as Quality of Service (QoS) requirements. From here the architecture deviates from conventional WS-Framework due to its complete dynamic nature for P2P environment. Thus to define this framework, a *Distributed System (DS)* needs to be established against the existing conventional WS-Framework. Following is a definition for this distributed system which can be denoted using the following tuples-

$$DS = (M, P, WR, HP, NT_m, WS, D_m, MS) \tag{1}$$

where,

| | |
|---|---|
| M = | Map Size, nodes in the system<br>$M = \{M_1, M_2, M_3, ...., M_n\}$ |
| P = | Node Properties |
| $NT_m$ = | A mapping between the nodes and WSPs and HPs according to their properties<br>$NT_m : M \rightarrow P(WR) \wedge M \rightarrow P(HP)$ |
| WR = | Set of Web Service Providers (Chord Ring) (at a particular instant) |
| HP = | Set of Host Providers (at a particular instant) |
| WS = | Set Of Web Services hosted in the Network |
| $D_m$ = | List of nodes where the Web Services are deployed<br>$D_m : WS \rightarrow HP$ |
| MS = | Node Monitoring system |

Each element $M_i$ in $M$ (for $i = 1$ *to* $n$) is defined as
$M_i = \langle N_i, IP_i, L_i \rangle$
where,

| | |
|---|---|
| $N_i$ = | Name of the Node |
| $IP_i$ = | IP Address of the node |
| $L_i$ = | Load of the node (current) |

$$P = (Cl, T, I) \tag{2}$$

where,

Cl = Category of the node
T = Type of the node, i.e, either it is WSP or HP
I = Node's basic resource information

$$WS = (\Sigma, R, S, D_f) \tag{3}$$

where,

$\Sigma$ = Request messages
R = Resources Requirement
S = State of the Service
$D_f$ = List of WSPs who are the owner of the web services
$D_f$: WS $\rightarrow$ WSP

$$MS = (Th, E, Em) \tag{4}$$

Th= $\{N_m, N_c, S_m, BF_n, W_sr\}$
$E_m$ : E $\rightarrow$ Th

where,

E = Set Events
Th = Set of Threads
$N_m$ = NodeMonitor
$N_c$ = NodeCommunicator
$S_m$ = ServiceMonitor
$BF_n$ = BestNodeFinder
$W_sr$ = WebServer

The overall system for dynamic service provisioning in distributed architecture can be defined as above using the tuples 1 to 4. Thus this formal representation denotes each and every functionality of the proposed architecture. First, M (the Map size) denotes the nodes actively present in the system at any particular moment, and changes accordingly as and when nodes join or leave the system. A node joining the system either act as WSP or HP. Thus if we consider that there are $n$ number of nodes actively present in the system, then $M = \{M_1, M_2, M_3, ...., M_n\}$, where $(WR \cup HP) = M$. However, $(WR \cap HP) \neq \phi$, because some nodes may act as both WSP, as well as HP. $NT_m$ defines mapping relationship between each node with respect to its property. Whenever a node joins the system, based on its own property it is decided whether it joins the list of Web Service Providers (implemented as a Chord Ring) or remains in the system only as a Host Provider. Thus the nodes joining the network with a role as WSP (as specified in P) form a ring (via Chord protocol) and share web service information among each other and are denoted as WR. Every WSP maintains a list of nodes where the web services are already deployed ($D_m$). This list is created dynamically after each deployment of the web services to some suitable host providers which is denoted by

A node in the system is defined by its certain properties (P) as defined in *Equation 2*. Node property is broadcast to all other nodes in the system, when a node joins the network. In a heterogeneous system, a node (WSP or HP), is categorised according to its type and resource information. Type defines whether the node is a WSP or HP. Resource information includes processor speed, memory size, operating system installed and other information. Node properties (P) according to the categories are defined in *Equation 3*. This information, though static in nature, is used for various purposes like, to meet the service requirements during service provisioning, load distribution, node collaboration and resource sharing among the nodes in the network.

A newly created web service is made available to a WSP in the system as the owner of the web service. Each web service (WS) (as defined in *Equation (3)*) in the system is accompanied

with information and service criteria like the service name, current state (S) denoting whether it is deployed or not, minimum Resource requirements (R) (in terms of requirements for processor, memory, operating system etc.) for optimum performance. A mapping $D_f$ provides list of owners of the web services from the available WSPs. A web service can have either of the following two statuses:

— **Available:** The service is ready, but has not been deployed yet in the system. In such a case, the consumer can request for the service which will then be deployed on an available resource (HP) for processing the request.

— **Deployed:** If the service has one or more ready deployments in the system, then the service URI of those instances will be provided.

The system maintains a list of nodes where web services are already deployed ($D_m$). This list is changed dynamically and has a strong relationship with basic resource requirements for the services (R), present status of the service (S) and current load ($L_i$) of the nodes. A service can also have more than one deployments in several nodes. Now when a request comes for the service, only the best suited node responds to the service request based on some load or job scheduling strategies. If the service is not deployed on any of the nodes or all the nodes are overloaded, a suitable HP from all the available HPs is selected for deployment on the basis of their current load information collected dynamically during runtime, after a certain interval of time. Consumer requests are all made to the WSP. The host providers only serve as the computational resources on which services can be deployed and requests from consumers can be processed. The WSP accepts the request and finds the most suitable HP to serve it, if the services is already deployed in the system, else a new deployment is triggered for the service. Once the service deployment is complete the consumer requests are served from it unless the HP get loaded and hence new deployments are triggered.

The subsequent operations that take palce once a service request reaches the Service Provider are described through node *Monitoring System (MS)*. The *Monitoring System(MS)* is responsible for monitoring current node and make sure all the supporting threads(Th) are running. Threads in the system have their own functions which are defined based on different modules of the monitoring system, like *NodeMonitor, Node-Communicatior, ServiceMonitor, BestNodeFinder, WebServer*. The *NodeCommunicator* module establishes inter and intra-node communication of *events(E)*. The *ServiceMonitor* component is responsible for publishing the service and making it discoverable based on its status. The *BestNodeFinder* component (only on WSPs) monitors all the nodes on the basis of their properties, runtime CPU and memory utilization to select one HP for new deployments and/or processing consumer requests.

### 3.1 Functional Overview

The framework discussed above is formed by an application running on nodes which facilitates the nodes to join the networks and play their roles as WSP or HP. Though the roles may seem to be completely different from each other, but different aspects of WSPs and HPs are handled using a set of modules such as *Service Monitor*, *Node Communicator*, *Load Balancer*, *Registry Handler*, *Deployment Manager* as depicted in Figure 2. Each module bearing a unique responsibility collaborates among each other to achieve the required goal. Thus the flow of functionality between these modules begin when a consumer makes a request to the WSP for a paticular Web Service. The functionalities of these modules for dynamic web service provisioning are described below.

