A Chord-based Architecture for Efficient Dynamic Service Provisioning over Distributed Resources

D. Jaiswal1, S. Mistry2, A. Mukherjee3 and N. Mukherjee1
1Department of Computer Science and Engineering, Jadavpur University, Kolkata-700032, India
2School of Mobile Computing and Communication, Jadavpur University, Kolkata-700098, India
3Innovation Labs, Tata Consultancy Services, Kolkata-700091, India

Abstract—The demand for distributed applications has been increasing since the birth of internet. It has scaled geographical areas in search of information and computational resources for processing. With the advent of Grid computing, ad-hoc Virtual Organizations have added an impetus towards distributed applications by providing on-demand service provisioning. The use of job-based paradigms and strong coupling of current service-based paradigms with static registries such as UDDI hinder the achievement of complete dynamism over volatile resources of the Grid. A possible solution is the use of structured peer to peer overlay networks to keep a check on the resources as well as handle the volatility of the system. In this paper, we present an architecture of a web service based P2P Grid framework for managing the services and resources using the Chord protocol over a de-centralized registry which facilitates demand driven provisioning of web services and enables a virtual market place for computational resources.

Keywords: Grid Computing, SOA, Dynamic Deployment, P2P, Chord, UDDI.

1. Introduction

In the recent past distributed computing has evolved as a new paradigm where resources are shared among the consumers over the Internet in collaborative manner. With the advent of technologies like Service Oriented Architecture (SOA), Grid Computing and Cloud Computing, a new era of distributed computing has started which is characterised by availability of high-end resources and services over the Internet, access to the resources and services any time anywhere and non-requirement for the consumers of owning the resources - instead using them on demand and on pay-per-use basis. However, this era is also marked with challenges like scalability, heterogeneity and dynamism of resources which need to be handled for large-scale adoption of the new technologies by the consumers [1], [2].

As Internet forms the basic communication backbone of all the major distributed computing systems, provisioning web services is a keystone to operating pay-per-use services between businesses. Furthermore, to deal with the inherent dynamic characteristics within a distributed environment, there is a need for dynamic adaptation for provisioned services to accommodate the ever-changing business requirements externally, as well as the computing resource status internally, while maintaining the continuousness of service provisioning. Hence, dynamic web service provisioning has been one of the major research issues for quite some time [3], [4].

In order to deal with the challenges mentioned earlier, we have proposed a fully distributed SOA-oriented framework [5] which offers loose coupling, robustness, scalability, availability and extensibility for large-scale distributed systems. The proposed distributed architecture 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 [6], [4] used for dynamic service deployment in distributed environments. The main goals of the new architecture are mentioned below:

- To provide a distributed environment to overcome issues associated with centralized registry based architectures.
- To allow clients and service providers mention specific requirements for a service in order to achieve desired quality of service for clients.
- To provide scalability by load-balancing of deployed instances and re-deploying on demand using a P2P communication model.

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 consumer and take care of all the collaboration with hosts. Consumers are placed in the third layer. 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 metric of the Web Service.

In this paper we focus on the implementation of our proposed architecture on a structured overlay peer-to-peer (P2P) network, Chord [7], with the following goals:

1) de-centralizing the service registry in a structured
manually
2) making the registry adaptable with volatile set of resources
3) deployment cost of a given service incurred by a single
WSP is shared among all WSPs.
4) making the registry scalable, having definite time
bounds for a service query

In the earlier approaches [5], though the registry is de-
centralized as multiple single-site registries, a service may
become unavailable if the hosting WSP goes down. To
increase the service availability, we propose that the service
is to be hosted from more than one WSP incurring a separate
set of deployments for the same service. The framework
presented in this paper provides a robust approach towards
dynamically deploying web services, as well as decentral-
izing the registry to meet the increased availability of the
services and maintaining its scalability at the same time.

The rest of the paper is organised as follows: Section 2
discusses the existing technologies and frameworks related
to our work, Section 3 explains the proposed architecture and
its implementation. The experimental results are presented in
the section 4 and section 5 provides a discussion and future
scopes of the proposed architecture.

