On Demand Service Provisioning On Peer-to-Peer Networks

Thesis submitted

by

Sujoy Mistry

DOCTOR OF PHILOSOPHY (Engineering)

School of Mobile Computing and Communication
Faculty Council of Engineering and Technology
Jadavpur University
Kolkata-700032, India

2017

Dedicated to my beloved parents, wife and my daughter

CERTIFICATE FROM THE SUPERVISOR/S

This is to certify that the thesis entitled "On Demand Service Provisioning On Peer-to-Peer Networks", submitted by Shri Sujoy Mistry, who got his name registered on 21st May, 2012, for the award of Ph.D. (Engineering) degree of Jadavpur University is absolutely based upon his own work under the supervision of Prof. Nandini Mukherjee and Dr. Arijit Mukherjee and that neither his thesis nor any part of the thesis has been submitted for any degree / diploma or any other academic award anywhere before.

Prof. Nandini Mukherjee
Supervisor
Professor
Department of Computer Science & Engineering
Jadavpur University

Dr. Arijit Mukherjee
Supervisor
Senior Scientist
TCS Research & Innovation
Tata Consultancy Services

Acknowledgements

It is my pleasure to acknowledge the roles of several individuals who were instrumental for completion of my Ph.D research. It has been a long journey with all the great memories which were created around the DST-FIST lab of Department of Computer Science and Engineering, Jadavpur Unniversity.

I am tremendously fortunate to have a teacher and supervisor, Prof Nandini Mukherjee during this period of Ph.D research. I would like to express my sincerest gratitude and deep sense of respect to her for her inspiring suggestion and guidance. It was her sustained support, continuous monitoring and constant persuasion which help me to focus on achieving my goal. Her advice on both research as well as career has been invaluable. At this point where I am standing its only possible because of her enduring assistance, rigorous attention, motivation and encouragement.

I must give my high, respectful gratitude to my another supervisor, Dr. Arijit Mukherjee for his sincere guidance, supervision and help throughout this Ph.D research. I learned a lot from him throughout this period of research, specifically the technical aspects of this research, with many challenging yet valuable experiences in order to complete the work. His valuable guidance, scholarly inputs, quick paper correction and consistent technical support that I received throughout the research work helped me to persue my goal. Long discussion over problems with him helped me to get out from several critical situations during this period.

I would like to express my highest appreciation towards my beloved parents Shri Bimal Mistry and Smt. Kabita Mistry, who have always been there for me, whatever situation arises or where I am, for all unconditional support and patience. Thank you for being ever so understanding and supportive.

Also thanks to my beloved wife Smt. Smiti Mistry and my sweetest daughter Senjuti Mistry, for being around me, and for never ending motivations I have been getting all this while. I also like to convey my sincerest gratitude to my uncle Prof. Priti Km. Roy for his continuous mental support and encouragement during the entire Ph.D period.

I would like to give a warm thanks to Tata Consultancy Services(TCS) for their financial and other support through TCS-RSP research scholarship propram.

I would like to specially thank my friends and fellow lab-mates of DST-FIST lab, Department of Computer Science and Engineering, Jadavpur University for their encouragement and mental support throughout my Ph.D work. Firstly, I would like to thank passed out M.Tech students of Department of Computer science and Engineering, Mr Dibyanshu Jaiswal, Mr. Sagar Virani and Mr. Sukhen Das for their support during this research work. I would like to give a special thanks to a very special friend cum lab-mate Dr Subrata Dutta for his encouragement, mental support throughout this research period. I would like to thank Dr. Sarbani Roy, Associate Professor, Department of Computer Science and Engineering, Jadavpur University for her motivation towards the research work. I would also like to thank, Dr. Madhulina Sarkar, Dr. Monideepa Roy, Dr. Zeenat Rehena, Shri Tanmoy Moitra, Shri Dibyayoti Ghosh, Shri Rupam Mukhopadhyay, Smt. Shupi Choudhury, Dr. Suman Shankar Bhunia, Shri Binoy Ray, Shri Joy Dutta and all other lab mates with whom I have worked together and spent a wonderful time. I would like to thank Shri Prabhat Chatterjee who is a technical assistant in the Department of Computer Science and Engineering, Jadavpur University for his support towards maintaining DST-FIST Lab.

I express my sincere thanks to, Head of the Department, Department of Computer Science and Engineering, Jadavpur University and Director, School of Mobile Computing and Communication, Jadavpur University for allowing me to use the DST-FIST and their other supports towards fulfillment my research work.

Finally, I like to thank all the teachers and staff of Department of Computer Science and Engineering and School of Mobile Computing and Communication, Jadavpur University and also all my relatives, friends for their supports and good wishes.

Date: _____ _____

Sujoy Mistry

JADAVPUR UNIVERSITY

KOLKATA – 700 032, INDIA

INDEX NO: 8/12/E

1. **Title of the Thesis**:

On Demand Service Provisioning On Peer-to-Peer Networks

2. **Name, Designation and Institution of the Supervisor/s**:

(a) **Prof. Nandini Mukherjee**
Professor
Department of Computer Science and Engineering
Jadavpur University
Kolkata-700032

(b) **Dr. Arijit Mukherjee**
Senior Scientist
TCS Research & Innovation
Tata Consultancy Services
Kolkata-700091

3. **List Of Publication**:

(a) **Journal**:

I. ***Sujoy Mistry***, Dibyanshu Jaiswal, Arijit Mukherjee and Nandini Mukherjee, "P2P Based Service Provisioning on Distributed Resources". Published in November, 2016 issue of (INTRENATIONAL JOURNAL OF NEXT GENERATION COMPUTING), IJNGC- Vol-7, No 3, and ISSN: 2229-4678.

(b) **Conference**:

I. ***Sujoy Mistry***, Dibyanshu Jaiswal, Arijit Mukherjee and Nandini Mukherjee, P2P-based Service Distribution over Distributed Resources', In 29th IEEE International Conference on Advance Information Networking and Applications(AINA-2015).

II. ***Sujoy Mistry***, Dibyanshu Jaiswal, Sagar Virani, Arijit Mukherjee and Nandini Mukherjee, 'An Architecture for Dynamic Web Service Provisioning using Peer-to-Peer Networks', In 9th International Conference on Distributed Computing and Internet Technology, ICDCIT-2013, LNCS 7753, p 290

III. Dibyanshu Jaiswal, ***Sujoy Mistry***, Arijit Mukherjee and Nandini Mukherjee, "A Chord-based Architecture for Efficient Dynamic Service Provisioning over Distributed Resources", accepted in PDPTA'13 - The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications.

IV. Dibyanshu Jaiswal, ***Sujoy Mistry***, Arijit Mukherjee and Nandini Mukherjee, "Efficient Dynamic Service Provisioning over Distributed resources using Chord", In International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), 2-5 Dec. 2013,Kyoto, JAPAN, Page(s): 257 – 264, INSPEC Accession Number: 14064326 .

V. ***Sujoy Mistry***, Arijit Mukherjee, Nandini Mukherjee, "Towards a Dynamic On-Demand Service Grid Based on P2P Networks", Second International Conference on Emerging Application of Information technology 2011, Kolkata, India 2011, p- 165 – 170,ISBN-978-1-4244-9683-9.

4. **List of Patents**: None

5. **List of Presentations in National = International Conference** :

a) ***Sujoy Mistry***, Dibyanshu Jaiswal, Sagar Virani, Arijit Mukherjee and Nandini Mukherjee, 'An Architecture for Dynamic Web Service Provisioning using Peer-to-Peer Networks', In 9th International Conference on Distributed Computing and Internet Technology, ICDCIT-2013,KIIT, Bhubaneswar, LNCS 7753, p 290

b) Dibyanshu Jaiswal, ***Sujoy Mistry***, Arijit Mukherjee and Nandini Mukherjee, "Efficient Dynamic Service Provisioning over Distributed resources using Chord", In International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), 2-5 Dec. 2013,Kyoto, JAPAN, Page(s): 257 – 264, INSPEC Accession Number: 14064326 .

c) ***Sujoy Mistry***, Arijit Mukherjee, Nandini Mukherjee, "Towards a Dynamic On-Demand Service Grid Based on P2P Networks", Second International Conference on Emerging Application of Information technology 2011, Kolkata, India 2011, p- 165 – 170,ISBN-978-1-4244-9683-9.

d) ***Sujoy Mistry,*** "Towards a Dynamic On-Demand Service Grid based P2P Networks". Poster Paper Presentation at TCS Technical Architect's global conference in Delhi on 3rd and 4th May 2012.

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. The recent developments in areas like the grid, cloud and utility computing have enabled researchers in need of compute power to utilize resources from a globally shared pool. The emergence of Service Oriented Architectures and Web Services has contributed to the development of several platforms for grid/cloud computing which offer a new way to create loosely-coupled dynamic distributed systems. Thus introduction of Virtual Organizations has 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 distributed frameworks. A possible solution is the use of structured peer to peer overlay networks, which has emerged as a means of sharing data and computing power where the nodes act as peers of each other to keep a check on the resources as well as handle the volatility of the system.

In this thesis, the architecture of a demand-driven web service deployment framework is presented which allows sharing of data and computing capacity using p2p technology as its backbone. Thus the use of dynamic peer-to-peer (p2p) techniques within a web service based framework introduce the ability of the network to adapt to resource volatility which has been already established in p2p-based content-delivery models. Consideration of a service as well as user specification of resources for provisioning consumer requests increases performance of the entire architecture. The proposed framework also incorporates a proper load balancing approach

between the servers, thereby increasing the utilization of resources in the networks.

One of the main focus of this architecture is decentralization of the registry. The distributed registry in the proposed architecture has been implemented in such a way that it makes service discoverable form any part of the network, increases the service availability, and provides a better platform for handling the scalability of the framework. The use p2p file sharing techniques within this framework reduces the overhead of fetching the deployable code from the service repository and sharing it among the deployed instances to carry out successive deployments. Thus, this thesis focuses on various issues such as resource availability, scalability and abstraction. Demand-driven resource allocation is based on request parameters and availability of the resources to create the basis for a fully dynamic virtual market place of computational resources.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

> "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable." —— Leslie Lamport

With each passing day distributed computing encompasses new distributed resources connected through different modes of communication and enabling sharing of data, sharing of cycle, sharing of storage, collusion of computing address etc. As a matter of fact, with the growth of the inter-networking technology and the increase in the number of computing nodes, the complexity of sharing resources within and across enterprizes environment increased a lot. Many organizations now relying on collaborative infrastructure which leads to significant cost reduction by distributing their IT service requirements to various service providers. Although the fundamental distributed computing typically can handle heterogeneity, scalability, availability of any distributed systems but these changing burden of excessive resources poses new challenges for distributed application development and deployment.

As complexity of decentralization and sharing of resources in distributed environment are continuing it is important that, even with this diversity of resources there is a need to obtain desired Quality of Service(QoS) as per the requirements of enterprizes, service providers, or customer systems. There is a need for new concepts to allow the applications to access services and share resources across distributed environment or wide area networks, for better performance and other QoS metrics which are important in a particular context.

Frameworks like grid [7] creates the basis of sharing and coordinated use of dis-

tinct resources within dynamic, distributed virtual organizations (VO) [8] that is, the creation of virtual computing systems, by collaboration of remote resources operated by distinct organizations with different protocols to deliver the desired QoS. To support such VO environment, several tools, such as Globus [9], Condor [10] and SunGridEngine [11] allow the construction of distributed applications over grid-based resources. In order to satisfy the requirements, researchers have introduced many different other architectural styles and standards, such as Service Oriented Architecture(SOA) [12], peer-to-peer computing [13], grid computing and more recently cloud computing [14] over the past decade to ease the cost of discovery, deployment and maintenance, with varying degrees of success.

Although grid made it possible for the user to execute computationally expensive jobs on remote resources, but for service discovery in globally distributed environment with resource volatility, few challenges remain unsolved. In this context peer-to-peer (p2p) computing has emerged as a means of sharing data and computing power where the nodes act as peers of each other. This has resulted in the consideration of computing grids based on p2p concepts as a new paradigm for developing new application for solving large-scale problems in science, engineering, and commerce.

Continuing 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.

One major advantage of service-oriented framework over job based frameworks is that once the service is deployed, it stays on the resource until explicitly removed and the initial cost of deployment can be shared across multiple invocations. This is not the case in job-based frameworks where once the execution is over, the job is removed from the queue, and for each subsequent invocation, the execution code and data must be resubmitted.

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 [15] [16]. A three-tier architecture called DynaSOAr [3] is proposed to provide a generic infrastructure and to offer a service-only approach for deploying web services on demand and to exploit the computational resources offered by a host. However, DynaSoAr is based on a centralized registry and therefore its performance is degraded for large number of service requests. It also suffers from the problem of single point failure.

Not only DynaSOAr, some other research works like WSPeer [4], HAND [16], P2PWeb [17] etc. also suffer from the same bottleneck, limiting the service code to a single site.

In order to overcome the above problem, this thesis focuses on the development of 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. 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.

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.

1.1 Distributed Computing

Distributed computing [18] is a field of computer science that studies distributed systems. A distributed system consists of multiple autonomous computers that communicate through a computer network. The computers interact with each other in order to achieve a common goal, such as solving a large computational problem. A problem is divided into many tasks, each of which is solved by one or more computers. Basically, distributed computing is the study of distributed systems which can characterized by :

- Platform independence : be capable of dealing with heterogeneous devices, software stacks and operating systems in interpretable manner.
- Scalability : be easy to expand and scale
- Availability : be available all the time (even though parts of it may not be)
- Abstraction: hide communication from the users.

In a distributed system, there are several autonomous computational entities, each of which has its own local (distributed) memory. Each such entity exchanges the information by passing messages among themselves. Alternatively, each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users. All the systems in the network have to tolerate failures on individual computers. The structure of the system (network topology, network latency, number of computers) is not known in advance. The system may consist of different kinds of computers and network links, and the system may change during the execution of a distributed program. An important goal of a distributed system is to hide the fact that resources are physically distributed across multiple computers. A distributed system which is able to present itself to users and applications as a single computer system is said to be transparent.

1.1.1 Distributed Computing Architectures

Architectures of distributed computing differ with the shift in consanguinity of the nodes. There are various design styles available to show the orientation of a distributed system [19], but the most important aspect is to make a divergence between the logical orientation of the collection of software components and also the true physical organization. Thus the organization of distributed systems is mostly categorized as follows:

- Software Architectures - describe the organization and interaction of software components; focuses on logical organization of software (component interaction, etc.)

- System Architectures - describe the placement of software components on physical machines. The realization of an architecture may be centralized (most components are located on a single machine), decentralized (peer-to-peer: most machines have approximately the same functionality), or hybrid (some combination).

Till date different architectural styles have been established, of which some the most commonly available distributed systems are:

1. Layered architectures
2. Object-based architectures
3. Data-centered architectures
4. Event-based architectures

Primary form of distributed architecture is the style of coordination and communicating the work within concurrent processes. There are numerous message passing protocols available, but commonly a master/slave relationship exists and the processes communicate directly with a central process. Another substitute way is to use a shared database, a 'database-centric' architecture can facilitate distributed computing to be done without any mode of direct inter-process communication.

1.1.2 Applications in Distributed Computing

Several distributed applications continuously evolved for sharing of distinct resources over the Internet which use vastly in real-life. Some of these applications are given below:

- One of the important and hugely used application is telecommunication which is based on distributed systems like Internet, cellular networks, different type of wireless networks etc.
- Airplane control towers and different type of industrial applications are also based on such computing.

- Different kind of network applications are also based on this type of computing such as WWW (World Wide Web), peer to peer networks, distributed databases and many more.
- Distributed computing is also used with parallel computation in different applications such as. Scientific Computing and Data rendering in distributed graphics etc.

1.2 Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) [12] is approach for building reliable distributed applications where autonomous computational entities residing on distributed nodes are exposed as services and can communicate between themselves by exchanging messages to perform a computational process. Such an architecture is built on the concepts of loose coupling between interacting services allowing flexibility, scalability and fault tolerance which are considered as pillars of a dynamic distributed environment. Integrating different types of applications over a wide variety, of web based environments, with the use of multiple platforms, SOA provides a great means of improving the reuse of application logic while eliminating duplication of production environments. With respect to the underlying concept, a service is considered as a software component accessible by another software component or by a standard interface.

As shown in Figure 1.1 [1] of SOA at a very basic level, it uses a paradigm of *publish-find-bind-execute*. This is characterized by three components namely:

Service Consumers/Client are the users who make request for a service via an application.

Service Providers provides service as a functional unit to perform the application specific task or a specified business logic.

Service Registry is a store of services available along with the records of the service provider and corresponding interfaces.

Figure 1.1 illustrates a simple service interaction cycle where an organization or service provider *builds* services and makes them discoverable over the Internet. They

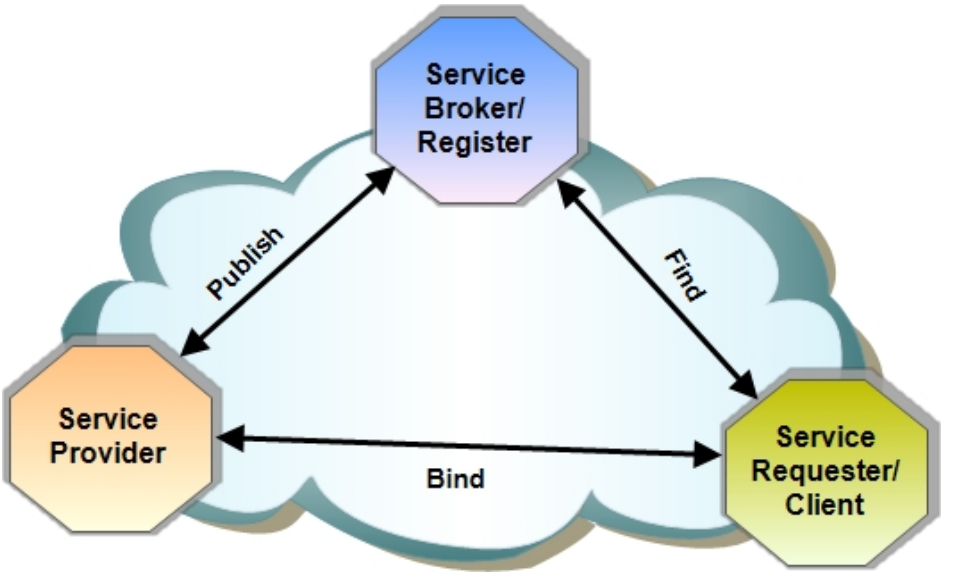


Figure 1.1: SOA Interactions [1]

register the service and hence *publish* the service information and interface in the Registry, making it discoverable. A client in search of a service, based on some criteria, *queries* the registry in order to discover the service for its own use. The Client then performs dynamic *binding*, to get a direct access of the service interface and its endpoint. The service is then invoked by the client and is executed.

Thus, SOA provides an architectural approach where all the components implemented in applications are accessed as modular services which commonly have the following characteristics:

- Service Composability and re-useability - that means it can be used to compose other services as well as to promote reuse of codes.
- Services communicate by the exchange of messages using well defined communication protocols.
- Services can perform in a function, where messages are sent and received in such a form that affects the result of the processes performed by a service.
- Services are generally fully self-sufficient otherwise they need to depend on some other resources such as database or they may depend on the availability of other

services.

- Services promote all the specifications such as their interfaces, communication protocols, capabilities, and other supported policies.

1.2.1 Advantages of SOA

SOA as an architectural style with appropriate implementation promises the following benefits [20] :

- Applications are loosely coupled and are accessible from transparent locations.
- Provides higher scalability.
- Applications maintain interoperability using consistent connectivity between them..
- Better reuse of existing modules of business logic and reduces the cost of application development and integration.
- For service provisioning much limited dependency on vendors.

1.3 Web Services

Web services are software entities which allow communication between application over the Internet to exchange information in the form of extensible Markup Language (XML) [21]. Unlike browsing web pages from web servers, web services typically define business objects that execute a unit of work (e.g., perform a calculation, read a data source, etc.) for the consumer and wait for the next request. However the back end implementation is completely independent from the consumer. Actually web services publish its functionalities and interface using WSDL [22], SOAP [23], and UDDI like standard protocols. Two main characteristics of web services are their re-usability and loosely coupled nature. Each web service is accompanied with a Web Service Description (WSD) - a machine processable specification of the web service written in WSDL, defining the message formats, datatypes, transport serialization and protocol formats to be used between the service provider and the client.

1.3.1 Important Classification used by Web Services

A service can be described, discovered and invoked using standardized XML technologies in the current web services technology stack. There are four main components:

XML (eXtensible Markup Language) is standard way to describe data and an easy way to create information formats and electronically share structured data via communication network such as Internet.

SOAP (Simple Object Access Protocol) is an XML-based messaging protocol for exchanging information among computers.

WSDL (Web Services Description Language) defines an XML schema for describing a web service. A WSDL document describes a web service as a collection of abstract items called 'ports' or 'endpoints'.

UDDI (Universal Description and Discovery Integration) standard provides a mechanism for businesses to 'describe' themselves and the types of services they provide and then register and publish themselves in a UDDI Registry. Such published businesses can be searched for, queried, or 'discovered' by other businesses using SOAP messages.

A web service is a paradigm of an SOA with a clear-cut set of implementation choices. Normally the technology preferences are SOAP and the Web Service Definition Language (WSDL) which are XML-based. WSDL describes the interface (the 'contract'), while SOAP describes the data that is transferred. Figure 1.2 [2] illustrates the use of WSDL. The steps involved in providing and consuming a service are:

1. A service provider describes its service using WSD file written in WSDL. This definition is published to a directory of services. The directory could use Universal Description Discovery and Integration (UDDI). Other forms of directories can also be used.

2. A service consumer issues one or more queries to the directory in order to locate a service and determine how to communicate with that service, i.e. to fetch WSD.

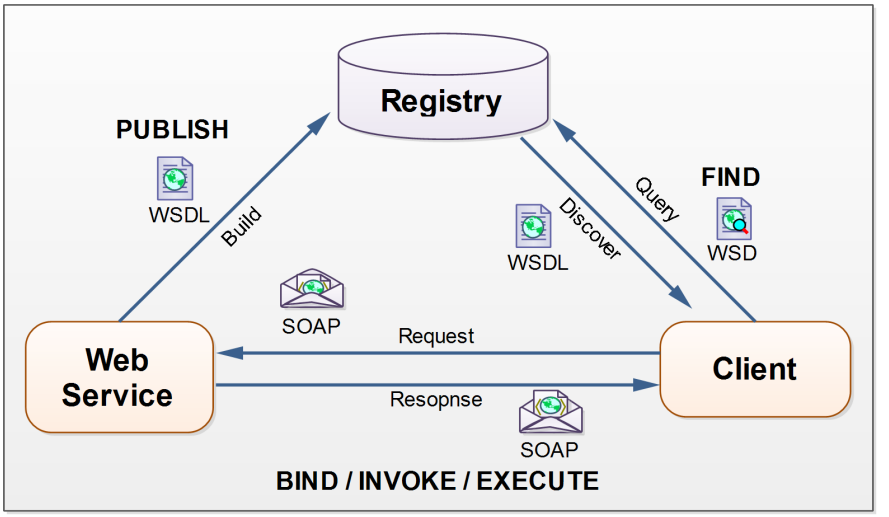


Figure 1.2: Working of Web Services with WSDL, SOAP, XML. [2]

3. Part of the WSD provided by the service provider is passed to the service consumer. This tells the service consumer what the requests and responses are for the service provider.

4. The service consumer uses the WSD to send a request to the service provider.

5. The service provider provides the expected response to the service consumer.

1.3.2 Static vs Dynamic Web service Provisioning

Web services provisioning is the procedure of attaching specific services to the fundamental functions of business processes. It represents an advantageous outline where services are dynamically discovered and invoked in a business process following their functional and non-functional proficiency. Hence, as a software unit web services need a platform or resources where it can be executed and the result will be returned. It may be possible that some of the applications or services will need extra resources which can be either provided by the provider itself or may be rented from third party resource provider. Thus, depending on how the web service is made available, there can be two methods for web service provisioning :

1. **Static web service provisioning** - Web Services are ready to use after the publication and deployment at more than one nodes. In such case, it may possible that the services are deployed at more than one locations and made ready to use after the publication. Such a policy may lead to very wasteful use of the resources and if the deployed instances has no consumer requests, the resources may remain under-utilized. On the hand, it may also be possible that there is huge increase in the consumer demand and the existing deployments fail to serve all the consumer requests.

2. **Dynamic web service provisioning** - Hence, the notion of deploying a web service on the basis of demand is followed. The web services are published over the Internet, no matter weather they are deployed or not. A consumer is capable of making a request for any of the service which is then followed by the decision of making appropriate deployments providing a cleaner and efficient use of the resources. In such a scenario the services presently in use only bear the cost of deployments. Not only this, once the service is deployed, it can be used by many consumers. Successive new deployments are made if the present deployed instances fail to provide a desired QoS. Similarly the services may be undeployed if the consumer demand subsides.

On comparing the above two techniques it may be observed that while static provisioning suffers from unconventional use of resources, it requires less overhead of resource managements in real time decision making process. On the other hand dynamic provisioning of services provide a better approach towards utilization of resources, though it involves extra overhead of monitoring the resources with time.

1.4 Motivation

Continuous advancement in distributed system increases association between remotely established distributed organizations which leading to newer challenges such as providing support for heterogeneity, scalability and availability in distributed computing. Researchers continuously made improvements to cope up with these challenges, thus some of the improved technologies like SOA, grid, cloud comes forward as major innovations and have been recently used by several organizations to provide solutions to

these challenges. Further advancement of the Internet creates the opportunity to reuse available online resources and also helps some of the popularly used technologies like grid, p2p and recently cloud to come out from the conventional nature of distributed computing for sharing resources among remotely distributed nodes. The emergence of the service-oriented model and Web Services paves the way for development of several grid and cloud computing platforms with a new approach for creating loosely-coupled dynamic distributed systems.

Initially distributed applications were divided into two categories- Client/Server model, i.e, centralized systems and decentralized systems better known as Peer-to-Peer systems. The boundaries between them were not crystal clear because classifying a system among the above two categories were determined based on the modes of discovery, availability and communication among the nodes or resources. Among the above three factors, discovery and availability decide the degree of centralization or decentralization of the system. Distributed applications in the context of web services which use XML specifics for its representation, advertisement WSDL [22], discovery and communication SOAP [23] use a centralized database called the *Registry* of web services maintained by UDDI [24] - a service discovery protocol. Hence we find a strong coupling of SOA framework with a centralized system leaving it vulnerable to single point failure of centralized systems. There have been various attempts to overcome this problem by replication of the information over multiple sites, but they fail to provide robust solutions with respect to scalability of the system as it incurs a lot of overhead to maintain the consistency of the system. Hence a system which can be scalable, fault tolerant and which can further increase the availability of the web services stands as one of the major motivation for this research.