3.1.1 *Node Joining . Node Communicator* creates the basis for the incomming nodes to join the network and also maintains communication link between each node. A node willing to join the system sends a request to the *Node Communicator* module for processing. Each node has its own node properties and based on that information *Node communicator* decides whether it will join the WSP set or HP set.

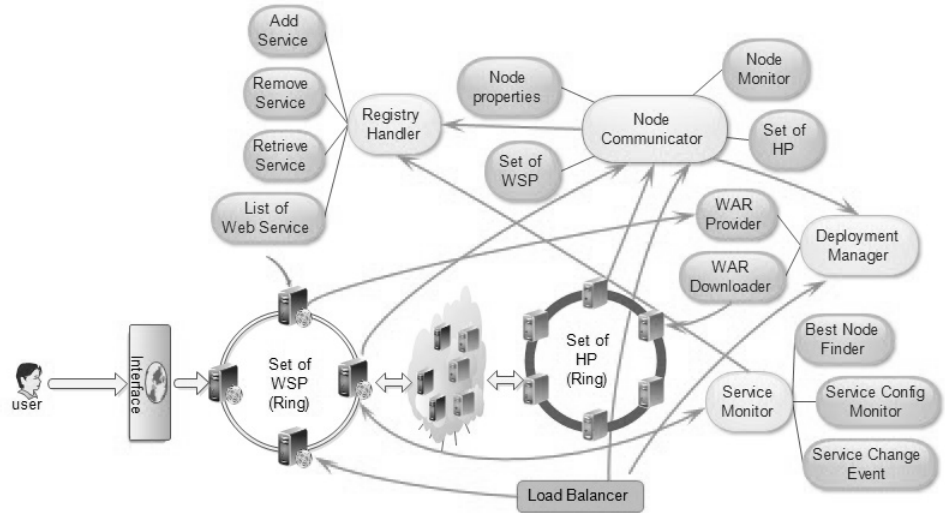


Figure 2. Components in the Architecture

3.1.2 *Web Service Registration .* WSPs willing to provide web services invoke *Registry Handler* module, which collects and maintains service information to publish the services. Based on the status of the web services (i.e, either available or deployed) and in collaboration with the *Service Monitor* module, statuses of all the services are kept updated and maintained in the system.

3.1.3 *First time Service deployments .* Whenever a WSP has one of its web services ready to be published, it invokes a *Registry Handler* to publish it to the registry with the 'available' status, which means it is now available to the world for providing services over the Internet. Thus, when a consumer requests a service for the first time, in such scenario, first time deployment of the web services is required and the status of the web service is changed from *available* to *deployed*. Now *Registry Handler* in collaboration with *Service Monitor* triggers *Service Configuration Change Event*, and manages the service deployments keeping track of the deployed instances. Next, the WSP finds a suitable HP in order to deploy the service. At this moment, *Service Monitor* plays an extra role for finding best node with the help of *Best Node Finder* module. At the HP's end when a deployment event is received, *Deployment Manager* module is triggered to download the web service from the WSP. Once the web service is fully deployed and it is ready to use, it can serve the incoming consumer requests and return the results.

3.1.4 *New deployments when current instances are busy .* When a node gets overloaded, that is when a service has its status "deployed", but it is unable to serve any further requests for the existing deployments, a need for fresh deployment arises. Once again WSP makes a decision based on dynamic load information, collected by the *Load Balancer* with assistance from the *Node Communicator*.

The architecture described above particularly focuses on the following aspects in order to improve the efficiency of the system:

—*Decentralizing the registry*

—*Dynamic deployment*

Following subsections discuss in detail how these functionalities are effectively implemented in our proposed architecture.

## 4. DECENTRALIZING THE REGISTRY

In our previous work Jaiswal et al. [2013] we presented an architecture enabling dynamic on-demand service discovery and deployment based on the concepts of P2P computing where our main focus was on decentralizing the registry, making it discoverable and thus increasing the service availability and reducing the deployment costs by sharing the current deployments and making the services available from more than one endpoints. In this section, we describe the improvements made to our earlier work by implementing decentralization of the registry using Chord.

Since P2P systems are well equipped in handling the volatility of networked resources, a fusion of the two concepts (P2P and SOA) may bring up new possibilities. P2P systems make use of *distributed hash tables* (DHTs)to keep track of the resources provided by the peers in the networks. Considering the web services as resources provided by peers in the system, i.e. WSPs, the service registry can be decentralized. Making use of a structured P2P overlay network with DHT implementation may further facilitate discovery and retrieval of resources within definite time bounds, making the registry scalable. DHT implementations such as CAN, Pastry, Tapestry and Chord can help to achieve the above characteristics for a decentralized registry.

Among the above mentioned DHT implementations, the architecture makes use of Chord due to its easy ring shaped structure achieved by the concept of consistent hashing providing an extra benefit for managing the volatility of peers in the network and the resources they wish to share. **Chord**, a DHT implementation over structured P2P overlay network, is a distributed lookup protocol that helps in efficiently locating a node that stores a particular data item in P2P applications. It can adapt itself with a changing set of resources (nodes) and hence can answer search queries even when nodes join and leave the system. Chord uses consistent hashing of the resources over a ring of *node identifiers*, i.e. unique identifiers of peers in the network. This is achieved by a single operation: given a key it maps the key onto a node identifier. The registry is composed of a DHT of web services stored as [*key, value*] pairs. It uses a hash function to generate unique keys from the byte version of a service name, that are mapped to node identifiers generated as hash value of nodes' IP addresses. Incorporating the benefits of Chord to the existing P2P network, the architecture decentralizes the registry among the WSPs.

### 4.1 Registry using Chord

To support decentralization, the proposed architecture is designed in such a way that at any time a node can join the network acting as a peer specifically depends on the type of node i.e, if it is a WSP then it may or may not carry Web Services to host. In either case it joins the de-centralized registry within the network and shares its own web services as *owners* with other WSP peers. This sharing and de-centralization of the registry is achieved by use of Chord protocol. Since the resources to be shared here are web services, all the DHT entries have its *key* as name of service and the corresponding *value* is set to service metadata, which consists of:

(1) Name of the service
(2) Status of the service (Available /Deployed)
(3) Owner of the service i.e. WSP hosting the service.
(4) List of HPs on which the service is currently deployed.

Each web service is owned by some WSP as identified by the *owner* field in service metadata. Such service metadata helps in identifying the service, its endpoints and other details necessary for proper execution in SOA framework. The use of only web service metadata instead of the entire web service package, improvises the architecture with the following unique benefits:

✓ Enables the security and confines administration of the web service from other WSPs. This may further encourage a business model for exchanging web services.
✓ Increases the availability of the web service, making is accessible from more than one sites.

✓ Keeps the service available even if the *owner* WSP is not available in the network, as the list of current deployed instances is made available in the network.

✓ Enables the *owner* WSP to resume its old state of web services with its existing deployments, when the WSP returns back to the network after some failure.

