Efficient Aspect Assignment in Heterogeneous Distributed Systems

S. Bulu, and F. Buzluca
Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey

Abstract - *In recent years, with increasing use of distributed systems, distributed Aspect Oriented Programming (AOP) arouses more interest. The way of distributing aspects over the network can affect the performance of the program. While assigning aspects to hosts, to achieve higher performance, properties of the distributed system and relation between aspects and objects must be taken into consideration. In this context as a solution for the aspect assignment problem in distributed systems we propose two algorithms, namely a Genetic Algorithm (GA) and an A* algorithm. We evaluate the efficiency of these two algorithms by using a simulator on heterogeneous distributed systems, where hosts and communication lines have different properties. It is shown that the GA is more favorable than A* algorithm for larger systems with many nodes while for smaller systems A* may be preferable. The simulation results indicate that the properly assignment of aspects to hosts improves the performance of a distributed aspect oriented program.*

Keywords: Aspect Oriented Programming, Distributed Systems, Task Assignment, Genetic Algorithm, A* Algorithm.

1 Introduction

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.

There are lots of AOP implementations that have been widely used. Some of these implementations are AspectJ [2], AspectWerkz [3], JBoss-AOP [4], PostSharp [5] and Spring AOP [6]. AspectJ, which was proposed as an extension of the Java language for AOP, is the most prominent implementation.

In an AOP cross-cutting 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.

Distributed systems are computer systems, in which the processing elements are connected by a network. They have become increasingly popular in recent years because of their high speed and high reliability. In such area, a new paradigm called distributed AOP has recently appeared. In distributed AOP, aspects can be deployed in a set of hosts. Remote pointcuts [7], which are similar to traditional remote method calls, invoke the execution of advices in aspects on remote hosts. DjCutter [7], JAC [8], AWED [9], Damon [10] and ReflexD [11] are approaches that address distributed AOP.

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. A similar problem also exists in distributed AOP: How to assign aspects to the hosts in the network so that the time required for program completion is minimized?

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.

In this paper we first formulate the aspect assignment problem in heterogeneous distributed systems by taking all necessary parameters into account and then propose two algorithms to solve this problem that occurs in distributed AOPs. The first algorithm is based on Genetic Algorithms (GA) [18], and the second one is an A* [19] based search technique. We also evaluate the efficiency of these algorithms for different systems and programs. We 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 the 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 randomly assignment.

The rest of this paper is organized as follows. Section 2 mentions related work. Section 3 gives background information on the aspect assignment problem. Section 4 and 5 present the two algorithms for the problem. Section 6 compares the efficiency of the algorithms and discusses the simulation results. Finally, section 6 concludes the paper.

2 Related work

Task assignment problem in distributed systems, which is similar to our aspect assignment problem, is well-known to be NP-hard [12]. Many approaches to this problem have been identified up to now. However, most of this reported approaches yield suboptimal solutions because of the large state space. Task assignment approaches can be classified into three categories, namely, graph theoretic, integer programming and heuristic methods.

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 approach is done by Stone [13]. In Stone's work, a Max Flow/Min Cut Algorithm is utilized to find assignments which minimize total execution and communication costs.

The integer programming method formulates the model as an optimization problem and solves it via mathematical programming techniques. For example, Chu [14] formulates the problem into a nonlinear integer zero-one programming problem and then reduced it to a linear zero-one programming problem.

As the first two approaches attempt to obtain the optimal solutions, they search the whole space and therefore need a lot of time and memory. Heuristic methods [15][16][17], 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.

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.

3 Problem definition

In this study we first formulate the problem, how to assign aspects of an AOP to hosts in a heterogeneous distributed system to increase the performance of the program. 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 form of heterogeneous in which the connected hosts have different processing capabilities. In the network X_{iq} denotes the execution cost of aspect a_i that is proportional to the execution time of the aspect when it is assigned to and executed on host h_q, $1 \le i \le k$, $1 \le q \le n$. 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 costs 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 [21]. Let there are 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_{iq}, 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, $1 \le L_j \le n$.

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 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_{iq}	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-Object 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-Object Relation Counts

$O(j)$	1	2	3	4	5
$h(L_j)$	1	2	2	3	1

e. Object Assignments (L_j)

Figure 1. An Example set of system and program parameters

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. 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 is the set of aspects that are assigned to host q then the load on host q is:

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

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 $F1$:

$$F1 = \max(Load_q), \quad 1 \le q \le n \qquad (2)$$

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 $F2$:

$$F2 = \sum_{q=1}^{n} Load_q \qquad (3)$$

4 Genetic algorithm

Genetic algorithms (GA) [18], 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 2.

