Efficient Service Provisioning through Dynamic Service Task Assignment in a Multi-Domain Distributed Computing Environment

Malamati Louta

Department of Business Administration
Technological Educational Institute of Western Macedonia
Koila, Kozani, 50100, Greece, louta@telecom.ntua.gr

Department of Information and Communication Technologies Engineering
University of Western Macedonia
Kozani, 50100, Greece

Angelos Michalas

Department of Informatics and Computer Technology
Technological Educational Institute of Western Macedonia
Fourka, Kastoria, 52100, Greece, amichalas@kastoria.teikoz.gr

Abstract: Highly competitive and open environments should encompass mechanisms that will assist service providers in accounting for their interests, i.e., offering at a given period of time adequate quality services in a cost efficient manner. Assuming that a user wishes to access a specific service composed of a distinct set of service tasks, which can be served by various candidate service nodes, a problem that should be addressed is the assignment of service tasks to the most appropriate service nodes. This scenario accounts for both the user and the service provider. Specifically, service providers succeed in efficiently managing their resources, while users implicitly exploit in a seamless way the otherwise unutilized power and capabilities of the provider's network. In general, service task assignment is founded on general and service specific user preferences, service provider's specific service logic deployment and current system & network load conditions. The pertinent problem is concisely defined, optimally formulated and evaluated through simulation experiments on a real network test bed.

Keywords: Service Provisioning, Service Task Assignment, Resource Allocation, Intelligent Agents, Distributed Computing Environments

1 Introduction

Service provisioning in liberalised, deregulated and competitive telecommunication market is a quite complex process since it involves various diverse actors (e.g., users, service providers, (third party) application (content) providers, brokers, network

providers). The following are some key factors for success. First, the efficiency with which services will be developed. Second, the quality level, in relation with the corresponding cost, of new services. Third, the efficiency with which the services will be operated, controlled, maintained, administered, etc. Fourth, the personalisation and tailoring of services and applications to the user needs and preferences. The aim of this paper is, in accordance with cost-effective QoS provision and efficient service operation objectives, to propose enhancements to the sophistication of the functionality that can be offered by service frameworks in open competitive communications environments.

In accordance with the service oriented architectures concept (Parlay; OSGi, 1999; Benatallah, 2003) and exploiting advanced software paradigms (e.g., distributed object computing (Vinoski, 1997) and intelligent mobile agents (Morreale, 1998; Jennings, 1998)), the service logic is realised by a set of autonomous co-operating components, which interact through middleware functionality that runs over Distributed Processing Environments (e.g., CORBA, Parlay). Limited by techno-economic reasons or considering administrative, management and resilience/ redundancy purposes it is assumed that each service provider deploys service components realising service logic in different service nodes, residing in the same and/or different domains. In the context of this paper, domains represent different network segments, thus, a hierarchical network structure is adopted. Moreover, it can be envisaged that a service will in general comprise a set of distinct service tasks, which could be executed by different service nodes.

Highly competitive and open environments should encompass mechanisms that will assist service providers in accounting for their interests, i.e., offering at a given period of time adequate quality services in a cost efficient manner, which is highly associated to efficiently managing and fulfilling current user requests. Thus, assuming that a user wishes to access a specific service composed of a distinct set of service tasks, which can be served by various candidate service nodes (CSNs), a problem that should be addressed is the allocation of service tasks to the most appropriate service nodes. In this paper, the pertinent problem is called *service task assignment*. The aim of this paper is to address the problem from one of the possible theoretical perspectives and to show the software architecture that supports its solution and how it can be incorporated in service architectures that run in the open environment.

This study is related to pertinent previous work in the literature, since efficient resource utilisation, load balancing and job scheduling are topics that attract the attention of the researchers as *computational grids* (interconnected networks of super-computing centers) have become an emerging trend on high performance computing (Special Issue, 2003). Most studies in the field of resource allocation schemes aim at efficiently utillising the otherwise unutilized powers of resources spread throughout a network. In most cases, the problem is reduced to load balancing among specific nodes. Different global objectives could be considered, such as minimization of mean service/task completion time, maximization of resources utilization (e.g., CPU time), minimization of mean response ratio (Tanenbaum, 2001).

The contribution of this paper lies in the following areas. First, the definition and mathematical formulation (one possible version) of the service task assignment problem, considering a multi domain distributed computing environment. Our approach takes into account the communication complexity introduced between the service components involved in service provisioning process and, thus, a model for the communication cost involved is provided. Through this work it is shown that the overall problem can be reduced to well-known optimisation problems, which can be solved by relevant standard

algorithms. Second, the presentation of a software architecture that supports the proposed solution and may be incorporated in service architectures that run in the open environment.

The approach in this paper is the following. The starting point (section 2) is the general description of the service task assignment concept, through the presentation of a relevant business case, while the software elements required for the realisation of the service task assignment process are identified. Additionally, our assumptions regarding the system model are presented. Sections 3 and 4 present a concise definition, mathematical formulation and optimal solution of the service task assignment problem, while one possible formulation of the communication cost taken into account in our framework is provided. Section 5 gives a set of experimental results on a network test bed, indicative of the efficiency of the proposed service task assignment scheme. In section 6 the related research literature is briefly revisited. Finally, in section 7 conclusions are drawn and direction for future plans are presented.

2 General Presentation of the Service Task Assignment Concept

This section starts from the description of the business case (sub-section 2.1), through which the role and importance of the service task assignment concept can be understood. Sub-section 2.2 provides the software architecture in terms of computational level components that supports the proposed framework, while in sub-section 2.3 our assumptions on the system model are given.

2.1 Description in terms of business level entities

Assume that a user wishes to access a specific service offered by a service provider. The service is composed by a distinct set of service tasks. Each service task can be served by various Candidate Service Nodes (CSNs), as depicted in Figure 1. The choice of the most appropriate service node engagement for the completion of each service task (service task assignment process) requires the realisation of the three general phases illustrated in Figure 2.

The first general phase involves service independent tasks like user authentication, authorisation, etc. It involves the user and an entity that will be called *Default Service Provider* (DSP) residing in the Default Domain (DD). In essence, at the end of this phase the user is enabled to request services. This phase will not be further addressed in this paper.

At the second phase, the service task assignment process is conducted by the *Service Provider* (SP) entity, which is specialised in the assistance of the service provider in the open competitive communication environment. The SP can accomplish this by providing, maintaining and hosting (essential parts of) the software that will conduct the service task assignment process. In this respect, the SP is assumed to play a co-ordinating role in the second general phase, which is the core of the service task assignment process. At this point the user has expressed the wish to access a given service. Involved in this phase will be the SP, the DSP, the service provider's CSNs that could be deployed for the provision of the service and the Network Provider related entities (NPs) in order to handle the network resources (e.g., bandwidth) required for service provision. In general, service task assignment is founded on general and service specific user preferences and provider's specific service logic deployment. It should be noted that the appropriate SP is

determined by the DSP at the end of the first general phase on the basis of the user preferences and requirements regarding the requested service.

In the third phase of the business case the result of the service task assignment is available, and hence the service usage can possibly start, in accordance with the specific task assignment provided during the previous stage.