2. Approaches for Dynamic Web Service
Discovery and Deployment

With the growth of the Internet and web technologies,
more and more complex services have come into exist-
ence spreading across organizational boundaries and multi-
layered architectures, which has also resulted in complexity
of cost-effective service discovery.

In this context, dynamic service discovery and deploy-
ment are two of the few most important issues for the
Grid/Web framework. The Web Services Resource Frame-
work (WSRF) utilizes tools like Monitoring and Discovery
System (MDS) [8] for such purpose. But, an MDS Index
Service registries are functionally similar to the centralized
UDDI [9] registries with some additional flexibilities, which
still are limited in terms of exploiting and reflecting the
complete dynamism of a Grid.

2.1 UDDI-based Approaches

Most service oriented architectures use the UDDI standard
for creating service registries. Since UDDI is based on XML,
platform independence and inter-operability is implicit, but
at a syntactic level. As UDDI uses a keyword-based service
query, the discovery process is limited to a certain precision.
With the increase in number of entries corresponding to
services in the registry, the efficiency of the service discovery
process and its scalability become a critical issue. The com-
mon model of a centralized UDDI is liable to single point of
failure when the demand is extremely high. Decentralizing
the registry, by having replicas of the service and the service
metadata over various sites may be considered as a solution
to the problem, but this distributed architecture makes the
discovery process even more complex and requires periodic
synchronisation to maintain an uniform view.

The concept of dynamic service deployment on available
resources using UDDI was introduced by DynaSOAr [3], [4],
[10] which was capable of deploying services on-demand,
based on consumer requests. The DynaSOAr architecture
offered a clear separation between Web Service Providers
(WSP) and Host Providers (HP) in order to better manage
the simultaneous tasks of service publication and discovery
and service execution. In this architecture, the consumers
send requests to the WSP, which in turn routes them to an
appropriate HP for completion. For an already deployed
service, the request is executed on the host and the result
is returned back to the consumer. In case of a request for an
yet undeployed service, the process involves its discovery
from the centralized UDDI registry used by WSP’s to publish
the available services, locating the repository for service
download and thereafter its deployment and the execution of
the request. But, due to that static nature of UDDI, there are
certain limitations related to service metadata and dynamic
nature of resources in the grid environment.

2.2 P2P-based Approaches

As opposed to the UDDI-based approaches, P2P networks
accomplish sharing of resources based on the discovery of
peers in the network involving various discovery strategies
and P2P network topologies. Hence decentralized P2P archi-
tectures coupled with Web Services as resources have the
potential to be able to exploit the dynamism of the Grid
more efficiently.

and store data items. Unstructured P2P overlays support
partial-match and complex queries, which fail to discover
rare items as compared to popular ones, within specified
time bounds. In contrast, structured P2P networks have
some definite algorithms to guide the resource discovery
process having an upper bound of the search time required
for example such as Chord [7], CAN [12], Pastry [13],
Tapestry [14]. Among these structured P2P networks the
architecture proposed in this paper is based on Chord. The
reason behind this is two-fold - (i) Chord is efficient in terms
of tracking the networked resources and (ii) Chord uses an
efficient mechanism of key assignment for network nodes as
well as the retrieval by locating the node responsible for the
key.

In recent years, the developments in decentralized P2P
techniques related to resource sharing and discovery have
ensured fault tolerance and scalability in the systems. At the
same time, researchers have also looked into the possibility
of incorporating P2P techniques with web service based
architectures to cater to dynamic provisioning of services.
WSPeer [15] emerged as an interesting framework which combines the benefits of P2P’s decentralized resource sharing with the XML based web service technologies. One of the major advantages of WSPeer is, consumers and service providers are located in remote places and can use it as an interface. This architecture provides the dynamic deployment facility such a way that anyone can easily deploy their application or part of application as web services. WSPeer has two different approaches towards service publication and discovery by the use of: (a) HTTP and UDDI coupled together, and, (b) a P2PS [16] implementation with a pluggable architecture of nodes. In the case of HTTP/UDDI implementation, the static centralized registry still remains a bottleneck of the infrastructure, which was removed in the P2PS implementation. Since the peer can act as both a service provider as well as service consumer a service endpoint was made available only when the node remains available in the network.