Convergence of p2p and SOA opens up a new dimension to the evolving distributed computing frameworks by adding functionalities to deal with dynamic changes of resources during service discovery and service deployment.

There have been lot of research works in the area of web service/ resource publication and discovery and dynamic service deployment like DynaSOAr [3], WSPeer [4], HAND [16], P2PWeb [17] etc., which fail to exploit the basic advantages enabled by different computing technologies such as peer-to-peer computing, grid computing and recently cloud computing. It is believed that working on the three main agenda of *publish-find-bind* of web service technologies and blending them in p2p networks can

bring better performance results to meet the requirements on demand for dynamic service deployment framework with respect to availability, scalability, volatility of web services and other resources. Moreover, since grid computing enables us to geographically distributed resources for a single application. A collection of such ready to use resources at disposal, provided by Virtual Organizations (VOs) can prove to be beneficial for hosting the services, managing them and providing a good QoS as per the policies of grid environments. Hence, a grid providing a back end for a service oriented framework can facilitate dynamic provisioning of services.

1.5 Objectives

The objective of this thesis is to provide a distributed architecture which is reliable for consumers, flexible in terms of performance requirements, scalable and adaptable towards the volatility of resources available in distributed environment like grid or cloud. In this research a fully distributed SOA-oriented framework is introduced that offers loose coupling, robustness, scalability, availability and extensibility for large-scale grid systems.

One of the major challenges in research on SOA-centric distributed systems are resource/service discovery and on-demand deployment of computational entities (jobs or services) on dynamically acquired resources.

In relation to the objective stated above, this thesis focuses on proposing an architecture for provisioning Web Services in a distributed environment based on a decentralized, peer-to-peer (p2p) network. The thesis considers to bring in the following new possibilities in the perspective of distributed computing merging it with the advancement of SOA and web services technologies as stated below:

- Peer-to-peer communication between the nodes in the system so as to facilitate scalability and fault tolerance over the resources.
- Enabling on-demand provisioning of web services among resources available based on some minimum resource requirement to deliver better QoS.
- De-centralizing the registry of services among the available peers in the network.

- Scheduling the consumer requests in the host environment, thereby balancing the load among the available resources.
- Minimizing the traffic to the service repository/service provider.

1.6 Contribution

The contribution of this thesis is towards the overall design and evaluation of a service-oriented distributed system that exploits dynamic deployment of web services. This thesis aims at providing an integrated framework which supports dynamic web service provisioning in a p2p based service oriented distributed environment. The distinguishing features of this framework in comparison with other systems are:

- A robust framework adaptable to the volatile nature of the nodes in the network.
- Services as well as the service requests can be categorized as per service and user specifications to opt for the execution environments to achieve better performance.
- A proper approach towards balancing the load of the servers as well as serving consumer requests for better utilization of resources available in the grid.
- A distributed registry to fulfill the purpose of service publication and discovery thereby increasing the service availability at the same time.
- Reducing overhead of fetching the deployable code from the service repository and sharing it among the deployed instances to carry out successive deployments.

This thesis thoroughly investigates various aspects of dynamic web service provisioning within a distributed framework where dynamic service discovery over the geographically distributed area and resource volatility still considered as a serious issue. In this context this thesis introduces a p2p-based architecture which not only tries to take advantage from dynamic peer-to-peer framework but also comes forward as a means of sharing resources and gives a complete dynamic nature. The work builds on two main approaches described in the thesis - *managing the services and resources using a decentralized p2p Protocol* and the *Dynamic Torrent Service deployment protocol (DynaTronS) for dynamic or demand-driven service deployment.*

The main pillar in the proposed architecture is the concept of deploying a service only when it is required using p2p as its backbone. As opposed to job-based grid frameworks, the deployment cost in this case is one-time, and is shared across multiple invocations of the service. In this approach the idle resources in the network are used via service deployments on the basis of their capability and load factors. The resulting DynaTronS protocol is a unique concept of dynamic on-demand service deployment using Bit-torrent-like p2p file sharing (Bit-torrent) framework. This technique enables handling the resource volatility of the nodes/peers in the network maintaining the consistency in the architecture to serve its purpose. Simplicity, ease of use, effective use of bandwidth makes the proposed architecture 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 entire network, the techniques proposed here can actually improve the performance of the network.

1.7 Structure of Thesis

The thesis is divided into chapters discussing the different aspects in detail. Chapter 1 provides introductory overview with necessary background by giving a brief description of the related technologies. Chapter 2 provides a background studies about the present scenario and surveys a range of projects providing provisioning of web services. Chapter 3 describes the basic architecture of the proposed framework. Chapter 4 provides a different approach to facilitate the SOA framework with a registry in a completely decentralized fashion. Chapter 5 puts emphasis on balancing load of the resources in the fframework. Chapter 6 presents a unique approach towards deploying the services by utilizing all the deployed instances for faster deployments. Chapter 7 provides a conclusion and discusses the future prospects of the thesis.

Chapter 2

Background and Related Work

2.1 Introduction

The popularization of the Internet has made it possible for users to consume globally distributed services, which has also increased the use of internet resources. Distributed applications have undergone a lot of changes after the introduction of service-orientation and web services within the distributed environment using standards like WSDL [22] for service description and SOAP [23] for message communication. Usefulness of web services increases significantly because of its re-usability and loosely coupled nature. The emergence of Service Oriented Architectures and Web Services has contributed to the development of several platforms, like grid/cloud computing which offer a new way to create loosely-coupled dynamic distributed systems. Integrating peer-to-peer (p2p) based distributed architecture with frameworks based on Service Oriented Architecture (SOA) and Web Services [12] is emerging as a powerful technology for different industry standard 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 [9], Condor [10] and SunGridEngine [11] allow the construction of distributed applications over grid-based resources. Researchers have introduced many different architectural styles and standards, such as web services, REST over the past decade to ease the cost of discovery, deployment and maintenance, with varying degrees of success.

This chapter explores the background behind the thesis concentrating on the work

relevant to the dynamic discovery and deployment over service oriented architecture distributed framework. It looks into the service-orientation and available service-oriented technologies, preliminary concepts on grid and its problem, SOA, web services, p2p within the grid and cloud context.

2.1.1 Service Oriented Grid

Service Oriented Architecture introduces a conceptual framework for increased intensity on the application layer, but modular services need much better hold of the enterprize resources for actual implementation during its execution. SOA has no role in formulating the distribution of processes and management of enterprize resources although it uses common Internet protocols to enable interoperability. However, collaboration between the advantages of SOA and goals of grid computing following rapid advances in web service technology opens up new platform for architecture like grid to service-oriented, standardized, enterprise-class grid for future [12].

Grid Computing supplement SOA by allowing virtualization of data and resources along with the necessary mechanism to achieve flexibility in monitoring and discovery of resources. Grid provides an effective collection and management of distributed resources providing a suitable infrastructure to meet the needs of SOA concept [20].

For execution of a computational logic (i.e. a service), an enterprize must have a proper command of resources making them available as and when needed. Hence, a fusion of grid, seeking management of resources when combined with service-oriented framework, all build in one box can help creating a successful SOA. Conventional grid environments work with only few specifically designed application running on dedicated servers. Expanding the boundaries with increase in applications may reflect a scenario similar to SOA framework. On coalition of SOA with grid [25], the enterprizes will have to deal with a huge amount of services spanning a large number of resources, which can be well handled by utility grids.

2.1.2 Benefits and Issues of SOA and Grid Combined

An architecture developed on the premises of grid, while implementing the SOA framework can help us to achieve the following benefits [20]:

- Flexibility of the applications with independence towards platforms for execution.
- Usage-based accounting and control for global management of services and workload distribution.
- Dynamic provisioning of service based on easy adaptable load balancing mechanisms.
- Rapid development and deployment of services due to inherent nature of service orientation and virtualisation.
- Reduction in cost over easy management of resources.

Collectively SOA and grid can be thought of as complementary to each other. While grid offers a complete framework for distribution of information and resources which is one of the key features for SOA, at the same time SOA offers an modular approach for grid architectural solutions by offering software mobility which is well suited for the dynamic nature of grid environments. However, there can be certain inherent issues to be dealt in such a integrated approach as enlisted below [20]:

- **Security of the services** - the provider of a service may or may not wish to share the service with other organisations. Migration of the service code to a host should involve secure transmissions. The hosts where deployments are carried out must be trustworthy, to prevent any unwanted use of the service.
- **Policies of management** - management policies are required for service deployments and use of services should be figured out. The underlying cost model should also be well defined.
- **Maintaining the QoS** - QoS are to maintained for its prospective clients.
- **Communication protocols** - protocols are adopted for inter and intra node communication of resources to achieve interoperability of the services
- **Platforms and environments** - several issues also come up while deciding platforms and environments for execution of the services.

- **Monitoring resources and load** - health of resources and their loads must be monitored regularly for better performance of the system and also the volatile nature of the resources must be taken into account.

The above issues may be dealt with effectively with the availability of appropriate standards, techniques and tools. There are many standard bodies and agencies such as Open Group Services Infrastructures (OGSI) [7], Global Grid Foundation (GGF) [26], World Wide Web Consortium (W3C) [27], Organization for Advancement of Structured Information Standards (OASIS) [28], which contribute towards defining various standards and implementing toolkits. For example,

- OGSA(Open Grid Services Architecture) uses the web service technology to render grid services and hence defines their behavior, description styles, communication protocols such as XML, WSDL and SOAP.

- Globus Toolkit [29] provides by IBM an approach towards resource management, data management and information services, providing a base for developing services and applications.

Though SOA and grid share the same objectives, SOA can virtualize services and business processes. On the other hand grid virtualize and manages hardware resources depending on the present needs, they both can use same technologies (XML, WSDL, SOAP, UDDI) complementing each other. Such an integrated approach of SOA with grid can be a treated as a Computational Market Place. Though grid environment itself implements the concept of pay per use, specially in the case of utility grid environments [30] [31], leveraging computational resources on demand can be beneficial for provisioning/hosting services. From the SOA perspective, services are implemented with well defined business logic with small granularity. Hence, these services can be used by various applications or other services. These services can work as a new paradigm for the business model, encouraging IT industry for development of cost effective service modules in terms of execution times, security, low communication overheads of the services. Not only this, every invocation of a service can be associated with a charge to be paid by the consumer. Depending on QoS, the charges may vary within some specified range.

In recent times, there have been several approaches to build architectures to implement SOA-driven frameworks. Though these architectures follow the notion of *publish-find-bind* technology,they have undergone changes in trends of implementation. During the early days of implementation, service publication and discovery were achieved through UDDI (Universal Description, Discovery and Integration) [24] protocol.UDDI is an XML based, platform independent registry, to locate web services over the Internet. It is an open industry initiative defined by OASIS. A provider publishes the WSDL of a given service to UDDI, which could be used by a prospective client and hence bind to the service. However, UDDI due to its centralized nature, suffers from single point failure and remains a bottleneck for the architectures adopting UDDI. OASIS have tried to refine such technicalities by de-centralizing the UDDI registry over multiple sites. Nevertheless accomplishing a fully de-centralized registry has not been successful.

Even distributed framework like grid, service or resource discovery and deployments over the widely spread distributed environment dynamic decentralization has not been successful. Thus a shift towards the upcoming concepts in computing technology such as peer to peer(p2p) computing and cloud computing has also been observed. These computing technologies not only help in carrying out deployments in an efficient manner but also enrich the framework with there specialities making them different from the existing grid environments.

In the rest of this chapter, the discussion on the different architectures and frameworks developed prior to this work providing different approaches towards service deployments.

2.2 Architectures and Frameworks

In recent years, distributed applications have experienced a lot of changes after the introduction of SOA. Web services being the model for implementation have provided a diverse set of approaches towards its discovery and deployments abiding to SOA framework. Key research for Service Oriented Computing is focused on convergence of p2p and web services for service discovery. These paradigms can add a new dimension to the evolving grid computing framework by adding functionalities to deal with dynamic changes of resources during service discovery and service deployment. In particular

DHT based p2p systems like CAN [32], Chord [33], Pastry [34], Tapestry [35] can overcome disadvantages of early p2p systems like Gnutella [36] and Napster [37] and provide efficient and effective service discovery mechanisms. There have been lots of research work for web service publication and discovery and deployment. In the literature, mainly two types of approaches are considered for service publication, discovery and deployments, these are:

- Centralized
- Decentralized

2.2.1 Approaches to Web Service Publication and Discovery

2.2.1.1 Centralized Approaches

In the early of stage of Internet, when Information Technology market started to grow rapidly, Prof. J Yannis Bakos, in the year 1997 has proposed a strong approach towards electronic market place in his paper [38]. He has attempted to build up a model for searching services over the network in a cost effective manner. Actually his work inspired us to think that service discovery over the Internet in a cost effective way is one of the major issues for virtual IT market place. After that several frameworks were proposed to increase the ability for rapidly locating useful on-line services.

Earlier service retrieval techniques were simply based on table-based approaches for matchmaking between tasks and on-line services. JiniTM [39], eSpeak [40], Salutation [41], UDDI/WSDL use this approach. Keyword based web service search also seems to be back dated, because, it does not fully support the underlying semantics of web services [42]. Generally, WSDL which is a simple web service description language with standard UDDI registries support this kind of keyword based search. Semantic web service has been supported by ontology based language like OWL-S [43]. WSMO [44] is another popular web service language for semantic web services. There are mainly two type of registries which match industry standard, one is UDDI and another is Universal Business Registry (UBR) [45]. Although UDDI has been used by different industry standard mechanism for service discovery but both of these registries i.e. UDDI and UBR are fully centralized and often the problems faced by them include single point failure, low availability, low accuracy, and bad performance. A quality

driven centralized discovery approach uses *Web Service Broker* (WSB). In this model, service providers publish service information in the UDDI or search engines. WSB performs the tasks of gathering web services spread over the web with the help of the UDDI registries and also monitors their behavior based on various *Quality of Service* (QoS) metrics automatically without needing any human intervention. Its interface allows clients to express appropriate service queries based on its QoS. When clients receive responses regarding availability of services, they can invoke services. A refinement of the above proposed model uses *Web Service Crawler Engine* (WSCE) [46], a crawler that can capture service information from various accessible resources over the Web like search engines and service portals, to help in searching of web services on the Web.

Centralized approach mainly focuses on discovering web services through a centralized UDDI registry. Centralized registries can provide effective methods for web service discovery, but they suffer from problems associated with centralized systems such as single point of failure, and delayed delivery of notification for updated service description. Above issues merging with other issues relating to the scalability of data replication and handling versioning of services from the same provider are driving researchers to find other alternatives.

2.2.1.2 Decentralized Approaches

As Internet grows, number of available services also increases rapidly and thus handling those services through a centralized process seems to be quite impractical. So, in the past few years researchers concentrate to develop decentralized registries which are more reliable, more scalable and more flexible compared existing centralized registries. From this perspective, p2p comes forward as a suitable platform for the web service discovery in a distributed manner because it offers interesting technical aspects like decentralized control, self organization, adaptation and scalability.

Decentralized approaches are used to overcome the problems associated with centralized systems. There have been attempts to decentralize the existing UDDI registry, by replicating the web service information over more than one sites. UDDI over JXTA [47] and UDDI on top of DHT were among the few successful outcomes in this context. But the major drawback of such decentralization has been maintaining the

consistency over the replicated instances and synchronization between them.

Several systems to exploit DHT-based p2p approaches for resource discovery in grids have been proposed. Two important issues investigated by these systems are range queries and multi-attribute resource discovery. Range queries look for resources specified by a range of attribute values (e.g., a CPU with speed from 1.2GHz to 3.2GHz). These queries are not supported by standard DHT-based systems such as Chord, CAN, and Pastry. To support range queries over DHTs, a typical approach is to use locality preserving hashing functions that retain the order of numerical values in DHTs. Multi-attribute resource discovery refers to the problem of locating resources that are described by a set of attributes or characteristics (e.g., OS version, CPU speed, etc.). Several approaches have been proposed to organize resources in order to efficiently support multi-attribute queries. Some systems focus on weaving all attributes into one DHT or one tree. Some others adopt one DHT for each attribute [48].

In some approaches Gnutella-based dynamic query strategy is used to reduce the number of messages generated by flooding. Instead of all directions, this strategy forwards the query only to a selected peer. If a response is not returned from a direction, another round of search is initiated in the next direction, after an estimated time. For relatively popular contents, this strategy significantly reduces the number of messages without increasing the response time. Broadcast in DHT-based p2p networks adds broadcast service to a class of DHT systems that have logarithmic performance bounds. In a network of N nodes, the node that starts the broadcast reaches all other nodes with exactly N - 1 messages (i.e., no redundant messages are generated). Some approaches proposed for dynamic resource discovery uses a DHT for broadcasting queries to all nodes without redundant messages, and adopts a similar incremental approach for dynamic query. It reduces the number of exchanged messages and response time. Below here a few architectures which try to provide web services with decentralized registry approaches are discussed

Searching and publishing web services have brought forward a new direction by introduction of ontology. Introduction of DAML-S [49] allows to create ontology which makes it possible to search web services not only based on keywords but also by its content. Most of the distributed systems have been so far developed based on p2p architecture for semantic web services and they use otologies for automated service discovery, such as Hypercube ontology-based p2p system [50] and Speed-R [51]. Recent p2p sys-

tems like CAN, Chord, Pastry, Tapestry use distributed hash tables (DHT) as their basic component to create the peer-to-peer network and provide new favorable service publishing and discovery mechanism for Web Services. Japster [52], CoDiP2P [53], SpiDeR [54] are some well known works in respect to collaboration of p2p and web services.

2.2.2 Dynamic Service Deployment and Invocation

Many of the current research works target to improve searching for web services. However, not much work has been done for service deployment. Very recently, collaboration peer-to-peer (p2p) based systems with frameworks based on Service Oriented Architecture (SOA) and web services are emerging as 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) [8] and the Web Service Resource Framework (WSRF) [7] for creating Virtual Organizations (VO) meant for secure resource sharing among several users. This research focuses on combining the advantages of p2p network within a WS-based grid computing framework. Dynamic deployment of services is considered with utmost importance in grid or cloud frameworks to allow services to be deployed on the fly on available resources. This can be compared to job-oriented frameworks as in Condor [10], 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. Projects like DynaSOAr [3], WSPeer [4] , HAND [16] provide an infrastructure where services can be dynamically deployed on-demand over a distributed network thereby creating an ad-hoc computational Grid. In the next following section, a brief description of some of the systems is given-

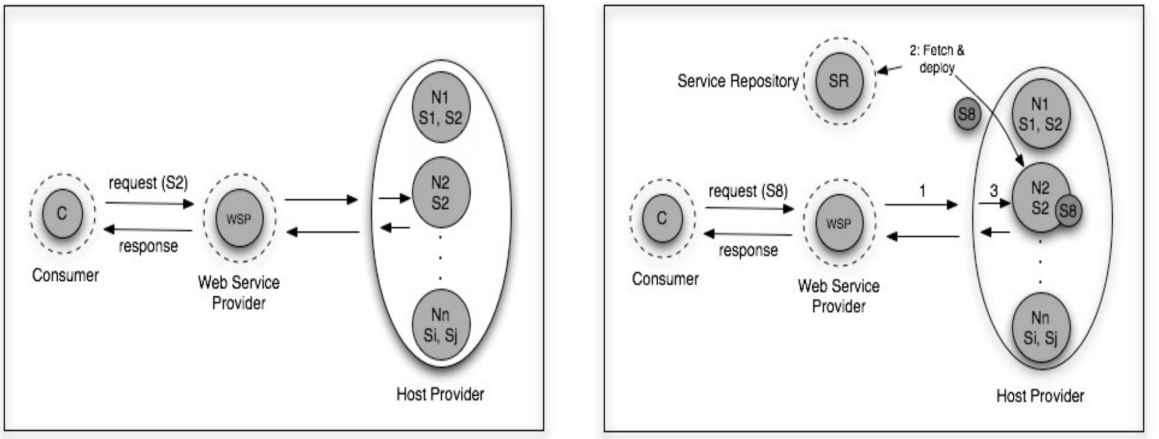
2.2.2.1 Centralized Approach : Dynamic Service Oriented Architecture (DynaSOAr)

Distributed job scheduling systems that can dynamically route client jobs to remote computing resources for execution are now widely available (e.g. Condor, Globus, SunGridEngine), and used by most Grid computing infrastructures. The job - a combination of code to be executed and the data on which it is to operate - is created by a client and submitted to distributed job scheduling systems which route it to a suitable host with sufficient the resources available to execute it. In recent years, an application is represented as a set of services using widely accepted standards like WSDL that communicate through the exchange of messages using protocols like SOAP. DynaSOAr project is a service-only approach that promotes service-oriented application design free of the job abstraction. It automatically deploys a service on an available host if no existing deployment is found, or if performance requirements cannot be met by existing deployments. A key feature of the architecture is that it makes a clear separation between two tires of it- namely, Web Service Providers, who offer services to consumers, and Host Providers, who offer computational resources on which services can be deployed, and messages sent to them are processed. These components are supported by Service Repositories that hold deployable versions of services, and Brokers that decide to which of a set of Host Providers a message should be routed.

DynaSOAr infrastructure is based on Tomcat/Axis for deployment of web services and GRIMOIRES [55] as the UDDI registry for serviced discovery. All these components are themselves realized as loosely-coupled web services, so enabling a wide range of deployment options. This approach has the potential to provide three main advantages over existing approaches that utilize both jobs and services:

1. It simplifies the development of applications and services by allowing designers and program developer to work entirely in a service-oriented framework;

2. It can improve performance by retaining the deployment of the code (in this case the deployed service) on a host. This allows the deployment cost to be shared over the processing of many messages sent to the service. In contrast, because jobs represent self-contained, often one-off executions, job schedulers lack the ability to share the cost of movement and installation of code across multiple executions; and third us,



(a) Existing Deployment (b) New Deployment

Figure 2.1: Process for service request in DynaSOAr Architecture [3]

3. The clear distinction between Service Providers and Host Providers enables new organizational/business models.

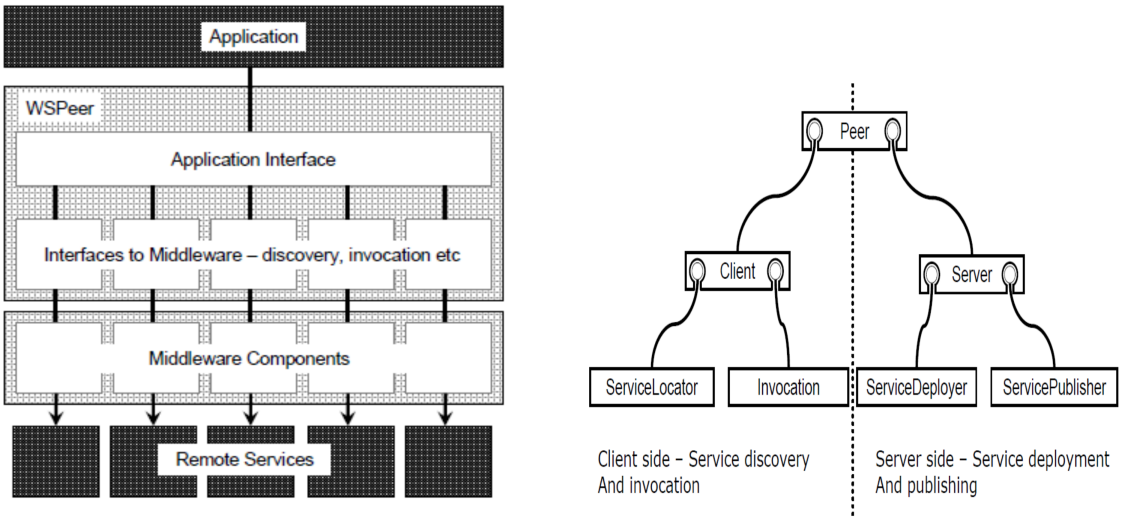
DynaSOAr provides a generic infrastructure for the dynamic deployment of web services. This is achieved by dividing the handling of the messages sent to a service between two components - a Web Service Provider (WSP) and a Host Provider (HP) - and defining an interface through which they interact. The Web Service Provider accepts incoming SOAP messages sent to an endpoint associated with a particular service and is responsible for arranging their processing. It does this by choosing a Host Provider and forwarding the SOAP message to it together with any associated Quality of Service (QoS) information. The Host Provider controls computational resources (e.g. a cluster or a grid) on which services can be deployed and messages to be processed. It accepts SOAP messages from the Web Service Provider (along with any associated information). If a response is generated after the processing of a message, the Host Provider returns it to the Web Service Provider.

Some of De-centralized approach for web service Deployment are discussed in the following section:

2.2.2.2 WSPeer

WSPeer [4] provides an interface for hosting and invoking web services. It allows application code to work with potentially varying and evolving service architectures while maintaining a consistent interface. Figure 2.2a [4] shows how WSPeer sits between application code and remote services acting as both buffer and interpreter. WSPeer has three main aims:

- To act as a complete interface for both publishing and invoking services, enabling applications to expose themselves as service oriented peers.
- To be applicable to a variety of network architectures including standard Web service architectures using technologies such as UDDI and HTTP, and p2p style networks.
- To be flexible and extensible, allowing users to develop application specific service capabilities of their choice.



(a) WSPeer Structure

(b) WSPeer - Tree of Interfaces

Figure 2.2: WSPeer Architecture [4]

WSPeer is constructed as a Tree of Interfaces (Figure 2.2b) [56]. Server side is responsible for publishing the web service as well as deploying the service when required. Whereas the client side is responsible for searching and invoking the web service. The main aim of the tree structure and the corresponding event driven notification system is to keep the elements of the tree loosely coupled, so as to allow individual nodes in the tree be replaced either at runtime or as part of a new implementation without disrupting the overall structure. WSPeer currently has two implementations. The first uses standard web service technologies with UDDI registry. The second uses a p2p framework called Peer-to-Peer Simplified(P2PS) [57].