At this point, some concepts concerning the business case can be outlined. Specifically, the scenario presented accounts for both the user and the service provider. Specifically, service providers succeed in better managing their resources, while users implicitly exploit in a seamless and transparent way the otherwise unutilised power and capabilities of the provider's network. Thus, the SP assists service providers in equitably and efficiently distribute their resources, in essence leading to a higher level QoS service provision to the users.

Based on the described business case we may provide the high level definition of the pertinent design problem. This means that we should define the cost function and specify the constraints that derive from the requirements of (primarily) the user with respect to the service requested and the provider's policies, in conjunction with the current load conditions as well as the capabilities of the service nodes and the network resources availability. The solution in our case should provide the minimum cost assignment of service tasks to service nodes.

The user requirements may be characterised in terms of service preferences. Service preferences yield the service tasks needed for the service provision, as well as the load that will originate from each service task, which may be expressed in terms of an associated with each service task, CPU time, memory and disk resources. In essence, these values correspond to the service node CPU time, memory and disk space required by the service task, so as it is adequately provided.

The cost function of the service task assignment problem may consist of the following factors. First, the cost of the service nodes that need to be deployed (involved in the solution). Second, the communication cost of components between the service node that has primarily undertaken the execution of the service task requested and the service nodes that may as well be involved in the accomplishment of the specific service task (e.g., one may consider the case of a service task requiring additional processing to data retrieved from a database server). Third, the management cost introduced due to the assignment of service tasks constituting a service to different service nodes. The constraints of the problem derive from the capabilities of the service nodes. These may be expressed in terms of their maximum resources (i.e., CPU time, memory and disk space), and probably, the maximum number of tasks they can control at the same time (e.g., number of parallel sessions). Regarding the network resources, the link capacity constraint is considered.

Taking into account the aspects outlined above, a general problem statement may be the following. Given the set of service nodes and their layout, the set of service tasks constituting the required service, the resource requirement of each service task in terms of CPU time, memory and disk space, the cost of deploying each service node, the current load conditions of each service node and of the network links, find the minimum cost assignment of tasks to service nodes (in terms of the number of nodes that need to be deployed, the communication cost introduced during the execution of service tasks, and the management cost imposed by the arrangement) subject to a set of constraints, associated with the capabilities of the service nodes and the network resources availability.

2.2 Description in terms of computational level components

Service Architectures (e.g., Parlay) comprise activities that allow user authentication, user profile control (inspection), and service invocation. In our framework, the *Default Service Provider Agent* (DSPA) is the component that enables the initial access to a domain.

The feature that is not supported is the overall task of the service task assignment. As a first step, this process requires a computational component that will act on behalf of the user. Its role will be to capture the user preferences, requirements and constraints regarding the requested service and to deliver them in a suitable form to the appropriate service provider entity. As a second step, service task assignment requires an entity that will act on behalf of the service provider. Each role will be to intercept user requests, acquire and evaluate the corresponding service node and network load conditions, and ultimately, to select the most appropriate service nodes for the realisation of the service. Furthermore, a monitoring module is required. Monitoring module consists of a distributed set of agents, which run on each service node of the service provider. Each agent is responsible for monitoring the load conditions and available resources of the service node and delivering them to the service provider related entity. Finally, a distributed set of network provider related entities will be responsible for providing the service provider entity with network load conditions and managing the network connections necessary for the service provision.

The following key extensions are made so as to cover the functionality that was identified above. First, the *Service Provider Agent* (SPA) is introduced and assigned with the role of selecting on behalf of the service provider the best service task assignment pattern. Second, the *User Agent* (UA) is assigned with the role of intercepting and processing user requests and promoting the service requests to the appropriate SPA. Third, the *Service Node Agent* (SNA) is introduced and assigned with the role of promoting the current load conditions of a CSN. In essence, the distributed set of the SNAs forms the monitoring module. Finally, the *Network Provider Agent* (NPA) is introduced and assigned with the task of providing current network load conditions (i.e., bandwidth availability) to the appropriate SPA. In other words, the SPA interacts with the UA in order to acquire the user preferences, requirements and constraints, analyses the user request in order to identify the service tasks constituting the service and their respective requirements in terms of CPU time, memory and disk space, identifies the set of CSNs and their respective capabilities, interacts with the SNAs of the candidate service nodes so as to obtain their current load conditions and with the NPAs so as to acquire the network load conditions, and ultimately selects the most appropriate service task assignment pattern for the provision of the desired service.

In more detail the interactions among the computational level components are as follows. The UA interacts with the SPA and the SPA interacts with the SNA of each CSN and with the NPAs handling the network resources / links among the service nodes. The aim of the UA-SPA interactions is to supply the SPA with user preferences and constraints, while the aim of the SPA-SNA and SPA-NPA interactions is to obtain the corresponding load conditions of each CSN and of the respective links in order to select at the final stage the most appropriate service task-service node deployment pattern.

The tasks outlined require a method that will enable service providers to process the user's request and generate a service task assignment scheme, satisfying user's

preferences, requirements, and constraints, at the same time resulting in an efficient resource management scheme on the service provider's side.

2.3 System model

We consider a set of service nodes SN and a set of links L. Each service node $n_i \in SN$ corresponds to a server, while each link $l \in L$ corresponds to a physical link that interconnects two nodes $n_i, n_j \in SN$. Our system operates in a multi-tasking environment, i.e., several tasks may be executed on a single service node sharing its resources (e.g., CPU time, memory, disk space). Let D_i denote a set of nodes grouped to form a domain. In essence, domains represent different network segments. A pattern for the physical distribution of the related software components to the service task assignment scheme is given in Figure 3. Each SPA controls the service nodes of a domain. Each SNA is associated with each node, while each NPA is associated with the network elements (e.g., switches or routers) necessary for supporting service node connectivity. The SNA, NPA role (in a sense) is to represent the service nodes or network elements, respectively, and to assist SPA by providing information on the availability of resources of the service nodes / network elements. Domain state information (load conditions of the service nodes of the particular domain and link utilisation) is exchanged between the SPA and the SNAs/NPAs residing in the specific domain, while SPAs residing in different domains exchange their domain state info. This approach increases scalability as it reduces the requirements in terms of computation, communication and storage. At this point it should be noted that for simplicity reasons the network elements needed for the service node connectivity are not depicted in Figure 3.

In the scope of this paper we consider that the service nodes constituting a specific domain are interconnected by a local area network, while different domains are interconnected by a wide area network. In the current version of this study we limit our attention to the cases where a service request may be served by service nodes residing in a single domain (the domain that is identified by the DSP), since we consider that the cost imposed due to information transfer through the WAN links is big, diminishing the net benefit of possible efficient resource utilisation. Thus, in our study, in case a service request cannot be served by the service nodes of a domain, it is transferred to the SPA of another domain in order to handle the request. However, the formal analysis of the service task assignment problem and its optimal formulation is given in a general mode, since the emergence of high performance backbone infrastructure and test-beds like Tera-Grid (TeraGrid, 2003) promises remarkable network bandwidth between distant sites, enabling thus load balancing with minimal cost.