Once the service is published in the registry, any further changes made to the state of the service such as status of the service and new deployments are all propagated simultaneously to the registry. A closer look towards the implementation can provide a clear picture of how the P2P overlay network is exploited to achieve the above benefits. Adhering to SOA framework, the next section describes implementation and working of the registry.

### 4.2 Service Discovery

All the nodes joining as Web Service Providers creates a P2P network using Chord protocol. If the requested service is provided by any of the service providers on the Chord ring, the information will be known to all other peers, and thus, the request will be forwarded to the corresponding Service Provider node. The provider then selects a suitable Host Provider, which may be a cluster of computational nodes, for processing the request and forwards the message to it. This selection may again depend on the consumer requirements or the status of the available hosts. If the service is not provided on any of the peers, a suitable Service Provider is selected (based upon the criteria such as QoS or provider preference given by the consumer) and the SOAP message is forwarded to the selected node with additional information which points to the location of the deployable code for the requested service. The designated provider again selects a suitable host and forwards the SOAP message in its entirety to it for deployment of the service and execution of the request.
A client can make a service request only when the service endpoint is made available to them via the interface. By default the interface enlists all the services with endpoints maintained locally as a *list* of DHT entries. Depending on the *list* a client request can be made in two ways:

—**Case 1:** If the service is in the *list* then the service endpoint is made directly available to client by which a service request can be made.

—**Case 2:** If the service is not in the *list* then a service query is made to the registry. As a result an endpoint for the same is returned if the service exists.

Chord uses an efficient routing algorithm for locating a key in the ring with an upper bound of O(log N), where N is the number of nodes taking part in the chord ring Stoica et al. [2003], thus even with increase in number of web services and WSPs, the registry remains scalable as compared to previous approaches. The WSPs always use a scheduling strategy to route the consumer requests for adequate resource management based on the dynamic load information collected from the HPs, with deployed instances of the services they own. The scheduling strategies used are time slice based, i.e. an instance among all the deployed instances for a given service is selected as a best node for a given time period. At the end of the time slice the best node is changed as per one of the following scheduling strategies:

— **Round Robin Reloaded (RRR)** - selects the best node in round robin fashion for every time slice, cycling over the deployed instances.

— **Least Recently Used Reloaded (LRUR)** - selects the least recently used instance as the best node if the current instance is loaded, for the next time slice, cycling over the deployed instances.

— **Minimum Loaded First (MLF)** - selects the instance with minimum load as the best node for every time slice among the deployed instances.

Later, in Section 7, the experimental results to demonstrate the efficiency of the framework with decentralized registry is given.

## 5. DYNAMIC DEPLOYMENT

When a service request is received at the WSP, there can be three interaction patterns depending on whether the service was deployed or not:

(1) **First time deployment:** At the initial stage of the processing, for the first consumer request of a given service (i.e. when the status of the service is *available*, the WSP uses some criteria to select the best HP to process the request. This HP will be directed by the WSP to download the corresponding service code from the repository and deploy it. The service status is then changed to *deployed*.

(2) **Fresh deployment:** When the status of a service is *deployed*, but the existing deployment instances are not able to serve the incoming requests (the nodes may be overloaded), the need for a fresh deployment arises. This decision is taken at the WSP based on a set of load information collected from the nodes. The consumer remains unaware of the fact that a new deployment is made, however access to the service is made possible.

(3) **Request for an existing service:** The requests from consumers for services already deployed on multiple nodes are redirected by service providers (WSPs) to the currently selected best node. The selection is made on the basis on some scheduling strategies (like round-robin or least-recently-used).

The architecture adopts two different modes to accomplish download of the service packages that are discussed below:

(1) **Direct Download** : where the HPs download the service package directly from the local repository of the WSP to complete the deployment process.

(2) **P2P Download** : where the HPs download the service package by requesting chunks of the WAR file from more than one site.

In the first approach, it portrays a client-server model where HP (client) needs to download the service package from the WSP (server). In such a scenario a service package may become unavailable if the owner WSP becomes unavailable at that time or in the midst of a deployment. This may again lead to the same old bottleneck of single point failure. Furthermore, it may require longer download times for large service packages resulting in higher response time for the service requests which trigger the successive service deployments. Since the whole architecture rests on the backbone of P2P network, exploiting the benefits of P2P model may prove to be advantageous for the architecture. This is where something like BitTorrent protocol can achieve faster download by proper utilization of the network bandwidth.

### 5.1 Dynamic Deployments with P2P

When a consumer requests a service to the WSP for the first time, WSP selects the best suited HP for the service and deploys the service by downloading the service package to the selected Host Provider. In case of on-demand service provisioning, the time required to download a service package stands to be an important factor for delivering low response times specifically for those requests which correspond to trigger the deployments. From the graphs (Figure 7 and Figure 8), it can be observed that the service requests which lead to trigger new deployments require more time than the average response time. Though the deployment time depends largely on many factors like present network load, size of the service package and efficiency of the WSPs and the HPs; To achieve this on the premises of P2P network, use of Bittorrent protocol seems to be a good solution for reducing the deployment time of a service.

5.1.1 *Rationale behind using BitTorrent.* Traditional *File Transfer Protocol (FTP)* Postel and Reynolds [1985] till now remains a standard for secure and reliable transmission of large files over the Internet. Nevertheless, its highly centralized client-server approach is inadequate for mass publication of files. With a client-server approach, when the number of clients requesting services

from the server increases, the performance of the server deteriorates. On the other hand, peer-to-peer (P2P) is an alternative network model which uses a decentralized model with each peer functioning as a client with its own layer of server functionality. On top of this network model, BitTorrent Pouwelse et al. [2005] is implemented as a peer-to-peer file sharing protocol. It is one of the most popular and successful protocols for transferring large files over the Internet. The protocol takes up a very simple approach by breaking up large files (typically of the order of hundreds of megabytes) into uniform blocks of considerably smaller size, such as 256 kilobytes, and these source components can be dynamically requested from multiple source machines as shown in Figure 3.

In our architecture, BitTorrent technology (implemented by *Node Communicator* $N_c$) can be used to download service packages from the existing deployments, thereby sharing the deployment cost over many nodes. Thus main advantages of P2P file sharing over dynamic web service provisioning can be achieved and it becomes possible to decrease the service failure, increase service availability and also to decrease the time taken for a service deployment.

The P2P approach is capable of removing single point failure for service package downloads. Based on this approach, a second mode of download is implemented using a protocol called ***Dyna****mic* ***Torren****t* ***S****ervice deployment protocol* (DynaTronS).