In recent years, there have been a lot of work on efficient ways of service discovery in Grid and SOA based frameworks [17] applying different approaches such as key word based matching, semantics or syntax based matching for the discovery process. Some approaches also use a ranking model to enhance the search procedure. All these approaches differ from each other and it is claimed in [17] that the suitability of the approaches depend on the application requirements, which in turn makes the selection of an appropriate service discovery process difficult. Further, it is desirable that the service discovery process should also be flexible enough for changing requirements. Hence, there is a need for a service discovery mechanism coupled with a registry which also caters to dynamic service provisioning. Frameworks which provide a complete solution for the entire cycle of web service publication to the deployment and management of the resources are rare or are in their initial stages of development.

A comparative study of the architectures mentioned in this section reveals that a static registry such as UDDI is a bottleneck for any system based on SOA in terms of availability, capability of handling volatile resources in the network and system scalability. Since P2P systems are capable of handling the volatility of networked resources, a merger of the two concepts (P2P and SOA) may have certain advantages. 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, the 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, Chord can help achieve the above characteristics for a decentralized registry.

3. Using Chord for Dynamic Service Provisioning

In the context of dynamic web service provisioning, availability of services as well as provisioning based on service requirements to meet its QoS play an important role. The performance of such architectures totally relies on two important factors: firstly, the computational resources on which the web services are deployed and secondly discovery of web services. Hence managing both i.e. web services and computational resources, turns out to be an important issue.

In our previous work [5] we presented an architecture enabling dynamic on-demand service discovery and deployment based on the concepts of P2P computing. It strives to use the idle resources in the network via service deployments among the distributed resources on the basis of their capability and load factors. The services are made available from the provider of services and are deployed on-demand, after a proper matchmaking of the service metrics with the capability of the resources available to provide better performance. Successive deployments are made if the existing deployments get loaded or fail to offer response with some specified QoS. Though this approach could efficiently manage the computational resources for service deployments and their execution, web service discovery was limited to multiple single-site registry, i.e. the web services known to a given service provider were not known to other service providers, hence the service was available from a single service provider only. To increase the availability of a given service, it must be hosted from multiple service providers. In such a case deployment costs of a given service are incurred separately from different service providers. Thus, with an objective to provision web services on-demand, in the proposed architecture we focus on de-centralizing the registry, making it discoverable and thus increasing the service availability and reducing the deployment costs by sharing the current deployments. In the remaining part of this section, we describe the improvements made to our earlier work by decentralizing the registry using Chord after giving a brief overview of the proposed architecture.

3.1 Overview of the Architecture

Figure 1 shows the architectural diagram of the proposed framework. It consists of three loosely coupled entities as follows:

- **Client**: To invoke web services as consumer/client requests, using an interface provided by some WSP.
- **Web Service Provider(WSP)**: It acts as provider of services in the system for establishing and managing the registry of web services and provisioning them on demand.
- **Host Provider(HP)**: It acts as provider of computational resources for deploying the web services and serving client requests.
The Client is the simplest component of the architecture. The clients can make service requests to the system by making a search query to the registry, via the interface provided by any of the WSP/s in the network. Apart from the client, all the nodes contributing to the system are peers to each other. Each peer can functionally act as either a WSP or a HP which is to be determined before a peer joins the network. The peers which aim to host web services, join the network as WSPs, whereas the peers that aim to provide computational resources join the network as HPs. The main role of HPs is to provide a platform for carrying out deployment of web service by WSPs. All the peers share their dynamic load information with all other peers to achieve better functionality and QoS for the clients.

WSP plays the most important role in the system. It is mainly responsible for:
1) Providing an interface for the clients.
2) Establishing and maintaining the service registry.
3) Taking care of new and successive deployments of the web services on the basis of current load information of the peers, on demand basis.
4) Scheduling the incoming consumer requests to the current deployed instances.