3 Formal Problem Statement

User u wishes to use a given service s. A fundamental assumption at this point is that service s may be decomposed in a set of distinct service tasks, which will be denoted as $ST(s)$. Among these service tasks, of interest to the user are those designated in the user profile and will be denoted as $ST(u,s)$ ($ST(u,s) \subseteq ST(s)$).

Let's assume the existence of multiple service nodes for the provision of service s, denoted by $SN(s) = \{n_1,...,n_{|s|}\}$. Each service node-$n_j$ contains a collection of

components, denoted as $A_{n_j}(i)$, which inter-work with other components that may reside in the same or in a different service node in order to accomplish each service task $i \in ST(s)$. Let A_{n_j} and C be the total set of components residing in the n_j service node and the various service nodes in total, respectively. Hence, the following relationship holds: $A_{n_j}(i) \subseteq A_{n_j} \subseteq C$. Each service task $i \in ST(s)$ may be executed on an associated set of possible candidate service nodes, represented by the set $SN(i)$, ($i \in ST(u,s)$). Thus, $SN(i) \subseteq SN(s)$. The service logic deployment pattern adopted by service providers determine each of these service node sets.

Task i, ($i \in ST(s)$) requires for its completion consumption of $r_{CPU}(i)$, $r_{mem}(i)$ and $r_{disk}(i)$ resources of service node(s) n_j, ($n_j \in SN(i)$). A realistic assumption is that SPA being in charge of assisting the service providers in the competitive telecommunication market, has a solid interest in as accurately as possible identifying the resources $r_a(i)$ (where $a \in \{CPU, mem, disk\}$) needed for the provisioning of service task i in terms of CPU utilization, memory and disk space. In this respect, the SPA can be the entity that configures these values based on the service task characteristics, user preferences and requirements, exploiting also previous experience.

Let c_D denote the cost of involving service node n_j, ($n_j \in SN(i)$), in the service provision. For notation simplicity it is assumed that the cost of involving a service node in the solution is the same for all service nodes. As an alternative this cost could be taken variant (depending on the cost of acquiring and/or maintaining the node etc.). Notation may readily be extended.

The objective of our problem is to find a service task assignment pattern, i.e., an assignment $A_{ST}(s)$ of service tasks i ($i \in ST(u,s)$) to service nodes n_j, ($n_j \in SN(i)$), that is optimal given the current load conditions and number of service tasks being served by each service node n_j, represented as $r_a^{pre}(n_j)$ and $k^{pre}(n_j)$, respectively. The assignment should minimise an objective function $f(s, A_{ST}(s))$ that models the overall cost introduced due to system/network resources consumption. Among the terms of this function there can be the overall cost due to the deployment of various service nodes to the service provisioning process, the communication cost introduced due to the interaction of the components A_{n_j} residing in n_j service node with the components A_{n_k} residing in service node n_k for the completion of each service task i, ($\forall i \in ST(s)$), as well as the management cost $c_M(i,i')$ introduced due to the assignment of $(i,i') \in ST^2(s)$ service tasks to different service nodes $(n_j, n_{j'}) \in SN^2(s)$.

The constraints of our problem are the following. First, each service task i ($i \in ST(u,s)$) should be assigned to only one service node n_j, ($n_j \in SN(i)$). Second, the capacity constraints of each service node should be preserved. Lets assume that $r_a^{\max}$ and $k^{\max}$ represent the maximum load and the maximum number of service tasks that a service node may handle. For notation simplicity, these parameters are assumed to be the same for each service node n_j, ($n_j \in SN(s)$). Thus, the constraints are

$r_a^{post}(n_j) \le r_a^{\max}$ and $k^{post}(n_j) \le k^{\max}$, $(\forall n_j \in SN(s))$, where $r_a^{post}(n_j)$ and $k^{post}(n_j)$ denote the potential load conditions of service node n_j, after the service task assignment process. Notation may readily be extended. The overall problem can be formally stated as follows.

Service Task Assignment Problem Description

Given:

(a) a user u who wants to use a service s,

(b) the profile of user u,

(c) the set of service tasks $ST(u,s)$ of service s that are of interest (relevant) to user u (this set is formed by the service specification, the user profile and the service provider's related capabilities),

(d) the set of service nodes $SN(s)$ and the set of candidate service nodes $SN(i)$ at which each service task i $(i \in ST(u,s))$ can be completed, according to the service specification, the service node capabilities and the preferences of user u,

(e) the communication cost introduced due to the interaction of the components A_{n_j} residing in n_j service node with the components A_{n_k} residing in service node n_k for the completion of each service task i, $(\forall i \in ST(s))$,

(f) the deployment cost c_D of each service node n_j involved in the service provisioning process, which derives from the assignment of service task i $(i \in ST(u,s))$ to service node n_j $(n_j \in SN(i))$,

(g) the management cost $c_M(i,i')$ introduced due to the assignment of $(i,i') \in ST^2(s)$ service tasks to different service nodes $(n_j, n_{j'}) \in SN^2(i)$,

(h) the current load conditions $r_a^{pre}(n_j)$ for each load type a and number of service tasks $k^{pre}(n_j)$ being executed on each service node n_j, $n_j \in SN(s)$,

(i) the capacity constraints of each service node $r_a^{\max}$ and $k^{\max}$,

(j) the resources $r_a(i)$ required for the completion of service task i, $(\forall i \in ST(s))$,

find the best service task configuration pattern, i.e., assignment of service tasks to service nodes $A_{ST}(s)$, that optimises an objective function $f(s, A_{ST}(s))$ that is related to the overall cost introduced by the assignment, under the constraints $r_a^{post}(n_j) \le r_a^{\max}$ and $k^{post}(n_j) \le k^{\max}$, and that each service task is assigned to exactly one service node.

In this respect, the combination of service tasks to service nodes that yields minimum cost will be selected.

4 Optimal Formulation

The general problem version presented is open to various solution methods. Its generality partly lies in the fact that the objective and the constraint functions are open to alternate implementations. Thus, the problem statement can be distinguished from the specific solution approach adopted hereafter. In order to describe the assignment $A_{ST}(s)$ of service tasks to service nodes we introduce the decision variables $x_{ST}(i,j)$

($i \in ST(u,s)$, $n_j \in SN(i)$) that take the value 1(0) depending on whether service task i is (is not) executed by service node-n_j. The decision variables $y_{SN}(j)$ assume the value 1(0) depending on whether candidate service node n_j ($n_j \in SN(i)$) is (is not) deployed (involved in the solution). In addition, we define the set of variables $z_{ST}(i,i')$ ($\forall(i,i') \in ST^2(u,s)$) that take the value 1(0) depending on whether the service tasks i and i' are (are not) assigned to the same service node. The variables $z_{ST}(i,i')$ are related to variables $x_{ST}(i,j)$, $x_{ST}(i',j)$, through the relation $z_{ST}(i,i') = \sum_{j=1}^{|SN(i)|} x_{ST}(i,j) \cdot x_{ST}(i',j)$, which may be turned into a set of linear constraints through the technique of (Papadimitriou, 1982). Assignment $A_{ST}(s)$ may be obtained by reduction to the following 0-1 linear programming problem.