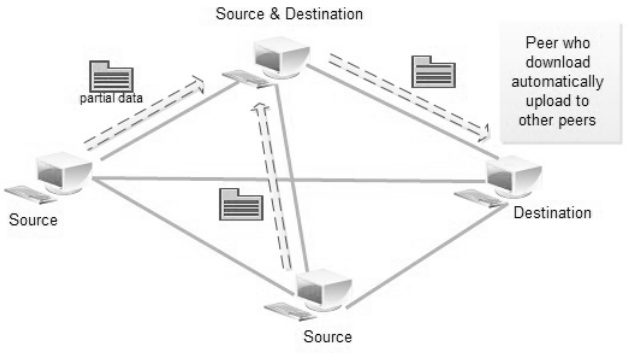


Figure 3. Bit-Torrent Protocal

## 5.2 DynaTronS Deployment Protocol

It is a download protocol specific to our architectural requirements, following the idea of simultaneous download from all the available peers to pull up the download speeds and hence to reduce the download time for the service package. The main aim of DynaTronS approach is to make an efficient use of all the peers in the network who bear the files required to complete a present deployment request. The underlying basic concept tries to download the service package from all the possible locations, i.e. the peers which contain the service package. In our architecture, the service package is available at the local repository of the WSPs as well as the HPs where the web services have already been deployed. After a WSP notifies a HP to carry out the deployment process, it also provides a *seed list*, i.e. a list of peers which contain the full service package. The architecture enables the WSP tier to host the web service, share its information via the registry, but does not allow to share the service package among the WSPs. Therefore, the only places (apart from WSP) where the service package can be found in the architecture are the current deployed instances of the web service in concern. Hence the list of HPs bearing the service deployments along with the WSP who is owner of the service, collectively form the *seed list* for a given web service. With every successive deployments of a given service its corresponding *seed list* is added with new peers. The HP selected to carry out a new deployment of the given web service, downloads the service package in *chunks* simultaneously from the peers enlisted in the *seed list*.

Figure 4 shows a conceptual diagram based on the DynaTronS protocol, where the service package is available at more than one site, i.e. a WSP and few HPs. An HP who needs to download the service package, requests chunks of the service package from all those peers in the network who posses the complete copy of the service package. Thus, the requesting HP makes a full use of the network bandwidth while downloading different parts of the service package concurrently.

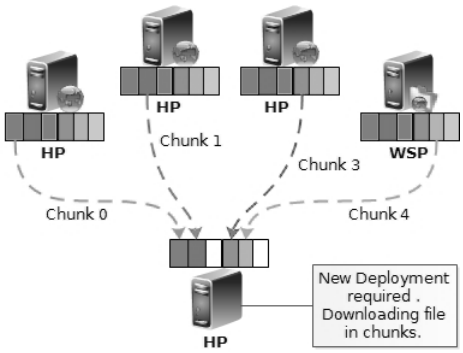


Figure 4. DynaTronS Deployment

A direct comparison of DynaTronS protocol with the bit-torrent protocol may bring up some conceptual differences between them considering the HPs as clients requesting for a service package. Important characteristics for DynaTrons implementation are discussed below:

(1) **No Tracker involved** : In DynaTronS protocol, the WSP itself plays the dual role of providing the torrent file as well as the tracker(as in BitTorrent protocol). The WSPs itself provides the *seed list*, accompanied with the *deployment events*, directly to the HPs.
(2) **First Gather then Share** : DynaTronS adheres to the concept of downloading the service package (in *chunks*) from all possible locations (as in the *seed list*), and then upload the *chunks* of service package (if asked) for next deployment. On the other hand, in case of BitTorrent protocol the client may download, as well as upload at the same time, which may sometime lead to choking of clients in the beginning of the download.
(3) **Single Seed - Simultaneous Download** : For the first time deployment of a web service, the web service package will have only a single peer in *seed list*, i.e. the WSP itself. In such a case if the service package is big, downloading it at one go may be troublesome. Thus DynaTronS protocol downloads the service package in small chunks simultaneously, even if the download is to take place from a single site.

Though the architecture employs DynaTronS protocol for simultaneous download, to achieve higher download speed, it incurs a some extra communication cost for requesting the required chunks of the service package, organizing the chunks as well as monitoring the completion of the download. All these tasks of managing and deploying the service is in the hands of *Deployment Manager* which takes care of downloading the service package.

## 6. IMPLEMENTATION OF SERVICE DOWNLOAD PROTOCOLS

Whenever a WSP requires deployment of a web service, after selection of the suitable HP from the HP tier, it sends a *deployment event* to the selected HP, which in turn initiates the *Deployment Manager* to start the deployment process at the HPs end. *Deployment Manger* then examines the deployment event received to extract the web service credentials such as its *configuration file, URI, length* and *checksum* of the service package. Depending on the deployment mode used by the HP tier, the service package download is carried out accordingly.

Implementation of the two modes, i.e. the *direct download* mode and the *P2P download* mode are detailed in the following two sebsections.

### 6.1 Direct Download Mode

In this mode, the deployments, i.e. first time deployments or be it the successive deployments, are facilitated by the WSPs only. For every deployment event received by an HP, after the retrieval of the service credentials, the HP contacts the WSP, requesting to download the service as specified by the URI. WSP then verifies identity of the HP that requests for the download. Download of a specific web service is enabled to an HP if and only if the HP belongs to list of deployed instances maintained by the Service Monitor at the WSPs end. Such an identity verification ensures that the service package is provided to only those HPs where the WSP intends to deploy the service. Once the requesting HP is verified by the WSP, a dedicated channel is established between the WSP and the HP to ensure the security of the service package and then the download for the requesting HP is started. Once the download is complete, the service package can be used to complete the deployment process. Binding the deployment process to such a constraint of downloading the service package from the WSP only may lead to longer download times specially for service packages of large sizes. In an attempt to reduce this download time, the architecture meets the requirement of another download mode.

### 6.2 P2P Download Mode

This mode implements the *Dynamic Torrent Service deployment protocal* (DynaTrons), enabling the HPs with deployed instances of the web service, along with the WSPs to provide the required service package. In this mode, at the HPs end, a receipt of a *deployment event* is followed by extracting the service credentials accompanied with the *seed list* as sent by the WSPs. With the complete information of the service URI and the seed list, the *Deployment Manager* starts the download process by invoking a sub module - *WAR Downloader* to download the WAR file corresponding to the web service deployment being carried out. Based on the service package length, an appropriate *chunk size* and *number of chunks* required to download are decided by the WAR Downloader. Each of such chunks is denoted by a *request chunk* object to acknowledge the chunks' identity such as :

—**Name** - name of the WAR file or the web service package.
—**ID** - unique id to identify the chunk's position in the complete service package.
—**Start Offset** - indicating the starting offset of the chunk in the complete file.
—**Chunk Size** - required specially to indicate the size of the last chunk as the last chunk of the WAR file may not be equal to the decided chunk size.
—**Buffer** - to store the downloaded chunk as a byte array.