HTTP/UDDI Implementation [58] [59] makes use of the standard technologies. Service Locator on the client side searches the UDDI registries for services after which a remote endpoint is invoked directly. The server side implementation launches a HTTP server that listen for requests. The server is launched once the application has deployed a service. The service publisher on the server side publishes the service to the UDDI registry.

P2PS Implementation makes use of a framework called Peer-to-Peer Simplified (P2PS) [57] and uses attribute based search as opposed to traditional key based search. P2PS uses XML to expose peers and services to the network. P2PS peer use abstract unidirectional communication channels called pipes. Such a level of abstraction is used as the peers are identified by their logical id instead of their physical addresses. Service publication and discovery follow a different pattern where peers broadcast there service information to other peers as advertisements to indicate a valid endpoint. Peers even search for a service by broadcasting messages to other peers. Another peer possessing the service, it returns an advertisement for a pipe which can be used to connect to the service. Both implementations have their own limitations. In case of HTTP/UDDI implementation, WSPeer suffers due to the tightly coupled UDDI registry. As a given node/peer in the system can act as both a client as well as service provider, a service may not be available to the system if the peer is not up, i.e. the peer is currently not in the network. Since P2PS implementation requires unidirectional pipes as it abstract communication channels, WSPeer requires two different pipes for both request and response. Also publishing a new web service and discovery of web service require a broadcast message hence leading to high network traffic. For a search query made of a service which is not available, no response is returned, hence keeping the client in

waiting state.

2.2.2.3 P2PWeb

P2PWeb [17] is one such infrastructure which gets its instinct for the de-centralizing approach towards UDDI registries whereas providing high availability of web service deployment infrastructure. The three main motives of P2PWeb are as follows:

- easy integration of web services into P2PWeb network,
- secure and de-centralized web services deployment,
- transparent location, load balancing and fault-tolerant p2p mechanism.

It aims at providing an easy way for developers of web services, to create them and just deploy them in the P2PWeb network. The infrastructure provides de-centralized structured p2p network, where every peer hosts a lightweight web server to permit deployment of web applications and services. P2PWeb platform has its foundation from middleware infrastructure to support peer-to-peer web application services, based on Distributed Remote Objects (Dermi) [60], a Component Model (P2PCM) [61] and Web Services Infrastructure (P2PWeb SOA). On top of such middleware, it provides a SNAP P2PWeb platform - a web application infrastructure over structured overlay network, by which developer can easily deploy any kind of J2EE compatible web application onto a worldwide structured peer-to-peer network. P2PWeb makes use of a decentralized registry D-UDDI [62], following a tag-oriented keyword based model for publishing the web service in the network. To provide a packaging model for web services, it offers P2PWeb service running on top of overlay network. After a P2PWeb service is deployed as one or more instance managed by P2PCM, it generates a P2PURI corresponding to it. A decentralized locator is used by the service provider to publish the service to the registry. A client requiring a p2pWeb service searches the registry, and the decentralized registry locates the service in the p2p network. Once the client obtains the P2PURI of the web service, it uses a p2p naming service provided by Dermi to resolve the specific endpoint and then binds to the service. The p2ps naming service resolves the P2PURI to make the closest instance available for the client.

In spite of the advantages promised by the infrastructure, it provides the services on the premises of static deployment made during publishing the service. In such a

scenario, a service may lay uselessly deployed even if it has no client requests to serve. Further p2pWeb employs a fully Java-centric approach abiding to J2EE platform.

2.2.2.4 Highly Available and Dynamic Deployment (HAND)

HAND [16] is highly available and capable dynamic deployment functionality based on the Java Web Services Core of Globus Toolkit. It uses two different approaches for dynamic deployment, these are: Service-level deployment (HAND-S) and Container-level deployment (HAND-C). In **Container-level deployment (HAND-C)**, the installation of any new service involves reloading (re-initializing and reconfiguring) the whole container. This is achieved by initially putting the container in reloading mode. The container will then return service unavailable error to any request that the container receives during the deployment. This step blocks until all currently executing requests finish or until a specified timeout expires. Then it stops and deactivates all services, resource homes, and so on. Cleanup operations to flush caches that might contain references to the resources and classes loaded by the original deployment are carried out. Then the deployment or undeployment scripts are executed. Afterwards, the whole container is reloaded. Lastly, the container is returned to the normal operating mode and it start accepting new requests.

In **Service-level deployment (HAND-S)**, one or more existing services are deactivated, new services are installed, and re-activate without reloading the whole container. It first checks the requested target service name. If it matches the already deployed service, then the class loader is switched to system level, or it uses its own class loader registered in the Service Package Manager (SPM). If requests being processed involve the services to be deployed, then deployment is suspended until those requests are completed or a timeout occurs. The services requested to be deployed is stopped if they are running. During this period, the container will return service unavailable to any request to the services. The stop operation will execute the persistency interface implemented by the services to keep persistency of the status of related resources. It also stops the services on which the pre-deployed services depend, and deactivates any related resources, and store the resources to persistency storage. Cleanup operations to flush caches that might contain references to the resources and classes loaded by the original deployment are carried out. Then the deployment or undeployment scripts are

executed and the working space context for the new services are created or updated and are registered to the SPM registry. Finally, the new services are initialized, activated and started

2.2.3 Loopholes in the Existing Frameworks and Future Direction

The DynaSOAr 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 implementations. In case of WSPeer, one implementation is based on UDDI which uses a centralized registry similar to DynaSOAr. The other implementation is P2PS-based [63], 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 the whole container, only the required service needs to be deployed.

Although these frameworks support dynamic web service provisioning, but none of these frameworks offer full dynamism over a volatile set of resources.

On the other hand, the advantages of peer-to-peer networks have often been tried to be leveraged in grid and distributed computing. In [39], the JINI framework was introduced for large-scale distributed computation and was based on the hybrid p2p network JXTA. The model proposed here comprises of separate levels of peer groups, such as monitors, task dispatchers and workers, which has limited similarity with the work proposed in this thesis. CompuP2P [64] 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 [53],

talks about computational resource sharing over a p2p network, rather than demand-driven service discovery and provisioning. GlobalStat [65] 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 [66], 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 Bit-Torrent protocol which implements a file sharing technique to download service packages from the existing deployments to cater to new deployments.

2.3 Summary

This chapter presents a detailed overview of the background related to the research work presented in this thesis. This chapter starts with a discussion on how advancement of SOA, grid, p2p, cloud provides efficient platform for web service provisioning. To understand the frameworks for service discovery and deployment of web services this chapter discusses some key research for web service publication, discovery and deployments. After that a detailed introduction of the existing work related to dynamic web service discovery and deployment over service oriented distributed architectures is presented here.

Detailed study of some existing dynamic service discovery and deployment systems for distributed frameworks like SOA, Grid, peer-to-peer computing along with their strengths and limitations are also discussed in this context. Based on this overview, this thesis aims to build up a new architecture for dynamic web service provisioning, overcoming the limitations of the existing architectures.

Chapter 3

System Overview

3.1 Introduction

This chapter presents the overall architecture for dynamic web service provisioning, based on the concept of peer-to-peer computing to enable dynamic on-demand service discovery and deployment on geographically scattered networked resources. To achieve this a SOA-oriented distributed architecture is proposed, which uses p2p technologies as a communication medium making it more flexible, scalable, reliable and robust compared to previous approaches used for dynamic service discovery and deployment in a distributed environment.

Virtual organizations introduced by Foster and Kesselman [8] allow collaborative sharing of computational and data resources over a wide area network. To support this, several tools, such as Globus, Condor and SunGridEngine have been built, who allow the construction of distributed applications over grid-based resources. Grid computing, peer-to-peer (p2p) computing and Service Oriented Architectures (SOA) have been major technologies adopted by various organizations to meet increasing demands of resource sharing by distributed applications during the last decade to ease the cost of discovery, deployment and maintenance with varying degree of success. One of the main target in Service Oriented Computing research has been on-demand or dynamic deployment of services or jobs on available resources. Job-based architecture such as Condor although provide dynamism in the sense that, it deploys jobs on the fly on available resources. However, dynamic discovery over geographically distributed area

and resource volatility were not considered in grid. In this context, we propose to introduce p2p-based architecture which provides a new approach for resource discovery and management suitable for dynamically changing distributed environments where smart decisions must be made to adapt to the changing environment and complex user requirements, and thus removes several redundancies and bottlenecks due to centralized registries and frequent resource alterations are removed. This chapter proposes a framework for dynamic service discovery and deployment in a distributed framework.

3.2 Basic Requirements of the Framework

The architecture presented here based on a service oriented framework which aims to serve its customer (using internet via an interface or by another application) through various web services, by dynamically deploying the services over the available distributed resources. To achieve this seemless service the architecture needed to be built with two aspects- from *customer point* of view and from *service provider's* point of view. When a customer wants to access web services which are made available by the service providers in the network, he can choose only those services which meet functional requirements and hence can make a service request to get back a desired response being totally unaware of the fact from where the request has been served. On the other hand from service provider's perspective, a registry must be maintained properly and efficiently in such a way that all the services made available in the network are published and any customer can use that service from any location any time. A static UDDI registry is thus not appropriate in the service discovery mechanism of a distributed web service architecture. After the advent of peer-to-peer (p2p) architecture, which is decentralized and distributed, a combination of the peer-to-peer concepts with those of web services appears to be the most promising model for a dynamic Web Service discovery and deployment framework. As the registry is de-centralized in nature, all the services available in the network are accessible from any of the service provider in the network. Depending on the service request made, a deployment of the same is carried out if no current deployment exists or the existing deployments are unable to serve the request within the desired QoS. Otherwise the client request is routed to an existing deployment using some scheduling strategy. Depending on the current network and service usage patterns a service provider may forcefully undeploy

one or more deployed instances of a service to optimize resource usage.

All the service providers need to be depend on their own or on third party resources for their service deployments. Thus, the resource providers joining the system with available resources need to allow service providers to carry out the required service deployments. Based on the service requirements *service providers* choose best possible resources for executing the web services. In this approach the idle resources among those distributed resources in the network are used for service deployments on the basis of their capability and load factors. The concept of separating s*ervice providers* and *resource providers* within this distributed framework also enhances the performance of data intensive jobs. In the real life scientific research fields, many research problems require lot of resources to analyze high volume of data. In such cases, the research organizations may try to get the computing resources from all over the world, and a p2p-oriented on-demand service deployment framework is likely to suit such requirements may be built to suit such requirements. Next section proposes an architecture based on the above mentioned requirements.

3.3 Overview of the Architecture

The overall architecture provides a new platform for dynamic web service provisioning by creating a network of computing resources using peer-to-peer computing as the basic platform. This is achieved by processing the incoming consumer request at different levels between three different components, namely Consumer, a Web Service Provider (WSP) and a Host Provider (HP), with a defined interface between them. Although each node plays a different role in the network, all the nodes are connected to each other and share their node information between each other. The HPs are resource owners in distributed environment, and provide required resources to perform computational intensive jobs with an abstraction of SOA framework. Combining the benefits of SOA and p2p computing, the key features of this architecture are as follows-

- Complete segregation of provider of services and provider of resources.
- All the nodes act as peers to each other providing p2p based service publication, discovery, deployment and management.

- Efficient load distribution mechanisms based on the concepts of grid computing environments.
- Resource discovery and allocation in a heterogeneous environment based on resource availability and QoS metrics of the Web Service.
- Scheduling of consumer requests following some basic scheduling algorithms like round-robin (RR) and least recently used (LRU).
- Faster deployment mechanism availing the benefits of Bit-torrent like protocol.

One of the key characteristics of this architecture is that it is composed of three distinct layers, each having separate functionalities and hence individually contributing to the workflow. Each layer is formed with a collection of nodes. Thus when a node joins the network based on its active role that it plays in the architecture(WSP or HP or Consumer),it announces the role and becomes part of some layer. It is clear that each level in this framework is distributed in nature and may consist of one or more peers so as to embed the dynamic features of a p2p network within the framework. Each peer node actively presented in the network must share their dynamic load information with all the other other peers to achieve better performance and maintain QoS for the clients and proper resource managements. Thus except the nodes in the consumer tier, all the nodes contributing in other two tiers 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, As shown in Figure 3.1, the three layers are described below:

1. **Consumer:** A consumer can request for any service available, from anywhere at any point of time. These services are made available to the consumer via the interfaces provided by the WSP using internet as a communication protocol.
2. **Web Service Provider (WSP):** Between the three layers, WSP plays the most important role, and aim to provide web services to its consumers while maintaining a de-centralized service registry. These services can be any computational entities, ranging from basic computational services to database services. Some of the main functionalities played by the WSPs are :

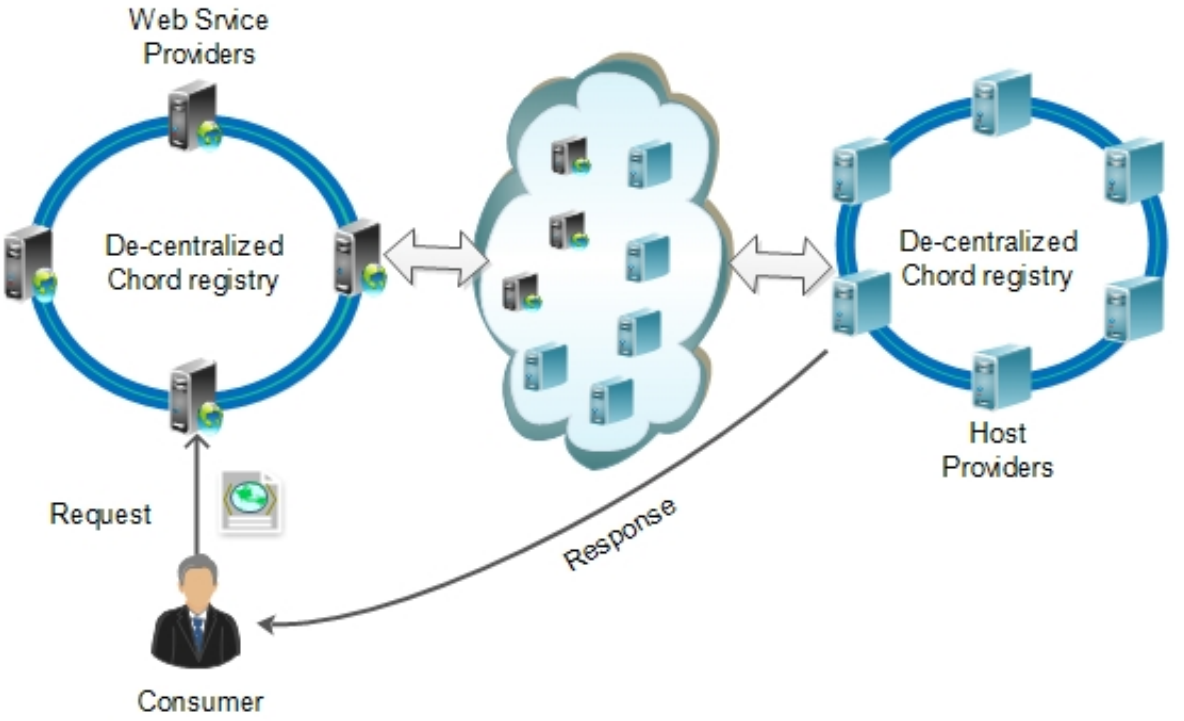


Figure 3.1: Overview of the Architecture

- Rendering a portal/interface for the Consumer tier.
- Establishing and maintaining the de-centralized service registry.
- Managing new and successive deployments of the web services on the basis of service metrics and current load information of the peers on demand basis.
- Scheduling the incoming consumer requests among the present deployed instances of a given service.
- Adapting the web service deployments and the service registry with respect to the volatility of the resources.

The WSP acts as the intermediary between consumers and computing hosts, to process service request while keeping the process completely transparent to the consumer. A WSP is responsible for managing service publication, handling consumer requests, choosing the best suited host for service deployment and/or request processing. During the request processing, WSP fetches every service

request form the Deployed nodes according to the present load of the host. The WSP maintains a local repository of Web Services and a list of hosts on which services are deployed in order to serve the consumer requests.

Each web service is accompanied with some information and service provisioning criteria like the service name, current status denoting whether it is deployed or not, as well as a set of minimum requirements with respect to the supported processor, required memory and operating systems for optimum performance. A service can have either of the 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 host 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.

Consumer requests are all made to the WSP. The WSP accepts the request and finds the most suitable HP to serve it.

3. **Host Provider (HP):** A *host provider*, provides resources and platform for service execution. All the *host providers* are connected as a decentralized hybrid p2p network and control the computational resources on which the services can be deployed and requests from consumers can be processed. The main job of a *host provider* is to maintain the computational resources in a distributed environment, such as a grid or a cluster or cloud, on which services can be deployed, and requests for those services can be processed. *Web service provider* forwards the user request to the *host provider*, HP accepts the massages on behalf of the services hosted by it, process the request and sends back the response to the Customer. Some of the main functionalities played by the HP are-

   - It must be able to receive service requests forwarded by the *service provider* and process them accordingly.
   - It must maintain a list of child *host providers* under its domain to whom the service request can be routed to. In order to achieve this it must allow other hosts to register and deregister with it.

- It must have the knowledge of the services supported by it.

The *host providers* are implemented in such a way that each *host provider* has the knowledge about all the other *host providers* and is capable of scheduling the processing request on any of them. It is also capable of downloading a service package, if multiple copies of the same service exist on different nodes in order to perform some load-balancing. Various scheduling algorithms such as Least Recently Used, Minimum Load First, Round Robin Reloaded are utilised within the host provider to make decisions about routing a request to a specific instance of the service.

3.4 Formal Description of the Framework

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 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-

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

where,

M = Map Size, nodes in the system
$M = \{M_1, M_2, M_3, ..., M_n\}$
P = Node Properties
NT_m = A mapping between the nodes and WSPs and HPs according to their properties
$NT_m : M \rightarrow P(WR) \wedge M \rightarrow P(HP)$
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
$D_m : WS \rightarrow HP$
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.

3.5 Functional Overview

The overall architecture described above is implemented with five activities, namely, node establishment or node joining, service publication, service discovery, resource discovery and service deployment. The system is implemented as a composition of independent modules taking care of different aspects of WSPs and HPs, which communicate amongst each other to meet some common goal.

Each of these nodes are distinguished by a set of node properties such as node name, node category - i.e. whether the corresponding node is a WSP or HP, and other static information such as host operating system, processor frequency and physical memory, all maintained within a configuration file. The nodes taking part in the architecture as WSP or HP are connected to each other in a p2p fashion by using JmDNS [67] and the node properties are known to each other.

The architecture discussed above is formed by an application running on nodes

which facilitates the nodes to join the network 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* and *Web Server* as depicted in Figure 3.2. Each module bearing a unique responsibility collaborates among each other to achieve the required goal. The functionality of each module as shown in Figure 3.3 is described below:

- **Service Monitor** - a module responsible for monitoring the service and its deployment status, hence mange the service deployments keeping track of the deployed instances.
- **Node Communicator** - a module that establishes connection to the network, registers the node and thereafter handle inter and intra-node communication of events.
- **Load Balancer** - a component that monitors the current load information based on free memory, CPU utilization of the nodes in the network to select a suitable HP to carry out deployments and/or scheduling consumer requests.
- **Registry Handler** - this module, in collaboration with the *Service Monitor* (only in case of WSP), maintains the registry to keep upgraded information about all the web services in the network.
- **Deployment Manger** - a module that handles the deployments od downloaded service code (i.e. WAR files) in an efficient manner as and when notified by the *Node Communicator*.
- **Web Server** - it provides a ready platform for execution of the web services being deployed on HPs. In the case of WSP, it is used to provide a portal/interface to the clients or other application to bind to the web services available in the network.

The hierarchy followed by the modules as depicted Figure 3.2, can be explained as follows: *Node Communicator* forms basis for the nodes to join the network and handles the communication, so it is placed at the lowest level. The *Deployment Manager* needs

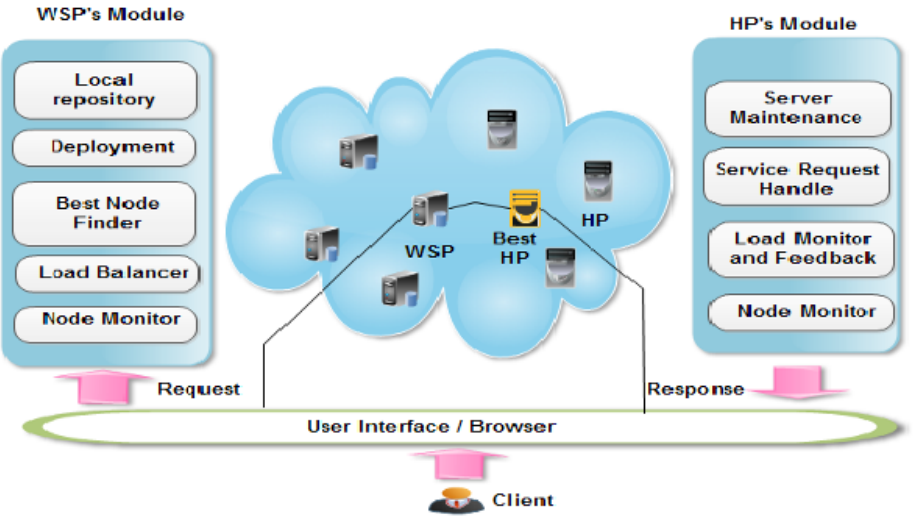


Figure 3.2: Basic Architecture as an Application

to serve downloading of files which requires the knowledge of both the service packages as well as the information of peers, hence is stacked in between the *Node Communicator* and *Service Monitor* modules. Monitoring the services being the most important part for all the nodes, the *Service Monitor* is placed in the midst. *Registry Handler*, which requires only the service information to publish the services is placed directly above *Service Monitor*, at the top most level, providing a list of web services for the interface enabled by the *Web Server*. Further the *Web Server* requires a direct contact with the *Deployment Manger* to carry out deployments at HPs once the web service package download is complete. On the other side the *Load Balancer* remains in touch with *Node Communicator* to send, receive dynamic load information of the nodes to each other; with *Deployment Manager* to download files based on the load of a node; with *Service Monitor* for selection of best nodes for a given service and also with *Registry Handler* which help in scheduling the requests. The modules of the application are activated and deactivated depending on the role the node plays in the architecture.

The architectural workflow described above is achieved by a series of steps that may occur in the process of deploying a web service starting form *Nodes Establishing the Network* followed by *Service Publication and Discovery, Resource Discovery, Dynamic Service Deployments* and *Service Request Scheduling.* In the following subsections details of the events that take place for each of the steps mentioned above are provided.

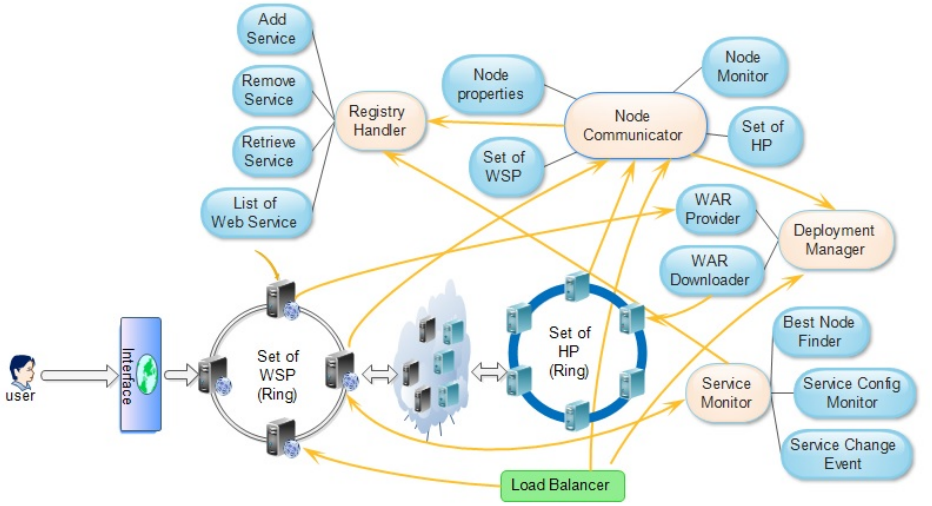


Figure 3.3: Components in the Architecture

3.5.1 Network Establishment: Node Joining

The main advantages of this architecture is its scalability and robustness while enabling on-demand service provisioning and resolving the problem of resource volatility by providing a network of interconnected nodes. To become part of this distributed system a node must need to join the network via Internet. Thus a node willing to join the network needs to register itself by sharing a detailed node configuration information called *node properties* as shown in Figure 3.4, similar to a [key, value] based property file with keys enlisted below:

- *Node Name :-* to identify the node in the network.
- *Node Type :-* WSP or HP, i.e. the role in which it wants to represent itself in the network (to be specified explicitly).
- *Host Operating System :-* used by the node.
- *Processor information :-* such as number of cores, frequency.
- *Physical Memory :-* Total physical memory installed.
- *Other :-* information such as operating port numbers, IP addresses etc.

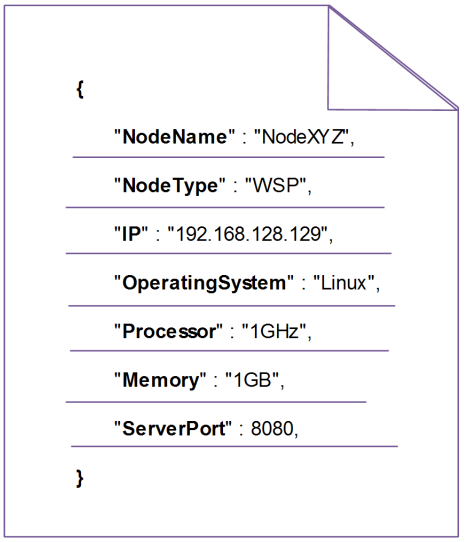


Figure 3.4: A sample Node Properties File

A node willing to join the system initializes the *Node Communicator* module. *Node Communicator* creates the basis for the incoming nodes to join the network and also maintains communication link between each node. As soon as the node registers itself with the network, its *node properties* are made available to all other nodes in the network using multicast DNS governed by a specific communication protocol(TCP/IP), enabling them to act as peers. Thus using this broadcast technique each peer maintains a list of all WSPs and HPs in the network for its own perusal. Thus the main function of the *Node Communicator* module is establishing inter and intra-node communication of events. After a node successfully joins the network the control of the events passes to the *Service Monitor* component, which is responsible for publishing the services and making them discoverable based on their status. *Node Communicator* waits until some inter or intra-node event occurs and at the same time monitors the changes in the network (if any) to adapt accordingly.

