ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF INFORMATICS

ASSIGNMENT OF ASPECTS IN HETEROGENEOUS DISTRIBUTED SYSTEMS

M.Sc. Thesis by
Samet BULU

Department : Computer Science

Programme : Computer Science

JUNE 2011

ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF INFORMATICS

ASSIGNMENT OF ASPECTS IN HETEROGENEOUS DISTRIBUTED SYSTEMS

M.Sc. Thesis by
Samet BULU
(704081013)

Date of submission : 06 May 2011
Date of defence examination : 09 June 2011

Supervisor (Chairman) : Assis. Prof. Dr. Feza BUZLUCA (ITU)
Members of the Examining Committee : Assoc. Prof. Dr. Zehra ÇATALTEPE (ITU)
Prof. Dr. Oya KALIPSIZ (YTU)

JUNE 2011

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

HETEROJEN DAĞITILMIŞ SİSTEMLERDE CEPHE ATAMA

YÜKSEK LİSANS TEZİ

Samet BULU
(704081013)

Tezin Enstitüye Verildiği Tarih : 06 Mayıs 2011
Tezin Savunulduğu Tarih : 09 Haziran 2011

Tez Danışmanı : Yrd. Doç. Dr. Feza BUZLUCA (İTÜ)
Diğer Jüri Üyeleri : Doç. Dr. Zehra ÇATALTEPE (İTÜ)
Prof. Dr. Oya KALIPSIZ (YTÜ)

HAZİRAN 2011

FOREWORD

I would like to express my deep appreciation and thanks for my advisor. This work is supported by ITU Institute of Science and Technology.

June 2011 Samet BULU

TABLE OF CONTENTS

Page

ABBREVIATIONS

OOP	**:** Object Oriented Programming
AOP	**:** Aspect Oriented Programming
GA	**:** Genetic Algorithm
PSO	**:** Particle Swarm Optimization
SA	**:** Simulated Annealing
HS	**:** Harmony Search
AJDT	**:** AspectJ Development Tools
SDK	**:** System Development Kit
CPU	**:** Central Processing Unit
XML	**:** Extensible Markup Language
CI	**:** Confidence Interval
RAA	: Random Assignment Algorithm

LIST OF TABLES

Page

LIST OF FIGURES

Page

ASSIGNMENT OF ASPECTS IN HETEROGENEOUS DISTRIBUTED SYSTEMS

SUMMARY

Nowadays, object-oriented programming has become the most preferred programming paradigm where a problem is decomposed into modular units called objects. Although object-oriented programming offers greater ability for separation of concerns, it has difficulty to implement crosscutting concerns like logging, profiling, caching, authentication, and authorization. Aspect oriented programming is proposed as a solution for separation of crosscutting concerns into single units called aspects. Aspects are then combined with a base program through a process called weaving.

In recent years, with increasing use of distributed systems, distributed AOP has become more popular. In distributed AOP, aspects can be deployed in a set of hosts where each host has unique memory and processing capabilities. Remote pointcuts, which are similar to traditional remote method calls, invoke the execution of method-like constructs called advices in aspects on remote hosts.

The way of distributing aspects over the network is critical and affects the performance of the program, because there is a relation between objects and aspects. When there is a call from an object to an aspect, data is exchanged between these object and aspect. This process consumes time and this time depends on the amount of the data and the capacity of the link, which is used during the data transfer. Therefore, while assigning aspects of an AOP to hosts in a distributed system several properties of the physical system and the program must be taken into consideration like processing capabilities of hosts, parameters of communication links, amount of data shared between objects and aspects.

Although a large number of task assignment approaches have been identified up to now, none of them is interested in assignment of aspect. In this thesis, first, the aspect assignment problem in heterogeneous distributed systems is formulated by considering all necessary parameters. Then we apply three algorithms namely A*, GA and PSO to solve this problem that occurs in distributed AOPs. Also a new algorithm which creates clones (copies) of necessary aspects while assigning them to hosts is proposed in order to improve the performance of the distributed AOP. Finally, we evaluate the efficiency of these algorithms for different systems and programs and compare the increase in the performance of the AOP obtained by these algorithms with an algorithm that assigns aspects to hosts randomly.

Experimental results show that GA and PSO are more favorable than A* algorithm for larger systems with many nodes, while for smaller systems A* may be preferable. On the other hand, using copies of aspects decreases the cost values up to a certain level and makes improvements in the performance. Finally, it is shown that proper assignment of aspects improves performance of the distributed AOPs.

HETEROJEN DAĞITILMIŞ SİSTEMLERDE CEPHE ATAMA

ÖZET

Problemi, nesne adı verilen modüler parçalara ayrıştıran nesneye yönelik programlama günümüzde en sık tercih edilen programlama tekniğidir. Nesneye yönelik programlama her ne kadar ilgilerin ayrıştırılması konusunda büyük imkanlar sağlasa da loglama, performas gözleme, ön bellekleme, kimlik doğrulama ve yetkilendirme gibi dik kesen ilgilerin ele alınmasında zorluklar yaşamaktadır. Cepheye yönelik programlama, dik kesen ilgileri cephe adı verilen parçalara ayrıştıran bir çözüm olarak önerilmiştir. Cephe adlı bu parçalar örme adı verilen bir işlem ile ana programa birleştirilir.

Son yıllarda dağıtılmış sistemlerin kullanımının artmasıyla birlikte dağıltılmış cepheye yönelik programlama da popüler hale gelmiştir. Dağıltılmış cepheye yönelik programlamada cepheler, her biri farklı bellek ve işlem gücüne sahip bir dizi düğüme yüklenir. Geleneksel uzak method çağrımına benzer şekilde uzak kesim noktaları tarafından cephe içinde yer alan method benzeri yapılar uzak düğümler üzerinde çalıştırılır.

Cephelerin sistem üzerinde nasıl dağıtıldığı önemlidir ve programın performansını etkiler. Çünkü nesneler ile cepheler arasında bir ilişki vardır. Nesneden cepheye bir çağrı olduğunda arada veri transferi gerçekleşir. Bu işlem bir süre gerektirir ve bu süre transfer edilen verinin miktarına ve transfer esnasında kullanılan iletişim yolunun kapasitesine bağlıdır. Dolayısıyla, cepheler dağıtılmış sistem üzerinde düğümlere atanırken düğümlerin işlem kapasiteleri, iletişim yolu parametreleri, transfer edilen veri miktarı gibi sistemin ve programın özellikleri dikkate alınmalıdır.

Her ne kadar günümüze kadar çok sayıda iş atama yöntemi tanımlanmış olsa bunların hiç biri cephelerin atanması ile ilgilenmemiştir. Tez kapsamında ilk olarak, heterojen dağıtılmış sistemlerde cephe atama problemi gerekli tüm özellikler dikkate alınarak tanımlanmıştır. Sonrasında dağıtılmış cepheye yönelik programlamada yer alan bu problemi çözmek üzere A*, GA ve PSO algoritmaları uygulanmıştır. Ayrıca dağıtılmış cepheye yönelik program performansını arttırmak üzere cepheleri düğümlere atama işlemi esnasında gerekli cephelerin kopyalarını oluşturan yeni bir algoritma önerilmiştir. Son olarak algoritmaların farklı sistemler ve programlar üzerinde etkinlikleri değerlendirilerek, rastgele atama yapan bir algoritmaya göre sağlamış oldukları performans artışı karşılaştırılmıştır.

Yapılan deneyler çok düğüme sahip büyük sistemlerde GA ve PSO algoritmalarının, daha küçük sistemlerde ise A* algoritmasının tercih edilebileceğini göstermiştir. Diğer taraftan cephelerin kopylarının kullanılması belirli bir seviyeye kadar maliyet değerlerini düşürmüş ve performansta artış sağlamıştır. Son olarak cephelerin uygun şekilde atanması dağıtılmış cepheye yönelik program performansını arttırdığı gözlemlenmiştir.

1. INTRODUCTION

Today, object oriented programming (OOP) is the most popular and preferred programming method in the software world. OOP groups operations and data into modular units called objects, and lets programmers combine objects into structured networks to form a complete program. However, some programming concerns (crosscutting concerns) cannot be neatly encapsulated in objects, but must be dispersed throughout the code like logging, tracing, profiling, policy enforcement, pooling, caching, authentication, authorization and transactional management. At this point, aspect oriented programming (AOP) [1] has been proposed as a technique to break all that crosscutting concerns out from the objects and apply it to the objects in some other way called aspect. AOP builds on previous technologies such as OOP that have caused improvements in the modularization of software.

Distributed systems [2], in which the processing elements are connected by a network, have become increasingly popular in recent years because of their high speed and high reliability. Distributed systems generally consist of dissimilar hosts where each host has unique memory and processing capabilities. Distributed systems allow programmers to divide applications into a number of tasks and execute concurrently on different hosts. This process obtains tremendous improvement in the performance when the task distribution and assignment are applied effectively.

In recent years, with increasing use of distributed systems, distributed AOP arouses more interest. In distributed AOP, aspects can be deployed in a set of hosts. The way of distributing aspects over the network can affect the performance of the program. While assigning aspects of an AOP to hosts in a distributed system several properties of the physical system and the program must be taken into consideration. These properties are processing capabilities of hosts, parameters of communication links, amount of data shared between objects and aspects. The assignment of aspects is critical and affects the program completion time because when there is a call from an object to an aspect (assuming that the object and the aspect are assigned to different hosts), exchanging data between these object and aspect consumes time. This time

depends on the amount of the data and the capacity of the link which is used during the data transfer.

Although there are a large number of task assignment algorithms, none of them is interested in assignment of aspect. To assign aspect to processing nodes properly we take the following two structures into consideration. Firstly, the parameters of the distributed system such as processing capabilities of nodes and bandwidths of communication lines between them. Secondly, structure of the aspect oriented program expressed by the relations between aspects and objects such as reference counts, amount of transferred data between them.

In this thesis, we first formulate the aspect assignment problem in heterogeneous distributed systems by taking all necessary parameters into account and then apply three algorithms to solve this problem that occurs in distributed AOPs. The first algorithm is an A* [3] based search technique, the second one is based on Genetic Algorithms (GA) [4], and finally the third one is Particle Swarm Optimization (PSO) [5]. We evaluate the efficiency of these algorithms for different systems and programs. It is shown that the GA and PSO are more favorable than A* algorithm for larger systems with many nodes, while for smaller systems A* may be preferable. GA obtains slightly better results compared to PSO, but PSO is faster than GA. Then we propose a new algorithm that creates clones (copies) of necessary aspects while assigning them to hosts in order to improve the performance of the distributed AOP. Creating clones of the aspects makes it possible to find a solution in a partially connected system and decreases communication costs. We also compare the increase in the performance of the AOP obtained by these algorithms with an algorithm that assigns aspects to host randomly. Simulation results indicate that assigning aspects to hosts properly using the proposed algorithms can reduce the completion time of a distributed aspect oriented program almost by half compared to random assignment.

2. BACKGROUND AND RELEATED WORK

2.1 Aspect Oriented Programming (AOP)

Aspect-oriented programming (AOP) [1] is a programming style that allows programmers to implement cross-cutting concerns like logging, tracing, profiling, policy enforcement, pooling, caching, authentication, authorization and transactional management in a modular way and then combine these concerns with a base program through a process called weaving. AOP aims at improving the quality of the software by decreasing the level of code scattering and code tangling known as primary symptoms of non-modularization. Code scattering occurs when a single issue is implemented in multiple modules. Code tangling occurs when a module is implemented to handle multiple concerns simultaneously.

There are lots of AOP implementations that have been widely used. Some of these implementations are AspectJ [6], AspectWerkz [7], JBoss-AOP [8], and Spring [9]. Table 2.1 lists AOP Frameworks and highlights their features. AspectJ, which was proposed as an extension of the Java language for AOP, is the most prominent implementation. It extends Java with support for two kinds of crosscutting implementation. First, it allows programmers to define additional implementation to run at certain points in the execution of the program which is called dynamic crosscutting mechanism. Second, it allows programmers to define new operations on existing types which is called static crosscutting mechanism.

