Resource Allocation and Scheduling of Multiple Composite Web Services in Cloud Computing Using Cooperative Coevolution

Lifeng Ai[1,2], Maolin Tang[1] and Colin Fidge[1]

[1]Queensland University of Technology, 2 George Street, Brisbane, 4001, Australia
[2]Vancl Research Laboratory, 59 Middle East 3rd Ring Road, Beijing, 100022, China
{l.ai,m.tang,c.fidge}@qut.edu.au

Abstract. In cloud computing, resource allocation and scheduling of multiple composite web services is an important and challenging problem. This is especially so in a hybrid cloud where there may be some low-cost resources available from private clouds and some high-cost resources from public clouds. Meeting this challenge involves two classical computational problems: one is assigning resources to each of the tasks in the composite web services; the other is scheduling the allocated resources when each resource may be used by multiple tasks at different points of time. In addition, Quality-of-Service (QoS) issues, such as execution time and running costs, must be considered in the resource allocation and scheduling problem. Here we present a Cooperative Coevolutionary Genetic Algorithm (CCGA) to solve the deadline-constrained resource allocation and scheduling problem for multiple composite web services. Experimental results show that our CCGA is both efficient and scalable.

Key words: Cooperative coevolution, web service, cloud computing

1 Introduction

Cloud computing is a new Internet-based computing paradigm whereby a pool of computational resources, deployed as web services, are provided on demand over the Internet, in the same manner as public utilities. Recently, cloud computing has become popular because it brings many cost and efficiency benefits to enterprises when they build their own web service-based applications.

When an enterprise builds a new web service-based application, it can use published web services in both *private clouds* and *public clouds*, rather than developing them from scratch. In this paper, private clouds refer to internal data centres owned by an enterprise, and public clouds refer to public data centres that are accessible to the public. A composite web service built by an enterprise is usually composed of multiple component web services, some of which may be provided by the private cloud of the enterprise itself and others which may be provided in a public cloud maintained by an external supplier. Such a computing environment is called a *hybrid cloud*.

The component web service *allocation problem* of interest here is based on the following assumptions. Component web services provided by private and public clouds may have the same functionality, but different Quality-of-Service (QoS) values, such as response time and cost. In addition, in a private cloud a component web service may have a limited number of instances, each of which may have different QoS values. In public clouds, with greater computational resources at their disposal, a component web service may have a large number of instances, with identical QoS values. However, the QoS values of service instances in different public clouds may vary. There may be many composite web services in an enterprise. Each of the tasks comprising a composite web service needs to be allocated an instance of a component web service. A single instance of a component web service may be allocated to more than one task in a set of composite web services, as long as it is used at different points of time.

In addition, we are concerned with the component web service *scheduling problem*. In order to maximise the utilisation of available component web services in private clouds, and minimise the cost of using component web services in public clouds, allocated component web service instances should only be used for a short period of time. This requires scheduling the allocated component web service instances efficiently.

There are two typical QoS-based component web service allocation and scheduling problems in cloud computing. One is the *deadline-constrained* resource allocation and scheduling problem, which involves finding a cloud service allocation and scheduling plan that minimises the total cost of the composite web service, while satisfying given response time constraints for each of the composite web services. The other is the *cost-constrained* resource allocation and scheduling problem, which requires finding a cloud service allocation and scheduling plan which minimises the total response times of all the composite web services, while satisfying a total cost constraint.

In previous work [1], we presented a random-key genetic algorithm (RGA) [2] for the constrained resource allocation and scheduling problems and used experimental results to show that our RGA was scalable and could find an acceptable, but not necessarily optimal, solution for all the problems tested. In this paper we aim to improve the quality of the solutions found by applying a cooperative coevolutionary genetic algorithm (CCGA) [3–5] to the deadline-constrained resource allocation and scheduling problem.

2 Problem Definition

Based on the requirements introduced in the previous section, the *deadline-constrained resource allocation and scheduling problem* can be formulated as follows.

Inputs

1. A set of composite web services $W = \{W_1, W_2, \ldots, W_n\}$, where n is the number of composite web services. Each composite web service consists of

several abstract web services. We define $O_i = \{o_{i,1}, o_{i,2}, \ldots, o_{i,n_i}\}$ as the abstract web services set for composite web service W_i, where n_i is the number of abstract web services contained in composite web service W_i.
2. A set of candidate cloud services $S_{i,j}$ for each abstract web service $o_{i,j}$, where $S_{i,j} = S^u_{i,j} \cup S^v_{i,j}$, and $S^v_{i,j} = \{S^v_{i,j,1}, S^v_{i,j,2}, \ldots, S^v_{i,j,\ell}\}$ denotes an entire set of ℓ private cloud service candidates for abstract web service $o_{i,j}$, and $S^u_{i,j} = \{S^u_{i,j,1}, S^u_{i,j,2}, \ldots, S^u_{i,j,m}\}$ denotes an entire set of m public cloud service candidates for abstract web service $o_{i,j}$.
3. A response time and price for each public cloud service $S^u_{i,j,k}$, denoted by $t^u_{i,j,k}$ and $c^u_{i,j,k}$ respectively, and a response time and price for each private cloud service $S^v_{i,j,k}$, denoted by $t^v_{i,j,k}$ and $c^v_{i,j,k}$ respectively.