Figure 2 shows a snapshot of the interface provided by the WSP to the client. It provides a list of services hosted by the WSPs in the network, along with their status and link for making the service requests. A prospective client can choose a service among the available services or can choose to use an already deployed service as per its service requirements, to the WSP, which is then scheduled an appropriate HP as discussed in [5]. If the client is unable to find a service, it can attempt to make search query to the registry with the name of the service via the search space provided by the interface.

3.2 Decentralizing the Registry using Chord

Chord [7] - a DHT [11]-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. 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 address.

Within our framework whenever a node wishes to join the network as a WSP, it may or may not have web services to host. In either case it joins the de-centralized registry within the network and shares the web services as resources 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, the value corresponding to a key is service metadata, which consists of:
1) Name of 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.

Such service metadata help in identifying the service, its endpoints and other details necessary for proper execution in SOA framework.

3.3 Workflow of Registry with SOA Framework

In order to maintain and use to the registry for the dynamic set of resources, three basic steps are required. These steps are responsible for publish-find-bind notion of the SOA framework that achieves interoperability of web services.

3.3.1 Publishing a Web Service

The first task performed by a WSP as it joins the network is to publish the web services it owns. If a WSP is the first
one to join the network, it initializes the service registry by uploading its services to the DHT. It also identifies itself as the bootstrap peer for the chord protocol, so as to facilitate other WSP peers to join the network and contribute to same DHT so formed. As the number of WSPs and the web services hosted by them increases in the network, the entries in the registry are distributed among the WSP peers with respect to the Chord protocol. This is done by mapping the keys to the respective node identifiers and distributing the information over the nodes/WSPs in the network. Each WSP also maintains a list of all web services (i.e. [key, value] pairs) locally, assigned by chord along with the web services it owns/host.

A WSP publishes a service as its owner in the registry, it is now discoverable by clients and other WSPs, until and unless it is explicitly removed by the owner itself. No other WSP has the rights to perform administrative tasks for a service such as publishing, deploying, replacing, modifying and removing the service to or from the registry, except the owner itself. Since the registry provides some service metadata (i.e. value), which is sufficient enough to make the service discoverable by different WSP and clients, it does not provide the whole service package in order to maintain the security of the service.

Once the service is published it may so happen that the owner undergoes failure or leaves the chord ring, thereby creating uncertainty of the environments. In such a scenario the service still remains discoverable in the network. This is possible because the keys of the services which the owner in concern is responsible for and the services which it owned are shared among the predecessor and successor of the concerned peer in a structured fashion as per the chord protocol. Such a phenomenon of exchanging information and re-mapping the keys provides higher service availability.

3.3.2 Discovery of Web Services from Registry

A client can make a service request only when the service endpoint is made available to the client 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 [7], thus even with increase in number of web services and WSP's the registry remains scalable as compared to previous approaches.

3.3.3 Binding of Web Services with clients

After the service endpoint is made available to the client, depending on the three parameters i.e. status of the service, the WSP through whom the service request is being made and the WSP who owns the service; a client request is scheduled accordingly among the available HPs. To represent the work flow, the requests are modeled as a tuple of three parameters i.e. [Name of Service, Request to WSP, Request of WSP]. For example as shown in Figure 3 a service request tuple [WS2, WSP#1, WSP#2] means a client request for web service WS2, the request is made via WSP#1 and the owner of WS2 is WSP#2. Describing the service request tuple in a generalized manner denoting WSx be the name of the web service owned by a WSP#x and WSy be a name of the web service owned by WSP#y and so on, with respect to these parameters four generic kinds of requests can occur as depicted in Table I along with there scheduling criteria.