Table 2.1: AOP frameworks [10]

Feature / issue	AspectJ	AspectWerkz	JBoss AOP	Spring
Weaving time	Compile/Load	Compile/Load	Compile/Load/Run	Run
Transparency	Transparent	Transparent	Choice	Factory
Per-instance	No	No	Yes	Yes
Aspect constructor, field, throw, and cflow interception	All	All	Some	Some
Annotations	No	Yes	Yes	Yes
Standalone	Yes	Yes	Yes	No
AOP alliance	No	No	No	Yes
Affiliation	IBM	BEA	JBoss	Spring

In an AOP crosscutting concerns are defined as a set of aspects. An aspect consists of method-like constructs called advice. An advice is used to define additional behavior at a set of well-defined points called join points in the program's execution. Join points are matched by a predicate called pointcut. AOP weaver maps various crosscutting elements to the object oriented constructs. For example, aspects map to classes where each data member and method in aspect become the members of the class. Pointcuts are intermediate elements that map to methods. Advice usually maps to one or more methods. The weaver inserts calls to these methods at potential locations matching the associated pointcut. During the execution of an AOP objects call methods of related aspects and mostly objects and related aspects operate on common data. This process can be seen in Figure 2.1.

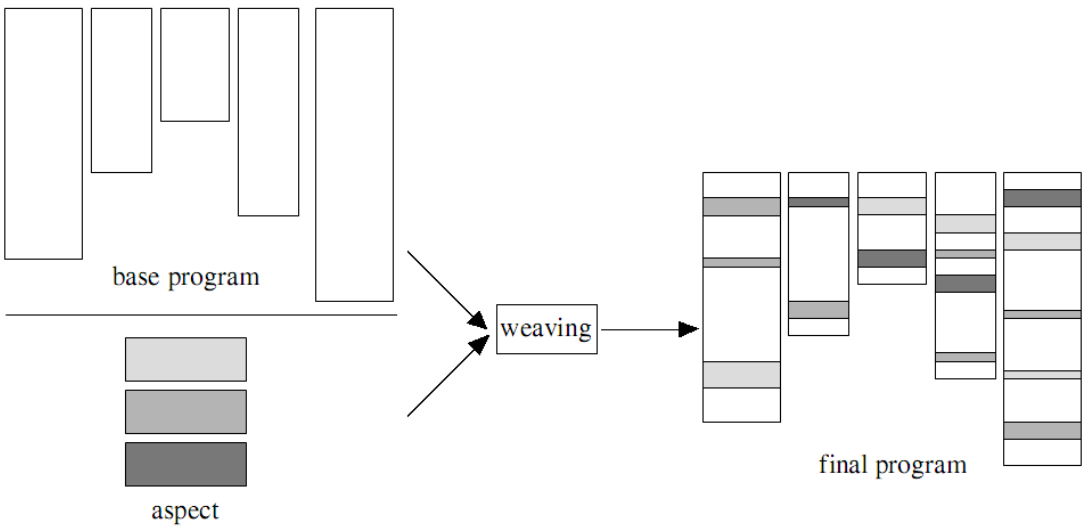


Figure 2.1 : AOP weaving mechanism.

2.2 Distributed Systems

A distributed system is a collection of independent computers (hosts) that appears to its users as a single coherent system [2]. The hosts interact with each other over a network in order to achieve a common goal, such as solving a large computational problem [11]. Large problems can be divided into a number of tasks and each task can be executed concurrently on different hosts. Since each host has its own resources, the hosts can run concurrently in parallel. Information is exchanged by passing messages between the hosts. Most of the distributed systems are defined in the form of heterogeneous in which the connected hosts have different processing capabilities. These types of systems are called heterogeneous distributed systems.

The advantages of distributed systems can be listed as follows:

- Provide high speed computing capabilities
- Achieve higher availability and improved reliability
- Offer modular expandability
- Hide the network structure and provide transparency
- Make it easy for the users to access remote resources

A distributed system can be modelled using a host connectivity graph $G_T = (V_T, E_T)$ where V_T corresponds to hosts in the network and E_T corresponds to links between the processors labelled by the communication costs. A sample host connectivity graph of a system is shown in Figure 2.2.

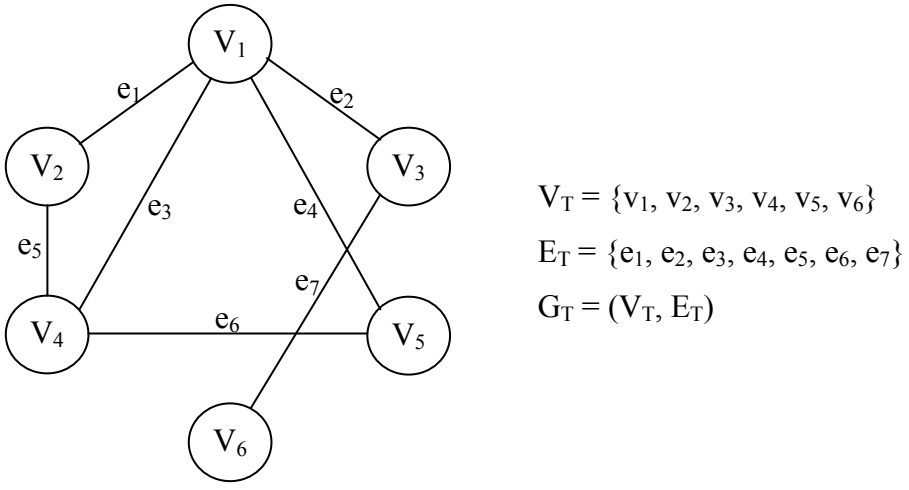
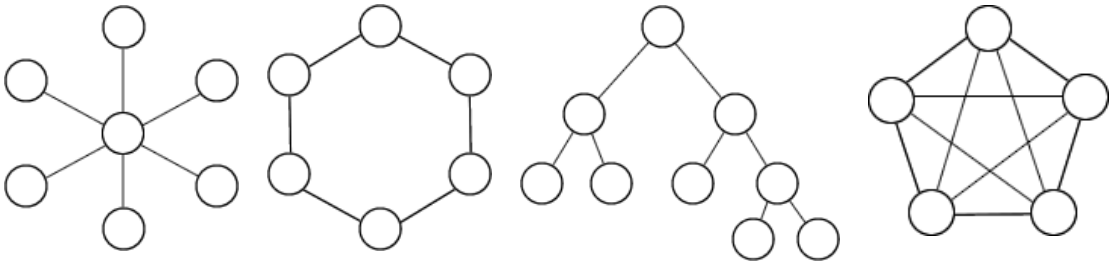


Figure 2.2 : Host connectivity graph.

The topology –in other words, how the hosts in the system are connected– of a distributed system is critical in order to guarantee the system performs well under favorable conditions. Topologies can be grouped into two categories:

- Partially connected
- Fully connected

In partially connected topologies, communication links exist only between some pairs of hosts, but not all. Star-structured, ring-structure and tree-structured topologies are some of the examples of partially connected topologies. On the other hand, in fully connected topologies, there are communication links between all pairs of hosts. These topologies are more complex than others are but they are more powerful**.**



(a) Star-structured (b) Ring-structured (c) Tree-structured (d) Fully-connected

Figure 2.3 : Distributed system topologies.

2.3 Task Assignment

To exploit effectiveness on a distributed system, tasks must be properly allocated to the hosts. This problem is called task assignment problem where it is well-known to be NP-hard [12]. Assuming that there are n hosts and k tasks, the total number of possible assignment cases is n^k. So, the optimal assignment is a problem of exponential complexity.

Task assignment can be performed statically or dynamically. Static task assignments are performed before running the application and remain unchanged until the end of the execution. In contrast, dynamic task assignments are performed at run time. Static task assignments are more favourable than dynamic task assignments when all information needed for the assignment is known before the application execution.

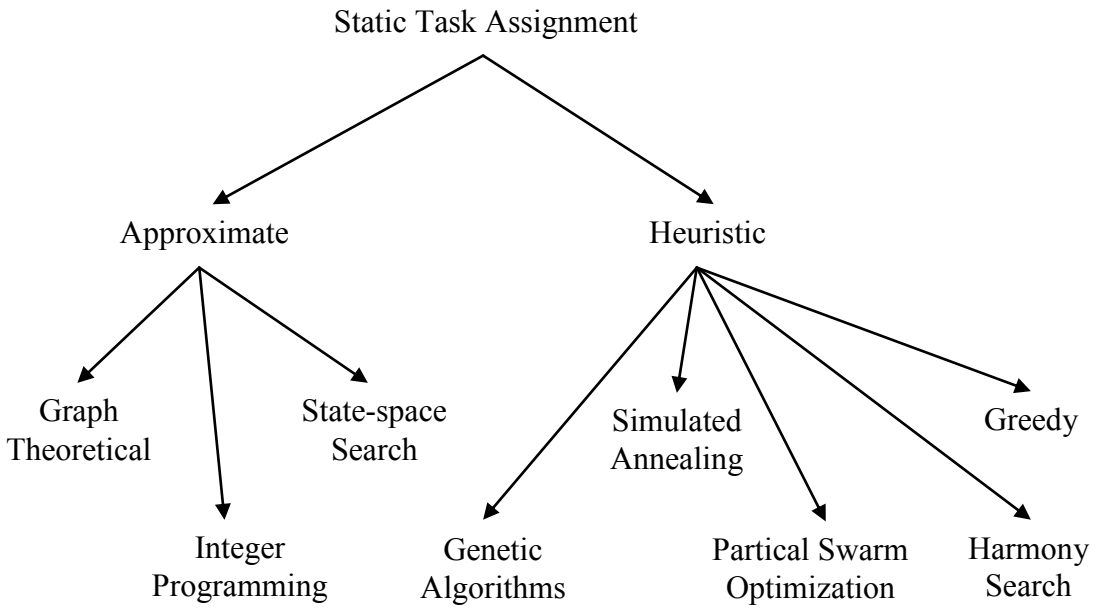


Figure 2.4 : Classification of task assignment approach.

Many approaches to static task assignment problem have been identified up to now. Detailed classification of these approaches is showed in Figure 2.4. They can be classified into main two categories

- Approximate connected
- Heuristic connected

Approximate methods attempt to obtain the optimal solutions by searching the complete solution space. Approximate methods are developed using different strategies.

In graph theoretical approach, each task or/and host is represented by a node and the cost induced by the communication delay between them is represented by a weighted edge. The first attempt in graph based task assignment is done by Stone [18]. In Stone's work, a Max Flow/Min Cut Algorithm is utilized to find assignments, which minimize total execution and communication costs. Stone uses Ford-Fulkerson algorithm for finding the maximum-flow in order to find an optimal partition of a program on a two-processor system. Then, he generalizes it to systems with three or more processors. In his work, Stone constructs a graph for the n-processor problem for which a minimal cost cut is a minimal cost partition of the graph into n disjoint sub graphs.

Stone's work is used as a starting point for new algorithms. V. M. Lo [19] develops a heuristic algorithm, which combines recursive invocation of Max Flow/Min Cut Algorithms with a greedy-type algorithm to find suboptimal assignments of tasks to processors.

The integer programming method formulates the model as an optimization problem and solves it via mathematical programming techniques. Chu [20] developed a model to find an optimal file allocation in a multiple computer system where the criterion of optimality is minimal overall storage and transmission costs. In his work, Chu assumed that the number of file copies to be stored in the fully connected network is a fixed and known quantity. He formulates the problem into a nonlinear integer zero-one programming problem and then reduced it to a linear zero-one programming problem.

State-space search techniques represent the problem in terms of states and systematically and intelligently enumerate on these states to find the solution. Shen and Tsai [21] proposed a graph-matching algorithm based on a minimax criterion for solving the static task assignment problem where the processors need not be fully connected. The algorithm combines graph homomorphism to restrict mapping of modules to processors, and an informed search technique, A* to produce an optimum assignment. Ramakrishnan and his colleagues [22] proposed an extension to Shen and Tsai's task assignment strategy by introducing several heuristics to choose the task to be assigned at each level.