Service Task Assignment Problem:

Minimise

$$f(s, A_{TN}(s)) = c_D \cdot \sum_{n_j \in SN(s)} y_{SN}(j) \cdot (1 + b \cdot \sum_{a \in \{CPU, memory, disk\}} w_a \cdot \frac{r_a^{pre}(n_j)}{r_a^{\max}(n_j)})$$
$$+ \sum_{i \in ST(s)} \sum_{n_j \in SN(i)} C(i,n_j) \cdot x_{ST}(i,j) + \sum_{i \in ST(s)} \sum_{i' \in ST(s)} c_M(i,i') \cdot (1 - z_{ST}(i,i')) \qquad (1),$$

where $C(i,n_j)$ denotes the communication cost introduced in case n_j service node has undertaken the responsibility for the execution of service task i ($i \in ST(u,s)$),

subject to the constraints:

$$\sum_{n_j \in SN(i)} x_{ST}(i,j) = 1 \qquad \forall i \in ST(s) \qquad (2),$$

$$r_a^{pre}(n_j) + \sum_{i \in ST(s)} r_a(i) \cdot x_{ST}(i,j) \le r_a^{\max}(j) \cdot y_{SN}(j) \quad \forall n_j \in SN(s) \qquad (3),$$

$$k^{pre}(n_j) + \sum_{i \in ST(s)} x_{ST}(i,j) \le k^{\max}(j) \cdot y_{SN}(j) \qquad \forall n_j \in SN(s) \qquad (4)$$

Cost function (1) penalises the aspects identified previously (i.e., cost of the service node involved in the solution, communication cost introduced during the realisation of each service task, and management cost of service tasks that are assigned to different service nodes). In order for the service providers to better utilize their resources, the cost of each service node deployment introduced in cost function (1) takes also into account the node's current load conditions in order to obtain a load balancing solution. Parameters β, ($\beta < 1$), and w_a denote the relative significance of load balancing and of each resource type a to the service provider. It is assumed that weights w_a for each resource type a are normalized to add up to 1 (i.e., $\sum_{a \in \{CPU, memory, disk\}} w_a = 1$). Constraints (2), guarantee that each service task will be assigned to one service node. Constraints (3) and (4) guarantee that each service node will not have to cope with more load and service tasks than those dictated by its pertinent capacity constraint.

Hereafter, we present a model for the overall communication cost $C(i,n_j)$ introduced in case n_j service node has undertaken the responsibility for the execution of

service task i ($i \in ST(u,s)$). In essence, the model covers the case in which the components of set $A_{n_j}(i)$ need to interact with the components of set $A_{n_k}(i)$ residing in service node n_k in order to provide service task i, ($i \in ST(s)$). It should be noted that service nodes n_j and n_k may reside even in different domains. At this point, a major assumption adopted in our study, is that part of A_{n_j} components are implemented as mobile agents, while the rest are supposed to be fixed service agent components. Let $A_{n_j}^M$ and $A_{n_j}^F$ be the subset of components of A_{n_j} that are implemented as mobile and fixed agents, respectively.

The volume of messages exchanged between each pair of components (e.g., dependent on the number of messages and size of each message) for the accomplishment of task i ($i \in ST(s)$) will be represented as $m_{wv}(i)$, $\forall (w,v) \in C^2$ and $\forall i \in ST(s)$. Let $cc(n_j, n_k)$ be the communication cost per unit message that is exchanged between service nodes n_j and n_k, $\forall (n_j, n_k) \in SN(s)^2$. This factor may be proportional to the distance (e.g., number of hops) between the two service nodes and the load conditions (e.g., bandwidth availability) of the communication link interconnecting the two nodes. Another factor that should be taken into account is the cost associated with the migration of a component (implemented as a mobile agent) from one service node to another. In this respect, let $mc(w, n_j, n_k)$ be the migration cost of component-w from service node n_j to service node n_k, $\forall w \in C$ and $\forall (n_j, n_k) \in SN(s)^2$.

The overall cost for the completion of task i ($i \in ST(s)$) can be calculated by the following formula.

$$C(i, n_j) = \sum_{\forall n_k \in SN(s)} [\sum_{w \in A_{n_j}^F} \sum_{v \in A_{n_k}} m_{wv}(i) \cdot cc(n_j, n_k) + \sum_{w \in A_{n_j}^F} \sum_{v \in A_{n_j}} m_{wv}(i) \cdot cc(n_j, n_j) + \sum_{w \in A_{n_j}^M} mc(w, n_j, n_k) + \sum_{w \in A_{n_j}^M} \sum_{v \in A_{n_k}} m_{wv}(i) \cdot cc(n_k, n_k)], \quad \forall i \in ST(s) \qquad (5)$$

In the previous formulation three main factors are identified. The first one is related to the communication cost deriving from the fixed components and is proportional to the messages (their number and size) that are exchanged between every pair of components (w,v) and the communication cost per unit message between different service nodes.

The second factor is associated with the migration cost of mobile agent components between two different service nodes. This factor is dependent on the path which the mobile agent will follow (i.e., number of hops) and the information encryption and code execution cost, as well as the load conditions of the communication links. The last factor is the communication cost within the same service node, which in practice may be negligible, and in the context of this study is taken equal to zero. It is noted that only the involved to the provisioning process components are taken into account.

Apparently, the designation of the components that will be included in sets $A_{n_j}^M$ and $A_{n_j}^F$ by the service providers may be based on factors such as the overall communication and migration costs as well as estimation of the respective component invocations. In our

study, the service logic deployment pattern (i.e., service components/nodes) adopted by the service providers is known.

Based on the aforementioned analysis, the service node selection algorithm, graphically illustrated in Figure 4, may be described as follows:

Service Node Selection Algorithm

Step 1. The UA component is acquainted with the preferences, requirements and constraints of user u regarding service s. These are expressed by the set of the service tasks $ST(u,s)$ that are of interest (relevant) to the user.

Step 2. At the end of the first general phase (user authentication & authorisation), the DSP determines an appropriate SPA (on the basis of user requirements and preferences with respect to the requested service) and provides the respective SPA with the UA reference.

Step 3. The SPA obtains from the UA user preferences, requirements and constraints, forms the set of the service tasks $ST(u,s)$ that are of interest to the user and retrieves from a database the set of candidate service nodes $SN(i)$ for the completion of each service task i, ($i \in ST(u,s)$), the deployment cost c_D of each service node n_j, ($n_j \in SN(i)$) and their respective capacity constraints $r_a^{\max}$ and $k^{\max}$, and the management cost $c_M(i,i')$ ($(i,i') \in ST^2(s)$). Additionally, the SPA computes for each service task i ($i \in ST(u,s)$) the corresponding resources $r_a(i)$ required for its completion in terms of CPU time, memory and disk resources.

Step 4. The SPA interacts with the SNAs in order to obtain the current load conditions $r_a^{pre}(n_j)$ and number of service tasks $k^{pre}(n_j)$ being executed on each CSN n_j, $n_j \in SN(s)$.