A node, when leaves the network, *Node Monitor* - a submodule of *Node Communicator*, at all the peers detect this change by the receipt of a *node removed* event, multicast by the node just before leaving the network. This change in network is then communicated to their respective *Service Monitor* so that appropriate actions are

performed to maintain the consistency of the system.

3.5.2 Functioning of a Node: with p2p Based Communication Protocol

At time of joining, each node assumes the roles in clearly two categories, i.e., either WSP or HP. Now on the basis of the categories, these nodes communicate with each other to accomplish different tasks. The communication is based on p2p computing technique. Thus each node acts as a peer to other. There can be following basic tasks leading to communication among the peers to accomplish functioning of the overall architecture.

- *Registry Maintenance* - All the WSPs maintain a registry to provide services to the consumer. In the architecture p2p Chord protocol is used to maintain this registry. This is a distributed registry. Thus each service is being owned by some WSP, and its current status is updated in the registry of its owner, that is then propagated among all the WSPs in a structured fashion.

- *Service Monitoring*- This task is carried out by WSPs, which are responsible for publishing the service and making it discoverable based on its status.

- *Service Deployments*- Whenever there is a requirement of a web service deployment, a suitable HP is selected from the list of HPs and a deployment message is forwarded to the selected HP.

- *Load Balancing* - It is carried out in the direction of HPs to WSPs. All the HPs send their dynamic load information to the WSPs to help in decision making process for service deployments and request scheduling for proper load distribution and resource utilization.

3.5.3 Decentralized Registry

In this architecture *registry service* is provided in distributed manner based on p2p protocol. When a service developer decides to make a service available via the Web Service Provider(WSP), the deployable service code and its information is uploaded

to the registry as a result of which the registry is updated with this information. 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 versions of service names, that are mapped to node identifiers generated as hash values of nodes IP addresses. Incorporating the benefits of distributed registry to the existing p2p network, the architecture decentralizes the registry among the WSPs.

3.5.4 Publication and Discovery of Services

The node joining the network as WSP is mainly responsible for performing the basic steps of service publication adhering to SOA framework. As discussed earlier, all the WSPs maintain a distributed Chord registry for managing the web services. Thus a WSP publishes all the services it owns to the registry maintained over the network in a de-centralized manner, whereas keeps deployable versions of the web service confined to its local repository.

WSPs publish services by providing the corresponding service in the form of a WAR (web archive) file. Each service is characterized by a set of service configurations and Service List defined in files. Both these files are metadata files corresponding to each service, with some identifiers which are discussed below. Basically this all operation maintained by *Service Monitor*. Each new service made available to the system makes their attributes known with the state as "available".

3.5.4.1 Requirements of Web Services and their Configuration

There is a service level agreement for all the web services provided by the WSPs. Developers during service development create and configure those services such a way that they meet some basic requirements before the deployment process. Thus web service management stand to be one of the most important issue for the architecture.

A WSP can launch a new web service providing the *web service package* (executable code), packed as a Web Archive File (WAR file), along with a *web service configuration file* (Figure 3.5) annotating the web service with certain parameters such as *name of service*, a *short description* of its functionalities, *service state* i.e. its deployment status (available or deployed) and *minimum deployment requirements* such as processor frequency, memory and host operating system required for proper execution; collectively

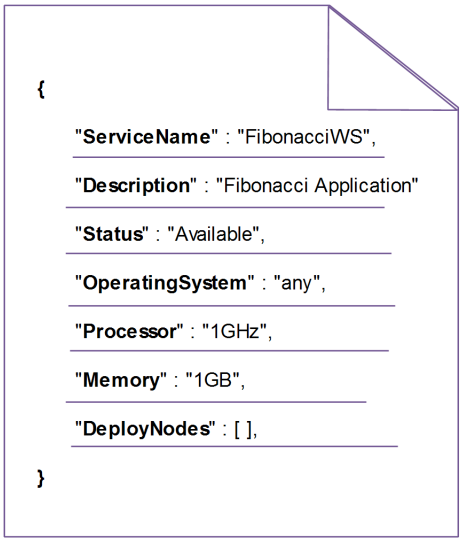


Figure 3.5: A sample Web Service Configuration File

all known as *service metrics*.

A web service is published to the registry by *Registry Handler* with its *status* set to *available*, rendering the service discoverable over Internet, after which consumers can search for the service and make requests to the system. For the first web service request, its *status* is changed from *available* to *deployed*. A change in the configuration of a given service triggers the *Service Monitor* by a *Service Config Change Event* which is processed by *Service Change Event* based on the processing logic depicted in Algorithm 1.

Among the different events portrayed in Algorithm 1, events ADDED and CHANGED drive the deployment process and hence invoke DEPLOYED or UNDEPLOY events. To remove a service from the current deployed instances, REMOVED event is triggered. Depending on the node type, i.e. WSP or HP, the event is delegated in appropriate direction. If the event takes place on a WSP, since WSP never deploys a service within itself, it forwards the event to a selected HP to carry out the deployment process. If the node is a HP it deploys/undeploys/removes the service as per the event type. After a service request is made to deploy a service, WSPs try to choose a suitable HP in order to deploy the service. This is where the next step, Resource Discovery comes into play.

Algorithm 1: Algorithm for Service Change Event

```
1  begin
2      serviceName ← getServiceName();
3      switch ServiceChangeEvent.getType() do
4          case ADDED:
5              // add or update service;
6              status ← addService(serviceName);
7          case CHANGED:
8              // update service;
9              status ← updateService(serviceName);
10         case REMOVED:
11             // undeploy service from node and remove from repository;
12             status ← undeployAndRemoveService(serviceName);
13         case REDEPLOYED:
14             // get service name and deploy service on a selected host;
15             node ← selectNode(nodeArray);
16             status ← deployService(serviceName, node);
17         case UNDEPLOY:
18             // undeploy service from node;
19             status ← undeployService(serviceName, node);
```

Service Monitor hence communicates with *Load Balancer* to select a suitable HP and carry out the deployment process.

3.5.5 Resource Discovery

The node joining the network as HP is mainly responsible for providing resources and platform for service execution. Nodes acting as Host Providers play an equally important role in the functioning of the architecture along with the WSPs. Since the aim is to serve the consumers in best possible way, a correct choice of the HP is required to meet the service requirements. Thus, resource discovery is an important task, Resource discovery process is performed in two steps-

Resource discovery process done in two steps-

- Dynamic requirements matching based on basic service requirements
- Load balancing

3.5.5.1 Dynamic Requirements Matching Based on Basic Service Requirements

When a user tries to execute a service by paying some amount, he or she expects the response time to be bare minimum along with some quality of service (QoS) requirements. In order to achieve this, the web service provider must state these requirements in advance or the user can mention it explicitly while executing a service so that when the service is deployed it chooses the right host provider based on these requirements. The requirements are specified by the web service provider using a service metadata file in which the minimum service requirements are specified as *service metrics*. Once an incoming consumer request triggers the deployment process, the WSP uses *Best Node Finder*, a submodule of *Service Monitor*, to search for a suitable HP with enough resources to deploy the service. The search process employs matchmaking on the basis of the service metrics with the node properties as demonstrated in Algorithm 2 to select a HP with sufficient The node acts as a peer joins the network as WSP mainly resposible for performing the basic steps of service publication adhering to SOA framework.resources to meet the service requirement for optimum performance. The first node which satisfies all the minimum service requirements and is not heavily loaded, is returned by the algorithm.

Algorithm 2: Algorithm for Best Node Finder

```
1  begin
2      serviceMetric ← ServiceToDeploy.getServiceMetric();
3      nodeProp ← NULL;
4      foreach NodeProperties ∈ HPNodeList do
5          nodeProp ← NodeProperties;
6          if nodeProp.getOperatingSystem() == serviceMetric.getOperatingSystem() then
7              if nodeProp.getCPU() >= serviceMetric.getCPU() then
8                  if nodeProp.getMemory() >= serviceMetric.getMemory() then
9                      // service metrics satisfied.  Check for Node not loaded.
10                     if LoadThreshold(nodeProp.getNodeName()) == FALSE then
11                         break;
12     return nodeProp;
```

The node acts as a peer joins the network as WSP mainly responsible for performing the basic steps of service publication adhering to SOA framework.

It may so happen that no such HP is available for the given service, in which case the algorithm returns *NULL*, and the deployment process is withdrawn. Otherwise a

deployment event is forwarded to the selected HP. The *deployment event* consists of the certain important information which help in the deployment process, as listed below:

- **Web Service Configuration File** to provide a detailed information of the web service.
- **URI** of web service package to facilitate the download from the selected HP.
- **Length** of web service package in bytes.
- **Checksum** of the web service package, to ensure its integrity.

3.5.5.2 Load Balancing

Whenever a Consumer sends a request to execute a service, the service is deployed on any one of the host providers based on the resource requirements and current load information for the first time. All the subsequent requests to that service are then redirected to that host provider and results are made available to the consumers. Minimum response time and quality of service desired by the consumers are the two major concerns when a request is handled. A load balancing approach is adopted for this architecture. Initially a simple threshold based load balancing approach has been implemented. Every host provider has a threshold or load criterion defined. It depends on parameters like memory and CPU usage of the overall system, CPU and memory usage per service, etc. Once a host provider exceeds that criterion, a new host provider node with a load lower than the threshold value is found from the remaining nodes in the network. Then the service is deployed on this new node. From then on, all the new consumer requests for the particular service are redirected to the new node and results are redirected to the clients from that node. Other load balancing approaches have also been explored as discussed in Chapter 4.

Upon receiving a of deployment event at the HPs end the *Deployment manager* is triggered to carry out the service package download (discussed in Chapter 6) and the service is deployed. Since the performance of the entire architecture depends on the QoS that can be delivered by the HPs, this step plays an important role in the service request flow to choose a suitable HP from the HP tier and trigger deployment.

3.5.6 Scheduling Strategies

Scheduling the incoming consumer requests to the current deployed instances is one of the important issue to meet the consumer satisfaction. A service is provided with an exposed endpoint in the system after the deployment by which the consumers can easily request for the web service. A proper distribution of load, i.e. web service requests is delivered by the use of some scheduling algorithms adopted by the WSPs. These scheduling strategies are are used to increase the performance of the overall architecture by achieving better response times, proper utilization of resources and also increasing the service availability. 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 the strategies/algorithms like- *Round Robin Reloaded (RRR)*, *Least Recently Used Reloaded (LRUR)*, *Minimum Loaded First (MLF)*.

This time slice based scheduling strategies provided by the *Load Balancer*, to distribute the service requests among the deployed instances. The algorithm is invoked at the end of every time slice (pre-decided), after the first deployment for the web service. A particular HP is selected as a *best node* from the current deployed instances and all the incoming consumer requests are routed to this *best node*. When the time slice expires depending on the algorithm used, a *best node* is selected again to serve the consumer requests for the next time slice and so on.

The algorithm is also responsible to trigger new deployments if the existing deployments become unavailable or fail to serve the consumer requests within some desired QoS. This decision is taken based on the present load of the deployed instances. If any HP among the deployed instances is found to be loaded, the scheduling algorithms does not assigns them as *best node* to serve consumer requests because they may fail to provide the response within the expected time violating consumers QoS. New deployments are triggered only if all the existing deployments are found to be loaded at the present instant. By the use of such scheduling strategies, the architecture tries to ensure proper distribution of load, by monitoring the deployed instances on the basis of their dynamic load information. Since new service deployments and selection of a best node are based on the present load of a HP, a clear idea on how the load is calculated and gathered, turns up to be a major concern. In the end, a service request scheduled

to the present best node processes the request and returns a response (if any) back to the consumer.

3.5.7 Dynamic Service Deployments

Web service requests are initiated from the consumer tier. A consumer can make a web service request only after web services are made available by the WSP tier. A WSP when joins the network it publishes the services to a de-centralized registry. Once the web service is made discoverable over Internet and ready for use, a proper endpoint is provided by the interface. A consumer can make a web service request to any WSP, via the interface. Once a consumer makes a web service request, the control is transferred to the WSP tier. A consumer request made to a WSP may have three types of interactions depending on the status of the service as described below-

- **First time deployment**: When the *status* of the web service is set to *available*, it implies there exists no deployment of the web service in the system. In such a scenario, first time deployment of the web service is required. WSP uses some criteria to select an HP which is best suited for the service, and the HP is directed to download the code from the repository and deploy it. The *status* of the web service is then changed to *deployed*.

- **Fresh deployment when current instances are busy**: When a service has its *status* set to *deployed*, but the existing deployed instances are not able to serve the request (the nodes may be overloaded), a need for a fresh deployment arises. This decision is taken at the WSP on the basis of a set of dynamic load information collected from the HPs. The consumer remains unaware of the fact that a new deployment is made.

- **Request for an existing service**: The consumer requests for a service already deployed on multiple nodes are redirected by service provider, to the currently selected best HP, based on some scheduling strategies.

All the consumer requests after reaching the HP tier are served and a response for the service is returned back to the consumer. To achieve dynamic deployments the architecture adopts two different modes to accomplish download of the service packages:

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

2. **P2P Download :** the HPs download the service package by requesting chunks of the WAR file from more than one site. 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 Dynamic Torrent Service deployment protocol (DynaTronS) which we discussed in the Chapter 6.

At the HPs end, when a deployment event is received, the *Deployment Manager* module is triggered to download the web service package or the WAR file of the web service and deploy the web service performing the basic steps of the application's *Web Server*. The architecture uses a light weight Jetty server, embedded in the application itself, for all its individual peers, which relieves architecture from the need of an external container to carry out the deployments. Once the web service is fully deployed and is ready to use, it can serve the incoming consumer requests and return desired response(if any).

The deployed instances are then used by the WSP to serve further incoming consumer requests, and new deployments are carried out if a WSP figures out that its existing deployments are not able to meet the consumer demands. The deployment process is termed as dynamic deployment because the first time deployments are carried out only when consumer requests are made i.e. when there is a need of a service execution. Furthermore successive deployments are carried out with increasing number of consumer requests rendering the architecture scalable with consumer demands providing efficient resource utilization and management. The decision to carry out new deployments is taken at the last phase of the workflow, i.e. *Service Request Scheduling*.

3.6 Summary

This chapter introduces a brief overview of the propose architecture for dynamic web service provisioning based on peer-to-peer networks. The functionalities of the proposed architecture for enabling dynamic on-demand service discovery and deployment on geographically scattered networked resources are also discussed in this context. A three tire architecture from consumers to Web Service Providers (WSP), and to Host

Providers (HP) is discussed here. The architecture targets to serve its consumers (using Internet via an interface) using various web services provided by Web Service Providers, by dynamically deploying the services over the available Host Providers. How a web service is published by a web service provider and how a consumer can get that service, each and every functionality of service publication, discovery and deployment is briefly discussed in this chapter. A formal description of the architecture also discussed this chapter. Thus, this chapter provides an SOA-oriented distributed architecture which uses p2p technologies as a communication medium making it more flexible, scalable, reliable and robust compared to previous approaches used for dynamic service discovery and deployment in a distributed environment.

Chapter 4

Dynamic Web service Discovery and Deployments using De-centralized Registry

4.1 Introduction

A fully distributed service oriented framework which uses peer-to-peer (p2p) techniques at its core has been introduced in Chapter 3. The framework offers loose coupling, robustness, scalability, availability and extensibility for large-scale distributed systems. The use of peer-to-peer concepts and techniques allow more flexibility and dynamism when compared with previous approaches such as DynaSOAr used for dynamic service deployment in distributed environments. A major change in the implementation of the framework is the use of distributed registry service for dynamic deployment of services. The primary objectives of this distributed registry for dynamic web service provisioning are mentioned below:

- To provide a distributed environment overcoming issues associated with centralized registry based architectures.
- To allow clients and service providers to mention specific requirements for a service in order to achieve desired quality of services 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 framework 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 collaborations 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 metrics of the Web Service. In this chapter the implementation of the proposed architecture on a structured overlay peer-to-peer (p2p) network is discussed using Chord [33] protocol. The major goals of the implementation are as follows:

1. de-centralizing the service registry in a structured manner,
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 [17], though the registry is decentralized as multiple single-site registries, a service may become unavailable if the hosting WSP goes down. To increase the service availability, it is proposed 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 chapter provides a robust approach towards dynamically deploying web services, as well as decentralizing the registry to meet the increased availability of the services and maintaining its scalability at the same time.

4.2 Approaches for Dynamic Web Service Discovery and Deployments

The phenomenal growth of Internet and web technologies has led to the development of more and more complex services with complicated interaction patterns, spread across

different organisations, platforms with different multi-layered architectures, which has in turn led to an immense complexity in cost-effective service discovery and management. In this context, dynamic service discovery and deployment are two of the most important issues for any service oriented framework. There are existing toolkits for such purposes, such as the Monitoring And Discovery System (MDS) [68] as a component of the Web Services Resource Framework (WSRF). However, an MDS Index Service registry is functionally similar to a centralized UDDI with some additional flexibilities, and is still limited in terms of exploiting and reflecting the complete dynamism of a Grid framework. In this section, the various approaches to dynamic service provisioning that are in existence are discussed.

- **UDDI-Based approach:**

  Most service oriented architectures use the UDDI standard for creating service registries. Since UDDI is based on XML, platform independence and interoperability 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 common 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 synchronization to maintain a uniform view.

  The concept of dynamic service deployment on available resources using UDDI was introduced by DynaSOAr 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 WSPs 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.

- **P2P-Based Approach:**

  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. 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 overlays generally make use of DHT [69] to index 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. Chord [33], CAN [32], Pastry [34], Tapestry [35] are some examples of such networks. From the above mentioned DHT implementations, the architecture in this thesis 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** [33]- 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.

  This chapter provides a brief overview about the implementation of the architecture 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.

As discussed earlier, WSPeer emerged as an interesting framework which combines the benefits of p2ps decentralized resource sharing with the XML based web service technologies. One of the major advantages of WSPeer is that consumers and service providers are located in remote places and can use it as an interface. This architecture provides the dynamic deployment facility in 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 [57] 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. However, this bottleneck is removed in the P2PS implementation. Since the peer can act as a service provider, as well as service consumer a service endpoint is made available only when the node remains available in the network.

In recent years, there have been a lot of research works on efficient ways of service discovery in Grid and SOA based frameworks applying different approaches such as keyword 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 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.

In the following sections, a brief overview of peer-to-peer networks and their advantages are discussed.

4.3 Peer to Peer Networks

P2P networking is defined by the Intel Working Group as the sharing of computer resources and services by direct exchange between systems [70]. These resources and services include the exchange of information, processing cycles, cache storage, and disk storage for files. In p2p, all the nodes act as simultaneous clients and servers are able to provide and consume resources at the same time. A p2p network is void of any centralized data sources. Any node can initiate a connection and a node can join or leave the system at any time without affecting the system performance thereby increasing the scalability. The malfunction on any given node does not affect the overall system thereby increasing the overall reliability of the system. Now from the beginning, several forms of architectural styles have been design on peer-to-peer framework. Two types of architectures, which categorizes peer-to-peer networks are structured and unstructured p2p. But in detail, p2p can be categorized in the following architectures -

Semi Centralized, Unstructured Decentralized, Decentralized or Hybrid p2p, Structured Decentralized

- Semi Centralized : Some dedicated servers are there to manage all the peer nodes. Napster is the earliest example of this type.

- Unstructured Decentralized : Purely Decentralized, all peer nodes have the equivalent capabilities, like Gnutella 0.4v [71].

- Hybrid p2p or Super Peer p2p : This is also a complete decentralized architecture for p2p but not all the nodes have the same capabilities. There are some extra powerful nodes called 'Super Peer Node', maximum message routing taken care of by this node, like Gnutella2.

- Structured Decentralized p2p :This is also a complete decentralized architecture and all the nodes have the same capabilities, but here all the peers organized in definite structure using some criteria or algorithms. Generally Distributed Hash Table (DHT) is used for building this type of structured peer-to-peer networks.

So the earlier p2p systems like Napster used centralized servers resulting in some obvious problems like a single point of failure and higher traffic. More recent p2p systems like Gnutella [72] or Freenet [73] do not depend on a centralized server and thus are more robust with respect to such issues. The latest p2p systems, including Chord, Content Addressable Networks (CAN), D2B [74], Tapestry, Pastry etc. incorporate distributed hash tables (DHT) to support scalability, load balancing and fault-tolerance [75].

4.3.1 Operations in p2p

Any peer participating in a p2p network applications must be able to perform the following operations:

- **Discovery of other Peers**: A peer may keep a list of other clients/peer on the application server, also known as trackers, so that a peer can use the list in order to find other peers. One may also use a Peer Name Resolution Protocol (PNRP) [76] infrastructure, which enables clients to find each other directly.

- **Connection with other Peers**: The connection problem is a more subtle one, and concerns the overall structure of the networks used by a p2p application. If there is one group of clients, all of which can communicate with one another, the topology of the connections between these clients can become extremely complex.

- **Communication with other peers**: Communication takes place via well defined protocols known over the network such as TCP/IP etc.

In p2p architecture the applications are generally Content Distribution Application like File sharing applications for example Napster [37], or are collaborative applications such as desktop sharing and shared white board applications. There are also multi-user communication applications, allowing them to communicate to exchange data, as well as distributed processing applications to performs tasks involving large amounts of data. The main idea behind the peer-to-peer systems was to exploit the resources distributed across internet to provide a single useful application, taking advantage of increased bandwidth and hard disk capacity to provide a file-sharing service. These system differ from each other by the methods of finding data among the resources. Naspter was the first largest p2p content delivery system, which used a *central index server*, to maintain the list of files available in the system. Due to this centralized architecture of Napster it was vulnerable to attacks and single point failure. Whereas Gnutella [36] used a *flooding query model* as a search method. Though it could avoid single point failure but was significantly less efficient than Napster. Freenet [77] was a fully distributed and employed a heuristic *Key-based routing*, associating each file with a key and clustering all files with similar keys on a similar set on nodes in the system. Search requests were routed from a node to other in the search of the file to the cluster without any need to visit many nodes, however it did not guarantee that the data will be found. Hence there was a need of an structured organization of data in a distributed architecture which formed the foundations of Distributed Hash Tables (DHTs). DHT is discussed in detail in Section 4.4.

4.3.2 Advantages of Peer-to-Peer networks

Distributed computing from its early versions has been accompanied with several benefits. But structuring the distributed network in different ways pointed out different advantages as per the system design [78]. Among them peer-to-peer networks stands to be a totally different approach in distributed computing with the following advantages:

1. It distributes the financial cost in terms of computing resources, which requires dedicated powerful servers or high performance computers to meet the needs.

2. The cost of the bandwidth of the whole system is spread over the network as compared to a single site.

3. Single point failure of the traditional Client/Server model is eliminated successfully proving its robustness towards increasing load and risk handling.

4. Capacity in terms of resources increases with the scalability of the system since the peers not only use the resources of the system but also provide the same to the system.

4.3.3 Disadvantages of Peer-to-Peer networks

Peer-to-peer networks, although provide an important concept in distributed computing but are still vulnerable with respect to scalability and security of the system [79]. It totally depends on the design of the systems as stated below:

1. Few peer-to-peer systems used flooding query models for the communication purpose whereby making the system infeasible, increasing the network load for large number of peers in the network.

2. Nodes in the network may not be trustworthy as per the application needs which may cause flooding and denial of service attacks.

3. File sharing systems may be prone to poisoning of files, loosing the users trust in the system.

4.4 Distributed Hash Tables (DHT)

Hash table has been proved to be an easier means of storing and retrieving data using keys associated with the data values. It uses a simple lookup function with a key as an input parameter and hence determines the location of the data value stored in the hash table. Distributed Hash Tables(DHT)[69] are similar to hash tables with a *structured key-based routing* lookup service for decentralized distributed system or say peer-to-peer systems. Using [*key, value*] pairs, any peer participating in the network can efficiently retrieve the value associated with a given key. The mapping form keys to values is maintained and shared among the nodes, in such way that a change in

a set of participants causes minimal amount of disruption of the hash table. with nodes coordinating with only a few other nodes in the system at most *O(log N)* of the N participants, only a limited amount of work needs to be done for a change in membership. DHTs characteristically emphasize the following properties:

- **Decentralization** : the nodes taking part collectively form the system without any central coordination.
- **Fault Tolerance** : the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.
- **Scalability** : the system should function efficiently even with thousands or millions of nodes.

DHT uses an abstract *keyspace*, which is partitioned among the participating nodes. This means each and every participating node is made responsible for a range of keys within the keyspace of the hashing algorithm. To store a file in DHT, a key is generated say by using a hash function on the file name. Now since the key space is partitioned among the nodes participating, the node assigned to the key generated stores the file in the system. Each node maintains a set of links to other nodes or say its neighbors, just like a routing table, all together forming an *overlay network*. For an incoming search request of a given key, firstly a lookup is performed to find the node responsible for the range of keys in the keyspace of the hashing algorithm. For this the nodes are arranged in such a fashion that the key addresses the nodes as well as the item in the hash table. The arrangement is done via a routing table maintained for structuring the network. A search for a key results in a node closer to node responsible for the key in every iteration. After the responsible node has been located the item can be retrieved as like a traditional hash table.

There can be different algorithms for defining the overlay networks and hence may have different time complexities for information retrieval. One of the very commonly used variant of the DHT is using Consistent hashing in a Cartesian space. Consistent hashing is a special type of hashing where for K number of keys and n slots, only K/n keys are re-mapped on average to n re-mappings being made for the change in size of the hash table. The choice of a good hash function endeavors uniqueness of key values, some real world implementations may use SHA-1 as its hash function. Varying the

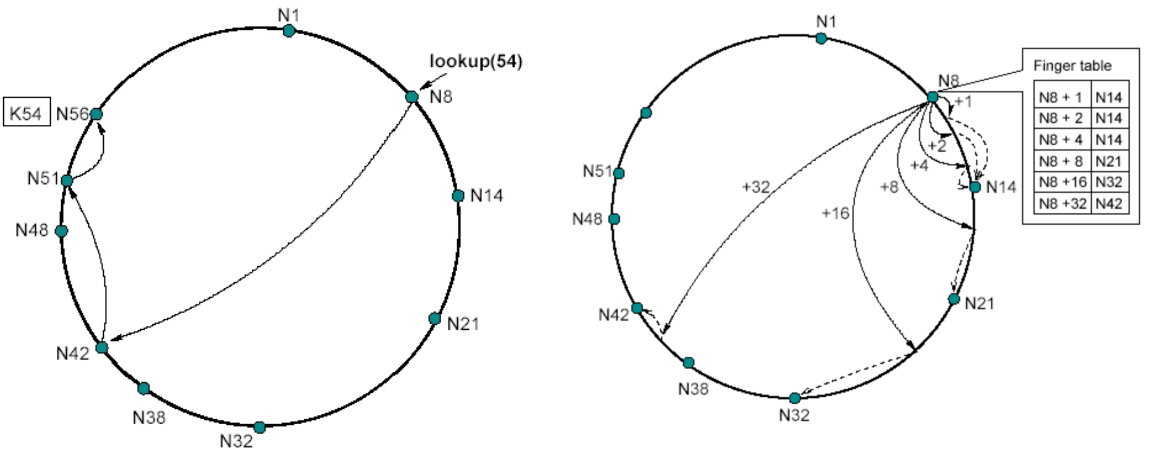
parameters of DHTs they can be used to build distributed file systems, domain name services, multicast, instant messaging, peer-to-peer file sharing and content distribution systems. Some of the well known DHT implementations are Apache Cassandra, CAN [32], Chord[33], Pastry[34], Tapestry[35] etc.