Aspect:	1	2	3	4
Host (gene):	2	3	1	2

Figure 2. 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 (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 (2).

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.

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 3.

```
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 3. Complete GA

5 A* algorithm

A* [19] is a best-first search algorithm, which can guaranteed to find the optimal solutions. 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 (1) and (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 is the set of unassigned aspects in node p, then $h(p)$ is calculated as follows:

$$h(p) = \sum_{i \in U} \left(\sum_{j=1}^{m} R_{ij} \right) \qquad (4)$$

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 study we chose the simple one in (4), which provides proper solutions.

As an illustration, for the sample system of three hosts, five objects and four aspects (see Figure 1) the resulting search tree of the A* algorithm is shown in Figure 4. 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 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.

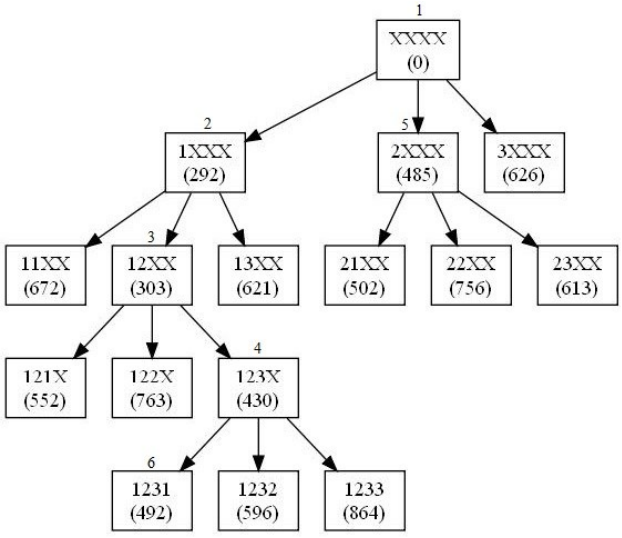

Figure 4. Search tree for A* algorithm

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, labels show the selection order of the nodes for the given system. We start with the root node labeled 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 5. Complete A* algorithm

6 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 speeds 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. Secondly, aspects of different programs are assigned to hosts according to solutions obtained by two algorithms and these programs are executed on a simulation tool called the Asynchronous Distributed System Simulator [20]. We compared completion time of programs related to the different aspect assignments.

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 with five hosts. The host connectivity graph of the system is shown in Figure 6, where labels on edges show the cost of delays of the communication links (D_{pq}). On this system 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

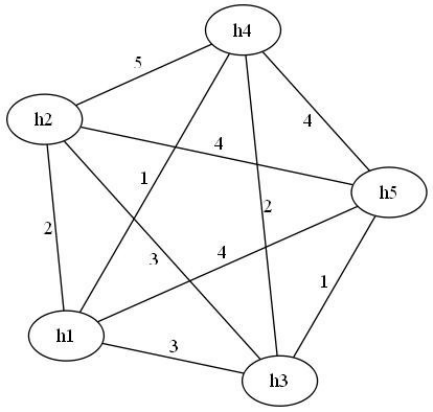

Figure 6. Host connectivity graph used in experiments

For each of these programs we generate 10 different datasets randomly, which include following properties of programs: aspect execution costs (X_{iq}), aspect-object relation counts (R_{ij}), aspect-object communication costs (C_{ij}) and object locations (L_j). X_{iq} 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]. These datasets can be found in [22].

Table 1. Obtained cost values execution times of two algorithms for P1 in milliseconds

# of dataset	GA			A*		
	F1	F2	Time	F1	F2	Time
1	1690	6287	441	1690	6287	25
2	1627	6505	379	1627	6505	43
3	1713	6173	371	1713	6173	42
4	1615	7240	425	1615	7240	31
5	2012	5751	418	2012	5751	43
6	1838	7057	378	1838	7057	42
7	1427	5194	368	1427	5194	24
8	1763	7023	359	1763	7023	38
9	1985	6994	399	1985	6994	47
10	1354	5307	363	1354	5307	34
Average			390			37

Table 2. Obtained cost values execution times of two algorithms for P2 in milliseconds

# of dataset	GA			A*		
	F1	F2	Time	F1	F2	Time
1	6836	31293	1392	6246	29085	671278
2	7013	30520	1289	6759	29237	441766
3	6895	29182	1342	6131	28809	317609
4	7110	32584	1373	7110	30025	524357
5	6604	27703	1349	6456	27601	389860
6	5388	22709	1211	5388	21194	478004
7	6412	28918	1301	5919	26361	383846
8	6127	28192	1307	6127	28051	680675
9	6716	29391	1153	6019	27461	238493
10	6758	27687	1291	6267	25851	174162
Average			1301			430005

Table 3. Obtained cost values execution times of two algorithms for P3 in milliseconds

# of dataset	GA			A*		
	F1	F2	Time	F1	F2	Time
1	14701	64427	3556	12078	56246	3935320
2	13672	61427	3642	12435	59121	9420174
3	14126	65993	3520	12871	63270	1956809
4	15334	69969	3387	13115	59602	5884341
5	13117	60436	3653	12045	55062	5229528
6	13725	65806	3549	12346	59445	3257284
7	13543	60250	3675	12053	56469	6977759
8	13667	64415	3487	12741	58318	7851088
9	16001	69825	3516	13734	65334	3139030
10	14588	66789	3520	13291	61567	2761528
Average			3551			5041286

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.

Tables 1, 2 and 3 provide performance comparison of two algorithms by considering obtained cost values (*F1* and *F2*) and their execution times for three different programs. Results in Table 1 show that GA and A* obtains always the same cost values for *P1*, which is a relative smaller program than *P2* and *P3*. In this case A* performs almost 10 times faster than GA. When the number of aspects increases A* obtains better (smaller) cost values than the GA as it is shown in Tables 2 and 3. This means that A* can distribute aspects more efficiently than the GA for bigger programs. A* achieves about 7% smaller *F1* values for *P2* and about 10% smaller values for *P3* compared to the GA. 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 for *P3* nearly 1400 times faster than the A*.