Step 5. The SPA estimates the communication cost $C(i,n_j)$ for each service task i, ($i \in ST(u,s)$) on the basis of equation (5), after contacting the NPAs in order to acquire the current load conditions of the communication links.

Step 6. The SPA solves the appropriate instance of the service task assignment problem (equations (1)-(4)).

Step 7. End.

5 Experimental Results

In this section, indicative results are provided in order to assess the proposed framework, which allows for effective service provisioning. In order to test the performance of the service task assignment scheme, we conducted experiments on a network test bed, assuming a simple application executing on a single PC performing a configurable number of queries on a database (that is, the service considered is composed of one service task that involves execution of one service component which interacts with the database).

Concerning the implementation issues of our experiments, the overall Service Provisioning System (SPS) has been implemented in Java. The Voyager mobile agent platform (The Voyager Platform) has been used for the realisation of the software components as well as for the inter-component communication. To be more specific, the system components (UA, SPA and the monitoring module SNAs, NPAs) have been

implemented as fixed agents and the service task constituting the service as intelligent mobile agent, which can migrate and execute to remote service nodes.

Two sets of experiments have been performed. In the first experiment, a copy of the database exists on each service node, thus, communication cost in practice is negligible and is taken equal to zero. In this case, only the service node deployment cost factor is considered and the performance of the system is tested using as decision parameter the load conditions of the service nodes. In the second experiment the database resides only on one of the service nodes. Thus, the communication cost is also taken into account in the service task assignment process.

The network topology that has been adopted for both experiments consists of five service nodes with the following configuration: two service nodes with 2GHz CPU and 2 GB RAM and three service nodes with 1GHz CPU and 1 GB RAM. All service nodes are running the Linux Redhat OS.

The idle states of the CPUs of the service nodes are simulated to follow the Normal, Uniform and Exponential distributions, respectively, with mean value 50,000 ms. and maximum value 100,000 ms. In all cases, the duration in which the CPU load of the service nodes is above 50% is 20,000 ms.

The graphical user interface of the SPA module, which implements the service task assignment process, is given in Figure 5.

Concerning the first experiment, all service nodes reside on a 100Mbit/sec Ethernet LAN. We have performed 100 experiments for each kind of CPU simulation with the mobile agent realising the service logic performing tasks varying from 100 to 1000 queries (with interval 100 queries). The same experiments have also been conducted without using our service task allocation scheme. In the latter case, service tasks are assigned randomly to service nodes.

The mean execution time for each CPU load distribution when the service task assignment process is applied and when the service node is selected randomly is illustrated in Figure 6. From the obtained results, we observe a decrease of the service completion time when the service task assignment system is used. At this point, it should be mentioned that this performance improvement is tightly related to the number of queries the service task needs to perform at the remote service node and the time that the service node's CPU is idle. It may be observed that for small and large tasks (from 100 to 300 and from 700 to 1000 queries) the improvement in performance is bigger than in medium sized tasks (from 400 to 600 queries). It may also be derived that we have about 6% improvement for small tasks and about 9% for the large ones, while for medium sized tasks the improvement in performance is minor. This could be explained as follows. From Figure 6, it could be extracted that the mean time required for initialisation of the mobile agent on a service node is 35,000 ms. Also the execution of a task consisting of 100 queries when CPU is idle is 5,500 ms. Thus, small tasks can be performed during one slope of a CPU load (i.e., time during which CPU load is below 50%), while large tasks require for their completion one CPU slope, one CPU peak (i.e., time during which CPU load is above 50%) and finally another CPU slope. The completion of medium tasks usually requires one CPU slope and one CPU peak. Thus, the application of service task assignment process results in minor performance improvement.

Concerning the second experiment, as depicted in Figure 7, three service nodes reside on one LAN, while the rest are located on a separate LAN. The two LANs are interconnected via a VPN connection, which utilizes a slow Internet connection (128Kb/sec). The SPA is located on the service node 'Center', while the database resides

on node C. In this experiment, the service task comprises execution of 100 queries to the database. The service task completion time has been measured on each node and the following results have been obtained (all in ms):

A→7200, B→7250, D→760, C→740

The same experiment has been performed 100 times applying the service task assignment scheme. The results obtained regarding node specialization are as follows: node C has been selected as the best service node 80% of the times, while node D has been selected 20% of the times. The average service completion time is approximately 750ms. The application of our proposed service task assignment scheme results in a decrease of the service completion time with respect to random service node selection which on average is 80%. It should be noted that the aforementioned percentage is tightly related to the data rate supported by the interconnection line.

6 Related Research

Most studies in the field of resource allocation schemes aim at efficiently utillising the resources spread throughout a network. In most cases the problem is reduced to load balancing among specific nodes. The design choices that the system architect has to face are quite vast ranging from deploying centralised vs. decentralised arrival and/or control systems, adopting static (model based) vs. dynamic (state based) schemes, considering different resource allocation strategies/algorithms incorporating or not the task migration concept, taking into account diverse load metrics, etc. The centralised resource allocation, referring to the arrival configuration of the service requests and the overall control of the service assignment scheme, provides sophisticated global control, throughput optimisation and relieves the network from the burden of continuous load information exchange between the system nodes in order to monitor and update their knowledge about the current system status. However, it increases the cost endured by the service provider due to the dedication of at least one node to the task assignment process, is quite impractical in case large scale networks are considered due to the computational complexity and storage burden imposed, especially when dynamic schemes are considered, while it is referred to as introducing a single point of failure or bottleneck in the system performance. On the other hand, the decentralised approach ensures scalability, overloading the network with load information due to the exchange among the nodes about the system status (Suguri, 2000).

Static schemes (Stone, 1997) use only information about the average system behavior, ignoring current system status, thus, in general they do not respond well to short term load imbalances among the service nodes. On the other hand, dynamic schemes are more complex and suffer from communication and computation overhead introduced due to current information acquisition and decision making. Learning from experience techniques could be exploited in order to update decision parameter values according to long term observations. For example, execution times or resource utilisation could be logged and reused as pre-estimations for the assignment of similar tasks.

Basic service task assignment strategies comprise the following (Schmidt, 2004): First, *Round Robin*, where the tasks are allocated to the nodes by simply iterating through the nodes list. Second, *Random*, where the nodes to be assigned with the tasks are selected randomly. Third, *Least Loaded* in accordance with which the tasks are assigned to a specific node until a pre-specified threshold is reached. Thereafter, all subsequent requests are transferred to the node with the lowest load and the aforementioned steps are

repeated. Fourth, *Load Minimum*, where the average load of the system is calculated. In case the load of a node is higher than the average node and of the least loaded node by a certain amount, all subsequent requests are transferred to the least loaded location.