4.4.1 Chord

Chord provides an algorithm for peer-to-peer DHTs having a different approach towards searching the location of keys in the system and how the nodes join and leave the system. It provides a fast distributed computation of hash function using consistent hashing for key value pairs to their hash buckets as physical nodes. Using the Chord lookup protocol [80], node keys are arranged in a circle that has at most 2^m nodes.The circle can have IDs/keys ranging from 0 to $2^m - 1$. IDs and keys are assigned an m-bit identifier using consistent hashing. The SHA-1 algorithm is the base hashing function for consistent hashing. Consistent hashing is integral to the robustness and performance of Chord because both keys and IDs (IP addresses) are uniformly distributed and in the same identifier space. Consistent hashing is also necessary to let nodes join and leave the network without disruption. Each node is accompanied with a *successor* and a *predecessor*. The successor to a node (or key) is the next node (key) in the identifier circle in a clockwise direction. The predecessor is counter-clockwise. Since the successor (or predecessor) node may disappear from the network (because of failure or departure), each node records a whole segment of the circle adjacent to it, i.e. the r nodes preceding it and the r nodes following it. This list results in a high probability that a node is able to correctly locate its successor or predecessor, even if the network in question suffers from a high failure rate.

The Chord protocol is one solution for connecting the peers of a p2p network. Chord consistently maps a key onto a node [81]. Both keys and nodes are assigned an m-bit identifier. For nodes, this identifier is a hash of the node's IP address. For keys, this identifier is a hash of a keyword, such as a file name. It is not uncommon to use the words"nodes" and"keys" to refer to these identifiers, rather than actual nodes or keys. There are many other algorithms in use by p2p, but this is a simple and common approach.

A logical ring with positions numbered *0* to 2^m - *1* is formed among nodes (as



(a) Chord Ring of 10 nodes

(b) Finger Table for Node N8

Figure 4.1: Chord Lookup Protocol [5]

shown in Figure 4.1) [5]. Key k is assigned to node *successor(k)*, which is the node whose identifier is equal to or follows the identifier of k. If there are N nodes and K keys, then each node is responsible for roughly K / N keys. When a new node joins or leaves the network, responsibility for $O(K / N)$ keys changes hands. If each node knows only the location of its successor, a linear search over the network could locate a particular key. This is a naive method for searching the network, since any given message could potentially have to be relayed through most of the network taking $O(N)$ time. Hence it can be said that Chord implements a faster search method [82].

Chord requires each node to keep a "finger table" containing up to m entries. The i^{th} entry of node n will contain the address of successor $((n + 2^{i-1}) mod 2^m)$. In a 64-node Chord network, the nodes are arranged in a circle labeled as N1, N8, N14 and so on as per the node identifier (Figure 4.1a). Each node is connected to other nodes at distances 1, 2, 4, and 8 away. The "fingers" for one of the nodes are highlighted in Figure 4.1b. A lookup for a given key k made to any node n, searches its finger table to find the exact node or the node nearest as per its finger table, responsible for the key, i.e. *successor(k)*. With such a finger table, the number of nodes that must be contacted to find a successor in an *N-node* network is $O(log\ N)$.

4.5 Chord-based Decentralized Registry

Chord [33] - a DHT [69]-implementation over structured p2p overlay network is a distributed lookup protocol in p2p applications that helps in efficiently locating a node/peer that stores a particular data item. Chord is adaptable to resource volatility (i.e. changing set of resources/nodes) and hence is capable of answering search queries even if nodes continue to join and leave the network. Chord uses consistent hashing of the resources over a ring of node identifiers where it can map a given key directly to a node identifier. The registry is composed of a distributed hashtable of web services stored as [*key, value*] pairs. Chord uses a hash function to generate unique keys from the byte-encoded service name that are mapped to node identifiers generated as hash value of nodes' IP address. In this framework, a node may attempt to join the network as a WSP (service provider) with services that are available, but may or may not be deployed at that point of time. In either case, the node joins the de-centralized registry within the network and shares its own web services as owner 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 the following items:

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.

Each web service is owned by some WSPs as identified by the *owner* field in service metadata. Such service metadata help in identifying the service, its endpoints and other details necessary for proper execution in SOA framework.

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. The consumers are provided with a user interface, maintained by the WSP with all the services it can provide along with their current status as shown in Figure 4.2. 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.

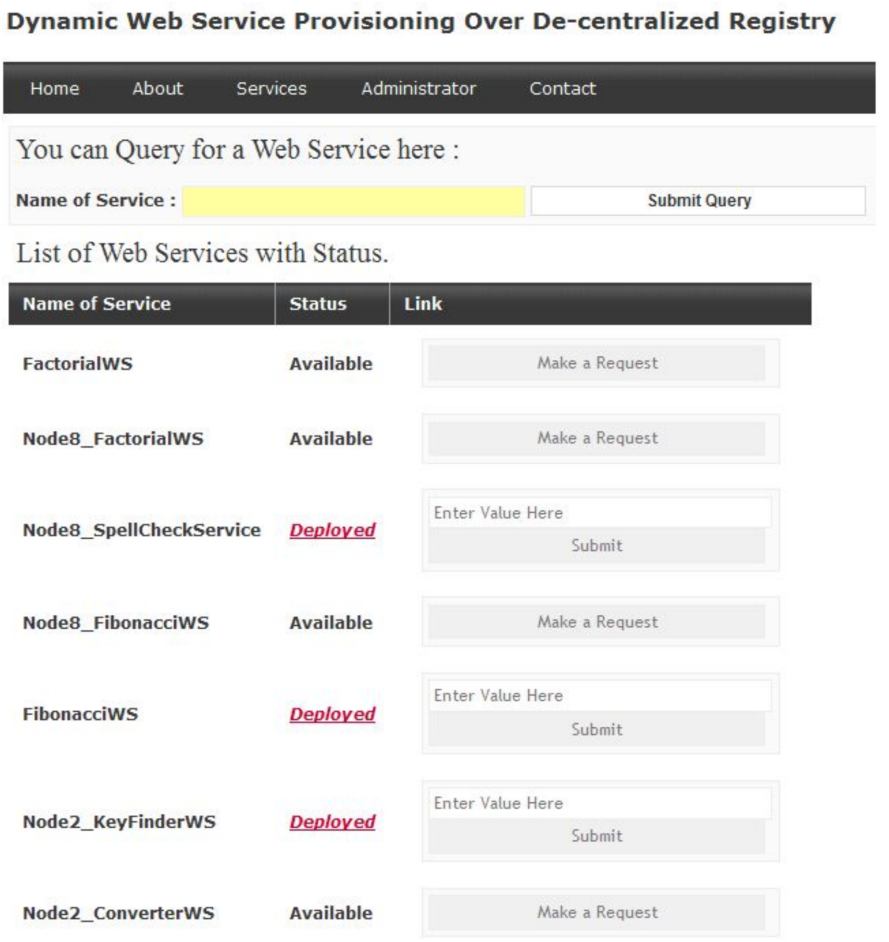


Figure 4.2: WSP Interface

4.5.1 The Registry Workflow

In order to maintain and use 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.

4.5.1.1 Publishing a Web Service

The first and foremost task performed by a WSP when it joins the network is to publish the web services it owns. In case the WSP is the first one to join the network, it initializes the DHT-based service registry by uploading the information about its services to the registry. Further, this node also identifies itself as the bootstrap peer for the Chord protocol which facilitates the other WSP peers to contribute to the DHT so formed when they join the network. With the increase in the number of WSPs and services provided by them increases, the entries in the registry are distributed among the peers based on the Chord protocol rules. This is achieved by mapping the hashed keys onto respective node identifiers and distributing the information among all the peers in the network. Apart from this, each WSP node also maintains a list of all services as [*key, value*] pairs locally, assigned by Chord along with the services it owns/hosts. When a service is published in the registry by the owner WSP, it becomes discoverable by all the consumers and other WSPs which are peers, until and unless the service is explicitly removed by the owner. Other WSP peers do not have the rights to perform any administrative task, such as publishing/deploying/modifying/removing the service owned by a particular WSP to/from the registry. The registry provides service metadata only which ensures that the service is discoverable by other WSP peers and consumers and it does not provide the entire service package for security reasons.

Once the service is published it may so happen that the owner undergoes a network/node failure or leaves the network, that is becomes unavailable, thereby creating uncertainty in the environment. In such a scenario the service still remains discoverable in the network due to the DHT-based implementation. This is possible because the keys of the services owned by a particular WSP 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 enables the architecture with higher service availability. When the WSP rejoins the network, it can resume its normal functioning from the same set of the deployed instances and schedules the incoming requests accordingly. This is possible because of the dynamic nature of the registry where dynamic service metadata such as the service status and the list of nodes where the service is deployed are maintained even if the owner is

unavailable leading to a high degree of fault-tolerance in the registry.

4.5.1.2 Discoverying a Web Service

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:

1. *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.

2. *Case 2* - If the service is not in the list, the client can submit a service query to the registry as a result of which the endpoint 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 [33]. As a result of this, the registry remains scalable even with an increase in number of web services and WSP's, as compared to previous approaches.

4.5.1.3 Binding with a Web Service

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.

In order to represent the workflow, we have modelled the request as a tuple with three attributes - [*service name, requesting WSP, target WSP*]. For example, as shown in Fig. 4.3, a service request tuple [*WS2, WSP#1, WSP#2*] implies that a client has made a request for service WS2 via the WSP#1 and the owner of the service 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 4.1 along with there scheduling criteria.

From the table it is evident that for a service which is already deployed, a client request made directly to the service owner (SR1) is routed using a default scheduling

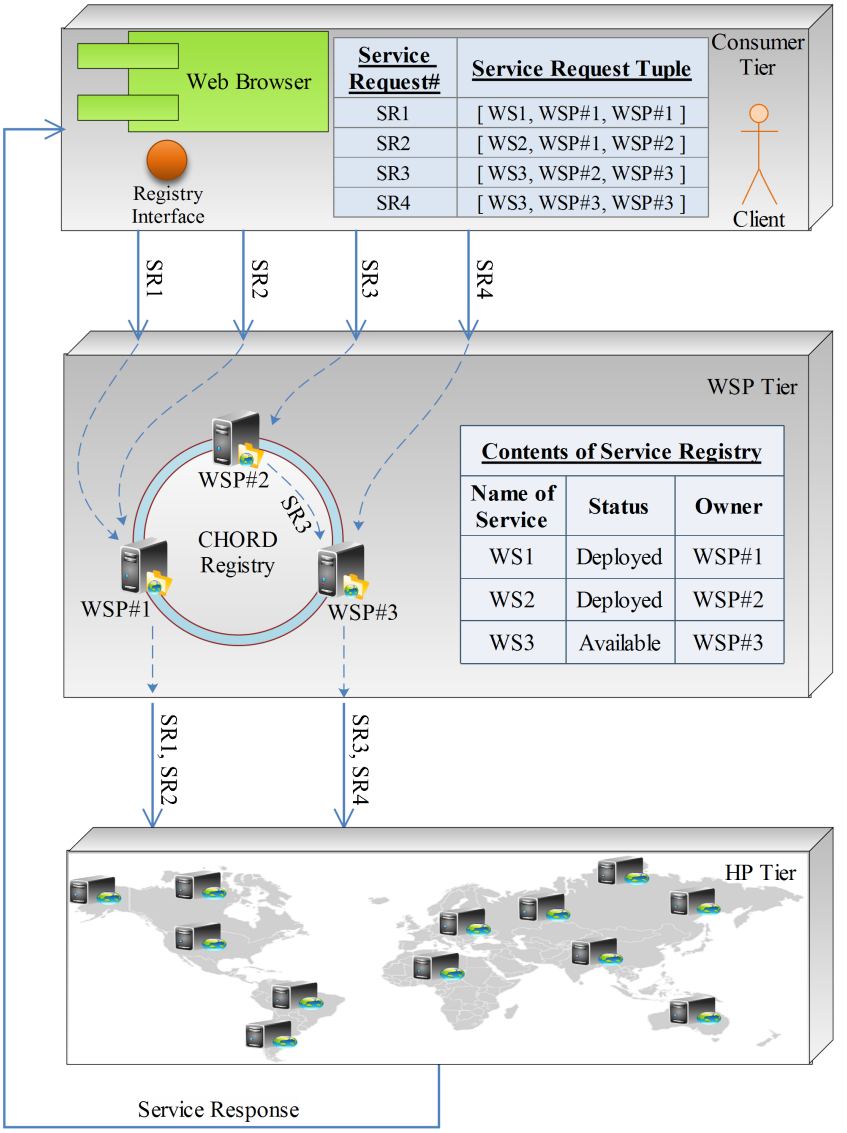


Figure 4.3: Service Request Types and Flow

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 the QoS parameters are 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

Table 4.1: Service Request Types

Request Type	Status	Request Tuple	Scheduling Criteria
SR1	Deployed	[WSx,WSP#x,WSP#x]	Routed to HP using WSP#x's scheduling strategy
SR2	Deployed	[WSx,WSP#y,WSP#x]	Routed to HP using minimum load criteria
SR3	Available	[WSx,WSP#y,WSP#x]	Routed to owner i.e. WSP#x for deployment
SR4	Available	[WSx,WSP#x,WSP#x]	Deployed on an appropriate HP

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 if it has already been deployed. Since the deployment rights are limited 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, by selecting a suitable node among the deployed instances decided by some scheduling strategy.

As discussed in Section 4.5, the registry entry for a given service includes the service meta-data, among which status of the service and the list of deployed instances play an important role to compensate for the absence of the WSP who owns the service. WSPs receiving service requests of type SR2 schedule them to appropriate HPs using the list of deployed instances from the registry. Once a WSP wants to rejoin the network it queries the registry for all the services it is supposed to provide and synchronizes its own service states, i.e. the service status and deployed instances, as per the registry, to avoid any inconsistency or conflict. Such an approach where consumer requests are handled while utilizing the benefits of the Chord protocol ultimately increases the service availability and provides a unique approach for rendering an architecture which can fully realize demand-driven provisioning of web services. Thus the use of only web service metadata instead of the whole 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 encourages a business model for exchanging web services.
- Increases the availability of the web service, making is accessible from more than one site.
- Keeping 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.
- Enabling 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.

4.6 Summary

The architecture presented in this chapter attempts to address the drawbacks of static UDDI-based registries that are used in distributed web based frameworks. It extends the ideas proposed in the architecture like DynaSOAr framework about decoupling the service providers and host providers and sharing the one-time deployment cost among many invocations and provides an alternative model which may be considered as complimentary to job-based grid paradigms. It employs service provisioning on the basis of balancing the load and meeting minimum service requirements to achieve better performance to cater to increasing demands for web applications. The incorporation of p2p technologies in the framework provides a robust approach towards handling the uncertainty of grid environments by sharing and distributing resources over the network. 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 inherent volatility of a grid-like framework.

Chapter 5

Request Scheduling : A Load Balancing Approach.

5.1 Introduction

The distributed architecture discussed in Chapter 3 aims at achieving the primary goals, such as high performance, availability and extensibility at low cost. However some of these benefits can be realized by overcoming the problem of allocating considerable processing capacity available in the distributed system so that it is used to its fullest advantage. In this system, the tasks, i.e. web service requests that are submitted to the WSP tier are of unpredictable nature and are submitted on the fly. Not only this, the service requests may vary from one web service to another. The performance of the architecture entirely depends on the HPs for executing service requests. But HPs may get overloaded if proper method of scheduling service requests is not adopted by the WSPs.

Alike commonly known distributed architectures, the system discussed in Chapter 3 consists of a collection of autonomous nodes connected by communication network where computing power of this distributed system can be realized by allowing its constituent computational elements (CEs), or nodes, to work cooperatively so that large loads are allocated among them in fair and effective manner. Different strategies are used for this load distribution. Any such strategy for load distribution among the nodes is called load balancing (LB). An effective LB policy ensures optimal use of the

distributed resources whereby no CE or nodes remains in an idle state while other CEs are being utilized.

Services must be submitted to the host computer (Host Provider) for processing whenever a consumer requests for that service. Now the problem is that the random arrival of services as shown in Figure 5.1 in such an environment can cause some nodes to be heavily loaded while other nodes are idle or only lightly loaded. Load distribution improves performance by transfering tasks from heavily loaded computers, where service is poor, to lightly loaded computers, where the tasks can take advantage of computing capacity that would otherwise go unused.

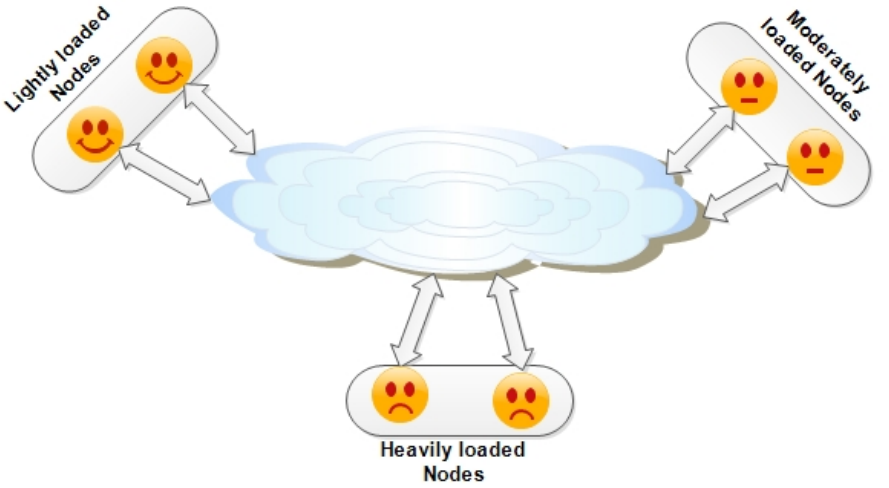


Figure 5.1: Load Balancing Model

In this chapter the focus is on the load distribution problem of the system among its nodes so that overall performance is maximized. Initially after discussing some key issues in load distribution for general purpose distributed systems, a survey on some load balancing policies is presented and algorithms used in this existing architecture are discussed. Conclusions are drawn about which algorithm might help in realizing the most benefits of load distribution.

5.1.1 Load Balancing Problem

Load balancing problem in distributed environment is an old problem, but some successors of distributed computing, such as grid computing, p2p added new challenges in it. One of the main challenges which creates a huge difference between the early distributed environments and the present day's distributed environments is that, in initial distributed systems nodes are homogeneous in nature and static, whereas in grid environments nodes are heterogeneous and any node can join and leave the network any time. In such an environment, system performance needs to be improved by suitably transferring the workload from heavily loaded computers to idle or lightly loaded computers. Originally a widely used performance metric considered to be the average response time of tasks. The response time of a task is the time elapsed between its initiation and its completion. Minimizing this average response time often becomes the goal of load distribution. Thus, if a system schedules the service request to a overloaded node then the request will take longer time to get served, which may lead to a service failure. Load balancing algorithms can solve this problem by distributing load evenly to the available resources. Basically, the following four basic steps are necessary for a load balancing system:

1. monitoring resource load and state,
2. exchanging load and state information between resources,
3. calculating new load distribution,
4. updating data movement,

5.1.2 Dynamic, Static and Adaptive Algorithms

Load-distributing algorithms can be broadly distinguished as dynamic, static or adaptive algorithms. Load balancing decisions can be taken either statically at compile time or dynamically at runtime. Static load distribution always assigns a given job to a fixed node. As shown in Figure 5.2, these decisions are based on a priori knowledge of the system, i.e. computing nodes and communication network are known and a prior information about all the characteristics of the jobs are required. Load balancing decisions are made deterministically or probabilistically at compile time, and remain

constant during run-time. The static approach is quite attractive for its simplicity and the reduced run-time overhead. However, the static approach cannot respond to the dynamic changes in the system, and may lead to load imbalance on some nodes and significantly increase the job response time. The majority of recent distributed system exhibit significant dynamic behavior. For these systems, dynamic scheduling, in which policy decisions are based on the load-state of nodes, is required.

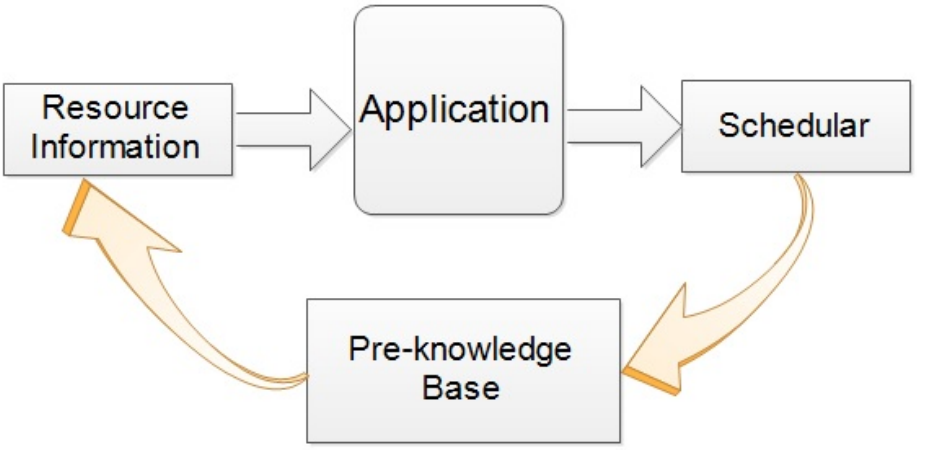


Figure 5.2: Static Load Balancing Model

Dynamic algorithms use system-state information to improve the quality of their decisions. For example, a simple dynamic algorithm, unlike its static counter part, will not transfer an arriving task if the node where it arrived is idle. Dynamic load balancing algorithms (Figure 5.3) use current load information while making distribution decisions. Multi computers with dynamic load balancing allocate/ reallocate resources at runtime based on a priori task information, and determine when and whose tasks can be migrated. As a result, dynamic load balancing algorithms can provide a significant improvement in performance over static algorithms [83].

Adaptive load-distributing algorithms are a special class of dynamic algorithms. They adapt their activities by dynamically changing their parameters, or even their policies, to suit the changing system state. For example, if some load distributing policy performs better than others under certain conditions, while another policy performs better under other conditions, an adaptive algorithm can choose between these policies based on the observations of the system state. Even when the system is uniformly so

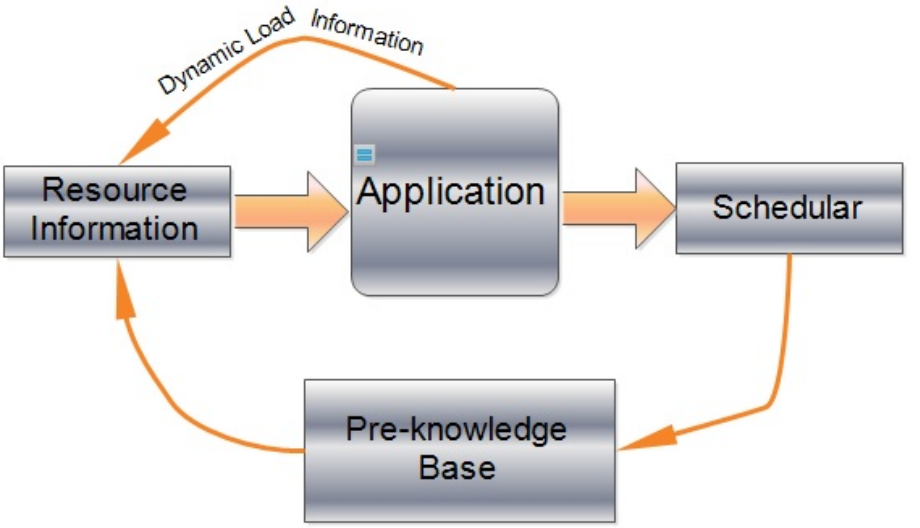


Figure 5.3: Dynamic Load Balancing Model

heavily loaded that no performance advantage can be gained by transferring tasks, a nonadaptive dynamic algorithm might continue operating and incur overhead. On the other hand an adaptive algorithm may stop the load distributing activity until some of the nodes become idle.

5.2 Load Balancing in the Proposed Framework

Today's distributed applications involving more than one server require a proper mechanism to distribute its clients requests to its servers. Depending on these mechanisms the servers may have varying workloads. Methods to achieve best performance needs to minimize the workload differences between the servers. Many approaches use prior estimations of jobs submitted to an application and hence perform sharing of workloads. Such a pre-decided load distribution leads to *static* load balancing approaches. In a distributed environment, any load balancing algorithm with greedy approach would seek for the least busy machine in the system. This decision for finding out the least busy machine or say the least loaded machine is subjective to the requirements of the application in consideration.

The architecture discussed in the Chapter 3 allows dynamic deployments of web

services where services are deployed on the fly on available resources or Host Providers (HPs). In this scenario once a service is deployed on a node (HP)after receiving a request from a consumer, it remains deployed on that node until explicitly removed, so that it can serve maximum requests for that service. So, in order to provide the consumer efficient and fast access to the service there is a need to minimize the average response time through efficient balancing of the load on these resources.

A web service may have more than one deployments to serve its consumers. Hence all the incoming service requests should be handled carefully to achieve proper utilization of the resources (HPs). In such a scenario, WSPs exercise different scheduling strategies to schedule the requests among the deployed instances of the service based on some suitable logic to balance the load among the deployed instances. As the requests are dynamic in nature, the load balancing algorithms should be able to properly mange the workload depending on the current load of the deployed instances specific to the web service request being made.