Once all the *request chunks* are initialized, *WAR Downloader* distributes these chunk requests to different peers in the *seed list* using a *Chunk Distribution Algorithm*. A peer can use either of the two algorithms to distribute chunk requests to the peers in the seed list as discussed below:

—**Serial Chunk Distribution** - The algorithm sends chunk requests to all the peers in the seed list in a cyclic fashion, until all the chunks have been requested (Algorithm 1).
—**Proportionate Chunk Distribution** - The algorithm decides a value called *proportion*, based on the number of chunks required and number of peers available in the seed list. This algorithm also cycles over the peers in the seed list, but makes more number of chunk requests to the peers which are lightly loaded (given by the variable *proportion*), and only one chunk request to the peers which have heavy load (Algorithm 2).

Among the two chunk distribution algorithms, *Serial Chunk Distribution* gives a naive approach for distributing chunk requests but may fail to achieve the goal as expected, because a need of new deployment arises only when the existing deployments get loaded. In such a scenario, these deployed instances may require more amount of time to provide a requested chunk, resulting in higher deployment cost as compared with *Direct Download* mode. In contrast, *Proportionate Chunk Distribution* algorithm tries to overcome the problem by requesting less number of chunks

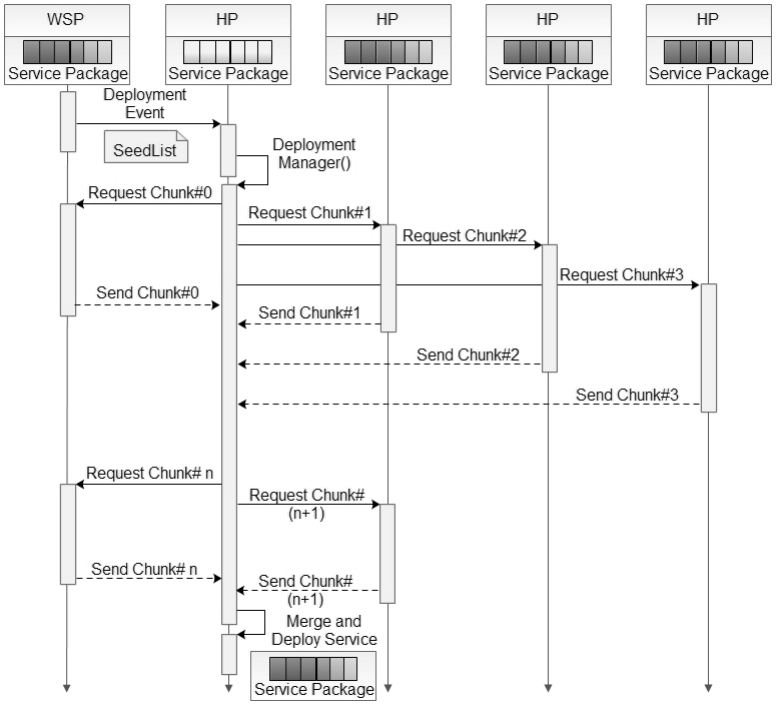


Figure 5. DynaTronS Sequence Diagram

**Algorithm 1:** Algorithm for Serial Chunk Distribution

```
1  begin
2      WarLength ← DeploymentEvent.getWarLenght();
3      SeedList ← DeploymentEvent.getSeedList();
4      ChunkSize ← CalulateChunkSize(WarLength);
5      NumberOfChunks ← WarLength/ChunkSize + 1;
6      Offset ← 0;
7      SeedCount ← 0;
8      for id ← 0 to NumberOfChunks do
9          peer ← SeedList.get(SeedCount);
10         SeedCount ← SeedCount + 1;
11         if SeedCount == SeedList.size() then
12             SeedCount ← 0;
13             // to cycle over the peers in SeedList
14         // requesting a chunk to a peer
15         RequestChunkToPeer(id, Offset, peer);
16         offset ← Offset + ChunkSize;
17         if Offset > WarLength then
18             // invalid Offset
19             break;
```

from the loaded peers. Though this approach does not guarantee to resolve the issue mentioned earlier, but still achieves a better performance over *Serial Chunk Distribution* algorithm.

**Algorithm 2:** Algorithm for Proportionate Chunk Distribution

```
1  begin
2      WarLength ← DeploymentEvent.getWarLenght();
3      SeedList ← DeploymentEvent.getSeedList();
4      ChunkSize ← CalulateChunkSize(WarLength);
5      NumberOfChunks ← WarLength/ChunkSize + 1;
6      Offset ← 0;
7      SeedCount ← 0;
8      Proportion ← NumberOfChunks/SeedList.size();
9      ProportionCount ← Proportion;
10     for id ← 0 to NumberOfChunks do
11         peer ← SeedList.get(SeedCount);
12         if peer.getThreshold() == TRUE then
13             // allow only one chunk request, so change SeedCount
14             SeedCount ← SeedCount + 1;
15             if SeedCount == SeedList.size() then
16                 SeedCount ← 0;
17                 // to cycle over the peers in SeedList
18         else
19             // request Proportion number of chunks, and then change SeedCount
20             ProportionCount ← ProportionCount − 1;
21             if ProportionCount == 0 then
22                 ProportionCount ← Proportion;
23                 SeedCount ← SeedCount + 1;
24                 if SeedCount == SeedList.size() then
25                     SeedCount ← 0;
26                     // to cycle over the peers in SeedList
27         // requesting a chunk to current peer
28         RequestChunkToPeer(id, Offset, peer);
29         offset ← Offset + ChunkSize;
30         if Offset > WarLength then
31             // invalid Offset
32             break;
```

Once the chunks are distributed, based on the *id* and *offset* of the chunks mentioned in each *request chunk*, different peers respond to provide chunks via *WAR Provider*, A chunk is provided only after performing a verification of the requesting peers identity (i.e. HP) in a similar fashion as discussed in *direct download* mode. Once the requesting HP downloads all the chunks of the WAR file, *WAR Downloader*performs a merge operation to organize the chunks in sequence based on the *start offset* to build the complete WAR file.

Figure 5 depicts a sequence diagram for the service package download carried out at the HPs end. As evident from the time line, at first WSP sends the *seed list* along with the *deployment event* to a selected HP. The deployment manager after extracting the *seed list* requests parts of the service package via the *WAR Downloader* (as per Algorithm 1). Each such request of a chunk is then provided by the *WAR Provider* at the other peer end. After all the chunks of the WAR file are downloaded, they are merged and then deployed.