Since approximate methods search the whole space to find the optimal solutions, they need a lot of time and memory. Heuristic methods on the other hand, do not pursue the optimal solutions but provide sub-optimal fast and effective solutions. They use special parameters that affect the systems in indirect ways.

Genetic algorithms (GA) generate solutions using techniques inspired by natural evolution. Simulated annealing (SA) is based on the manner in which liquids freeze or metals recrystallize in the process of annealing. Harmony search (HS) is derived from the improvisation of musicians that process of searching for better harmony. Particle swarm optimization (PSO) follows a collaborative population-based search model where each individual of the population, called a ‘particle’, flies around in a multidimensional search space looking for the optimal solution. Finally, greedy

algorithms follow iteration-based approach and seek to reach better solutions from one generation to the next.

There are wide varieties of heuristic algorithms used in task assignment problem such as genetic algorithms [23-24], simulated annealing [25-26], particle swarm optimization [27-28], harmony search [29], and greedy [30].

2.4 Distributed AOP

The relation between AOP and distributed computing is interesting. In distributed systems, decentralized crosscutting concerns can be found where distributed aspects are usually executed simultaneously in multiple hosts of the network. The first approximation in distributed aspects is using AspectJ for improving the modularity of RMI-based programs [13], splitting code and remote object logic into aspects. However, combination of AspectJ and an existing framework for distributed software is not a solution. Because this approximation does not provide general support for explicit distribution in the aspect language or weaver technology, but can only modify the distribution behaviour of a base program.

The next approximation in distributed aspects is the remote pointcut concept [14]. A remote pointcut is a function for identifying join points in the execution of a program running on a remote host. Remote pointcuts are similar to remote method calls, which invoke the execution of a method on a remote host. When the thread of control reaches the join points identified by a remote pointcut, the advice body associated with that remote pointcut is executed on a remote host different from the one where those join points occur.

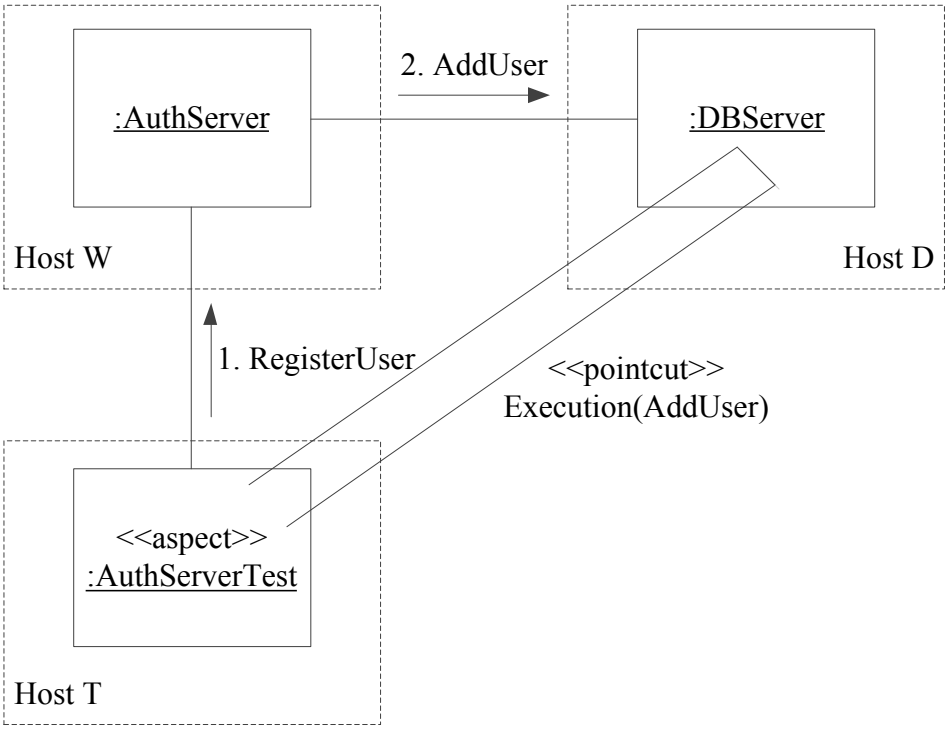


Figure 2.5 : Remote pointcut in distributed system [14].

In Figure 2.5, a simple distributed authentication service is illustrated. The service consists of two components: a front-end server *AuthServer* on a host *W* and a database server *DbServer* on another host D. When a client on host T needs to register a new user it remotely calls *RegisterUser* on the front-end server. Then the *RegisterUser* method remotely calls *AddUser* on the database server to complete the task. Let there is an aspect located on the client used to confirm that adding user to the database is correctly done. By the concept of remote pointcut, related advice(s) of aspect located on host *T* is executed when the thread of control reaches the *AddUser* method on the host *D*.

There are lots of approaches that address distributed AOP. DjCutter [14], JAC [15], AWED [16], Damon [17] and ReflexD [18] are some of these approaches. They are built on top of the previous AOP frameworks and introduce new pointcut predicates that can match events on remote hosts.

DjCutter is the first study which points out the remote pointcut concept. It proposes a centralized aspect-server where the server gathers joinpoint information of remote pointcut definitions and executes the related advices local to the server. It also provides another language construct named remote inter-type declaration, which allows developers to declare a new method and field in a class on a remote host. JAC does not introduce a dedicated aspect language, but use OOP constructs to describe aspects. JAC provides support to specify a named host that delimits the context in which the joinpoint should be detected. AWED and ReflexD make it possible to execute advices in several hosts and programmers can control where aspects are deployed. ReflexD also allows programmers to customize the remote parameter passed to a remote advice which provides greater flexibility. Damon, on the other hand, introduces distributed component model and aspect remoting service with one-to-one and one-to-many abstract.

3. PROBLEM DEFINITION

The assignment problem for aspects in distributed systems can be defined as the assignment of k aspects $A = \{a_1, a_2, \ldots, a_k\}$ to n hosts, $H = \{h_1, h_2, \ldots, h_n\}$. We define our distributed system in the heterogeneous form in which the connected hosts have different processing capabilities. In the network X_{qi} denotes the execution cost of aspect that is proportional to the execution time of the aspect a_i when it is assigned to and executed on host h_q, $1 \leq i \leq k$, $1 \leq q \leq n$. Here we assume that each advice in the same aspect has the same load. This means that the execution time doesn't depend on which advice of an aspect is executed.

Each communication link in the network has different amounts of delay, which can be represented by a delay matrix $D=\{D_{pq}\}$. D_{pq} denotes the communication cost between two hosts h_p and h_q, which arise because of the communication delay when an object located on h_p calls an aspect located on h_q. Further, $D_{pq}=D_{qp}$ and $D_{pp}=0$.

Another parameter that effects the performance of the AOP is the relation count defined as how many times each aspect instance will be called from each object. These values can be obtained from the AOP framework tools. For example, AspectJ Development Tools (AJDT) allows programmers to register a listener to obtain crosscutting relationship information whenever a project is built [35]. Let there be m objects, $O = \{o_1, o_2, \ldots, o_m\}$, then R_{ij} denotes aspect-object relation count between aspect a_i and object o_j. On the other hand, when there is a call from an object to an aspect, a communication cost is incurred because of exchanging data. So, let C_{ij} denotes the communication cost between aspect a_i and object o_j that is proportional to size of data transferred between a_i and o_j. All cost values (X_{qi}, D_{pq}, C_{ij}) are normalized by assigning one to the smallest positive value in each group.

In our study we assume that locations of objects are fixed and predetermined according to their specific jobs. We focus on distributing and assigning aspects, which are used by the objects. Therefore we don't consider the execution times of objects. So, let L_j denotes the host that object o_j is assigned to, $1 \leq L_j \leq n$

All parameters described in this section can be derived explicitly from the distributed system. As an example, the parameters of a simple system which is made up of three hosts, five objects and four aspects are represented in tabular form in Figure 3.1

D_{pq}	h_1	h_2	h_3
h_1	0	1	3
h_2	1	0	2
h_3	3	2	0

(a) Host Communication Costs

X_{qi}	a_1	a_2	a_3	a_4
h_1	1	4	2	3
h_2	3	1	6	2
h_3	5	6	2	3

(b) Aspect Execution Costs

C_{ij}	o_1	o_2	o_3	o_4	o_5
a_1	4	1	2	4	4
a_2	4	1	5	3	2
a_3	2	2	6	3	5
a_4	5	2	1	5	4

(c) Aspect-Class Communication Costs

R_{ij}	o_1	o_2	o_3	o_4	o_5
a_1	16	0	11	14	9
a_2	17	3	9	10	5
a_3	2	9	0	9	16
a_4	14	1	14	4	9

(d) Aspect-Class Relation Counts

$o\ (j)$	1	2	3	4	5
$h(L_j)$	1	2	2	3	1

(e) Object Assignments (L_j)

Figure 3.1 : An Example set of system and program parameters

Host connectivity graph and aspect-object relation graph of the sample system are shown in Figure 3.2. Host connectivity graph has three circle nodes corresponding to hosts and edges between them indicating the communication cost as the edges label. Each host in the graph contains a list where the elements of the list represent the execution costs of aspects on that host. On the other hand, aspect-object relation graph has five square nodes corresponding to objects, four triangle nodes corresponding to aspects and edges between objects and aspects indicating the communication cost and relation counts respectively as the edges label. Object assignments are shown on top of each object node.

The solution of the aspect assignment problem is a proper mapping of k aspects to n hosts that will minimize the running time of the aspect-oriented program. To evaluate the efficiency of the assignment procedure we consider the host that is maximally loaded by the aspects.

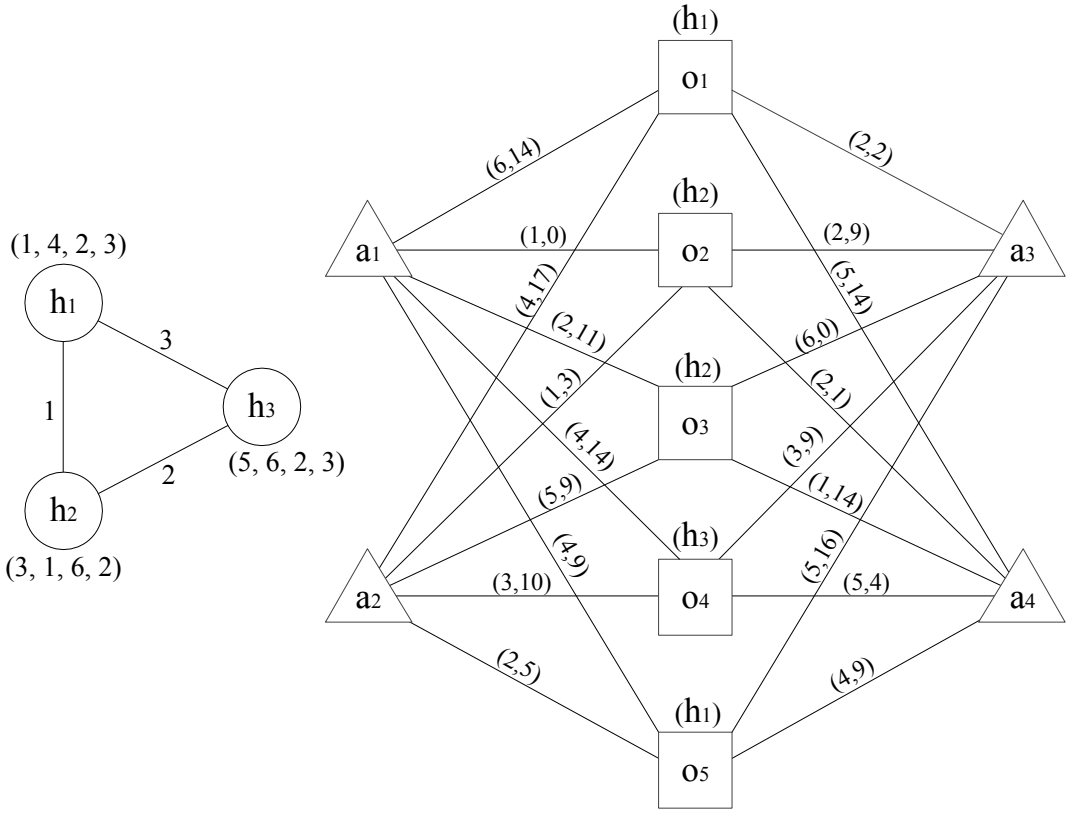


Figure 3.2 : Host Connectivity Graph and Aspect-Object Relation Graph

Here the load of a host is defined as a metric that is proportional to the total time consumed by the aspects located on this host during the execution of the program. As the hosts in a distributed system run parallel, the host that needs the longest time to complete its aspects is taken into consideration, because it will determine the completion time of the whole AOP. The load metric of a host consists of two components. First, one is the total running time of the aspects on this host and second one is data transfer time between these aspects and related objects. Let T be the set of aspects that are assigned to host q then the load on host q is:

$$Load_q = \sum_{\forall i \in T} \left(\sum_{j=1}^{m} R_{ij} X_{qi} + \sum_{j=1}^{m} R_{ij} C_{ij} D_{qL_j} \right) \qquad \textbf{(3.1)}$$

where m is the number of the objects and L_j is the host number of j^{th} object.