In this thesis both static as well as dynamic load balancing approaches are used to ensure better performance for the consumers. Static approach considers the minimum service requirements along with node configuration information, to prevent over utilization of the resources. This is achieved by constraining the deployments on HPs with insufficient resources as compared to the minimum service requirements. To achieve dynamic load balancing of the service requests, a major pre-requisite is to calculate the present workload of an HP. Once the workload is determined and collected by the WSPs, it runs some scheduling algorithm to decide a best suitable node to serve the consumer requests. In the next few sections the different approaches taken up within the framework to balance and manage the resources are discussed.

5.3 Resource Selection

5.3.1 Concept

The architecture works on the principle of assigning resources which have at least equal or greater computational efficiency required by the web service from the heterogenous pool of HPs. Thus, the WSPs ensure the all the deployments made do not over-utilize the HPs resources, leaving them unavailable for the services which may match their

static configuration. A web service (e.g. WSx and WSy as shown in Figure 5.4), having a given set of minimum requirements as *service metrics*, if deployed on a HP with lower configuration may lead to inefficiency of the system. Such a scenario will not only violate the QoS for the consumer, but at the same time may lead to higher response time for the web services which match the static configuration of the HPs (i.e. WSz). By constraining the deployments to the HPs with enough resources to meet the service requirements, the HPs with lower configuration may be pretended from being overloaded or becoming unavailable in the network.

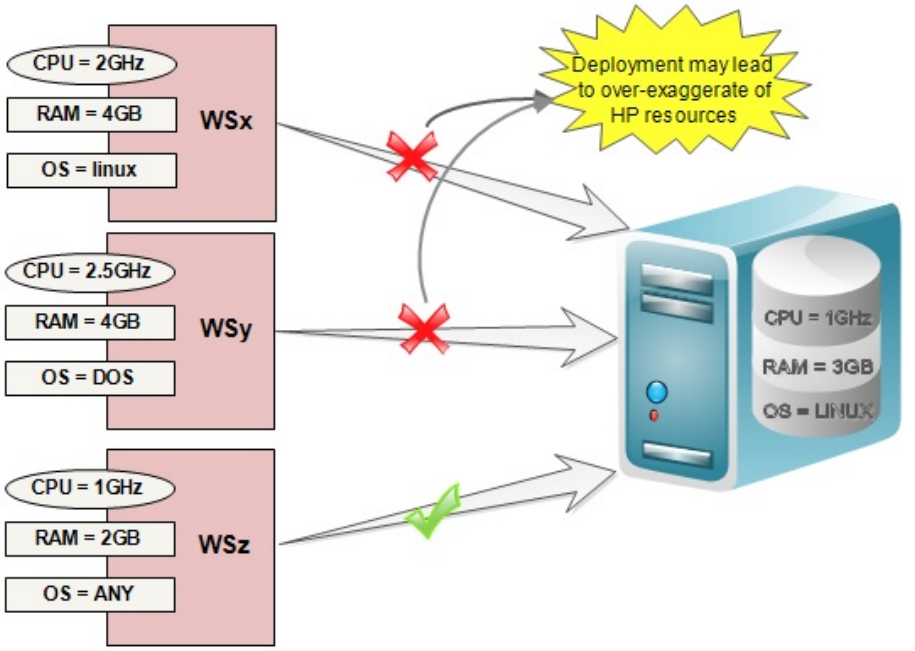


Figure 5.4: Static Load Balancing Model

5.3.2 Implementation

As mentioned in the earlier chapter, a web service hosted by a WSP is annotated with a *service metric* which consists of fields specifying minimum physical resources required such as - processor, memory, host operating system etc., for its proper execution. All these requirements are provided by the manufacturer of the service to specify the resource capabilities which would provide us with the desired QoS. In future more

parameters can be added to meet SLA for its consumers. These specifications are hence used to search for a suitable resource in the network.

The WSPs perform matchmaking of the *service metrics* with *node properties*, i.e. static physical information made available as shown in Algorithm 2. Such matchmaking process helps the WSP tier to prevent deployments on insufficient resources hence avoiding over utilization of the resources. By this approach the *Best Node Finder* ensures that no web service is deployed on a resource which is not compatible with the service requirements.

In spite of the above effort, the HPs may still get overloaded if a large number of consumer requests are scheduled to a single HP, resulting in higher response time and under utilization of other deployed instances. Hence static load balancing does not provide us with a solution for the dynamic nature of incoming service requests. To handle such uncertain behavior of consumer requests, the architecture makes a move towards dynamic collection of the present workload of the current deployed instances and hence schedule the service requests among them.

5.4 Dynamic Load Balancing

A correct evaluation of a server's workload can be determined by the amount of processing or response time needed to execute all its clients' requests. Researches have shown [84] the best mechanism to accomplish optimal response time by distributing the workload evenly among the servers. But there is no known way to determine how much processing time a request may take prior to actually executing it. As a consequence, determining the actual workload of a server remains an unresolved issue.

Parameters, such as CPU utilization, CPU queue length may provide with approximate estimates to determine the present workload of the server. The performance of jobs which require access to a large database may depend on amount of physical memory available. In such cases amount of physical memory in use may also prove to be an important parameter to determine the present workload of a server. A correct choice and a right combination of such parameters can help in achieving a better estimate of the overall workload.

5.4.1 Overview

Our architecture deals with a situation where the WSP tier needs to distribute the workload, i.e. the web service requests among the deployed instances in the HP tier. To achieve this, it employs methods which make use of the dynamically collected current load information of the HPs. All HPs send their current load information to all WSPs at regular intervals. All such load information are stored by the WSPs and are processed whenever required. The decision making process is carried out at two stages:

1. First it is checked weather a HP is loaded or not before deploying a web service (as shown in Algorithm 2in Chapter 3).

2. Next a *best node* is selected for distributing the service requests based on some scheduling algorithm.

Though at the first stage, Algorithm 2 discussed in Chapter 3, depicts a static approach to load balancing, but deploying a service on an overloaded node may lead to higher cost of deployment and higher response times for the service requests scheduled to that HP. Hence performing a check on the current load of a HP provides an added advantage for selecting a suitable HP which is lightly loaded. At the second stage, WSP uses the load information with the principle, that service requests should be scheduled to only those HPs which are not overloaded at present. It utilizes the load information (discussed next) in different ways depending on the scheduling algorithm in use.

5.4.2 Load Information

One of the important issue for any dynamic load-distribution algorithm is identifying a suitable load parameter. A load parameter indicates the expected performance of a job that which may be executed at some particular node. To be effective, load parameter readings are taken when tasks are initiated and should correlate well with task-response times.

Load indexes which have been studied and used in this thesis include the CPU utilization, the average CPU queue length over some period, and the amount of available memory. Researchers have consistently found significant differences in the effectiveness of such load parameters - and that simple load indexes are particularly effective.

In case of the architecture which has been discussed earlier, the current load of the given servers such as CPU utilization and physical memory in use can be useful. A waiting queue length of the given server may also contribute to get an estimate of the number of jobs still pending at the server end. The architecture attempts to use these parameters to estimate the current load of the HPs. All the HPs send their average or present CPU usage, physical memory in use, the server queue length and size of server thread pool in use. Collectively, the load information of a given HP consists of the values as follows:

- Average CPU utilization (in percentage) over the last minute, i.e. just before sending the load information.
- Physical memory (in percentage) currently in use.
- Queue length of the server.
- Size of the thread pool and number of idle threads in the thread pool.
- Overall *load* of the node calculated based on the above parameters.
- A boolean value to state the node is loaded or not, based on some threshold.

The load information is wrapped as an object and is communicated to all the WSPs in the network. Among the above parameters the first four are basic load parameters whereas the last two parameters are derived from the basic parameters and help in the decision making process for the load balancing algorithms. Hence a proper derivation and selection of such variables is of great concern and may effect the performance of the system.

5.4.3 Load and Load Threshold

The HPs in the architecture may be of heterogenous nature, hence calculating load information in terms of percentage of utilization provides a simple approach for handling the workloads. Mapping all the above mentioned load information on one scale provides an easier approach for deciding the difference in present workload of the HPs. To obtain such a value, the architecture uses a liner equation of two major parameters as shown in 5.1.

$$load = (average\ CPU\ utilization \times 0.5) + (physical\ memory\ in\ use \times 0.5) \quad (5.1)$$

Equation 5.1, estimates the *load* of a given HP as a linear combination of average CPU utilization and physical memory in use, by assigning equal weights to both the parameters, assuming that both parameters play equal role in estimating the present workload of a HP.

Based on the above estimate of the workload, the decision that whether a node is loaded or not depends on a *threshold* value. If the *load* exceeds the *threshold*, the HP is said to be *overloaded* or simply *loaded* and may not be able to serve more incoming service requests with the same proficiency as it could if the *load* lies below *threshold*. Scheduling algorithms try to avoid the HPs if their corresponding *load* exceeds the *threshold*. Similarly, no new deployments are made to the nodes if they are found to be *loaded*.

The *threshold* value as discussed above is also pre-decided for all the nodes depending on the configuration of the nodes. Selection of an optimum threshold still remains a research issue as performance of the system may vary with respect to different thresholds.

5.4.4 Implementation

5.4.4.1 Gathering Load Information

Load Balancer provides the encapsulation of the current load information of a given peer in the system. All peers can share their dynamic load information among each other. The load information as described above are collected during runtime and is wrapped as a *load object* and is sent to all WSPs by *load Sender* of each HP. All the *load objects* received by the WSPs are stored in a *load map* (a map of *load objects*), which contains only the latest *load object* corresponding to a given HP. The scheduling algorithms running at the WSP makes use of the *load map* for monitoring the current *load* of a HP. *Load sender* sends the load objects at regular intervals, equal to the time slice (discussed in next section) for web service request scheduling. This enables the scheduling algorithms to work on the latest load information for every time slice. Once

the WSP has a ready database, i.e. *load map* of current *load objects* of the HPs, it uses those to schedule the incoming web service requests to appropriate HPs.

5.4.4.2 Scheduling Strategies

The WSPs use a scheduling strategy to route the consumer requests based on the dynamic load information collected from the HPs with deployed instances of the services they own. The scheduling strategies used by *Resource Scheduler*, are 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. To facilitate this process *Service Monitor* maintains a sperate list of the deployed instance of HPs along with the *best node*, specific to each web service. The list is known as *service list*. A set of all such *service lists* is maintained as a *Service List Map*. At the end of every time slice, the *best node* is changed to some other HP from the *service list* 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 (Algorithm 3).
- **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 (Algorithm 4).
- **Minimum Loaded First (MLF)** - selects the instance with minimum load as the *best node* for every time slice among the deployed instances (Algorithm 5).

5.5 Experimental Results

In this section we present the performance of dynamically deployed web services to serve a large number of consumer requests made simultaneously. These tests have been conducted using a simple web service for calculating the N^{th} Fibonacci term (N=40), with one WSP and many HPs. Initially requests are made for a service which is not deployed, calculating response time for each request, keeping the load factors as 50% of CPU usage and a minimum free memory. The physical nodes used in the experiments are of configuration which Intel Core 2 Duo with 1.86GHz frequency processor, 1GB

Algorithm 3: Algorithm for Round Robin Reloaded (RRR)

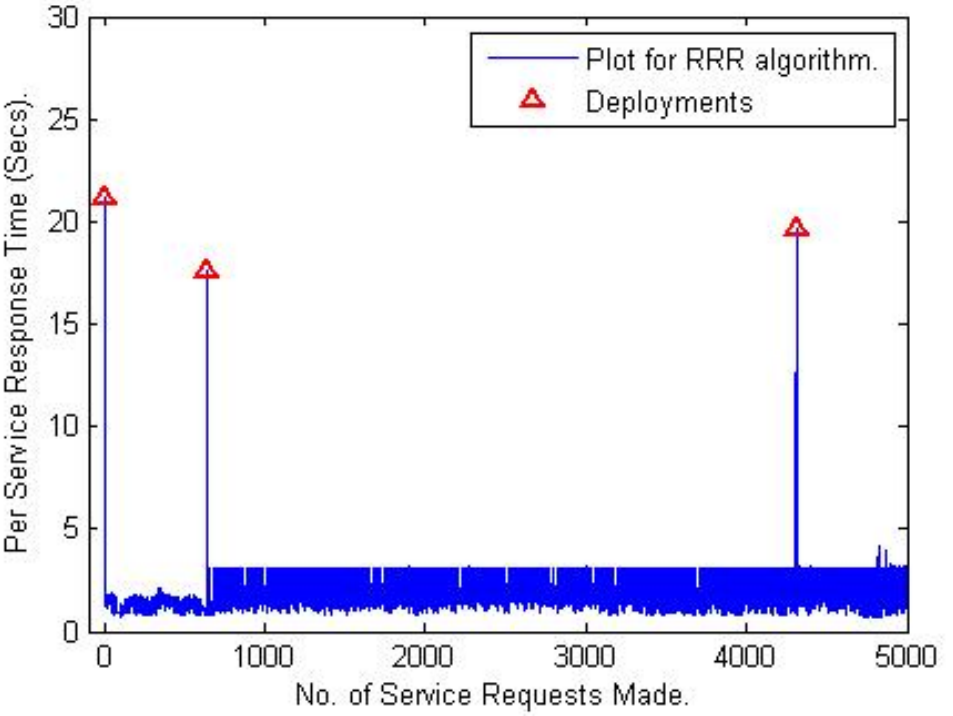
```
1  begin
2      foreach ServiceList ∈ ServiceListMap do
3          NodeFound ← FALSE;
4          Marker ← ServiceList.getPointer();
5          while not found suitable node do
6              ServiceList.incrementPointer();
7              tempNode ← DeployNodes.get(ServiceList.getPointer());
8              if LoadThreshold(tempNode) == TRUE then
9                  if Marker == ServiceList.getPointer() then
10                     // all nodes loaded
11                     NewDeploymentRequired();
12                     NodeFound ← TRUE;
13                 else
14                     ServiceList.setBestNode();
15                     NodeFound ← TRUE;
16             // update ServiceListMap with new ServiceList
17             UpdateServiceListMap(ServiceList);
```

Algorithm 4: Algorithm for Least Recently Used Reloaded (LRUR)

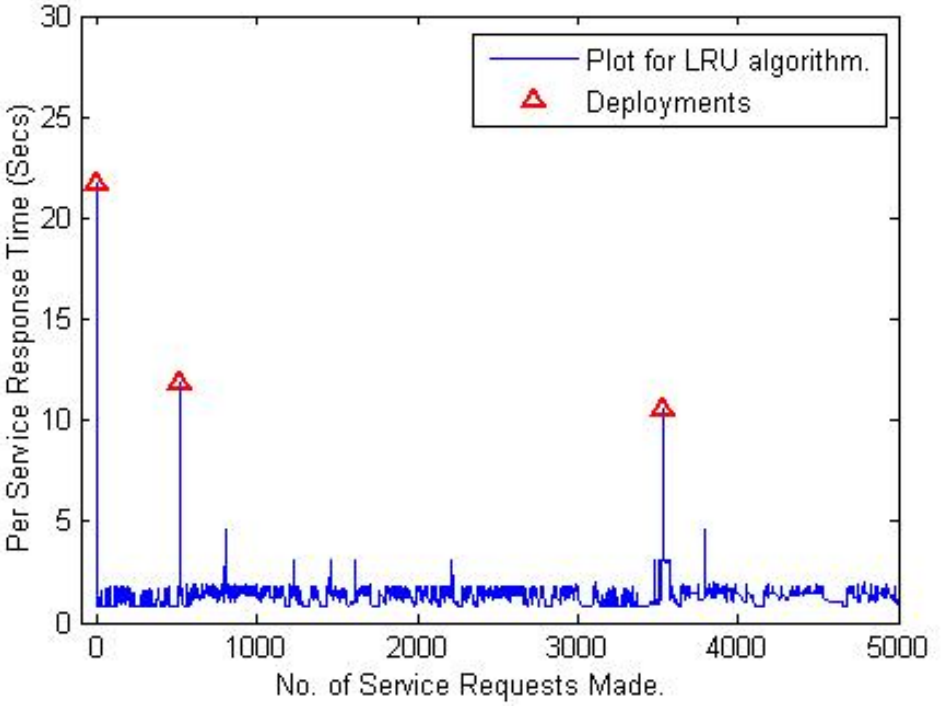
```
1  begin
2      foreach ServiceList ∈ ServiceListMap do
3          NodeFound ← FALSE;
4          Marker ← ServiceList.getPointer();
5          while not found suitable node do
6              tempNode ← DeployNodes.get(ServiceList.getPointer());
7              if LoadThreshold(tempNode) == TRUE then
8                  ServiceList.incrementPointer();
9                  if Marker == ServiceList.getPointer() then
10                     // all nodes loaded
11                     NewDeploymentRequired();
12                     NodeFound ← TRUE;
13                 else
14                     ServiceList.setBestNode();
15                     NodeFound ← TRUE;
16             // update ServiceListMap with new ServiceList
17             UpdateServiceListMap(ServiceList);
```

RAM with Windows 7/Windows XP and Linux (Ubuntu) installed. The graphs in Figure 5.5, show the plots of 5000 consumer requests made to the corresponding service. Response time in seconds for each service request is plotted for the different scheduling strategies mentioned in the Section 5.4.4.2.

It may be noted that the time taken by the very first request to be served is quite high as compared to the subsequent requests. The high response time for the first request is caused due to the overhead of a new service deployment. Once the



(a) Plot for Round-Robin Reloaded



(b) Plot for Least Recently Used Reloaded

Figure 5.5: Experimental results for Load Balancing

Algorithm 5: Algorithm for Minimum Loaded First (MLF)

```
1  begin
2      foreach ServiceList ∈ ServiceListMap do
3          NodeFound ← FALSE;
4          while not found suitable node do
5              LoadObject ← FindMinimumLoad(ServiceList.getDeployedInstances());
6              if LoadObject.getThreshold() == TRUE then
7                  // if minimum load exceeds load threshold
8                  // all nodes loaded
9                  NewDeploymentRequired();
10                 NodeFound ← TRUE;
11             else
12                 ServiceList.setPointer();
13                 ServiceList.setBestNode();
14                 NodeFound ← TRUE;
15         // update ServiceListMap with new ServiceList
16         UpdateServiceListMap(ServiceList);
```

service is deployed, subsequent requests are processed within a short duration till the instance becomes *loaded* and a new deployment is triggered. When the new instances are deployed, the response time again increases for those particular requests for which the deployment was triggered. On the whole, the " deploy once, use many times " philosophy of web services is well observed in the experiment. As the service requests use the computational resources scheduled to them, the load of the nodes may increase leading to higher response times - and such situations lead to fresh deployments.

Based on the experiments conducted with different parameters, and comparing the results, we find the cumulative response time for LRUR (Figure 5.5b) is less than that of RRR (Figure 5.5a) with same parameter set. As LRUR algorithm schedules the service requests to the HP until it gets loaded, we get lower response time for longer times (for good HPs) but this leaves the other deployed HPs less used. In contrast the observation in RRR is just opposite as in this case the *Resource Schedular* schedules the service requests evenly among all the HPs which may lead to higher response time if the HPs have a *load* factor just below the *threshold*.

5.6 Summary

A load balancing approach for the proposed architecture and its implementation is discussed in this chapter. To achieve the desired QoS of the propose architecture, an

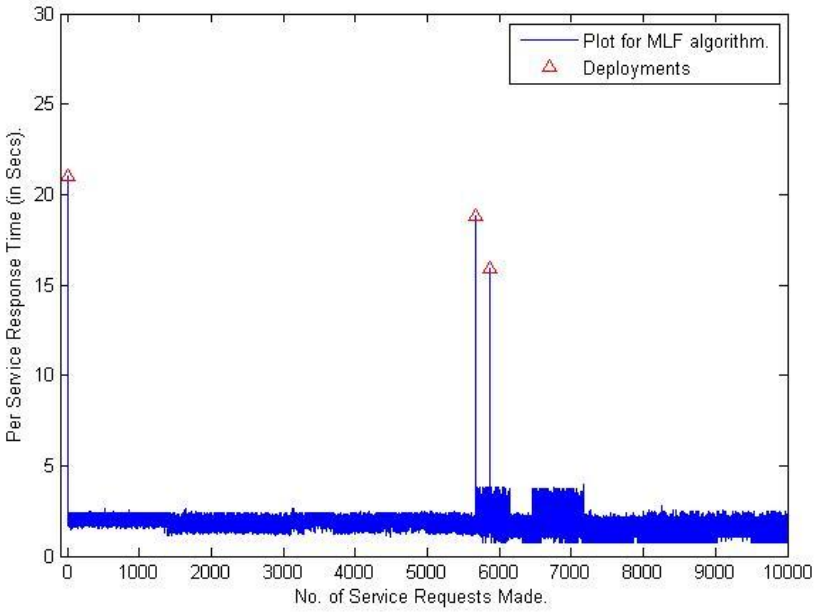


Figure 5.6: Dynamic Load Balancing Model

efficient and effective load balancing approach is needed. Thus, whenever a consumer requests for a service, in order to execute that service, it needs to be deployed on any of the available host providers based on its resource requirements and the current load information of the host provider. After the first deployment, all the subsequent requests for the service are then just redirected to the host providers. Some load balancing approaches are discussed in this chapter where initially a simple threshold based load balancing approach has been implemented. Every host provider has a threshold or load criterion defined. It depends on parameters like memory and CPU usage of the overall system, CPU and memory usage per service, etc. Once a host provider exceeds that criterion, a new host provider node with a load lower than the threshold value is found from the remaining nodes in the network. Then the service is deployed on this new node. Next, some good scheduling strategies are also used to serve consumer requests as the service is available on more than one host providers. These scheduling strategies and their implementations with results are also discussed in this chapter.

Chapter 6

P2P-Based Service Distribution: DynaTronS protocol

6.1 Introduction

Dynamic or demand-driven service deployment in a distributed 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 chapters, but the proposed methods face one major difficulty. It incurs considerable deployment cost because in many cases, a service deployment operation may require additional data to be accompanied that increases the service package size. In this context, fast p2p file sharing techniques (such as Bit-Torrent 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. One significant contribution of the thesis is incorporation of Bit-Torrent like 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.

In this chapter, a unique peer-to-peer based approach is proposed for dynamic

service provisioning. The approach is based on a Bit-Torrent like protocol for provisioning the service on a remote node. Thus, this fast p2p file sharing technique can make a value addition to the framework. *Bit-Torrent* [85] is quite popular as it increases the speed of the download, decreases the possibility of download failure and increases the availability of the files and services. This chapter focuses on the inclusion of Bit-Torrent protocol in the proposed framework and demonstrates its advantages by making use of such a file sharing technique to download service packages from the existing deployments to cater to new deployments. Thus, being built around a p2p model, the proposed framework discussed in this chapter caters to resource volatility and also incurs lower provisioning cost .

6.2 Enhancement of the Proposed Architecture

The proposed framework discussed in the earlier chapters with decentralized service registry, efficient catering of consumer requests is only possible if requested services can be deployed in a timely manner. This three tire architecture where all the nodes either serves as WSP or HP are connected in p2p network provides a new platform for dynamic web service provisioning.

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. To achieve this on the premises of p2p network, use of Bit-Torrent protocol seems to be a good solution for reducing the deployment time of a service.

The architecture adopts two different modes to accomplish download of the service package. These are:

1. **Direct Download** : where the HPs download the service package directly from the local repository of the WSP to carry out 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.

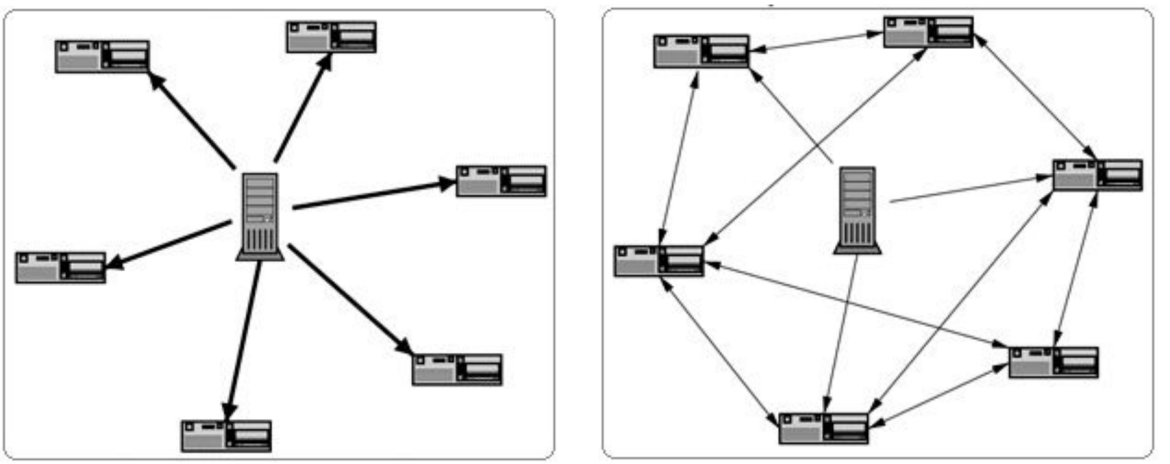
In the first approach, it portrays a client-server model where HP (client) needs to download the service package from the WSP (server). In such a scenario a service

package may become unavailable if the owner WSP becomes unavailable at the time or in the midst of a deployment. This may again lead to the bottleneck problem or 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 be advantageous for the architecture. This is where something like Bit-Torrent protocol can achieve faster download by proper utilization of the network bandwidth. In the following subsections, a brief overview of Bit-Torrent protocol and its advantages re discussed.

6.2.1 Bit-Torrent Protocol

Bit torrent protocol was specifically designed for transfer of files, ensuring better performance with respect to download times. This is achieved by the use of p2p networks for resource sharing. In traditional client/server model [86], where each node acts as a client or a server, enabling sharing of files between them, server acts as the central/control point for the whole architecture. The server stores all the files/resources hosted in the network, which can be downloaded by more than one clients at the same time (as shown in Figure 6.1a) [6]. The main drawback of such an architecture is that it suffers from single point failure, i.e. the failure of the server [87]. Moreover, it requires a maximum cost of resources for the server to simultaneously cater many clients possibly for same or different files.