According to the task farming paradigm (Andrews, 1991), a pool of tasks and one worker on each node of the system is considered. Each worker repeatedly claims a task from the pool, executes it and claims the next task. This way, the system load is efficiently distributed to the available resources. Considering dynamic, distributed controlled resource allocation, schemes in most cases follow three basic types (Agrawal, 1987): *Sender-Initiated*, where congested nodes (nodes where the load reaches a predefined threshold) take the initiative and probe other nodes in order to determine the most suitable node (e.g., least loaded node) for remote task execution, *Receiver-Initiated*, where lightly-loaded nodes search for work in a similar manner (probe other nodes in order to determine the node(s) that should be relieved from tasks e.g., the most loaded node), *Symmetrically-Initiated*, according to which both congested and lightly loaded nodes take the initiative. In (Lazowska 1986, Krueger 1988) the performance of these schemes is evaluated. The sender-initiated scheme is shown to perform better in light or moderate loaded systems, while the receiver-initiated paradigm is preferable at higher load conditions, under the assumption that the cost of transferring a task between the nodes is comparable for the two schemes. Both sender-initiated and symmetrically-initiated schemes become unstable at high load conditions, especially when the cost of probing other nodes is taken into account.

In general, many approaches have derived and encourage the necessity of adaptive switching between strategies (Svenson, 1992) and dynamic adjustment of decision parameters (e.g., node's load predefined threshold, time interval upon which load information exchange between the nodes should take place) (Xu, 1993). However, depending on the number of nodes in the network, the load balancing technique adopted, the network status, the time required and the complexity indroduced, the resource allocation scheme itself may diminish the net benefit of the overall procedure. In (Eager, 1986), the relative benefits of simple versus complex load sharing policies are examined. Using an analytical model for a homogeneous network, the authors concluded that simple policies that require only a small amount of state information perform as well as complex policies.

Researchers also borrow notions from economic fields (particularly, dynamic pricing and game theory) in order to efficiently allocate network resources through the construction of market-based systems (Chavez, 1997). In (Buyya, 2002), a computational economy framework for resource allocation and for regulating supply and demand in grid computing environments is proposed. Specifically, economic models (commodity market models, posted pricing schemes, tender and auction mechanisms), system architectures and policies for resource management are provided for computational grids and peer-to-peer computing systems.

7 Conclusions

This paper provides a mechanism for assisting service providers in efficiently managing and fulfilling current user requests. Specifically, one possible version of the service task assignment problem has been addressed. Our objective is to find the best service task assignment pattern, i.e., an assignment of service tasks to service nodes that is optimal given the current load conditions and number of service tasks being served by each

service node. Experimental results on a real network test bed indicate that the proposed framework produces good results in relatively simple contexts (e.g., a service, which is composed of one service task that involves execution of one service component). Specifically, when the load conditions of the service nodes is the only factor considered for deciding on the most appropriate service node for the service provisioning, an overall improvement in service completion time of about 7% is introduced (especially, for the small and the large sized service tasks). In case the communication cost factor is considered for determining the service node to be involved in the service provisioning process, our scheme succeeds each time in acquiring a node requiring only local / LAN based component interactions for service completion, minimizing, thus, network resources consumption. What remains is to evaluate the performance of the proposed service task assignment scheme in complex contexts.

Directions for future work include, but are not limited to the following. First, the realisation of further wide scale trials, so as to experiment with the applicability of the framework presented herewith. Second, the experimentation with alternate approaches (e.g., market-based techniques) for solving the service task assignment problem.

8 References

Agrawal R. and Ezzat A. (1987) "Location Independent Remote Execution in NEST", IEEE Transactions on Software Engineering, vol. 13, no. 8, pp. 905-912.

Andrews G.R. (1991) "Paradigms for Process Interaction in Distributed Programs", ACM Computing Surveys, vol. 23, no. 1, pp. 49-90.

Benatallah B., Sheng Q., and Dumas M. (2003) "The Self-Serve Environment for Web Services Composistion", IEEE Internet Computing, vol. 7, no.1, pp.40-48.

Buyya R., Abramson D., Giddy J., and Stockinger H. (2002) "Economic models for resource management and scheduling in Grid computing", Concurrency and Computation: Practice and Experience, vol. 14, pp. 1507-1542.

Chavez A., Moukas A., and Maes P. (1997) "Challenger: A Multi-agent System for Distributed Resource Allocation", Proc. 1st International Conference on Autonomous Agents.

Eager D., Lazowska E., and Zahorjan J. (1986) "Adaptive Load Sharing in Homogenous Distributed Systems", IEEE Transactions on Software Engineering, vol. 12, pp. 662-675.

Jennings N., Sycara K., and Wooldridge M. (1998) "A Roadmap of Agent Research and Development", Autonomous Agents and Multi-Agent Systems, vol. 1, no. 1, pp. 7-38.

Krueger P. and Livny M. (1988) "A Comparison of Preemptive and Non-Preemptive Load Distributing", Proc. 8th International Conference on Distributed Computing Systems, pp. 123-130.

Lazowska E., Eager D. and Zahorjan J. (1986) "A Comparison of Receiver-Initiated Sender-Initiated Dynamic Load Sharing", Performance Evaluation, vol. 6, no. 1, pp. 53-68.

Morreale P. (1998) "Agents on the move", IEEE Spectrum, vol. 35, no. 4, pp. 34-41.

OSGi (1999) Open Service Gateway Initiative, http://www.osgi.org

Papadimitriou C. and Steiglitz K. (1982) Combinatorial optimisation: Algorithms and complexity. Prentice Hall, Inc.

Schmidt J., Dowdy D., Othman L. (2004) "Evaluating the Performance of Middleware Load Balancing Strategies", Proc. 8th International IEEE Enterprise Distributed Object Computing Conference, pp. 135- 146.

Special Issue (2003) "Special section on grid computing", ACM SIGMETRICS Performance Evaluation Review, vol. 30, no. 4, pp. 12-49.

Stone H. (1997) “Multiprocessor Scheduling with the Aid of Network Flow Algorithms”, IEEE Transactions on Software Engineering, vol. 3, no. 1, pp. 85-93.

Suguri T., Yamashita H., Kinoshita S., and Okada Y. (2000) “Load Balancing in Distributed Autonomous Cooperative Systems”, Systems and Computers in Japan, Vol. 31, No. 6, pp. 74-89.

Svenson A., (1992) “Dynamic Alternation between Load Sharing Algorithms”, Proc. 25th Hawaii International Conference vol. 1, pp. 193-201.

Tanenbaum A. S. (2001) Modern Operating Systems, Englewood Cliffs, New Jersey: Prentice-Hall, 2nd ed.

The TeraGrid Project (2003) A distributed computing infrastructure for scientific research. http://www.teragrid.org.

The Parlay Group http://www.parlay.org/

The Voyager Platform, Recursion Software Inc. http://www.recursionsw.com/

Vinoski S. (1997) “CORBA: Integrating diverse applications within distributed heterogeneous environments”, IEEE Communications Magazine, vol. 35, no. 2, pp. 46-55.

Xu J. and Hwang K. (1993) “Heuristic Methods for Dynamic Load Balancing in a Message-Passing Multicomputer”, Journal of Parallel and Distributed Computing, vol. 18, pp. 1-13.