The solution has to fulfill two objectives. First, we try to minimize the load of the maximally loaded host, which is represented by the following cost function *F1*:

$$F1 = \max(Load_q), \qquad 1 \le q \le n \qquad \textbf{(3.2)}$$

This first objective is related to the completion time of the aspect-oriented program under assumption that all hosts operate parallel. Secondly, if there are many aspect assignment possibilities, which minimize the *F1*, the second objective is to minimize the sum of load on all nodes, which is expressed by the following function *F2*:

$$F2 = \sum_{q=1}^{n} Load_q \qquad \textbf{(3.3)}$$

4. THE PROPOSED ALGORITHMS

To solve the aspect assignment problem in distributed systems we propose three algorithms, namely an A* algorithm, a Genetic Algorithm (GA) and a Particle Swarm Optimization (PSO). Each of these algorithms has drawbacks and advantages. A*, which is widely used in artificial intelligent, is an effective search algorithm that allows to solve a large number of problems with greater ease. However, its memory requirement is the main drawback of A* algorithm. On the other hand, GA is a randomized searching technique where it is used for optimization and classification problems. PSO is similar to the GA in the sense that these two algorithms use some heuristics to solve the problems. GA has strong ability of global searching but it requires more computational effort than PSO. They are both fast and effective, but they usually find sub-optimal solutions.

4.1 A* Algorithm

A* [3] is a best-first search algorithm, which can guaranteed to find the optimal solutions by using admissible heuristics. In a tree representation it starts from the root node, expands the intermediate nodes and finally reaches one of the leaf nodes. At each node, one of the aspects is assigned to a specific host as an addition to assignments made at its ancestors. Root node is a null solution of the problem. Intermediate nodes represent the partial solutions and leaf nodes represent the complete solutions.

Each node p in the tree maintains a cost function $f(p)$ which is computed as $f(p)=g(p)+h(p)$, where $g(p)$ is the cost of getting from the root to node p and $h(p)$ is the estimated cost of getting from p to the goal node. In our algorithm $g(p)$ is calculated using Equations 3.1 and 3.2 as the load on the heaviest-loaded host ($F1$) of partial assignment. Since, at intermediate nodes all aspects have not been assigned yet, $g(p)$ is not sufficient solely to express the greatest load $F1$. Future assignments to the same host may increase this load. To be able to compare cost values of nodes in different levels fairly, possible effect of unassigned aspects on the load is added as

h(p) to *g(p)*. In our algorithm *h(p)* is calculated as the sum of the object relation counts of aspects that are unassigned at node *p*. Let *U* be the set of unassigned aspects in node *p*, then *h(p)* is calculated as follows:

$$h(p) = \sum_{i \in U} \left(\sum_{j=1}^{m} R_{ij} \right) \qquad \textbf{(4.1)}$$

Here *h(p)* is not a real load value; it is just an estimation of the effect of future assignments that is used to compare cost values of different nodes fairly. Different functions may also be used as *h(p)*. In our thesis, we chose the simple one in (Equation 4.1), which provides proper solutions.

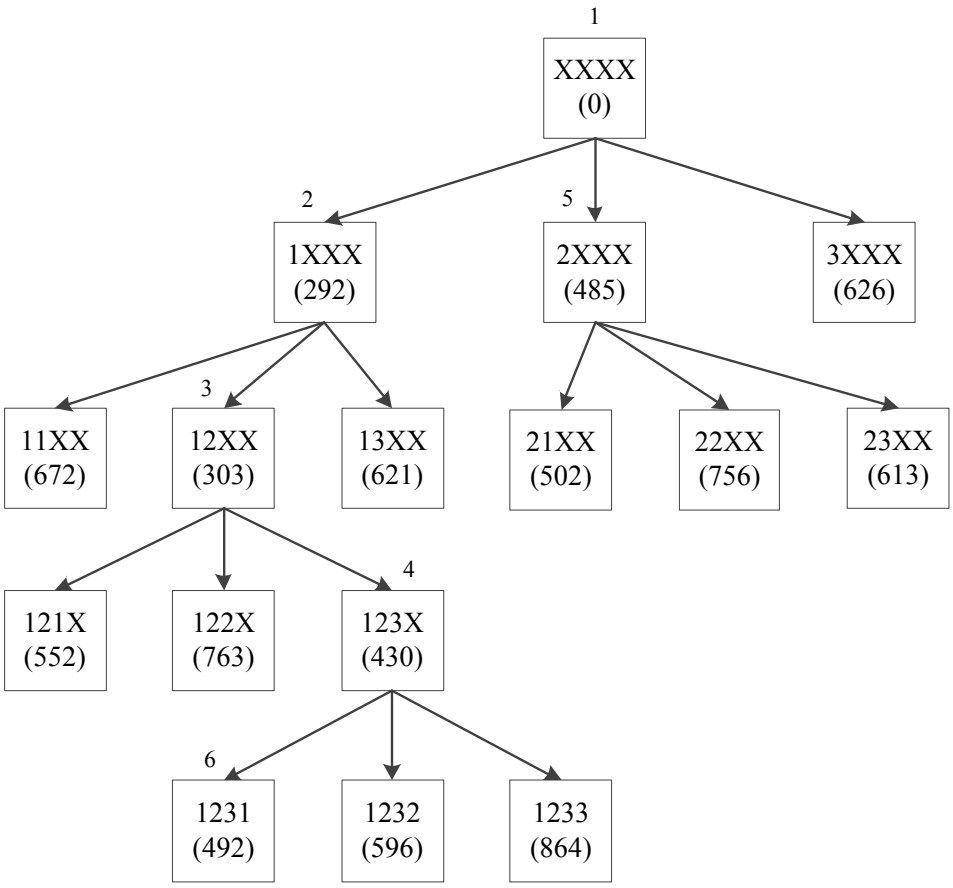


Figure 4.1 : Search tree for A* algorithm.

As an illustration, for the sample system of three hosts, five objects and four aspects (see Figure 3.1) the resulting search tree of the A* algorithm is shown in Figure 4.1. A search-tree node includes partial assignment of aspects to hosts, and the value of the cost function. A partial assignment means that some aspects are unassigned; if

there is an 'X' in the place of aspect a_i it indicates that i^{th} aspect has not been assigned yet. For example in Figure 4.1 the node with label 5 shows that aspect a_1 has been assigned to host 2, and the value produced by the cost function *f(p)* is 485. The search tree's depth equals the number of aspects, and any node of the tree can have a maximum of n successors, which is the number of the hosts.

The algorithm maintains two lists named OPEN and CLOSED. The OPEN list keeps nodes that need to be examined, while the CLOSED list keeps nodes that have already been examined. When a node is selected from OPEN list to be examined, its child nodes are generated and put into the OPEN list. The nodes in the OPEN list are ordered before the selection according to cost function *f(p)*; that is, the algorithm selects the node with the minimum cost. Initially, the OPEN list contains just the root node, and the CLOSED list is empty.

In the example, given in Figure 4.1, labels show the selection order of the nodes for the given system. We start with the root node labelled as 1. We examine children of the root and select node 2 because it has the lowest cost value (292). Then all children of node 2 are added to the OPEN list, where children of root still exist. Now node 3 is selected from the OPEN list because it has the smallest cost value and its children are added to the list. After that, nodes 4, 5 and finally 6 are selected from the OPEN list according to their cost values. Since node 6 is a leaf node the algorithm terminates and the final solution is the assignment of aspects as presented on this node. If more than one node have the same smallest cost value then the second objective function (*F2*) is taken into account, and the node with the smallest sum of load is selected. The complete A* algorithm is as follows:

```
Initialize OPEN and CLOSED lists (OPEN=root node; CLOSED=EMPTY)
while the OPEN list is not empty {
  Get node p off the OPEN list with the lowest f(p)
  Add p to the CLOSED list
  if p is the leaf node then return p as solution
  Generate each successor node p' of p
  Add p' to the OPEN list
}
```

Figure 4.2 : Complete A* algorithm.

4.2 Genetic Algorithm

Genetic algorithms (GA) [4], which are used for solving many search and optimization problems, generate solutions using techniques inspired by natural evolution. A GA starts by generating a random population of solutions (called chromosomes in GAs literature). At each iteration, a number of solutions are selected for the mating pool according to their fitness. Crossover and mutation operations are then applied to mating pool in order to produce new solutions. The algorithm terminates when either a maximum number of generations has been produced or population is converged.

The first step in designing a GA is to develop a suitable representation for chromosomes in the population. In our algorithm, we use integer representation, with considering the relationship between hosts and aspects. For k aspects there are k elements (called gene in GAs literature) in the chromosome. The value of each gene in the chromosome represents the host to which that aspect is allocated. As an example, a chromosome with four genes is shown in Figure 4.3.

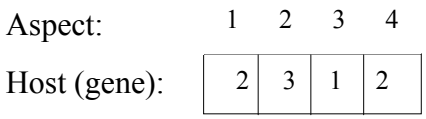


Figure 4.3 : Chromosome representation.

The fitness value of each gene in the chromosome is the load on the host that the gene represents ($Load_q$), which is calculated using the equation giving in 3.1. On the other hand, the fitness value of the chromosome is the maximum gene fitness, which is the load on the heaviest-loaded host that is represented by *F1* as given in Equation 3.2.

For selection phase, we use Roulette-Wheel Selection, which is a very common probabilistic selection method for GAs. It simulates a toss in a roulette-wheel to select an individual. Each individual is assigned a segment on the roulette-wheel proportional to its selection probability. This selection scheme is repeated until a number (size of population) of individuals have been selected. Figure 5.4 illustrates the Roulette-Wheel Selection.

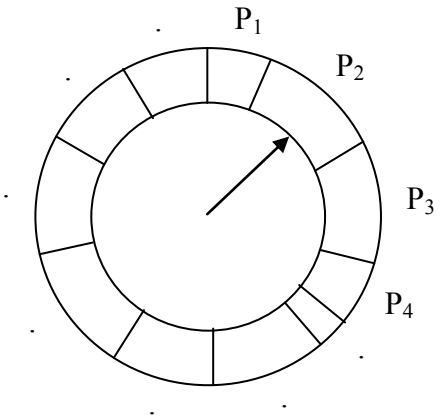


Figure 4.4 : The roulette-wheel selection method.

In our algorithm, we apply one point crossover operation on a pair of chromosomes, which is randomly selected from the mating pool. One point crossover is accomplished by randomly choosing a point along the length of the chromosome, and exchanging all genes beyond that point in either chromosome. This operation yields two new chromosomes. After crossover operation, a mutation operation is performed on a randomly selected gene of each chromosome with a certain probability. In mutation operation the value of a gene is replaced by randomly generated host number. These operations are illustrated in the Figure 4.5 and Figure 4.6.