The main goal of Bit-Torrent protocol is to reduce the server and network impact of sharing large files by providing a swarm of hosts over a peer-to-peer network, where each node acts as a peer to each other playing a dual role of client as well as server. The peers who provide the files or parts of files are called *seeds*, whereas those who wish to download the file are called *leechers* [88]. A file to be shared is divided into pieces. All the pieces of the given file are equal in size, for example a file of 100MB may be shared as ten pieces of 10MB each or twenty pieces of 5MB each. Once a peer downloads a complete piece of the file it becomes the seed (source) of that piece while downloading the rest of the pieces. Peers request different pieces from different peers while downloading the complete file (as shown in Figure 6.1b). The pieces of the file are generally downloaded in a random manner, and are arranged in order after all the



(a) Client/Server Model

(b) Peer-To-Peer Model

Figure 6.1: File/Resource Sharing Models [6]

pieces have been downloaded by the bit torrent client, to produce the complete file. In this approach of downloading the files in pieces, the download of the whole file can be stopped in between, without loosing the information of the pieces which have been downloaded completely. Due to this advantage it offers an easy way to download larger files.

6.2.1.1 Operation

A node can act as a peer in the Bit-Torrent protocol only if it operates via a *Bit-Torrent Client* - a program that implements bit torrent protocol rendering the node capable of preparing, requesting and transmitting any type of file over internet using this protocol. A node with an instance of Bit-Torrent client is considered as a *peer*.

To share a file, a peer creates another file called *torrent*, containing some metadata about the file to be shared and about the *tracker* - a computer that coordinates file distribution over the network. Once the torrent file is created, the peer publishes it over web. A peer who wants to download the file, first downloads the torrent file and connects to the specified tracker. Tracker then provides a list of peers from where to download the file. In this process, the Bit-Torrent client makes small data requests over different TCP connections to different peers. The selection of which piece to download

first follows an approach of rarest first, in contrast to the sequential download [89].

6.2.1.2 Advantages and Disadvantages over Classical Downloads

It provides a much lower cost to the content provider alleviating the overhead to cater large number of clients to download the same file again and again. At the same time it removes the bottleneck of single point failure, as not much of difference is made if one of the seeds become unavailable. By this approach it reduces the resources and bandwidth cost of the original distributor of the file. Since the downloads take place from more than one site at the same time, it provides a better/full utilization of the available bandwidth for a peer by offering higher transfer speeds. From a different point of view, it also provides redundancy of data over many sites and reduces dependence on the original distributor.

In spite of using the concept of decentralized peers in a p2p network, the whole architecture can suffer if the centralized tracker fails. There have been approaches to decentralize the tracker using DHT implementations. Another major disadvantage of the protocol is that downloads achieve full download speed after some time, when all or maximum connections to different peers have been established, whereas in classical download system, full download speed is made available by the server from the beginning of the download. Not only this, as per this protocol, a peer is offered high download speeds only if it has pieces to share, otherwise the peer is choked.

6.2.2 Why Bit-Torrent?

Traditional *File Transfer Protocol* (FTP) [90] till is considered as a standard for secure and reliable transmission of large files over the Internet. Nevertheless, it is based on a client-server approach which is centralized and 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 use a decentralized model with each peer functioning as a client with its own layer of server functionality. On top of this network model, Bit-Torrent [85] 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 6.2. In the proposed architecture, Bit-Torrent technology can be used to download service packages from the existing deployments, thereby sharing the deployment cost over many nodes. Thus, the speed advantage of p2p file sharing over dynamic web service provisioning can be achieved with increased service availability, decreased possibilities of service failure and reduce time for aa service deployment.

Based on such a p2p approach capable of removing single point failure for service package downloads, the architecture uses a second mode of download which is named as *Dynamic Torrent Service* deployment protocol or DynaTronS.

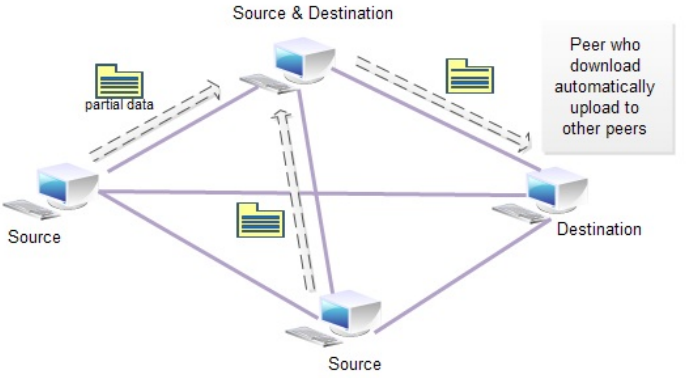


Figure 6.2: Bit-Torrent Protocol

6.3 DynaTronS Deployment Protocol

DynaTronS is a download protocol specific to the requirements of the architecture proposed in Chapter 3. It allows simultaneous download from all the available peer to pull up the download speeds and hence reduces 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 store the files required to complete a deployment request. The underlying concept is to download the service package from all the possible locations, i.e. peers which contain the service package. In the proposed architecture, the service package is available at the local repository of the WSPs, as well as the HPs where the

web service has been deployed. After a WSP notifies an 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. Since the architecture enables the WSP tier to host the web service and to share its information via the registry, but does not allow to share the service package among the WSPs, the only place where one can find the service package (apart from WSP) in the architecture is from the currently deployed instances of the web service in concern. Hence the list of HPs bearing the service deployments along with the WSP who owns 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 listed in the *seed list*.

Figure 6.3 shows a conceptual diagram following the DynaTronS protocol, where the service package is available at more than one sites, i.e. a WSP and few HPs. A 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 making a full use of the network bandwidth, while downloading different parts of the service package concurrently.

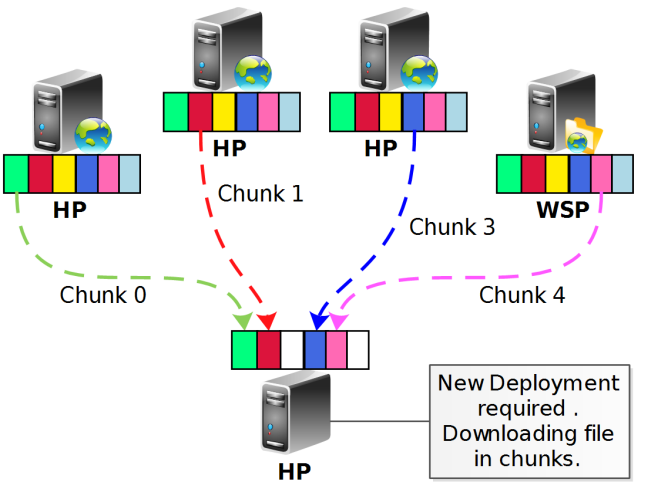


Figure 6.3: Gather and Deploy Protocol

6.3.1 DynaTronS vs Bit-Torrent

In early file transferring protocols, transferring file was a costlier affair in terms of time, because as the size of the file increases, with limited bandwidth and speeds the time to download the file also increases. After many p2p implementation Bit-Torrent comes forward as a most popular form of file sharing protocol over the Internet. Bi-tTorrent is a p2p-based communication protocol, that is used to share data and electronic files over Internet. Mostly BitTorrent protocol is used for transferring large file such as digital video files containing TV shows or video clips or digital audio files containing songs over the Internet between geographically distributed nodes. Now considering the advantages of Bit-Torrent protocols, DynaTrons protocol has been designed for Service Oriented Architectures, where services are not limited to only video, audio or some other content files. This protocol works behind some architectural framework like service oriented grid or cloud, in such a way that whenever a service deployment or service migration is needed for computationally intensive services, this protocol provideds new platform for dynamic service deployment. Though the architecture employs DynaTronS protocol for simultaneous download to achieve higher download speed, it incurs 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. Thus based on the functionalities of Bit-Torrent protocol, DynaTronS protocol is designed some different architectural aspects where purpose remains the same but roles of different elements change due to its architectural demand.

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 in comparison with Bit-Torrent protocol 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 Bit-Torrent protocol). The WSP 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 for next deployment. On the other

hand, in case of Bit-Torrent 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 the *seed list*, i.e. the WSP itself. In such 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 be carried out from a single site.

6.4 Downloading the Service Package

Whenever WSP requires a deployment of a web service after selection of the suitable HP form 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 HP 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. Two download modes discussed in Section 6.2 are discussed detail in the following subsections.

6.4.1 Direct Download Mode

In this mode, the deployments, i.e. first time deployment or in some cases even the successive deployments all 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 requesting for the download. Download of a specific web service is enabled to an HP if and only if the HP belongs to the 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 hence starts the download for the requesting HP. 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.

6.4.2 P2P Download Mode

This mode implements the *DynaTronS protocol*, enabling the HPs with deployed instances of the web service, along with the WSPs, to provide the required service packages. In this mode, at the HP 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 the *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 identity of the chunk such as :

- **Name** of the WAR file or the web service package.
- **ID** - a 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** - Sends chunk requests to all the peers in the *seed list*, in a cyclic fashion, until all the chunks of a service package are requested (Algorithm 6).
- **Proportionate Chunk Distribution** - 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 proportion number of chunk requests to the peers which are lightly loaded, and only one chunk request to the peers which have heavy load (Algorithm 7).

Among the two chunk distribution algorithms, *Serial Chunk Distribution* gives a naive approach for distributing chunk requests and may fail to achieve the goal as expected, because a need for 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 from the loaded peers. Though this approach does not guarantee to resolve the issue mentioned earlier, but still achieves a better performance compared *Serial Chunk Distribution* algorithm.