Once the service package download is complete in either of the download modes, *Deployment Manager* performs an integrity check of the downloaded WAR file using file length and checksum

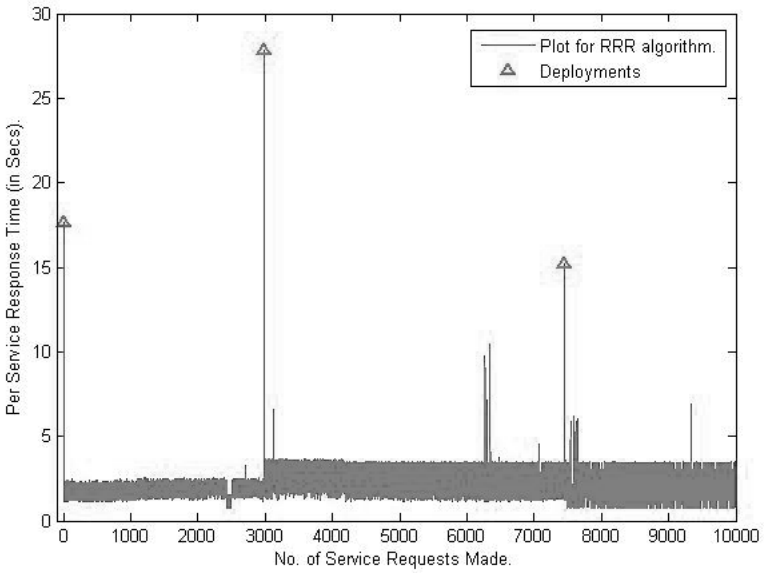


Figure 6. Plot for Round-Robin Reloaded]

retrieved from the *deployment event*. Any discrepancy may lead to download of the WAR file again. Otherwise the *Deployment Manager* relinquishes the control to the *Web Server* to deploy the web service.

## 7. EXPERIMENTAL RESULT

This section includes experimental results to demonstrate the efficient functioning of the framework. First the framework is implemented with decentralized registry and resource discovery and its behavior is shown in Section 7.1. Next, the DynaTronS Protocol discussed in the previous sections is implemented in the framework and the experimental results are discussed in Section 7.2.

The experiments have been carried out in a laboratory environment with a handful number of nodes. Thus, although the functioning of the system is demonstrated here, the scalability issue has not been handled in these experiments. Nevertheless, the scalability issue of a P2P-based system has been handled by other researchers Chiola and Cordasco [2009], Mantyla [2005], Rosen [2016] and this research work only puts forward the concepts of decentralized registry, resource discovery and distributed service download protocols based on a P2P system.

### 7.1 Experimental Results for Decentralized registry

In this section we present the results of the experiments performed for dynamically deploying web services and managing client requests over a distributed registry. The tests have been conducted in a laboratory environment where some of the nodes have been assigned as Web Service Providers (WSP) and others as Host providers (HP). In our distributed testbed we have considered 3 WSP peers, and 7 HP of different node configurations.

A web service for calculating the $N^{th}$ Fibonacci term has been used with value of N=40. Nodes taken into account have been of configuration ranging from 1GB-4GB of physical memory, 1.86GHz-3GHz Dual-core processors. The tests have been conducted by making 10000 client requests, made to different WSPs each time, and the response times have been measured. The graphs are obtained by using different scheduling strategies as described in Section 4.2 and then plotting the service response times with number of service requests made.

From the graphs shown in Figure 6, Figure 7, and Figure 8, we can observe that few requests

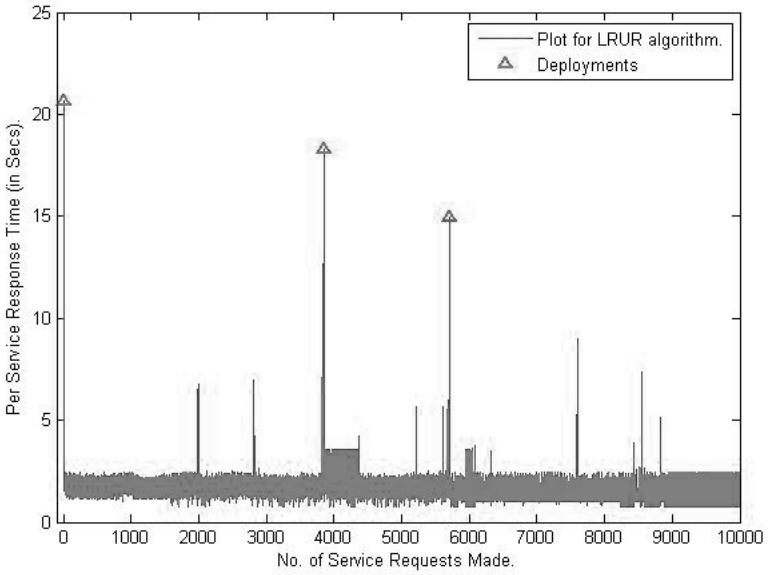


Figure 7. Plot for Least Recently used Reloaded

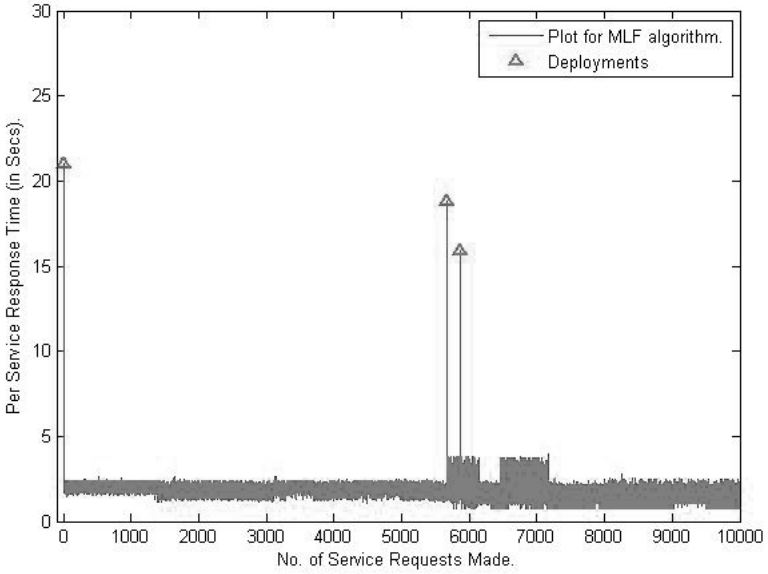


Figure 8. Plot for Minimum Loaded First

which incur the deployment costs (shown as higher peaks) take higher response times. For rest of the requests, the response time is comparatively low. Thus it can be concluded that the initial deployment cost is shared over successive consumer requests and the idea of "deploy once and use many times" is well implemented, proving to be a major advantage over the job-based framework.

Comparing the plots we can observe that the cumulative response time for RRR (Figure 6) strategy is high as compared to the other two strategies. This is because an instance with lower capacity is selected as *best node* in every cycle leading to higher response time, which is not the case in other two strategies. Hence RRR strategy leads to better utilization of resources at the cost of high response time. In contrast, LRUR (Figure 7) strategy achieves lower response

times for a longer time, till the current instance does not get loaded. At the same time it also guarantees the utilization of all the instances. $NT_m : M \rightarrow P(WR) \vee M \rightarrow P(HP)$ defines mapping relationship between each node to HP or WSP with respect to its Property type (P). Thus, whenever a node joins in the system, based on its own property it is decided whether it joins Web Service Provider's Chord Ring or remains in the system only as a Host Provider. The nodes joining the network with a role as WSP (as specified in P) form an intranetwork via Chord protocol and share web service information among each other and are denoted as WR. The instances are arranged in a cyclic fashion, thereby providing an approach to web service provisioning with lower response time and better utilization of resources as well.