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. But 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 uses 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. However it is expected that the A* can find optimal solution in all cases, while the GA can obtain optimal aspect assignments only for relative small systems.

In order to validate the efficiency of the aspect assignments of two algorithms we run three aspect oriented programs 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(GA), and T(RAA)/T(A*), where T(GA) and T(A*) are completion times of the AOPs, when aspects are assigned according to the GA and A*, respectively. Results are given in Table 4. 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 GA or A*.

Table 4. Speedup of programs using proposed algorithms relative to random assignment

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

We deduce from Table 4 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 1.7 and 2.9 times. Secondly, we see that A* achieves slightly higher speedups then the GA except for *P1*, where the GA obtains also the same values. This result was expected, since the cost values (*F1*) given in Tables 1, 2 and 3 are related to the completion time of the AOPs and they have almost the same characteristic as the speedup values in Table 4.

7 Conclusion

In this paper 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 two different algorithms to solve this problem. One of these

algorithms is GA which is based on the laws of natural evolution and the second one is A* algorithm which is based on best-first search.

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 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 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 performs better, namely up to 1400 faster for one of the tested programs.

To evaluate proposed algorithms and examine the effect of assignment of aspects on the speed of the AOPs, we distributed aspects in three different ways, namely according to GA, A* 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 1.7 and 2.9 times. We also see that A* provides approximately 10% higher speedups then the GA 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 GA can be preferred to solve this assignment problem.

8 References

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

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

[3] AspectWerkz, "AspectWerkz Dynamic AOP for Java Overview", http://aspectwerkz.codehaus.org/, 2004.

[4] B. Burke and al. JBoss-AOP. www.jboss.org/developers/projects/jboss/aop.

[5] PostSharp. http://www.postsharp.org/.

[6] R. Johnson et al. *Spring – Java / J2EE application framework. Reference Manual Version 2.0.6*, Interface21 Ltd., 2007.

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

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

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

[10] R. Mond´ejar, P. Garc´ıa, C. Pairot, and A. Skarmeta. "Building a distributed AOP middleware for large scale systems," *In Proceedings of the 2008 Workshop on Next Generation Aspect Oriented Middleware (NAOMI 08)*, New York, USA, April 2008, pp. 17–22.

[11] É. Tanter and R. Toledo. "A Versatile Kernel for Distributed AOP," *In Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2006)*, Bologna, Italy, June 2006, pp. 316–331.

[12] M. Gursky, "Some complexity results for a multi-processor scheduling problem," Private Communication from H. S. Stone, 1981.

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

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

[15] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. On Computers*, Vol. 37, No. 11, 1988.

[16] B. Shirazi, M. Wang and G. Pathak, "Analysis and evaluation of heuristic methods for static task scheduling," *J. Parallel Distrib. Comp.*, Vol. 10, 1990, pp. 222-232.

[17] S.S. Wu and D. Sweeping, "Heuristic Algorithms for Task Assignment and Scheduling in a Processor Network," *Parallel Computing*, Vol. 20, 1994, pp. 1-14.

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

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

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

[21] Obtaining crosscutting relationship information from AJDT, http://wiki.eclipse.org/Developer's_guide_to_ building_tools_on_top_of_AJDT_and_AspectJ

[22] http://web.itu.edu.tr/bulusa/DS/