![Fig. 3: Basic Architecture Diagram](image)

<table>
<thead>
<tr>
<th>Request Type</th>
<th>Status</th>
<th>Request Tuple</th>
<th>Scheduling Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR1</td>
<td>Deployed</td>
<td>[WSx,WSP#x, WSP#x]</td>
<td>Routed to HP using WSP#x’s scheduling strategy</td>
</tr>
<tr>
<td>SR2</td>
<td>Deployed</td>
<td>[WSx,WSP#y, WSP#x]</td>
<td>Routed to HP using minimum load criteria</td>
</tr>
<tr>
<td>SR3</td>
<td>Available</td>
<td>[WSx,WSP#y, WSP#x]</td>
<td>Routed to owner i.e. WSP#y for deployment</td>
</tr>
<tr>
<td>SR4</td>
<td>Available</td>
<td>[WSx,WSP#x, WSP#x]</td>
<td>Deployed on an appropriate HP</td>
</tr>
</tbody>
</table>

From the above table it is evident that for a service
which is deployed, a client request made directly to the service owner (SR1) is routed using a default scheduling strategy. A service request made to a WSP other than the owner (SR2) of the service is routed to a minimum loaded HP so that QoS is not compromised. If the service is not deployed yet, a request made to a WSP other than the owner (SR3) first routes the request to the owner for the deployment of the service. After which a normal deployment procedure is carried out similar to deployment request made to same owner (SR4). As mentioned earlier, if the owner of a service goes down, the service still remains discoverable and hence can be used by client via SR2. Since the deployment rights are constrained to the owner of the service, further new deployments will not occur and all the incoming client requests will be serviced from the existing deployments.

Catering the client requests employing the benefits of chord protocol, thereby increasing the service availability by de-centralizing the registry, provides a unique approach for rendering a demand driven architecture for web service provisioning. For proper utilization of resources at the same time the incoming client requests needs to scheduled to appropriate resources for execution.

3.4 Scheduling Strategies

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 as time slice based, i.e. an instance among the 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 the scheduling strategy used below:

- **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.

All the incoming requests of WSx made to WSP#x are scheduled by the use of any one of the above mentioned strategy. Whereas the incoming requests of WSx made to WS#y are always scheduled via MLF strategy, to avoid the overhead of constantly assigning a best node for the services which they do not own.

4. Experimental Results

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 were conduct with 3 WSP peers, and 7 HP of different node configuration. A web service for calculating the Nth 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 10000 client requests, made to different WSPs each time, and hence calculating the response times. The graphs are obtained by using different scheduling strategies as discussed earlier and hence plotting the service response times with number of service requests made.

From the graphs we can observe that few requests which incur the deployment costs (shown as red peaks) take higher response times. For rest of the requests, the response time required 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 the cumulative response time for RRR (Figure 4) strategy is high as compared to other two strategies. This is because an instance with a 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 implements better utilization of resources at the cost of high response time. In contrast LRUR (Figure 5) strategy achieves lower response times for a longer time, till the current instance does not gets loaded. At the same time it also guarantees to use all the instances in a cyclic fashion, hence provides an approach with lower response time and better utilization of resources as well. Since MLF (Figure 6) does not cycles over the deployed instances and hence provides the best possible response time for every time slice, hence low cumulative response time as compared to other strategies, though is suffers from poor utilization of resources as the instances with lower capacity may have load values higher as compared to instances with higher capacity.

5. Discussion and Future Work

The architecture presented in this paper overcomes the drawbacks of distributed environments by strongly coupling it with the SOA. It uses on-demand web service provisioning frameworks and employs service provisioning on the basis of load balancing and meeting minimum service requirements to achieve better performance with emerging demands of web applications.

Further, incorporation of P2P technologies provides a robust approach towards handling the uncertainty of grid environments. It overcomes the scenario of single point failure as it is devoid of any centralized mechanism of service registry. The architecture not only decentralizes the registry but at the same time dynamically adapts the registry...
to the volatile changes of the network. Since the service information is distributed over the registry, and not the service itself, this approach restricts WSPs other than the owner of the service to perform any administrative tasks of the service. This is because the security of the service (executable code and service metrics) is an important issue, still to be taken care of. The authors are of believe that the security of the service is a separate research problem focusing on what information and to whom and how the service must be made available.

The framework also establishes new scopes for virtual organizations which can take part as provider of services as well as provider of resources. The approach towards de-centralizing the registry may further enhance business opportunities for cross hosting the web services, agreed on some cost model. Combining SLA for client requests is a future aspect still to be achieved which would provide freedom for client to choose a cost model best suited for them.

References