In contract to the above approaches, MLF (Figure 8) does not cycle over the deployed instances. However, it provides the best possible response time for every time slice. Thus, cumulative response time is low as compared to other strategies, though it suffers from poor utilization of resources as the instances with lower capacity may have higher load values as compared to instances with higher capacity.

### 7.2 Experimental Results for dynamic deployment of web services using DynaTronS Protocol

In this section we present the results of the experiments performed to compare the performance of the two download modes for dynamic deployment of web services and managing client as discussed in the previous sections. As before, we have created a P2P-based distributed environment in the laboratory. Here, some of the nodes are considered as Web Service Providers (WSP) and others as Host providers (HP). Thus the tests have been carried out by making web service requests to a given WSP, and 13 HPs of different node configurations have been kept at disposal.

A web service for calculating the $N^{th}$ Fibonacci term is used with value of N=40. Nodes taken into account were of configuration ranging from 1GB-4GB of physical memory, 1.86GHz-3GHz dual-core processors. The tests were conducted by making 500 client requests. To compare the deployment times required in the different modes of deployment, the load threshold value is deliberately kept low so that a lower number of requests can trigger more deployments. A large service package of 36MB is used, so that the deployment requires a considerable amount of download time for both the approaches and the service response times are plotted against the number of service requests made.

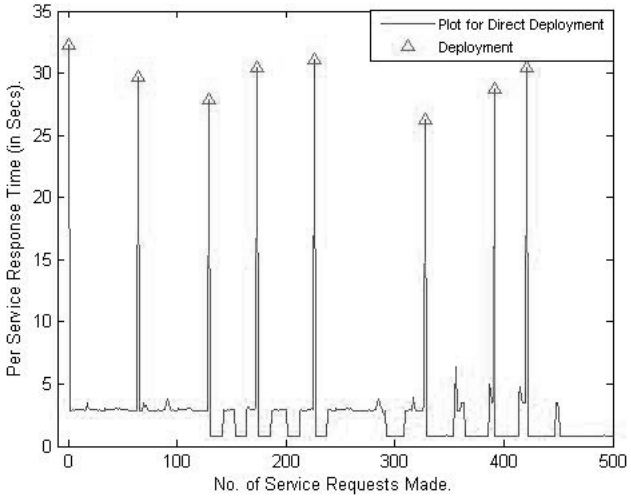


Figure 9. Experimental Results for Direct Download Mode

Figure 9 and Figure 10 depict the *service response time* for the two download modes. Figure 9 plots the time required for fetching a service using the *direct download* mode, and Figure 10

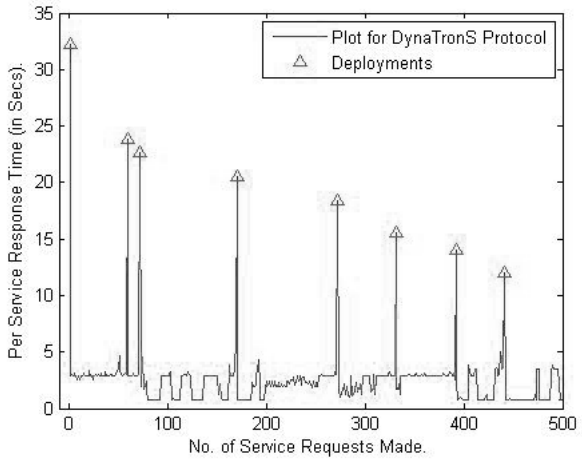


Figure 10. Experimental Results for P2P Download Mode

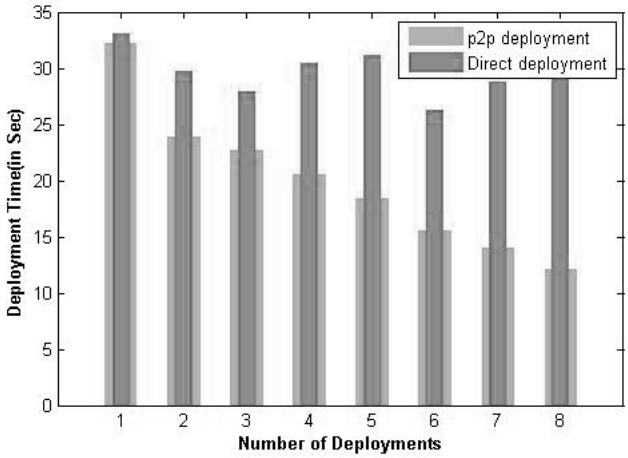


Figure 11. Experimental Results for Comparing P2P vs Direct Download Modes

shows the time required for fetching service using our P2P-based DynaTronS protocol. In our experiments, because of multiple service invocations, eight deployments were triggered. The experimental results shown in Figure 11 compare the service response time and also deployment time for the two download modes. It is evident from the graphs that as the number of deployments increases, the deployment time in the p2p mode decreases as there is an increase in the number of peers; whereas, in the direct download mode, the deployment time remains more or less same depending upon the network condition at that time.

The successive deployment events as depicted in Figure 9 also require similar deployment times as required by the first deployment in *Direct Download* mode. This is due to the fact that all deployments are handled by the WSP alone for fetching the service package. In contrast, the *P2P Download* mode in Figure 10 shows a considerable overall improvement. Though the first time deployment requires almost same amount of time as compared to *Direct Download* mode, but successive deployments start using the benefit of the peers in the *seed list* resulting in a sharp decrease in deployment time.

In a Grid environment, resources may be volatile in nature, which is handled well in a P2P framework. In our experiments, we simulated node failures by turning off nodes at random and observed that the service download process continued even if one or more of the peers left the network. If the failed node was an HP where deployment was being carried on, another

deployment was triggered on another designated HP.

## 8. CONCLUSION

This paper provides a robust approach to dynamic deployment of web services in a distributed environment. It also incorporates the benefits of P2P systems by de-centralizing the registry of web services as resources. This architecture proposes a unique concept of dynamic on-demand service deployment using BitTorrent-like P2P file sharing protocol. The proposed techniques enable handling the resource volatility of the nodes/peers in the network maintaining the consistency in the architecture to serve its purpose. Its simplicity, ease of use, effective use of bandwidth make this architecture appreciably suitable for deploying large, highly computational intensive services over the internet. Unlike typical network scenarios where a high demand for a resource can create a bottleneck thereby degrading the performance of the whole network, our proposed techniques can handle increases in demand of services and can actually improve the performance of the network.

The experiments discussed in this paper have been carried out in a laboratory environment. Thus, the scalability of the system could not be demonstrated in this paper. However, the system needs to be tested on a Internet scale which will be taken up as future work.

The performance of the whole architecture depends on how efficiently we can choose a best suitable *host provider* for a service. The need for an optimized load-balancing algorithm is also understood while carrying out the experiments. Hence, we intend to work on load balancing issues in future. Till now, we have not looked into the security issues. In future, we shall also focus on the security issues and investigate the possibilities of incorporating the work by other researchers on P2P and Grid security into our framework.