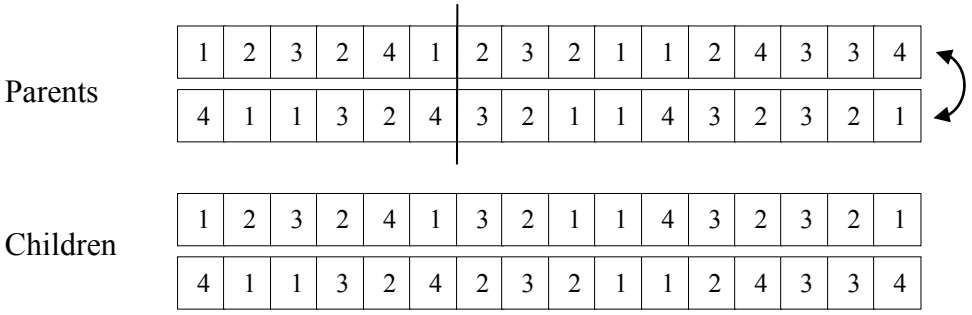


Figure 4.5 : Crossover operation.

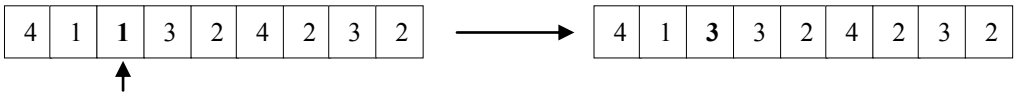


Figure 4.6 : Mutation operation.

After crossover and mutation operations the worst chromosomes, the chromosomes with the highest value of fitness (*F1*) in the population are replaced by new ones in the mating pool. This means that the best chromosomes (the chromosomes with the

lowest value of fitness) in the population are carried to the next generation (called elitism in GAs literature). In our algorithm we replace 1 chromosome by new ones with better fitness values. If there are many chromosomes with the same fitness value (*F1*), then the second objective (*F2*) comes into play, and the chromosomes with the smallest *F2* value are selected. The complete GA is given in Figure 4.7.

```
Generate initial population (chromosomes represent different
aspect assignment possibilities. Fitness = F1)
do {
  Create mating pool
  Apply crossover operation
  Apply mutation operation
  Apply elitism (Select chromosomes with smallest F1 values. If
    these values are equal select chromosomes with smallest F2
    values.)
  Carry new chromosomes from mating pool to population
}until(max generation is reached or converged)
```

Figure 4.7 : Complete GA.

4.3 Partical Swarm Optimization

Particle Swarm Optimization (PSO) [5] is a population based stochastic optimization technique first proposed by Kennedy and Eberhart in 1995. The algorithm is inspired by the social behavior of organisms such as bird flocking or fish schooling. The system consists of multiple candidate solutions and searches for optimal solution by updating generations. Each solution candidate, called a 'particle', flies in the problem search space looking for the optimal position to land.

Each particle keeps track of its position in the problem space, which is associated with the best cost value (fitness) it has achieved so far. Also the best position, obtained so far by any particle in the population is tracked as time passes through particle quests. These local and global best solutions are used to balance exploration and exploitation of the algorithm.

The algorithm is initialized with a population of random solutions. At each time step, the travelled distance (velocity) of each particle is determined toward its local and global best positions using the equation giving in 4.2. Then the position of each

particle is updated according to its velocity value using the equation giving in 4.3. After each step, particles renew their local best position if they get better cost value. Also the global best position is updated according to new local best positions. When updating the global best position, the local best position with the lowest cost value (*F1*) is chosen among the particles. If there are many particles with the same cost value (*F1*), then the second objective (*F2*) comes into play, and the local best position of the particle with the smallest *F2* value are selected. Concept of modification of a searching point by PSO is shown in Figure 4.8 where X_i^k is the current position, X_i^{k+1} is the modified position, V_i^k is the current velocity, V_i^{k+1} is the modified velocity, V_i^{Pbest} is the velocity based on local best position, and V_i^{Gbest} is the velocity based on global best position.

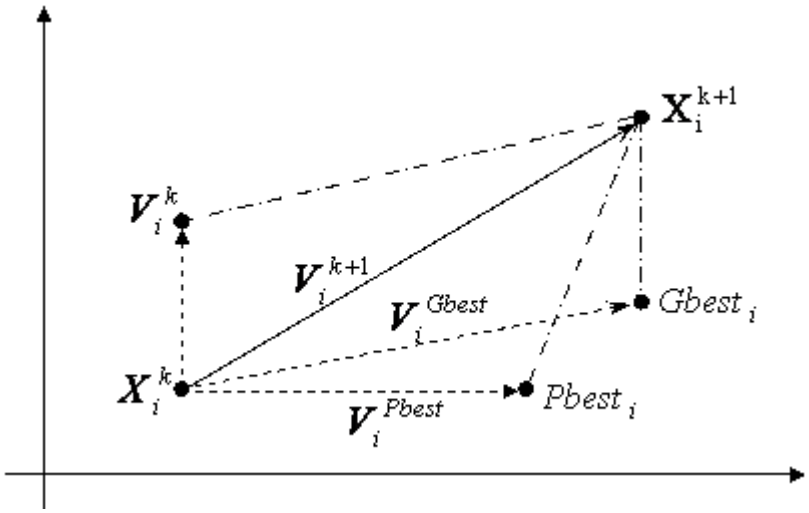


Figure 4.8 : Concept of modification of a searching point by PSO [32].

The algorithm terminates when either a maximum number of generations has been produced or population is converged. At the end, the global best position gives the result.

$$V_i = WV_i + c_1 Rand_p()(P_i - X_i) + c_2 Rand_g()(G - X_i) \quad \textbf{(4.2)}$$

$$X_i = X_i + V_i \quad \textbf{(4.3)}$$

In Equations 4.2 and 4.3, V_i, called the velocity for particle *i*, represents the distance to be traveled by this particle from its current position, X_i represents the particle position, P_i represents its best previous position (local best), and G represents the best position (global best) among all particles in the population. $Rand_p()$ and $Rand_g()$

are two random functions with a range [0,1]. c_1 and c_2 are positive constant parameters used to control the impact of previous historical values of particle velocities on its current one. Usually the value 2 is suggested for both parameters in the literature. W, on the other hand, is called inertia weight and it is used in order to keep the balance between the local and global optimality. In our algorithm W is assigned to 0.9, and $c_1 = c_2 = 2$. The complete PSO algorithm is given in Figure 4.8.

```
Generate initial population with the random position and velocity
Initialize the local best position of each particle with a copy of
  initial position
Initialize the global best position of population within local
   best positions
do {
  Update the velocity of each particle
  Update the position of each particle according to its velocity
  Calculate the fitness of each particle (Fitness = F1)
  Renew the local best position of each particle if it gets
    smallest F1
  Renew the global best position of population (Select the local
    best position with smallest F1 values. If these values are
    equal select the local best position with smallest F2 values.
}until(max generation is reached or converged)
```

Figure 4.9 : Complete PSO.

4.4 Aspect Copy Assignment Algorithm

In order to increase efficiency in distributed AOP we propose an algorithm, which works after finding proper aspect assignment. First, the algorithm finds the host h_p that is maximally loaded after the aspect assignment, and examines all communication costs related with aspects assigned to this host to find the highest one. After this examination, the algorithm finds out an aspect a_i on h_p, and a host h_q which are caused to highest communication cost on h_p. Let M be the set of objects that are located on h_q then the communication cost between h_p and h_q for a_i is calculated as follows:

$$Cost = \sum_{j \in M} R_{ij} C_{ij} D_{pq} \qquad \textbf{(4.4)}$$

Secondly, a copy of aspect a_i which can be called a_{k+1} (k is the number of aspects) is assigned to host h_q. This means that there is no need for a communication between h_p and h_q for a_i because h_q can now use a_{k+1} instead of a_i. After this assignment a new vector for a_{k+1} is added to aspect execution costs matrix, aspect-object relation counts matrix, and aspect-object communication costs matrix. For aspect execution costs matrix and aspect-object communication costs matrix, the values of the vector for a_{k+1} are the same as the values of the vector for a_i, because these aspects are the same one. On the other hand, for aspect-object relation counts matrix, all the relation counts between a_i and the objects which are located on h_q are copied to the vector for a_{k+1}, and then, these values on the vector for a_i and the rest of the values of the vector for a_{k+1} are initialized to zero.

	…	a_i	…	a_{k+1}
h_1		1		1
h_2		3		3
…		…		…
h_n		4		4

(a) Aspect Execution Costs

	o_1	o_2	…	o_m
…				
a_i	3	2		5
…				
a_{k+1}	3	2		5

(b) Aspect-Object Communication Costs

	o_1	o_2	…	o_m
…				
a_i	12	0	…	16
…				
a_{k+1}	0	9	…	0

(c) Aspect-Object Relation Counts

Figure 4.10 : Adding a new vector for a_{k+1} to input matrices.

Finally, a further examination is performed on the other hosts different then h_p and h_q. We check the objects on these hosts which are related with a_i and determine if they will use a_i or a_{k+1} according to the communication cost between hosts. For example in Figure 4.11, let h_p is the maximally loaded host and after examination, a copy of a_i is assigned to h_q. Also let o_m which is located on h_r has a relation with a_i. We compare the communication costs D_{pr} and D_{qr}. If D_{qr} is less than D_{pr} then we decide that o_m will use a_{k+1} which is located on h_q instead of a_i which is located on h_p. We make all necessary updates on related input matrices.

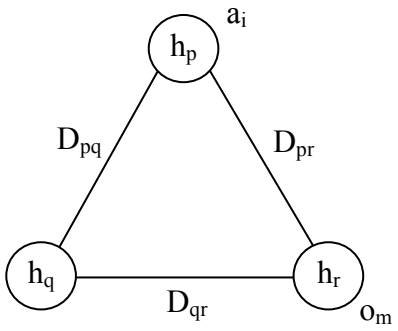


Figure 4.11 : A simulation of aspect copy assignment.

The cost value *F1* is calculated after the above-described steps. If there is a decrease in the cost value, we repeat the process again to find the new aspect copy assignment. The process continues until there are no decreases in the cost value. The complete algorithm is given in Figure 4.12.

```
do
  hp = findMaximallyLoadedHost();
  cost = 0;
  a = 0;
  hq = 0;
  for(int i=1; i <= number of hosts; i++) do
      if(h(i) != h(p)) then
         for(int j=1; j <= number of aspects; j++) do
            if(a(j) is assigned to hp)then
               temp = 0;
               for(int k=1; k <= number of objects; k++) do
                  if(o(k) is located on h(i))then
                     temp += D(hp,i)*R(j,k)*C(j,k);
                  end
               end
               if (temp > cost) then
                  cost = temp;
                  a = a(j);
                  hq = h(i);
               end
            end
         end
      end
   end
   /* Update Aspect-Object Relation Counts Matrix */
   for(int i=1; i <= number of objects; i++) do
      if(o(i) is located on hq)then
         R(number of aspects+1,i) = R(a,i);
         R(a,i) = 0;
      else
         R(number of aspects+1,i) = 0
      end
   end
   /* Update Aspect Execution Cost Matrix */
   for(int i=1; i <= number of hosts; i++) do
      X(number of aspects+1,i) = X(a,i);
   end
   /* Update Aspect-Object Communication Cost Matrix */
   for(int j=1; j <= number of objects; j++) do
      C(number of aspects+1,i) = C(a,j);
   end
   for(int i=1; i <= number of objects; i++) do
      if(o(i) is not located on hp and hq) then
         if(D(hq,i) < D(hp,i)) then
            R(number of aspects+1,i) = R(a,i);
            R(a,i) = 0;
         end
      end
   end
   number of aspects = number of aspects + 1;
while(cost value (F1) is decreases)
```

Figure 4.12 : Complete aspect copy assignment algorithm.

5. EXPERIMENTAL RESULTS

To evaluate the performance of our algorithms firstly, we coded our algorithms in Java programming language using Eclipse SDK 3.4.2 and compared their efficiency for different aspect oriented programs by executing them on a server with four quad-core 2.60 GHz Intel Xeon CPU processors and 15 GB main memory, running the Ubuntu Linux 10.04.1. The system specifications are shown in Figure 5.1. Secondly, aspects of different programs are assigned to hosts according to solutions obtained by three algorithms and these programs are executed on a simulation tool called the Asynchronous Distributed System Simulator [33]. We compared completion time of programs related to the different aspect assignments.