Algorithm 6: 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;
```

Algorithm 7: 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* 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. After the requesting HP completes downloading 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 6.4 depicts a sequence diagram for the service package download carried out the at 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

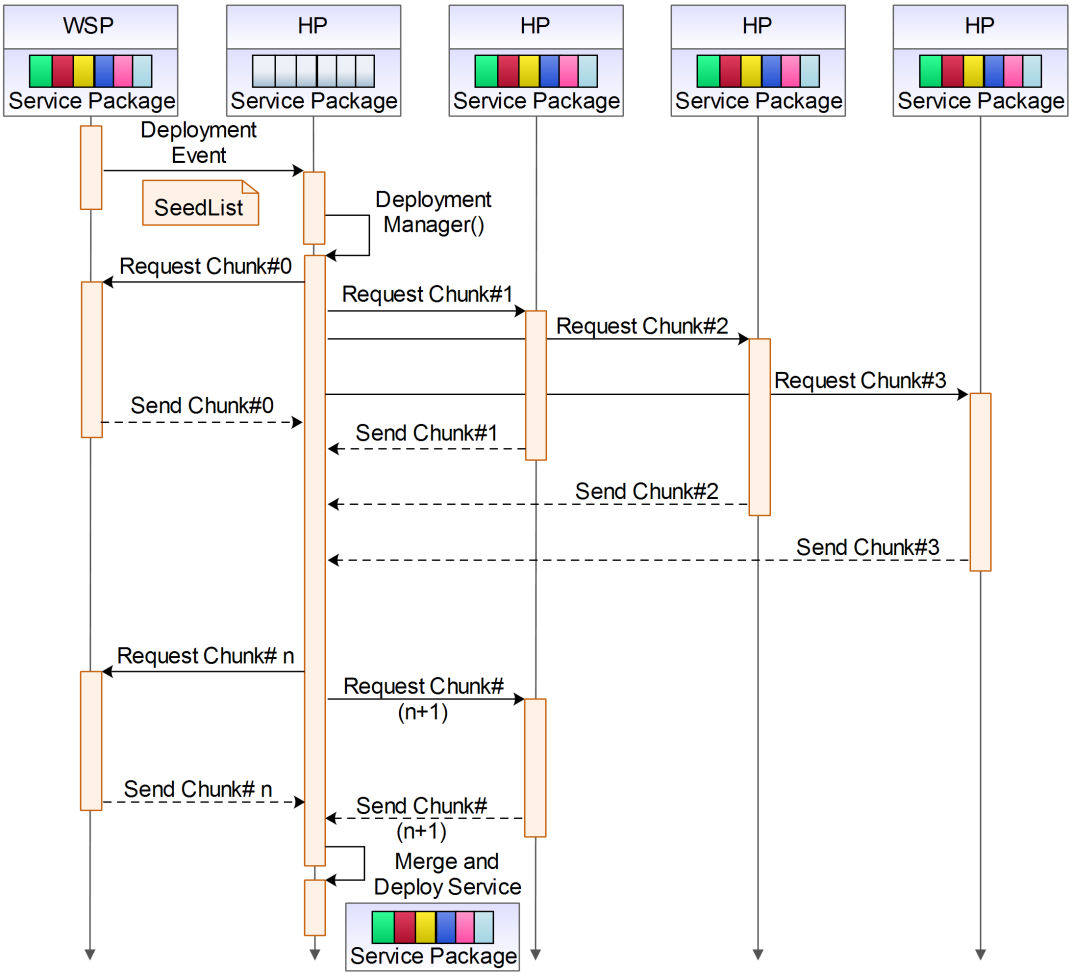


Figure 6.4: Gather and Deploy Sequence Diagram

Algorithm 6), 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 is 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 with that of file length and checksum 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.

6.5 Experimental Results

A set of experiments have been 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. The results of the experiments are presented in this section. As before, a distributed environment is created for conducting the test where 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 sending the 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 are of configurations ranging from 1GB-4GB of physical memory, 1.86GHz-3GHz dual-core processors. The tests have been 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 small number of requests can trigger more deployments. A large service package of 36MB is used, so that a 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.

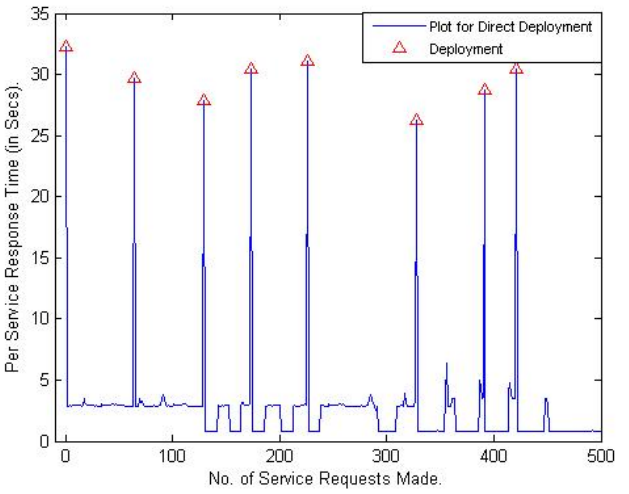


Figure 6.5: Experimental Results for Direct Download Mode

Figure 6.5 and Figure 6.6 depict the *service response time* for the two download modes. Figure 6.5 plots the time required for fetching a service using the *direct down-*

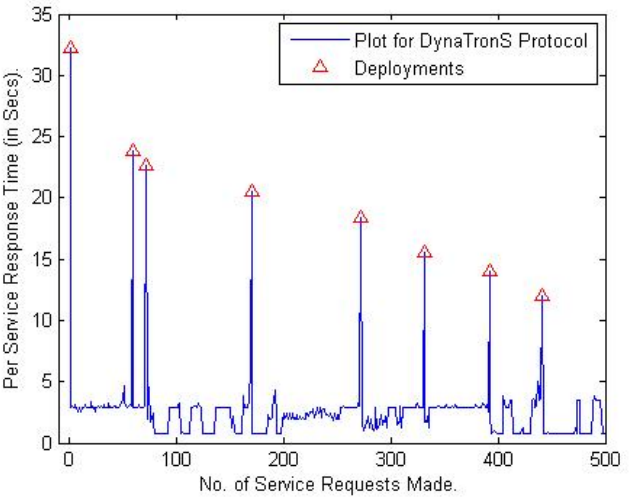


Figure 6.6: Experimental Results for p2p Download Mode

load mode, and Figure 6.6 shows the time required for fetching service using the p2p-based DynaTronS protocol. In these experiments, because of multiple service invocations, eight deployments were triggered. The experimental results shown in Figure 6.7 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 6.5 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 6.6 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 showing the benefit of the second mode using the peers in the *seed list* resulting in a sharp decrease in deployment time.

In a distributed environment, resources may be volatile in nature, which is handled well in a p2p framework. In the above experiments, node failures have been simulated by turning off nodes at random and observing that the service download process

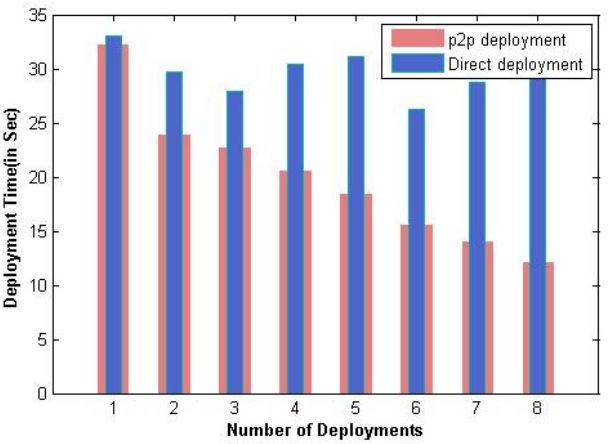


Figure 6.7: Experimental Results for Comparing p2p vs Direct Download Modes

continues even if one or more of the peers leave the network. If the failed node is an HP where deployment is being carried on, another deployment is triggered on another designated HP.

6.6 Summary

This chapter provides a robust approach to dynamic deployment of web services in a distributed environment. It also incorporates the benefits of p2p systems by decentralizing the registry of web services as resources. This architecture proposes a unique concept of dynamic on-demand service deployment using Bit-Torrent-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, the proposed techniques can handle increases in demand of services and can actually improve the performance of the network.

Chapter 7

Conclusions and Discussion

Come to an end of the thesis a summary of the overall research and its contributions are presented. A discussion about the opportunities for further work and possible improvements to the concept of dynamic service provisioning are also presented.

7.1 Overview of the Thesis

This section summarizes and discusses the main work and contributions presented in earlier chapters. The claims made in the introductory chapters are scrutinized with respect to the results obtained during the evaluation to verify their validity.

In this research work an architecture is designed for enabling dynamic on-demand service discovery and deployment based on the concepts of p2p computing. The conceived framework attempts to use the idle resources within the network by deploying services on distributed resources on the basis of their capabilities and load factors. The services are made available by the service providers and are deployed on-demand, after proper matchmaking of the service metrics with the capabilities of resources in order to provide better performance. Successive deployments of an already deployed service are triggered if the existing deployed instances are overloaded or fail to offer any response with the desired QoS.

In *dynamic service provisioning, service management* (that is availability of the service), resource management (scalability of the service and load balancing), dynamic service deployment (deplyment cost management) are important issues to achieve better performance of the overall architecture. The major contributions of this work are

summarized in the following sections.

7.1.1 Dynamic Web Service Discovery and Deployments using De-centralized Registry- Chapter 4

An architecture is proposed for enabling dynamic on-demand service discovery and deployment based on the concepts of p2p computing. The main focus is on decentralizing the registry, making it discoverable and thus increasing the service availability and reducing the deployment cost by sharing the current deployments and making the services available from more than one endpoints. In order to increase the availability of any given service, the service must be hosted by multiple providers. In such a scenario, the cost of deployment of a service is incurred separately at different service providers. Thus, with the objective of provisioning web services on-demand, in this architecture on de-centralizing the registry is considered, thereby making it discoverable and thus increasing the service availability and reducing the deployment costs by sharing the current deployments.

The implementation mentioned in this portion is based on DHT implementations, such as CAN, Pastry, Tapestry and Chord. Among these DHT implementations, the architecture presented in this thesis makes use of Chord due to its easy ring shaped structure which is achieved by the concept of consistent hashing and provides 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.

7.1.2 Request Scheduling : A Load Balanced Approach- Chapter 5

The implementation of the architecture requires a load balanced approach which would provide the desired QoS. In dynamic web service provisioning, services are deployed

on the fly on available resources, i.e. on Host Providers (HPs). In this scenario, once a service is deployed on a node (HP) by receiving a request from consumer, it remains deployed on that node until explicitly removed so that it can serve maximum request of that service. So, to offer the consumer efficient and fast services, it is required to minimize the average response time. However, for that managing the resources (HPs), i.e. balancing load of those resources is important. In this architecture, the tasks, i.e. web service requests which are submitted to the WSP tier are of unpredictable nature and are submitted on the fly. Not only this, the service request may vary from one web service to another. It is obvious that the performance of the architecture may bank on the HPs for executing service requests. But HPs may get loaded if proper method of scheduling service requests is not adopted by the WSPs.

This architecture uses dynamic load balancing approaches to ensure better performance for its consumers. It first makes use of the minimum service requirements information along with node configuration information, to prevent over utilization of the resources. This is achieved by constraining the deployments on HPs with insufficient resources as compared to the minimum service requirements. To achieve dynamic load balancing of the service requests, a major pre-requisite is to calculate the present workload of an HP. Once the workload is determined and collected by the WSPs, it runs some scheduling algorithm to decide a best suitable node to serve the consumer requests. Thus, a proper load balancing approach and a good scheduling strategies are needed to ensure better performance for dynamic web service provisioning.

7.1.3 P2P-Based Service Distribution: DynaTronS protocol- Chapter 6

Within the framework, decentralized service registry, efficient catering of consumer requests are only possible if requested services can be deployed in timely manner. However, 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 Bit-Torrent 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 file and services. This thesis proposes incorporation of Bit-Torrent protocol in the existing framework [91] and demonstrates its advantages by making use of such a file sharing technique to download service packages from the existing deployments to cater to new deployments.

A robust approach to dynamic deployment of web services in a distributed environment is provided. 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, the techniques proposed in this thesis can handle increase in demand of services and can actually improve the performance of the network.

7.2 Limitations

One of the shortcoming of the architecture can be its security in terms of migration of the service packages. Not only this, the HPs on which the service may be deployed may not be trustworthy and may put the service to wrong use. At present the architecture does not provide with methods ensuring security of the services and authentication of the HPs. Though it provides a separate communication channel to carry out the service deployments and checksum to ensure the service's integrity, these approaches may not be enough to cope with the present Internet technologies. Also the terms security and privacy of the service prove to subjective in nature, in the sense of how, what, and to whom the services must be made available; and can be considered as a separate research topic all over.

Another limitation which drives in the architecture is by the use of DHT for the service registry. DHT enables only precise-match search queries and not partial-match search queries [13]. Due to this, a random search query made to the registry will not provide with good search results, until the query matches exactly to some names of service presently hosted in the network.

Though the architecture tries to provision web services, it may run out of resources

that may match the service requirements. During the resource discovery process as discussed in Chapter 3, the *Best Node Finder* (Algorithm 2) may not find any resource suitable for the current web service. From another point of view, since the algorithm uses a policy of finding an HP with a configuration greater than or at least equal to that of the minimum service requirement, it may so happen that a web service may be deployed on a HP of higher configuration. This may not only lead to improper use of resources but may also require higher cost for leveraging a better HP. With respect to balancing the load over the HPs, categorizing and calculating the load on the basis of web service being data intensive or CPU intensive may provide a better formalization for calculating the present load of the HPs. Taking up more scheduling algorithms and comparing them with existing works, may provide a wider perspective of resource scheduling and hence adopting the best suited strategy among them. A proper trial and error approach or a mathematical background for selection of time slice depending on the web service execution time can further provide a robust mechanism for load optimization.

The test beds upon which the experiments were performed are of very small scale, hence a better performance measure can be obtained if the tests are conducted over a large number of nodes with better configuration and dedicated servers.

7.3 Future Work

Looking into the future aspects of the architecture, it envisages a combination of versatile and flexible environment for provisioning web services on demand. With advancement in technology and increase in devices accessing the Internet, a demand of distributed applications is at its peak. One major need for the proposed architecture is that it requires computational resources for deploying the web services. The architecture makes a choice among HPs available depending on the requirements of the web services. At present, it utilizes the resources provided as HPs, which are made available as resources within the distributed environment. These resources may be a part of a virtual organizations or may be provided by individual users whose configurations may be unalterable or may not meet our requirements in some way or the other. To cope with such scenarios of provisioning computational resources as well, a fusion with

Platform as a Service (PaaS) in a cloud environment may provide a good solution.

Further, starting from highly efficient computational resources to portable devices like mobile phones, computing devices are becoming a part of the Internet and hence can contribute their computational power as well as other resources for the betterment of the proposed architecture. This can be accomplished by further integrating the standard web service technologies like SOAP and WSDL to interface with the web services. Not only this, for such portable devices the application needs a proper transformation to suit the device requirements like power management.

As the QoS maintained at present is load calibrated, it opens up wide opportunities for researches to enable the architecture with facilities where the user chooses a cost model appropriate for its own requirement directly from the interface, adhering to Service Level Agreement (SLA) and hence may further enhance the flexibility of the architecture.

Bibliography

[1] L. Srinivasan and J. Treadwell, "An overview of service-oriented architecture, web services and grid computing," *HP Software Global Business Unit*, vol. 2, 2005. xv, 6, 7

[2] D. B. Claro, P. Albers, and J.-K. Hao, "Web services composition," in *Semantic Web Services, Processes and Applications*. Springer, 2006, pp. 195–225. xv, 9, 10

[3] P. Watson, C. Fowler, C. Kubicek, A. Mukherjee, J. Colquhoun, M. Hewitt, and S. Parastatidis, "Dynamically deploying web services on a grid using dynasoar," in *Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006. Ninth IEEE International Symposium on*, april 2006, p. 8 pp. xv, 3, 12, 24, 26

[4] a. Harrison and I. Taylor, "Wspeer-an interface to web service hosting and invocation," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 175a–175a. xv, 3, 12, 24, 27

[5] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 17–32, 2003. xv, 69

[6] Napster. [Online]. Available: http://www.bittorrent.org/introduction.html xv, 95, 96

[7] I. Foster, "The anatomy of the grid: Enabling scalable virtual organizations," in *Euro-Par 2001 Parallel Processing*, ser. Lecture Notes in Computer Science,

R. Sakellariou, J. Gurd, L. Freeman, and J. Keane, Eds. Springer Berlin Heidelberg, 2001, vol. 2150, pp. 1–4. 1, 19, 24

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.8105 2, 24, 33

[9] The globus toolkit. [Online]. Available: http://www.globus.org/toolkit 2, 16

[10] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor - a distributed job scheduler," in *Beowulf Cluster Computing with Linux*, T. Sterling, Ed. MIT Press. 2, 16, 24

[11] W. G. S. Microsystems), "Sun grid engine: Towards creating a compute power grid," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, ser. CCGRID '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 35–. [Online]. Available: http://dl.acm.org/citation.cfm?id=560889.792378 2, 16

[12] D. Sprott and L. Wilkes. (2004) Understanding service-oriented architecture. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa480021.aspx 2, 6, 16, 17

[13] C. Wang and B. Li, "Peer-to-peer overlay networks: A survey," Tech. Rep., 2003. 2, 112

[14] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010. 2

[15] P. Watson and C. Fowler, "Dynasoar: An architecture for the dynamic deployment of web services on a grid or the internet." 3

[16] L. Qi, H. Jin, I. Foster, and J. Gawor, "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.* IEEE, 2007, pp. 155–162. 3, 12, 24, 30

[17] R. Mondéjar, P. García, and C. Pairot, "Towards a decentralized p2pweb service oriented architecture," 2006. 3, 12, 29, 59

[18] A. Y. Zomaya *et al.*, "Parallel and distributed computing handbook," 1996. 3

[19] F. Schneider and A. Tanenbaum, "Distributed systems," *ch. Replication Management using the State Machine Approach*, pp. 169–198, 1993. 4

[20] Z. Mahmood, "Synergies between soa and grid computing," *Communications of the IBIMA*, vol. 8, 2009. 8, 17, 18

[21] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (xml)," *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*, vol. 16, p. 16, 1998. 8

[22] Web services description language (wsdl) 1.1, w3c note 15 march 2001,. [Online]. Available: http://www.w3.org/TR/wsdl 8, 12, 16

[23] Simple object access protocol (soap) 1.1, w3c note 08 may 2000. [Online]. Available: http://www.w3.org/TR/2000/NOTE-SOAP-20000508 8, 12, 16

[24] Universal description, discovery and integration(uddi) v3.0.2,. [Online]. Available: http://uddi.org/pubs/uddi-v3.0.2-20041019.htm 12, 20

[25] Z. Mahmood, "Synergies between soa and grid computing," *Communications of the IBIMA*, vol. 8, pp. 164–169, 2009. 17

[26] C. Catlett, "Standards for grid computing: Global grid forum," *Journal of Grid Computing*, vol. 1, no. 1, pp. 3–7, 2003. 19

[27] R. Cailliau, "A little history of the world wide web," *World Wide Web Consortium*, 1995. 19

[28] G. A. Moore, "Crossing the chasm," 2002. 19

[29] I. Foster, "The globus toolkit." 19

[30] R. Buyya and S. Venugopal, "The gridbus toolkit for service oriented grid and utility computing: An overview and status report," in *Grid Economics and Business Models, 2004. GECON 2004. 1st IEEE International Workshop on.* IEEE, 2004, pp. 19–66. 19

[31] P. D. Gurav, M. R. Hans, and N. A. Kulkarni, "Review: Role of cloud computing in grid empowerment," in *Automatic Control and Dynamic Optimization Techniques (ICACDOT), International Conference on.* IEEE, 2016, pp. 258–263. 19

[32] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *IN PROC. ACM SIGCOMM 2001*, 2001, pp. 161–172. 21, 61, 68

[33] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 1, pp. 17 – 32, feb 2003. 21, 59, 61, 68, 70, 73

[34] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," 2001. 21, 61, 68

[35] B. Zhao, J. Kubiatowicz, A. Joseph *et al.*, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," 2001. 21, 61, 68

[36] Gnutella. [Online]. Available: http://gnutella.wego.com 21, 65

[37] Napster. [Online]. Available: http://www.napster.com 21, 65

[38] J. Y. Bakos, "Reducing buyer search costs: Implications for electronic marketplaces," *Manage. Sci.*, vol. 43, no. 12, pp. 1676–1692, Dec. 1997. [Online]. Available: http://dx.doi.org/10.1287/mnsc.43.12.1676 21

[39] K. Arnold, "The jini architecture: dynamic services in a flexible network," in *Design Automation Conference, 1999. Proceedings. 36th.* IEEE, 1999, pp. 157–162. 21, 31

[40] J. Duddington, "espeak 1.36," *See http://espeak. sourceforge. net*, 2008. 21

[41] S. Consortium *et al.*, "Salutation architecture specification," 1999. 21

[42] E. Al-Masri and Q. H. Mahmoud, "Investigating web services on the world wide web," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 795–804. 21

[43] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne *et al.*, "Owl-s: Semantic markup for web services," *W3C member submission*, vol. 22, pp. 2007–04, 2004. 21

[44] R. Lara, D. Roman, A. Polleres, and D. Fensel, "A conceptual comparison of wsmo and owl-s," in *Web services*. Springer, 2004, pp. 254–269. 21

[45] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, "Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services," *Information Technology and Management*, vol. 6, no. 1, pp. 17–39, 2005. 21

[46] E. Al-Masri and Q. H. Mahmoud, "Wsce: A crawler engine for large-scale discovery of web services," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, 2007, pp. 1104–1111. 22

[47] L. Gong, "Jxta: A network programming environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88–95, 2001. 22

[48] D. Talia, P. Trunfio, J. Zeng, and M. Högqvist, "A dht-based peer-to-peer framework for resource discovery in grids," *Institute on System Architecture, CoreGRID Technical Report*, 2006. 23

[49] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne *et al.*, "Daml-s: Web service description for the semantic web," in *International Semantic Web Conference*. Springer, 2002, pp. 348–363. 23

[50] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl, "Hypercuphypercubes, ontologies, and efficient search on peer-to-peer networks," in *International Workshop on Agents and P2P Computing*. Springer, 2002, pp. 112–124. 23

[51] U. Thaden, W. Siberski, and W. Nejdl, "A semantic web based peer-to-peer service registry network," *Technical Report*, 2003. 23

[52] S.-D. Wang, H.-L. Ko, and Y.-Y. Zhuang, "Japster: An improved peer-to-peer network architecture," in *International Conference on Embedded and Ubiquitous Computing*. Springer, 2004, pp. 1044–1054. 24

[53] D. Castella, I. Barri, J. Rius, F. Giné, F. Solsona, and F. Guirado, "Codip2p: A peer-to-peer architecture for sharing computing resources," in *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*. Springer, 2009, pp. 293–303. 24, 31

[54] O. D. Sahin, C. E. Gerede, D. Agrawal, A. El Abbadi, O. Ibarra, and J. Su, "Spider: P2p-based web service discovery," in *International Conference on Service-Oriented Computing*. Springer, 2005, pp. 157–169. 24

[55] W. Fang, L. Moreau, R. Ananthakrishnan, M. Wilde, and I. Foster, "Exposing uddi service descriptions and their metadata annotations as ws-resources," in *2006 7th IEEE/ACM International Conference on Grid Computing*, Sept 2006, pp. 128–135. 25

[56] A. Harrison and I. Taylor, "Dynamic web service deployment using WSPeer," in *Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies*. Louisiana State University, Feb. 2005, pp. 11–16. 28

[57] I. Wang, "P2ps (peer-to-peer simplified)," in *Proceedings of 13th Annual Mardi Gras Conference-Frontiers of Grid Applications and Technologies*, 2005, pp. 54–59. 28, 62

[58] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," Tech. Rep., 1999. 28

[59] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Importing the semantic web in uddi," in *International Workshop on web services, e-business, and the semantic web*. Springer, 2002, pp. 225–236. 28

[60] C. P. Gavaldà, P. G. López, and A. G. Skarmeta, "Dermi: A new distributed hash table-based middleware framework," *IEEE Internet Computing*, vol. 8, no. 3, pp. 74–84, 2004. 29

[61] C. Pairot, P. García, R. Mondéjar, and A. F. G. Skarmeta, "p2pcm: a structured peer-to-peer grid component model," in *International Conference on Computational Science*. Springer, 2005, pp. 246–249. 29

[62] R. Mondejar, P. Garcia, C. Pairot, and A. F. G. Skarmeta, "Enabling wide-area service oriented architecture through the p2pweb model," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE'06. 15th IEEE International Workshops on*. IEEE, 2006, pp. 89–94. 29

[63] V. Mesaros, B. Carton, and P. Van Roy, "P2ps: Peer-to-peer development platform for mozart," in *International Conference on Multiparadigm Programming in Mozart/OZ*. Springer, 2004, pp. 125–136. 31

[64] R. Gupta, V. Sekhri, and A. K. Somani, "Compup2p: An architecture for internet computing using peer-to-peer networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 11, pp. 1306–1320, 2006. 31

[65] S. Wu and Z. Du, "Globalstat: A statistics service for diverse data collaboration and integration in grid," in *High-Performance Computing in Asia-Pacific Region, 2005. Proceedings. Eighth International Conference on*. IEEE, 2005, pp. 6–pp. 32

[66] P. Danielis, J. Skodzik, V. Altmann, B. Kappel, and D. Timmermann, "Extensive analysis of the kad-based distributed computing system dude," in *Computers and Communication (ISCC), 2015 IEEE Symposium on*. IEEE, 2015, pp. 128–133. 32

[67] Napster. [Online]. Available: http://jmdns.sourceforge.net/ 43

[68] Monitoring and discovery system. [Online]. Available: http://www.globus.org/toolkit/mds/ 60

[69] F. Dabek, "A distributed hash table," Tech. Rep., 2005. 61, 66, 70

[70] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005. 63

[71] S. Ertel, "Unstructured p2p networks by example: Gnutella 0.4, gnutella 0.6," *Proceeding of Dresden University of Technology*, 2014. 64

[72] I. Ivkovic, "Improving gnutella protocol: Protocol analysis and research proposals," *Prize-Winning Paper for LimeWire Gnutella Research Contest*, 2001. 64

[73] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 46–66. 64

[74] P. Fraigniaud and P. Gauron, "The content-addressable network d2b," 2003. 64

[75] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," *IEEE/ACM Transactions on Networking (ToN)*, vol. 12, no. 2, pp. 219–232, 2004. 64

[76] C. Huitema and J. L. Miller, "Peer-to-peer name resolution protocol (pnrp) and multilevel cache for use therewith," Jun. 20 2006, uS Patent 7,065,587. 64

[77] Freenet. [Online]. Available: https://freenetproject.org/ 65

[78] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell, "A survey of peer-to-peer storage techniques for distributed file systems," in *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, vol. 2. IEEE, 2005, pp. 205–213. 65

[79] M. Parameswaran, A. Susarla, and A. B. Whinston, "P2p networking: an information sharing alternative," *Computer*, vol. 34, no. 7, pp. 31–38, 2001. 66

[80] R. Morris, M. F. Kaashoek, D. Karger, H. Balakrishnan, I. Stoica, D. Liben-Nowell, and F. Dabek, "Chord: A scalable peer-to-peer look-up protocol for internet applications," *IEEE/ACM Transactions On Networking*, vol. 11, no. 1, pp. 17–32, 2003. 68

[81] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, "Building peer-to-peer systems with chord, a distributed lookup service," in *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on.* IEEE, 2001, pp. 81–86. 68

[82] S. Zoels, M. Eichhorn, A. Tarlano, and W. Kellerer, "Content-based hierarchies in dht-based peer-to-peer systems," in *Applications and the Internet Workshops, 2006. SAINT Workshops 2006. International Symposium on.* IEEE, 2006, pp. 4–pp. 69

[83] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load distributing for locally distributed systems," *Computer*, vol. 25, no. 12, pp. 33–44, 1992. 80

[84] W. Winston, "Optimality of the Shortest Line Discipline," *Journal of Applied Probability*, vol. 14, no. 1, pp. 181–189, 1977. [Online]. Available: http://dx.doi.org/10.2307/3213271 84

[85] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis," in *International Workshop on Peer-to-Peer Systems.* Springer, 2005, pp. 205–216. 94, 97

[86] M. D. Hanson, "The client/server architecture," *Server Management*, p. 3, 2000. 95

[87] K. Davis, J. W. Turner, and N. Yocom, "Client-server architecture," in *The Definitive Guide to Linux Network Programming.* Springer, 2004, pp. 99–135. 95

[88] R. Toole and V. Vokkarane, "Bittorrent architecture and protocol," *The University of Massachusetts, Dartmouth*, 2006. 95

[89] J.-F. Paris and P. Shah, "Peer-to-peer multimedia streaming using bittorrent," in *Performance, Computing, and Communications Conference, 2007. IPCCC 2007. IEEE Internationa.* IEEE, 2007, pp. 340–347. 97

[90] J. Postel and J. Reynolds, "File transfer protocol," 1985. 97

[91] S. Mistry, D. Jaiswal, S. Virani, A. Mukherjee, and N. Mukherjee, "An architecture for dynamic web service provisioning using peer-to-peer networks," in *Distributed*

Computing and Internet Technology, ser. Lecture Notes in Computer Science, C. Hota and P. Srimani, Eds. Springer Berlin Heidelberg, 2013, vol. 7753, pp. 290–301. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36071-8_23112

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 ~~re-sources~~. 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

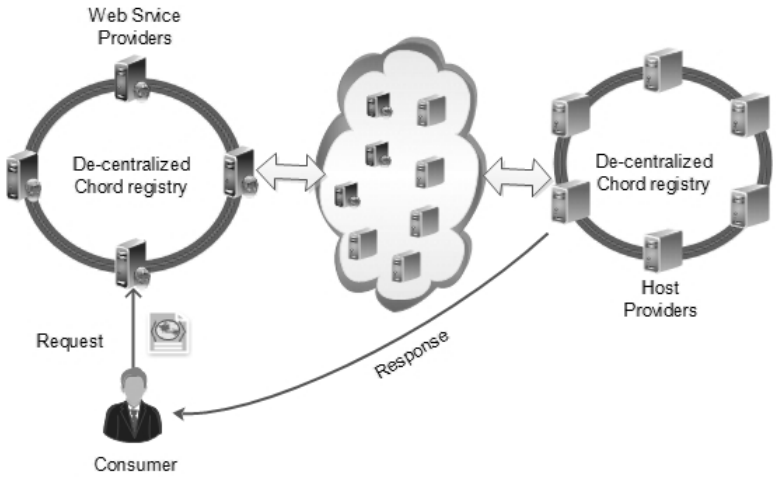


Figure 1. Basic Architecture

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 $M = \{M_1, M_2, M_3, ..., M_n\}$
P =	Node Properties
NT_m =	A mapping between the nodes and WSPs and HPs according to their properties $NT_m : M \rightarrow P(WR) \wedge M \rightarrow P(HP)$
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 $D_m : WS \rightarrow HP$
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,

Σ = 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*, *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.

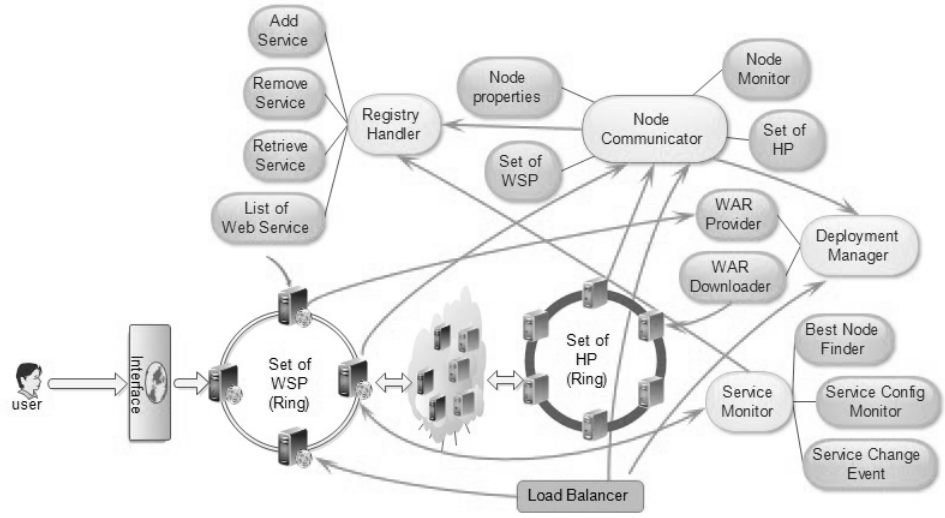


Figure 2. Components in the Architecture

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).

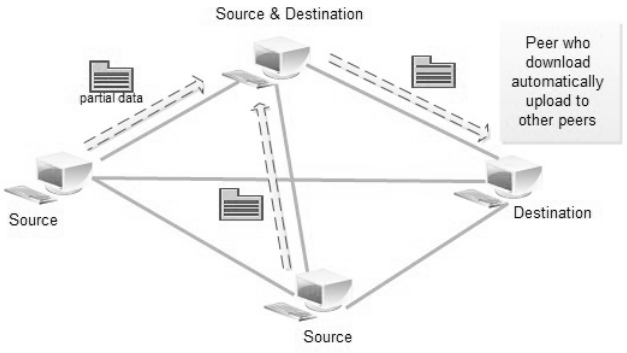


Figure 3. Bit-Torrent Protocal

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.

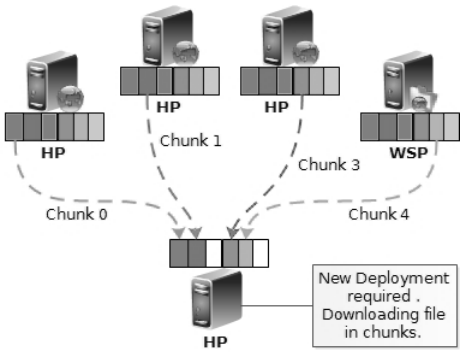


Figure 4. DynaTronS Deployment

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

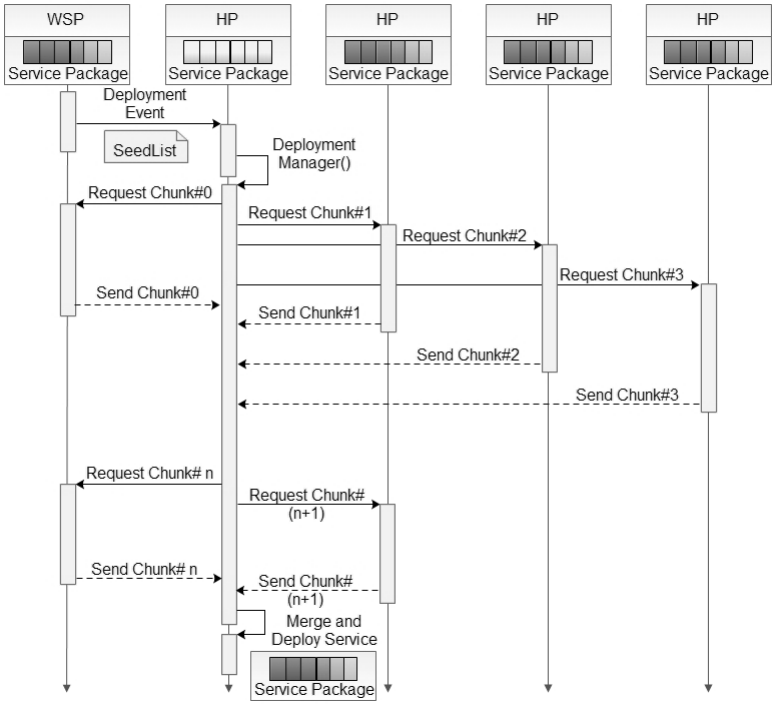


Figure 5. DynaTronS Sequence Diagram

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

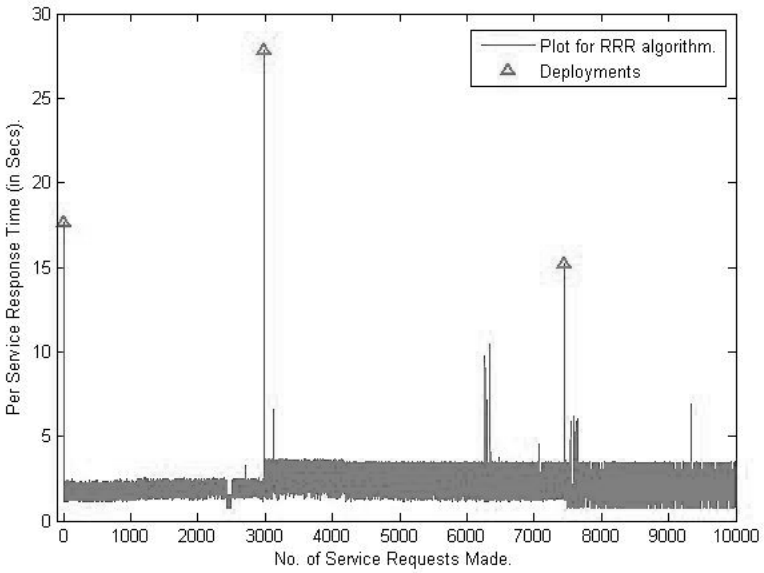


Figure 6. Plot for Round-Robin Reloaded]

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

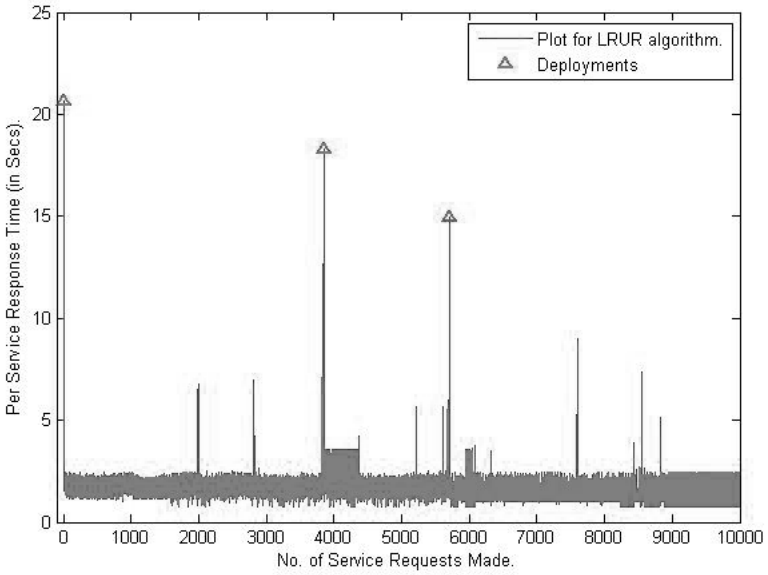


Figure 7. Plot for Least Recently used Reloaded

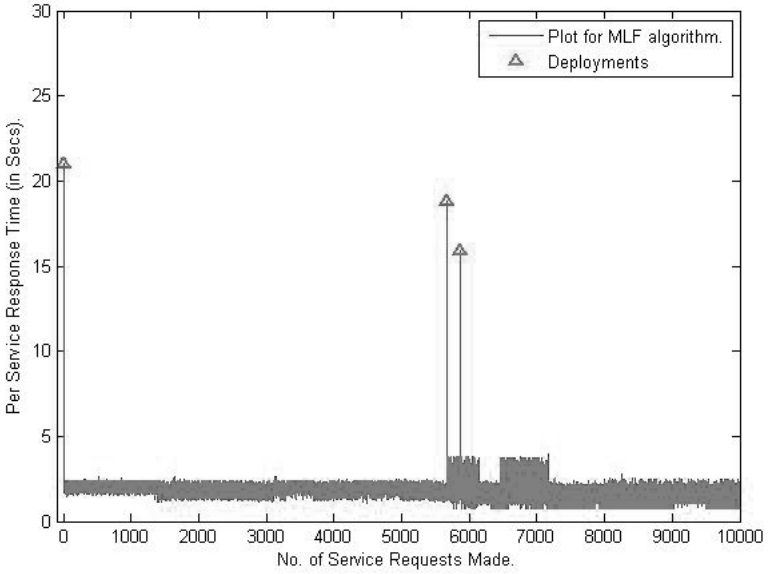


Figure 8. Plot for Minimum Loaded First

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.

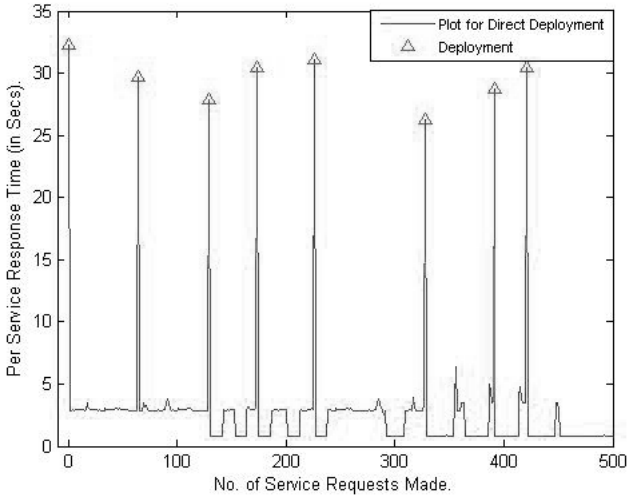


Figure 9. Experimental Results for Direct Download Mode

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

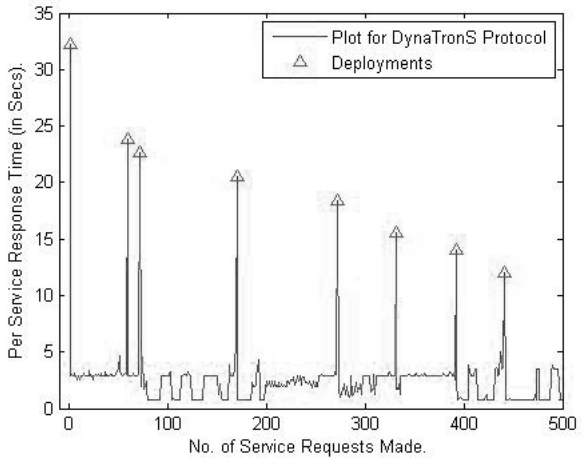


Figure 10. Experimental Results for P2P Download Mode

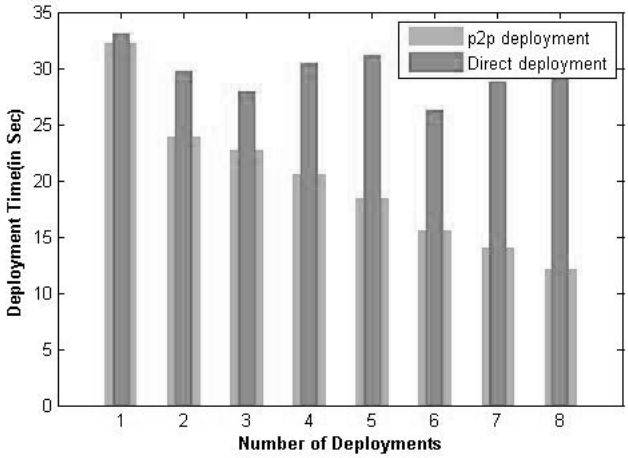


Figure 11. Experimental Results for Comparing P2P vs Direct Download Modes

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.