## References


Brooks, T. A. 2010. World wide web consortium (w3c). In *Encyclopedia of library and information sciences*. 5695–5699.

Castellà, D., Barri, I., Rius, J., Giné, F., Solsona, F., and Guirado, F. 2009. *CoDiP2P: A Peer-to-Peer Architecture for Sharing Computing Resources*. Springer Berlin Heidelberg, Berlin, Heidelberg, 293–303.

Chiola, G. and Cordasco. 2009. Degree-optimal routing for p2p systems. Number 1. 43–63.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. 2002. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing 6,* 2, 86.

Danielis, P., Skodzik, J., Altmann, V., Kappel, B., and Timmermann, D. 2015. Extensive analysis of the kad-based distributed computing system dude. In *2015 IEEE Symposium on Computers and Communication (ISCC)*. 128–133.

Feng, X., Shen, J., and Fan, Y. 2009. Rest: An alternative to rpc for web services architecture. In *Future Information Networks, 2009. ICFIN 2009. First International Conference on*. IEEE, 7–10.

Foster, I. 2002. The physiology of the grid: An open grid services architecture for distributed systems integration.

Foster, I. 2005. Globus toolkit version 4: Software for service-oriented systems. In *Proceedings of the 2005 IFIP International Conference on Network and Parallel Computing*. NPC'05. Springer-Verlag, Berlin, Heidelberg, 2–13.

Foster, I., Kesselman, C., and Tuecke, S. 2001. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl. 15,* 3 (aug), 200–222.

Gentzsch, W. 2001. Sun grid engine: towards creating a compute power grid. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*. 35–36.

Gupta, R., Sekhri, V., and Somani, A. K. 2006. Compup2p: An architecture for internet computing using peer-to-peer networks. *IEEE Trans. Parallel Distrib. Syst. 17,* 11 (nov), 1306–1320.

Harrison, A. and Taylor, I. 2005. Wspeer-an interface to web service hosting and invocation. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 175a–175a.

Jaiswal, D., Mistry, S., Mukherjee, A., and Mukherjee, N. 2013. Efficient dynamic service provisioning over distributed resources using chord. In *Signal-Image Technology Internet-Based Systems (SITIS), 2013 International Conference on*. 257–264.

Mantyla, J. 2005. Scalability of peer-to-peer systems. In *Seminar on Internetworking, Spring 2005*. Citeseer.

Mondejar, R., Garcia, P., Pairot, C., and Gomez Skarmeta, A. F. 2006. Enabling wide-area service oriented architecture through the p2pweb model. In *Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. WETICE '06. IEEE Computer Society, Washington, DC, USA, 89–94.

Mukherjee, A. and Watson, P. 2006. Adding dynamism to ogsa-dqp: Incorporating the dynasoar framework in distributed query processing. In *European Conference on Parallel Processing*. Springer, 22–33.

Mukherjee, A. and Watson, P. 2012. Case for dynamic deployment in a grid-based distributed query processor. *Future Gener. Comput. Syst. 28,* 1 (jan), 171–183.

Newcomer, E. and Lomow, G. 2005. *Understanding SOA with Web services*. Addison-Wesley.

Postel, J. and Reynolds, J. K. 1985. File transfer protocol.

Pouwelse, J., Garbacki, P., Epema, D., and Sips, H. 2005. The bittorrent p2p file-sharing system: Measurements and analysis. In *Proceedings of the 4th International Conference on Peer-to-Peer Systems*. IPTPS'05. Springer-Verlag, Berlin, Heidelberg, 205–216.

Qi, L., Jin, H., Foster, I., and Gawor, J. 2007. Hand: Highly available dynamic deployment infrastructure for globus toolkit 4. In *Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on*. 155–162.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. 2001. A scalable content-addressable network. *SIGCOMM Comput. Commun. Rev. 31,* 4 (Aug.), 161–172.

Rosen, A. 2016. Towards a framework for dht distributed computing.

Rowstron, A. and Druschel, P. 2001. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, M., Dabek, F., and Balakrishnan, H. 2003. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on 11,* 1 (feb), 17 – 32.

Tan, Y. 2009. A peer-to-peer based web service discovery mechanism. In *Proceedings of the 2009 Second Pacific-Asia Conference on Web Mining and Web-based Application*. WMWA '09. IEEE Computer Society, Washington, DC, USA, 175–177.

Tannenbaum, T., Wright, D., Miller, K., and Livny, M. 2010. Condor - a distributed job scheduler. In *Beowulf Cluster Computing with Linux*, T. Sterling, Ed. MIT Press.

Verbeke, J., Nadgir, N., Ruetsch, G., and Sharapov, I. 2002. *Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12.

Walsh, E. A., Ed. 2002. *Uddi, Soap, and Wsdl: The Web Services Specification Reference Book*. Prentice Hall Professional Technical Reference.

Wang, I. 2003. P2ps (peer-to-peer simplified).

Watson, P., Fowler, C., Kubicek, C., Mukherjee, A., Colquhoun, J., Hewitt, M., and Parastatidis, S. 2006. Dynamically deploying web services on a grid using dynasoar. In *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*. 8.

Wu, S. and Du, Z. 2005. Globalstat: a statistics service for diverse data collaboration and integration in grid. In *Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)*. 600–602.

**Sujoy Mistry** has completed his M.Tech in Computer Science and Engineering from University of Calcutta, Kolkata, West Bengal, India in 2009. He is now pursuing PhD from School of Mobile Computing and Communication, Jadavpur University, Kolkata, India. Currently he is doing research in the area of Distributed Computing, P2P Networks and Grid Computing.

**Dibyanshu Jaiswal** has completed his Bachelors in Computer Science from Dr. B. C. Roy Engineering College, Durgapur, West Bengal in 2011 and Master of Engineering in Computer Science form Jadavpur University, Kolkata West Bengal in 2013. He joined Innovation Labs of Tata Consultancy Services a Systems Engineer in September 2013.

**Dr. Arijit Mukherjee** had completed his PhD in 2008 from Newcastle University. From June 2008, Arijit worked as a lead researcher at Connectiva Systems (I) Pvt. Ltd., in Kolkata, India, for three years. He joined the TCS Research and Innovation of Tata Consultancy Services in September 2011 and is currently working as a Senior Scientist.

**Prof. Nandini Mukherjee** has joined as a faculty member in Department of Computer Science and Engineering, Jadavpur University, India in 1992. She has completed her PhD in Computer Science from University of Manchester, UK in 1999. She has become a professor in Jadavpur University in 2006. She has acted as the director of School of Mobile Computing and Communication, an interdisciplinary school of research in Jadavpur University from 2008 to 2014. Her research interests are in the area of High Performance Parallel Computing, Grid Computing and Mobile Computing. She is a senior member of IEEE.