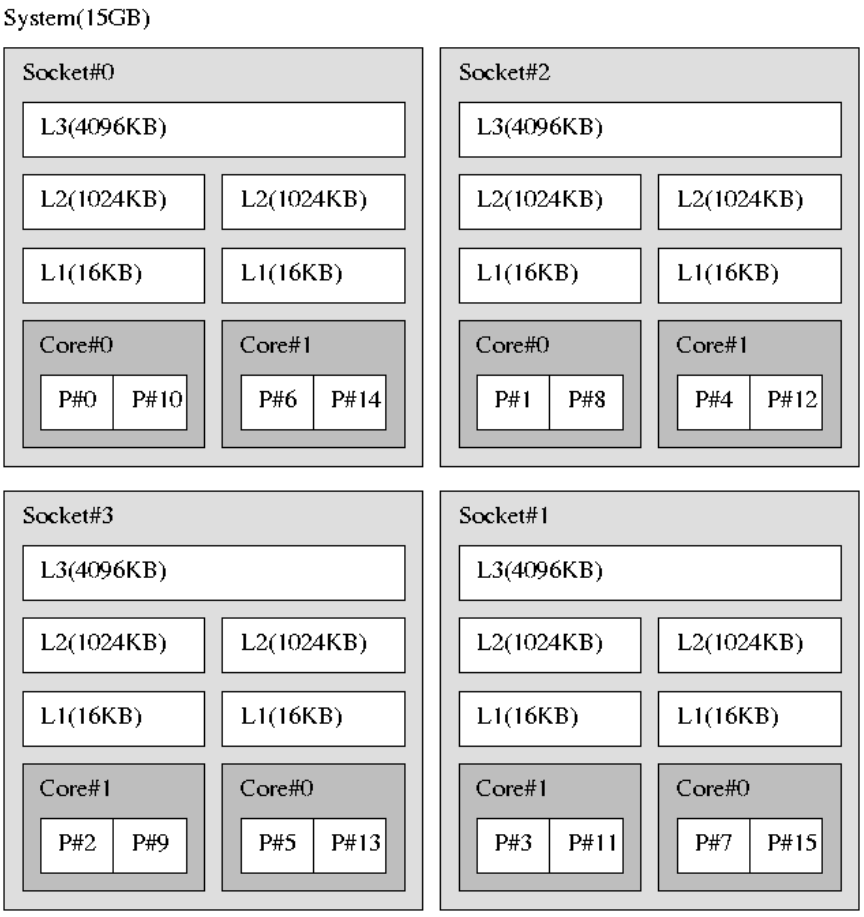


Figure 5.1 : System specifications.

The Asynchronous Distributed System Simulator is written in Java programming language using a threaded architecture and can simulate any algorithm that has been designed for the distributed system network. It takes input parameters through an XML file which specifies the nodes in the network, the links between the hosts and the algorithm to be run on the distributed system. The simulator has a queue of messages that represents messages that are in transit on the network. Each link has a delay associated with it and messages sent using a link are not delivered until after the delay period has passed.

In our experiments we test our algorithms on a fully connected distributed system and a partially connected distributed system with five hosts. The host connectivity graphs of the systems are shown in Figure 5.2 and Figure 5.3 respectively, where labels on edges show the cost of delays of the communication links (D_{pq}). The partially connected distributed system is obtained from the fully connected distributed system by removing some connections between hosts. On these systems we try to distribute aspects of three aspect oriented programs with different sizes. The programs are detailed below:

- *P1* : 10 objects and 5 aspects
- *P2* : 20 objects and 10 aspects
- *P3* : 30 objects and 15 aspects

For each of these programs we generate 10 different datasets randomly, which include following properties of programs: aspect execution costs (X_{qi}), aspect-object relation counts (R_{ij}), aspect-object communication costs (C_{ij}) and object locations (L_j). X_{qi} and C_{ij} cost values are generated randomly in the range of [1, 10]. Similarly, R_{ij} values are generated randomly in the range of [0, 20]. We run GA and PSO for 50 trials for each of the dataset, and then we calculate the confidence intervals (CI) for 95% confidence level for the cost value *F1*. CI for 95% confidence level means that there is a probability of 95% to get the result during the specific range. We calculate CI as follows:

$$CI = \frac{t * \sigma}{\sqrt{n}} \qquad \textbf{(5.1)}$$

In Equation 5.1, n is the sample size, σ is the standard deviation of the sample, and t is the critical value from the t-distribution [34] with the degrees of freedom of n. For sample size 50 and the probability of 95% confidence t is equal to 2.009.

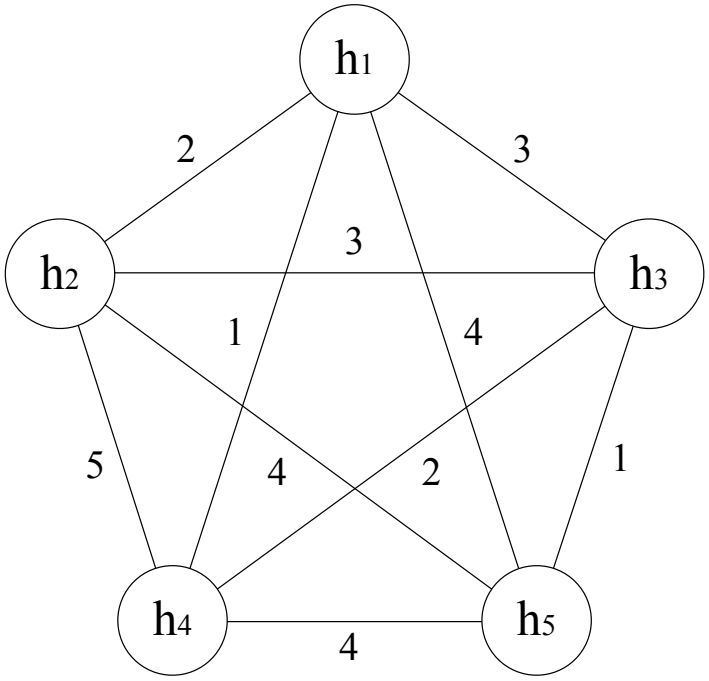


Figure 5.2 : Host connectivity graph for fully connected distributed system.

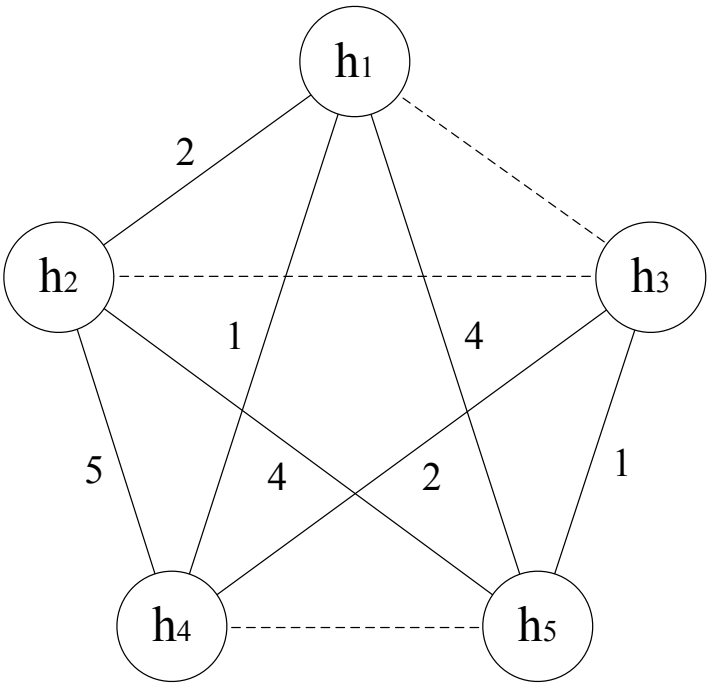


Figure 5.3 : Host connectivity graph for partially connected distributed system.

Using the results of simulations, we evaluate performance of our algorithms in two ways. Firstly, we consider execution times of algorithms, spent to find a solution. Secondly, we investigate the completion time of the AOP, when the aspects are assigned to host according the solutions provided by the algorithms.

5.1 Fully Connected Distributed System

Table 5.1: Obtained cost values and execution times of A* on a fully connected distributed system for P1, P2, and P3 in milliseconds

# of	P1			P2			P3		
DS	F1	F2	Time	F1	F2	Time	F1	F2	Time
1	1690	6287	20	6246	29085	671278	12078	56246	3935320
2	1627	6505	25	6563	30585	441766	12435	59121	9420174
3	1713	6173	32	6131	28809	317609	12871	63270	1956809
4	1615	7240	24	7100	32391	524357	13115	59602	5884341
5	2012	5751	20	6377	26695	389860	12045	55062	5229528
6	1838	7057	30	5397	21439	478004	12346	59445	3257284
7	1427	5194	26	5969	26092	383846	12053	56469	6977759
8	1763	7023	25	6127	28620	680675	12741	58318	7851088
9	1985	6994	39	6019	27461	238493	13734	65334	3139030
10	1354	5307	39	6267	25851	174162	13291	61567	2761528
Avg.			27			430005			5041286

Table 5.2: Obtained cost values and execution times of GA and PSO on a fully connected distributed system for P1 in milliseconds

# of	GA				PSO			
DS	F1	CI	F2	Time	F1	CI	F2	Time
1	1690	0	6287	441	1690	0	6287	43
2	1627	0	6505	379	1627	0	6505	44
3	1713	0	6173	371	1713	0	6173	43
4	1615	0	7240	425	1615	0	7240	43
5	2012	0	5751	418	2012	0	5751	43
6	1838	0	7057	378	1838	0	7057	43
7	1427	0	5194	368	1427	0	5194	43
8	1763	0	7023	359	1763	0	7023	44
9	1985	0	6994	399	1985	0	6994	43
10	1354	0	5307	363	1354	0	5307	43
Avg.				390				43

Table 5.3: Obtained cost values and execution times of GA and PSO on a fully connected distributed system for P2 in milliseconds

# of DS	GA				PSO			
	F1	CI	F2	Time	F1	CI	F2	Time
1	6723	62	30529	1391	7044	58	32279	133
2	6712	37	29880	1280	6882	37	31171	133
3	6575	60	29123	1343	6864	62	30523	139
4	7242	35	31615	1378	7379	30	32682	136
5	6383	25	28424	1349	6447	38	29207	137
6	5405	12	21993	1212	5603	34	23592	140
7	6017	38	26367	1300	6085	33	26851	135
8	6367	99	28411	1308	6927	53	30632	136
9	6463	87	28796	1154	6645	56	30234	135
10	6363	53	27032	1291	6646	59	28803	134
Avg.				1301				136

Table 5.4: Obtained cost values and execution Times of GA and PSO on a fully connected distributed system for P3 in milliseconds

# of DS	GA				PSO			
	F1	CI	F2	Time	F1	CI	F2	Time
1	13091	117	60384	3556	13906	98	63839	315
2	13061	102	60037	3642	14062	105	64667	314
3	14189	115	65718	3520	14944	114	69034	307
4	13736	102	62893	3387	14972	116	67587	317
5	12923	99	59053	3653	13832	123	63036	308
6	13491	115	62978	3549	14317	97	66134	317
7	12813	88	58830	3675	13767	83	62940	316
8	13144	92	61229	3487	13874	105	64529	312
9	14566	103	67908	3516	15368	96	71022	311
10	13883	131	63811	3520	14813	107	68284	308
Avg.				3551				312

Tables 5.1, 5.2, 5.3 and 5.4 provide performance comparison of three algorithms by considering obtained cost values (*F1* and *F2*) and their execution times for three different programs. Results in Table 6.1 show that A*, GA, and PSO obtain always the same cost values for P1, which is a relative smaller program than *P2* and *P3*. In this case A* performs more than 10 times faster than GA and almost 2 times faster than PSO. When the number of aspects increases A* obtains better (smaller) cost values than the GA and PSO. This means that A* can distribute aspects more efficiently than the GA and PSO for bigger programs. A* achieves about 7% smaller *F1* values for P2 and about 10% smaller *F1* values for *P3* compared to the GA. Also it achieves about 12% smaller *F1* values for *P2* and about 14% smaller *F1* values for *P3* compared to the PSO. On the other hand, with the increase in the number of

aspects and objects in the program, the execution time of A* increases very fast. For *P2* the GA proposes a solution 300 times faster and the PSO proposes a solution 3000 times faster than the A*. Similarly, for *P3* the GA proposes a solution nearly 1400 times faster and PSO proposes a solution 16000 times faster than the A*. When we compare GA and PSO, it is shown that GA achieves about 7% smaller *F1*values than PSO for both of *P2* and *P3*. However, PSO proposes a solution 10 times faster than the GA for all of the programs.