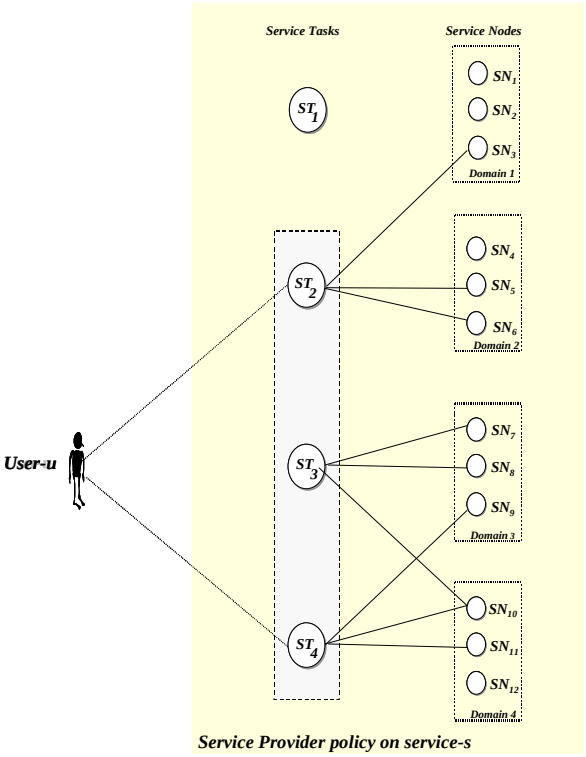


Figure 1 User-u wishes to access service-s, which is composed of 4 different service tasks (of interest to the user are 3 out of the 4 service tasks)

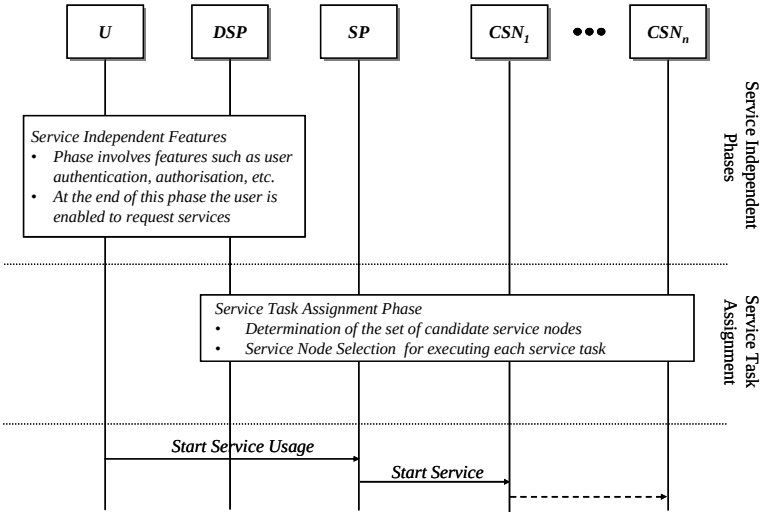


Figure 2 Interactions among the business level entities during the service task assignment case

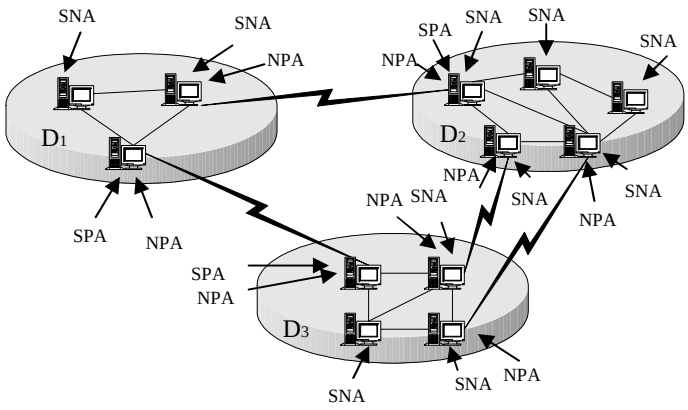


Figure 3 System model and physical distribution of the service task assignment related software components