The relation between the performance of the algorithms and the number of objects and aspects in the program can be explained as follows. A* algorithm uses a best-first search technique that builds a search-tree by visiting the most promising nodes first. When the number of nodes in the search tree is smaller, it quickly reaches the solution node. However, if the number of aspects increases, nodes in the tree also increase and the algorithm spends more time to visit these nodes. On the other hand, GA and PSO use a random search technique, which requires only a certain number of iterations to obtain a solution. Therefore if the number of aspects increase the execution time of the A* is increased much more that the GA and PSO. However it is expected that the A* can find optimal solution in all cases, while the GA and PSO can obtain optimal aspect assignments only for relative small systems.

Since the execution time of A* increases very fast with the increase in the number of program elements, we do further experiments on GA and PSO. We execute these algorithms for different sizes of hosts, objects, and aspects. For each execution, we generate a dataset randomly. The properties of the datasets used in this experiment are the same as the ones used in the previous experiment.

Table 5.5 provides performance comparison of two algorithms by considering obtained cost values (*F1* and *F2*) and their execution times for different sizes of hosts, objects, and aspects. Results in Table 6.4 shows that GA achieves about 18% smaller *F1* values and about 7% smaller *F2* values compared to the PSO. On the other hand, PSO performs almost 10 times faster than GA.

In order to validate the efficiency of the aspect assignments of three algorithms we run three aspect-oriented programs (*P1*, *P2*, and *P3*) on the simulator and measure the completion time of these programs under different assignments of aspects. To evaluate the performance improvement achieved by our algorithms, we created a rival algorithm, namely the random assignment algorithm (RAA). The RAA assigns

aspects to hosts randomly without taking any properties of the system and program into consideration. This is our baseline algorithm that helps us to observe the speedup obtained by the proposed algorithms. We performed the RAA on three programs (*P1*, *P2*, *P3*) for each dataset 10 times. We ran these programs on the simulator for 10 different random assignments produced by the RAA and calculated the average of the completion time *T(RAA)* for each dataset. To get the speedup of the AOPs we do the following calculations: *T(RAA)/T(A*)*, *T(RAA)/T(GA)*, and *T(RAA)/T(PSO)*, where *T(A*)*, *T(GA)*, and *T(PSO)* are completion times of the AOPs, when aspects are assigned according to the A*, GA, and PSO, respectively. Results are given in Table 5.6. For example, the value 2.6 in the first row and column of the table denotes that the execution time of the AOP *P1* for dataset #1 takes 2.6 times longer if the aspects are assigned by the RAA then the case where aspect assignment is performed by the A*, GA, and PSO.

Table 5.5: Obtained cost values and execution times of GA and PSO for different sizes of hosts, objects, and aspects in milliseconds

# of Hosts	# of Objects	# of Aspects	GA				PSO			
			F1	CI	F2	Time	F1	CI	F2	Time
10	20	10	7144	87	52982	1203	7972	126	56866	123
10	40	20	26723	202	221873	4231	29700	238	231638	467
10	60	30	59613	368	525417	10755	63989	379	536408	1155
20	40	20	16303	252	223507	3577	19975	278	226594	412
20	60	30	37096	270	527756	8081	41365	388	544472	905
20	80	40	70215	534	1045199	15914	78327	683	1054328	1745
30	60	30	33355	369	556523	7535	37899	308	566392	837
30	80	40	53917	277	1015205	13157	59619	585	1035816	1486
30	100	50	78940	920	1597086	21747	92570	897	1601167	2354
40	80	40	50145	340	1022945	12223	54847	267	1039298	1352
40	100	50	71510	409	1641617	19714	79145	820	1661243	2147
40	120	60	94863	820	2336423	29651	113424	842	2343305	3184
50	100	50	67551	399	1647067	18899	73176	398	1682990	2078
50	120	60	85000	642	2334146	28438	95506	1320	2369123	3064
50	140	70	109906	1615	3147700	40311	129996	881	3145409	4285

Table 5.6: Speedup of programs using proposed algorithms relative to random assignment

# of	P1	P2			P3		
Dataset	A*, GA and PSO	A*	GA	PSO	A*	GA	PSO
1	2.6	1.9	1.8	1.8	2.0	1.9	1.7
2	2.3	1.7	1.7	1.6	2.2	1.8	1.8
3	2.2	2.6	2.3	2.2	2.1	1.9	1.7
4	2.5	2.0	2.0	1.8	2.0	1.8	1.7
5	1.8	2.1	2.0	2.0	2.3	2.1	1.9
6	1.9	2.1	2.1	2.0	1.9	1.8	1.6
7	2.8	2.1	1.9	1.9	2.2	1.9	1.8
8	2.4	2.2	2.2	1.9	1.9	1.8	1.7
9	2.4	2.0	1.8	1.7	2.3	2.0	2.0
10	2.9	2.2	2.0	1.9	2.0	1.8	1.7

We deduce from Table 5.6 two main results. Firstly, properly assignment of aspects improves the performance of a distributed AOP. Experimental result show that the proposed algorithms can speed up the AOPs between 1.6 and 2.9 times. Secondly, we see that A* achieves slightly higher speedups then the GA and PSO except for *P1*, where the GA and PSO obtain also the same values. Also, it is shown that GA achieves slightly higher speedups then PSO for *P2* and *P3*. This result was expected, since the cost values (*F1*) given in Tables 6.1, 6.2, 6.3 and 6.4 are related to the completion time of the AOPs and they have almost the same characteristic as the speedup values in Table 5.6.

We deduce from Table 5.6 two main results. Firstly, properly assignment of aspects improves the performance of a distributed AOP. Experimental result show that the proposed algorithms can speed up the AOPs between 1.6 and 2.9 times. Secondly, we see that A* achieves slightly higher speedups then the GA and PSO except for *P1*, where the GA and PSO obtain also the same values. Also, it is shown that GA achieves slightly higher speedups then PSO for *P2* and *P3*. This result was expected, since the cost values (*F1*) given in Tables 6.1, 6.2, 6.3 and 6.4 are related to the completion time of the AOPs and they have almost the same characteristic as the speedup values in Table 5.6.

We run our aspect copy assignment algorithm for the results of three algorithms. Results are given in Table 5.7, 5.8, and 5.9.

Table 5.7: Obtained cost values of aspect copy assignment algorithm on a fully connected distributed system for P1

# of DS	A*			GA			PSO		
	# of copy	F1	F2	# of copy	F1	F2	# of copy	F1	F2
1	0	1690	6287	0	1690	6287	0	1690	6287
2	4	1063	3822	4	1063	3822	4	1063	3822
3	0	1713	6378	0	1713	6378	0	1713	6378
4	2	1568	5891	2	1568	5891	2	1568	5891
5	4	923	2538	4	923	2538	4	923	2538
6	6	806	3143	6	806	3143	6	806	3143
7	0	1427	5194	0	1427	5194	0	1427	5194
8	1	1568	5908	1	1568	5908	1	1568	5908
9	6	729	2264	6	729	2264	6	729	2264
10	0	1354	5307	0	1354	5307	0	1354	5307

Table 5.8: Obtained cost values of aspect copy assignment algorithm on a fully connected distributed system for P2

# of DS	A*			GA			PSO		
	# of copy	F1	F2	# of copy	F1	F2	# of copy	F1	F2
1	0	6246	29085	3	5696	25474	4	5957	26747
2	0	6563	30585	3	5946	26040	3	6164	26532
3	1	6006	25947	2	6045	25628	3	5963	25202
4	0	7100	32391	2	6467	28617	2	6744	29897
5	6	4967	17642	3	5372	23489	4	5297	23477
6	0	5397	21439	1	5237	21337	2	5092	21418
7	9	3657	14660	6	4405	19102	6	4591	19868
8	0	6127	28620	4	4471	17984	6	4287	16751
9	0	6019	27461	7	4310	18597	7	4740	20152
10	7	3737	16869	4	4992	21780	2	5856	25358

Table 5.9: Obtained cost values of aspect copy assignment algorithm on a fully connected distributed system for P3

# of DS	A*			GA			PSO		
	# of copy	F1	F2	# of copy	F1	F2	# of copy	F1	F2
1	0	12078	57378	3	12018	54474	3	12603	57638
2	1	12085	56246	3	11482	52337	4	12049	55159
3	0	12871	63270	4	12349	56348	3	13243	60150
4	6	10457	47754	3	12509	56545	3	13086	59191
5	1	11364	53185	3	11615	52415	3	12269	54853
6	0	12346	59445	3	12326	56551	3	12983	59822
7	4	10035	45502	6	9980	44453	6	10683	48133
8	0	12741	58318	4	11481	52291	3	12556	57546
9	0	13734	65334	5	12160	55858	4	13151	60354
10	5	10946	51161	3	12207	55660	4	12724	58216

Since both of the algorithms find the same assignments for P1, results indicated from Table 5.7 for each of the algorithm are identical. On the other hand, for P2 and P3, more copies of aspects are used for the assignments of GA and PSO than the assignments of A*. Therefore, the cost values obtained for GA and PSO are less than the cost values obtained for A*. It is obvious that using more copies of aspects results in decreasing the cost values. Because, using copies of aspects reduces the communication costs. On the other hand, aspect execution costs are increased by using copies of aspects, however, since communications costs are higher than execution costs, overall cost values decrease up to a certain level.

5.2 Partially Connected Distributed System

The experiments on partially connected distributed system show that algorithms cannot be able to find any solution. They were able to find a solution only with three datasets for *P1*. This can be explained as follows. The aspects are used by the objects where the locations of the objects are fixed and predetermined according to their specific jobs. So, there must be a communication link between the hosts that the objects are located and the hosts that the aspects are assigned. Since each aspect is used by many objects, algorithms cannot be able to find any host for aspects that the communications can be done properly.

To be able to find a result on a partially connected distributed system we add virtual links between hosts where there are no links actually. To make it sense, we set the communication cost for these virtual links to very high value, for example 1000000. Since the results have to use at least one of the virtual links, obtained cost values are over 1000000 expectedly.

We run our aspect copy assignment algorithm for the results of three algorithms on a partially connected distributed system. Results are given in Table 5.10, 5.11, and 5.12. Since we use one or more copies of aspects, the obtained cost values (*F1* and *F2*) decrease drastically. It is obvious that after using the copies of aspects, there is no longer need to use virtual links.

Table 5.10: Obtained cost values of aspect copy assignment algorithm on a partially connected distributed system for P1

# of DS	A*			GA			PSO		
	# of copy	F1	F2	# of copy	F1	F2	# of copy	F1	F2
1	5	1788	3821	5	1788	3788	5	1788	3821
2	10	665	1709	10	665	1709	10	665	1712
3	8	594	1796	8	594	1796	8	594	1796
4	8	772	2254	8	772	2254	8	772	2254
5	12	496	2056	12	496	2056	12	496	2056
6	11	543	2021	11	543	2021	11	543	2021
7	10	471	1653	10	471	1653	10	471	1653
8	10	1526	3246	10	1526	3246	10	1526	3246
9	7	287	988	7	307	985	7	287	988
10	9	703	1798	9	703	1798	9	703	1798

Table 5.11: Obtained cost values of aspect copy assignment algorithm on a partially connected distributed system for P2

# of DS	A*			GA			PSO		
	# of copy	F1	F2	# of copy	F1	F2	# of copy	F1	F2
1	14	2654	10258	14	2649	10506	14	2717	11361
2	16	2193	8714	17	2182	8900	17	2323	9342
3	11	3797	11675	11	3470	12030	12	3780	12760
4	16	3448	11083	16	3397	11233	17	3434	10799
5	15	1960	8964	15	2160	9602	15	2247	9701
6	13	2866	9323	13	3202	9959	12	3141	10157
7	18	1446	6170	18	1449	6166	18	1459	6078
8	12	3546	12179	13	3128	11712	14	3459	12189
9	10	3032	10072	13	2873	9516	14	2587	9089
10	14	3404	12488	14	3337	12199	15	3152	11528

Table 5.12: Obtained cost values of aspect copy assignment algorithm on a partially connected distributed system for P3

# of DS	A*			GA			PSO		
	# of copy	F1	F2	# of copy	F1	F2	# of copy	F1	F2
1	20	6813	23127	20	7310	23385	22	7197	23173
2	19	5552	25490	18	5670	25620	18	5646	25069
3	21	4712	20906	21	5487	22818	20	5607	22465
4	20	5338	24446	21	5616	24378	21	5619	23973
5	26	6062	22871	28	5236	21671	27	5200	21911
6	31	3814	18019	26	4672	21059	25	4562	20690
7	19	6568	25716	19	6331	24901	19	6966	2487
8	25	6207	22086	23	6583	22576	23	6464	22458
9	18	6352	27527	21	5348	23978	21	5457	24130
10	24	4356	19486	24	5236	22422	23	5050	21438

Results in Tables 5.10, 5.11, and 5.12 indicate that using copies of aspects on a partially connected distributed system shows same characteristic as using on fully connected distributed system. However, in this case nearly the same number of copies of aspects is used for both of the algorithms. Results show that the cost values obtained for A* is smaller than the cost values obtained for GA and PSO in most cases for *P2* and *P3*. This is expected because A* produces better results than GA and PSO for *P2* and *P3*. Similarly, the cost values obtained for GA is smaller than the cost values obtained for PSO as expected.

6. CONCLUSION AND FUTURE WORK

In this thesis, we first formulate the aspect assignment problem for distributed AOP. During this formulation, we consider properties of heterogeneous distributed systems and distributed AOPs, such as processing capabilities of hosts, delays of communication links, amount of transferred data between objects and related aspects. Then we propose there different algorithms to solve this problem. One of these algorithms is A* algorithm which is based on best-first search technique, the second one GA which is based on the laws of natural evolution and the third one is PSO which is inspired by the social behavior of organisms such as bird flocking or fish schooling. We improved these algorithms by adding the feature of cloning necessary aspects. Without cloning feature sometimes, it is not possible to find a solution for partially connected systems, because locations of objects are predetermined and fixed. This feature also decreases the communication cost in the system.

Experimental results show that the proposed algorithms have their own advantages and disadvantages compared to each other. Firstly, we noticed that the A* algorithm obtained the optimal assignments for each of the programs with all datasets we used. On the other hand, the GA and PSO found the optimal assignments for small sized programs and sub-optimal solutions if the size of the programs increased. Secondly, the solution time for A* algorithm is considerably shorter than GA and PSO when the search space is smaller. However, the duration of the A* algorithm increases with the growth of the search space very fast and GA and PSO perform better. When we compare GA and PSO, the results show that GA achieves a bit smaller cost values than PSO for big sized programs. However, PSO proposes a solution 10 times faster than the GA for all of the programs.

To evaluate proposed algorithms and examine the effect of assignment of aspects on the speed of the AOPs, we distributed aspects in four different ways, namely according to A*, GA, PSO and randomly. Then we compared the completion time of the AOPs under different aspect assignments. The simulation results indicate that properly assignment of aspects can speed up the AOPs between 1.6 and 2.9 times.

We also see that A* provides approximately 10% higher speedups then the GA and PSO for relatively larger programs. In conclusion, properly assignment of aspects improves performance of the distributed AOPs, and because it's shorter response times the proposed PSO can be preferred to solve this assignment problem.

A further study can be carried out to find a solution for aspect assignment problem on multicore systems. Different from the distributed systems, multicore systems have a number of tightly connected cores using level one and two caches. Each core has its own level one cache. Level two cache is shared between the cores of processor. The way of distributing aspects over the cores can affect the performance of the program. Since caches are used frequently in systems and have limited capacity there is a need to propose new algorithms for aspect assignment problem on multicore systems.

Dynamic aspect assignment can be also performed as a further study. In static aspect assignment, an initial assignment of aspects is computed and this assignment is used for the duration of the program execution. However, in the case of dynamic aspect assignment, the process of assignment is done continuously over time. These types of assignments are quite difficult because it is necessary to find a proper assignment very quickly to respond to new information.

REFERENCES

[1] **Elrad, T., Filman, R. E., and Bader, A.**, 2001: Aspect-oriented programming. *Communications of the ACM*, Vol. 44, No. 10, pp. 29-32.

[2] **Tanenbaum, A. S., and Steen, M. V.**, 2002. Distributed systems: Principles and Paradigms. Prentice Hall.

[3] **Hart, P., Nilsson, N., and Raphael, B.**, 1968: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, pp. 100-107.

[4] **Goldberg, D. E.**, 1989. Genetic Algorithms in Search, Optimization and Machine Learning, Massachusetts: Addison Wesley.

[5] **Kennedy, J., and Eberhart, R.**, 1995: Partical Swarm Optimization, *In Proceedings of IEEE International Conference on Neural Networks*, pp. 1942–1948, Piscataway, NJ, USA.

[6] **Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W.**, 2001: An overview of AspectJ, *In 15th European Conference on Object-Oriented Programming*, pp. 327–353, Budapest, Hungary.

[7] **Boner, J.**, 2004: Aspectwerkz - dynamic aop for java, *In Proceedings of the 3rd International Conference on Aspect-Oriented Software Development*, Mancaster, UK.

[8] **Burke, B.**, JBoss-AOP, http://www.jboss.org/developers/projects/jboss/aop.

[9] **Johnson R.**, 2007. Spring - Java / J2EE application framework, Reference Manual Version 2.0.6, Interface21 Ltd.

[10] **Schweisguth, D.**, 2004. Second-generation aspect-oriented programming, JavaWorld.com, 07/05/04, http://www.javaworld.com/javaworld/jw-07-2004/jw-0705-aop.html.

[11] **Ghosh, S.**, 2007. Distributed Systems – An Algorithmic Approach, Chapman & Hall/CRC.

[12] **Soares, S., Laureano, E., and Borba, P.**, 2002: Implementing distribution and persistence aspects with AspectJ, *In ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp. 174-190.

[13] **Gursky, M.**, 1981. Some complexity results for a multi-processor scheduling problem, private communication from H. S. Stone.

[14] **Nishizawa, M., Chiba, S., and Tatsubori, M.**, 2004: Remote pointcut – a language construct for distributed AOP, *Proc. ACM Int'l Aspect Oriented Software Development*, pp. 7-15.

[15] **Pawlak, R., Seinturier, L., Duchien, L., Florin, G., Legond-Aubry, F., and Martelli, L.**, 2004: JAC: an aspect-oriented distributed dynamic framework, *Software: Practice and Experience*, Vol. 34, No. 12, pp. 1119–1148.

[16] **Benavides Navarro, L. D., Südholt, M., Vanderperren, W., De Fraine, B., and Suvèe, D.**, 2006: Explicitly distributed AOP using AWED, *In Proceedings of the 5th International Conference on Aspect-Oriented Software Development*, pp. 51–62, Bonn, Germany.

[17] **Mondejar, R., Garcia, P., Pairot, C., and Skarmeta, A.**, 2008: Building a distributed AOP middleware for large scale systems, *In Proceedings of the 2008 workshop on Next generation aspect oriented middleware*, pp. 17–22, New York, NY, USA.

[18] **Tanter, E., and Toledo, R.**, 2006: A Versatile Kernel for Distributed AOP, *In Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Systems*, volume 4025 of Lecture Notes in Computer Science, pp. 316–331, Bologna, Italy.

[19] **Stone, H. S.**, 1977: Multiprocessor scheduling with the aid of network flow algorithms, *IEEE Trans. Software Eng.*, Vol. SE-3, pp. 85-93.

[20] **Lo, V. M.**, 1988: Heuristic algorithms for task assignment in distributed systems, *IEEE Trans. Comp.,* Vol. C-37, pp. 1384–1397.

[21] **Chu, W. W.**, 1969: Optimal file allocation in multiple computing system, *IEEE Trans. Comp.*, Vol. C-18, pp. 885-889.

[22] **Shen, C. C., Tsai, W. H.**, 1985: A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion, *IEEE Trans. Comp.*, Vol. C-34, No. 3, pp. 197-203.

[23] **Ramakrishnan, S., Chao, H., and Dunning, L. A.**, 1991: A Close Look at Task Assignment in Distributed Systems, *Proc. IEEE Infocom'91*, IEEE Computer Society Press, pp.806–812.

[24] **Chockalingam, T., Arunkumar, S.**, 1995: Genetic algorithm based heuristics for the mapping problem, *Computer and Operations Research*, Vol. 22, No. 1, pp. 55–64.

[25] **Hadj-Alouane, A.B., Bean, J.C., Murty, K.G.**, 1999: A hybrid genetic / optimization algorithm for a task allocation problem, *Journal of Scheduling*, Vol. 2, No. 4, pp. 189–201.

[26] **Hamam, Y., Hindi, K.S.**, 2000: Assignment of program modules to processors: a simulated annealing approach, *European Journal of Operational Research*, Vol. 122, No. 2, pp. 509–513.

[27] **Lin, F. T., Hsu, C. C.**, 1990: Task assignment scheduling by simulated annealing, *IEEE Region 10 Conference on Computer and Communication Systems*, pp. 279–283.

[28] **Yin, P.Y., Yu, S.S., Wang, P.P., Wang, Y.T.**, 2006: A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems, *Computer Standard and Interface*, Vol. 28, No. 4, pp. 441–450.

[29] **Salman, A., Imtiaz, A., and Al-Madani, S.**, 2001: Particle swarm optimization for task assignment problem, *IASTED International Conference on Artificial Intelligence and Applications*, Marbella, Spain.

[30] **Zou, D.X., Gao, L.Q., Li, S., Wu, J.H., Wang, X.**, 2010: A novel global harmony search algorithmfor task assignment problem, *Journal of Systems and Software*, Vol. 83, No. 10, pp. 1678–1688.

[31] **Chen, D. J., and Huang, T. H.**, 1992: Reliability analysis of distributed systems based on a fast reliability algorithm, *IEEE Trans. Parallel Distributed Syst*., Vol. 3, No. 2, pp. 139–154.

[32] **Allaoua, B., Laoufi, A., Gasbaoui, B., and Abderrahmani, A.**, 2009: Neuro-Fuzzy DC Motor Speed Control Using Particle Swarm Optimization, *Leonardo Electronic Journal of Practices and Technologies*, Issue 15, pp. 1-18.

[33] **Burgess, S.**, 2007. Asynchronous Distributed System Simulator, University of Western Ontario, Computer Science.

[34] **Walpole, R., Myers, R., and Ye, K.**, 2002. Probability and Statistics for Engineers and Scientists, Pearson Education, 7th edition, pp. 237.

[35] **Url-1** < http://wiki.eclipse.org/Developer's_guide_to_building_tools_on_top_of _ AJDT_and_AspectJ >, accessed at 12.04.2011.

CURRICULUM VITAE

Candidate's full name: Samet BULU

Place and date of birth: Emet, 27.04.1985

Permanent Address: Cumhuriyet Mah. Canlar Sok. No:18 Arifler Sitesi D Blok. Daire:12 Üsküdar/İSTANBUL

Universities and Colleges attended: Istanbul Technical University

Publications:

Bulu, S., and Buzluca, F., 2011: Efficient Aspect Assignment in Heterogeneous Distributed Systems. *The 2011 International Conference on Software Engineering Research & Practice (SERP'11)*, July 18-21, 2011, Las Vegas, USA