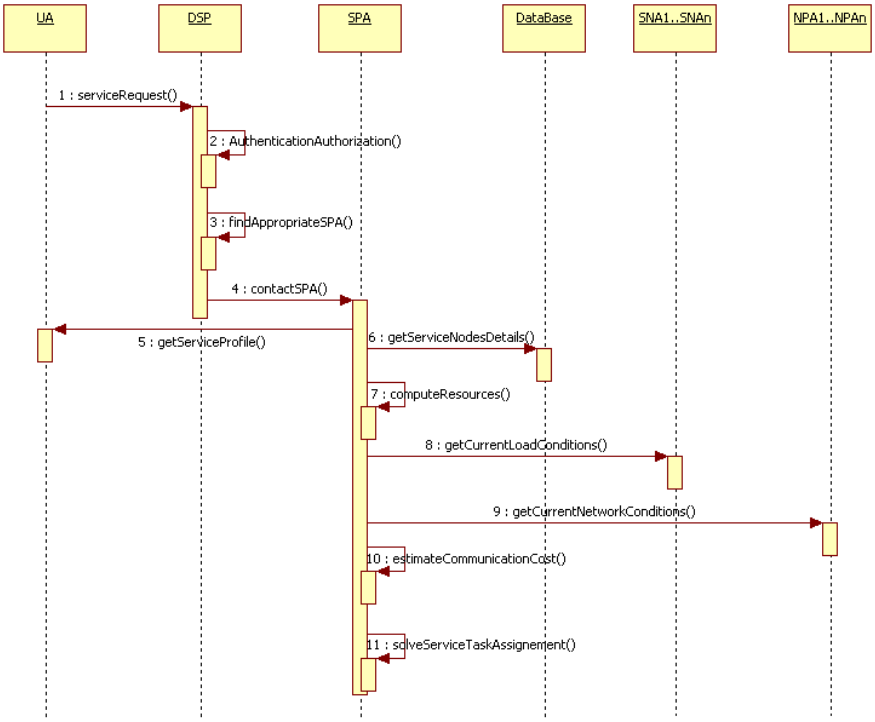


Figure 4 Service Node Selection Algorithm

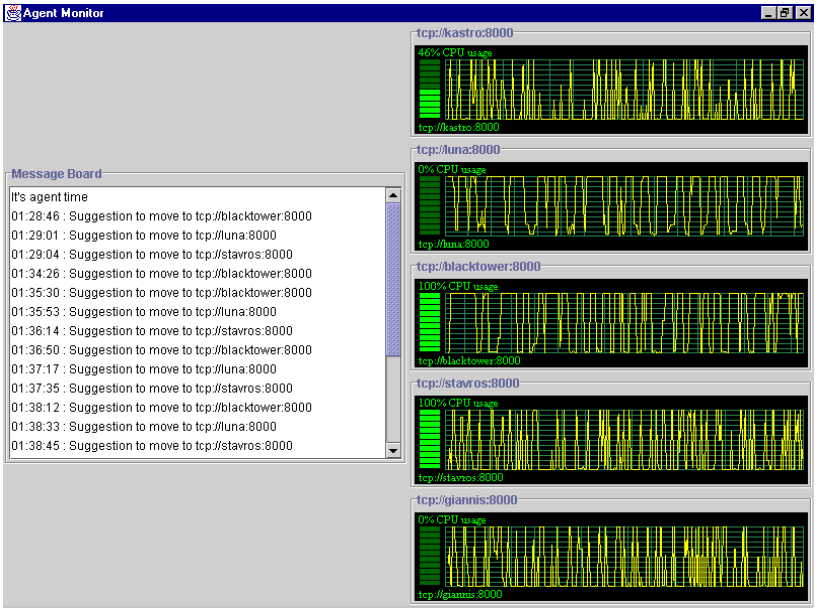
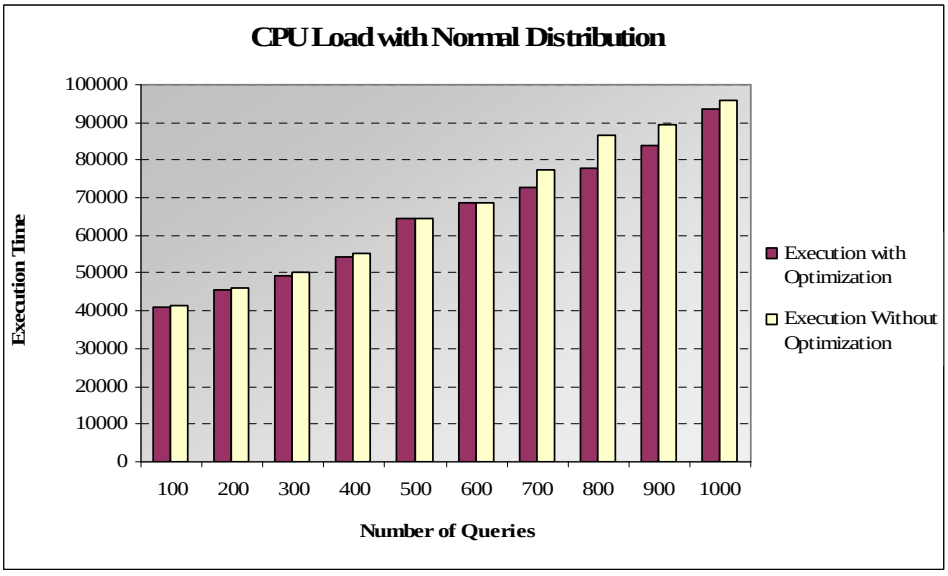
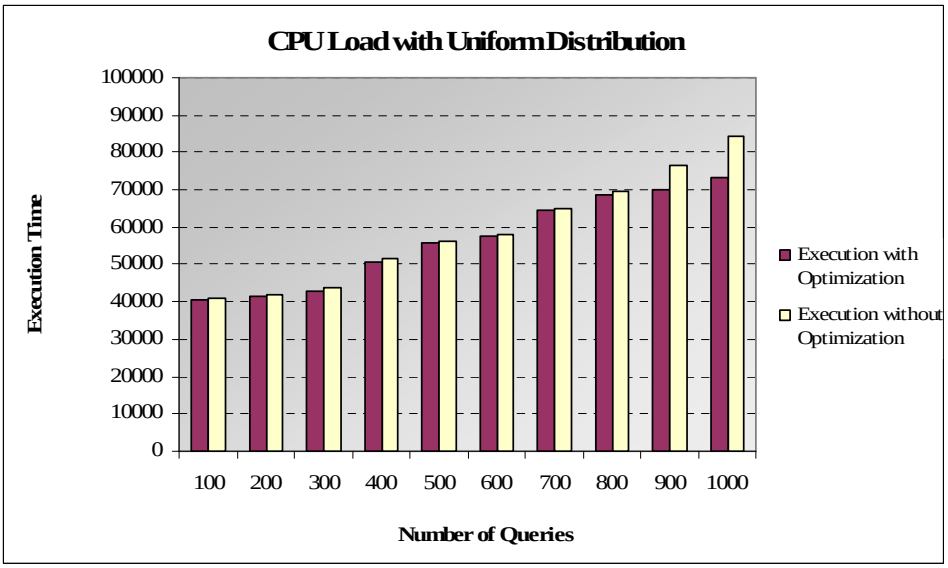


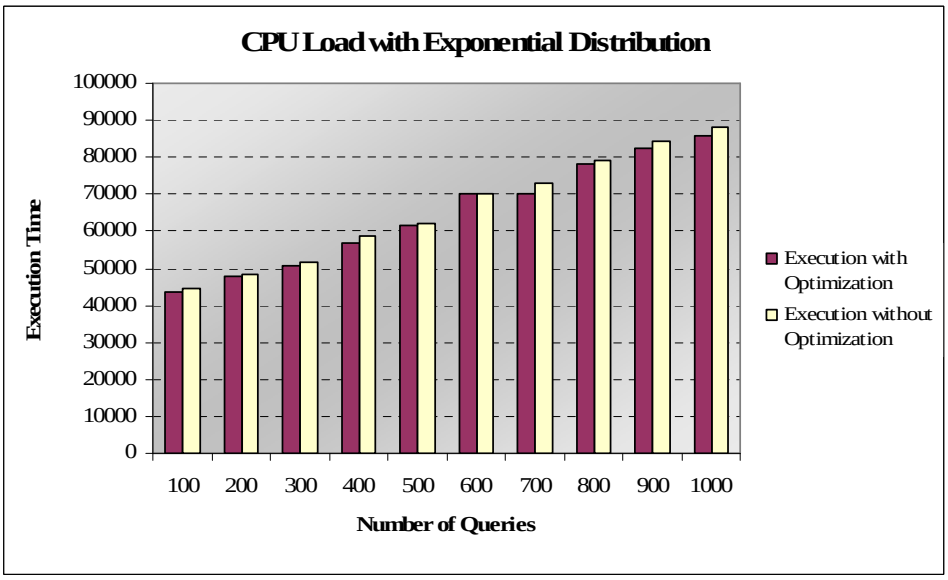
Figure 5 Graphical User interface of the SPA module



(a)



(b)



(c)

Figure 6 Execution times with and without optimisation for (a) Normal, (b) Uniform and (c) Exponential CPU load distributions

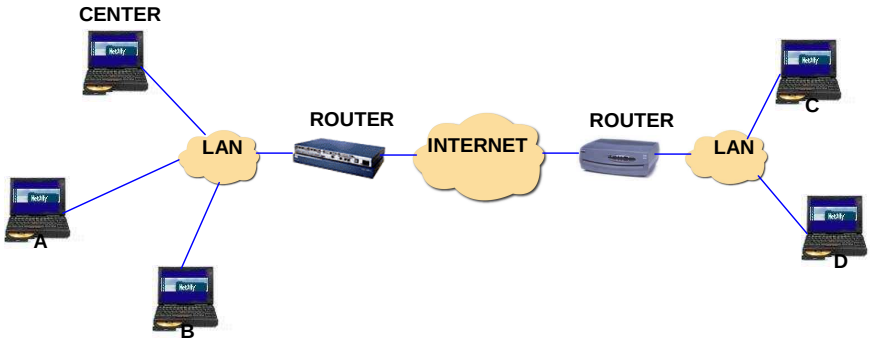


Figure 7 Network Topology adopted for the second experiment