Output

1. An allocation and scheduling plan $X = \{X_i \mid i = 1, 2, \ldots, n\}$, such that the total cost of X, i.e., $Cost(X) = \sum_{i=1}^{n} \sum_{j=1}^{n_i} Cost(M_{i,j})$, is minimal, where $X_i = \{(M_{i,1}, F_{i,1}), (M_{i,2}, F_{i,2}), \ldots, (M_{i,n_i}, F_{i,n_i})\}$ denotes an allocation and scheduling plan for composite web service W_i, $M_{i,j}$ represents the selected cloud service for abstract web service $o_{i,j}$, and $F_{i,j}$ stands for the finishing time of $M_{i,j}$.

Constraints

1. All the finishing-time precedence requirements between the abstract web services are satisfied, that is, $F_{i,k} \leq F_{i,j} - d_{i,j}$, where $j = 1, \ldots, n_i$, and $k \in Pre_{i,j}$, where $Pre_{i,j}$ denotes the set of all abstract web services that must execute before the abstract web service $o_{i,j}$.
2. All the resource limitations are respected, that is, $\sum_{j \in A(t)} r_{j,m} \leq 1$, where $m \in S^v_{i,j}$ and $A(t)$ denotes the entire set of abstract web services being used at time t. Let $r_{j,m} = 1$ if abstract web service j requires private cloud service m in order to execute and $r_{j,m} = 0$ otherwise. This constraint guarantees that each private cloud service can only serve at most one abstract web service at a time.
3. The deadline constraint for each composite web service is satisfied, that is, $F_{i,n_i} \leq d_i$, such that $i = 1, \ldots, n$, where d_i denotes the deadline promised to the customer for composite web service W_i, and F_{i,n_i} is the finishing time of the last abstract service of composite web service W_i, that is, the overall execution time of the composite web service W_i.

3 A Cooperative Coevolutionary Genetic Algorithm

Our Cooperative Coevolutionary Genetic Algorithm is based on Potter and De Jong's model [3]. In their approach several *species*, or *subpopulations*, coevolve together. Each individual in a subpopulation constitutes a partial solution to the problem, and the combination of an individual from all the subpopulations forms a complete solution to the problem. The subpopulations of the CCGA

evolve independently in order to improve the individuals. Periodically, they interact with each other to acquire feedback on how well they are cooperatively solving the problem. In order to use the cooperative coevolutionary model, two major issues must be addressed, *problem decomposition* and *interaction between subpopulations*, which are discussed in detail below.

3.1 Problem Decomposition

Problem composition can be either static, where the entire problem is partitioned in advance and the number of subpopulations is fixed, or dynamic, where the number of subpopulations is adjusted during the calculation time. Since the problem studied here can be naturally decomposed into a fixed number of subproblems beforehand, the problem decomposition adopted by our CCGA is static.

Essentially our problem is to find a resource allocation scheduling solution for multiple composite web services. Thus, we define the problem of finding a resource allocation and scheduling solution for each of the composite web services as a subproblem. Therefore, the CCGA has n subpopulations, where n is the total number of composite web services involved. Each subpopulation is responsible for solving one subproblem and the n subpopulations interact with each other as the n composite web services compete for resources.

3.2 Interaction Between Subpopulations

In our Cooperative Coevolutionary Genetic Algorithm, interactions between subpopulations occur when evaluating the fitness of an individual in a subpopulation. The fitness value of a particular individual in a population is an estimate of how well it cooperates with other species to produce good solutions. Guided by the fitness value, subpopulations work cooperatively to solve the problem. This interaction between the sub-populations involves the following two issues.

1. *Collaborator selection*, i.e., selecting collaborator subcomponents from each of the other subpopulations, and assembling the subcomponents with the current individual being evaluated to form a complete solution. There are many ways of selecting collaborators [6]. In our CCGA, we use the most popular one, choosing the best individuals from the other subpopulations, and combine them with the current individual to form a complete solution. This is the so-called *greedy collaborator selection method* [6].
2. *Credit assignment*, i.e., assigning credit to the individual. This is based on the principle that the higher the fitness value the complete solution has—constructed by the above collaborator selection method—the more credit the individual will obtain. The fitness function is defined by Equations 1 to 3 below. By doing so, in the following evolving rounds, an individual resulting in better cooperation with its collaborators will be more likely to survive. In other words, this credit assignment method can enforce the evolution of each population towards a better direction for solving the problem.

$$Fitness(X) = \begin{cases} F_{Max}^{Cost}/F_{obj}(X), \text{ if } V(X) \leq 1; \\ 1/V(X), \qquad \text{otherwise.} \end{cases} \tag{1}$$

$$V(X) = \prod_{i=1}^{n}(V_i(X)) \tag{2}$$

$$V_i(X) = \begin{cases} F_{i,n_i}/d_i, \text{ if } F_{i,n_i} > d_i; \\ 1, \qquad \text{otherwise.} \end{cases} \tag{3}$$

In Equation 1, condition $V(X) \leq 1$ means there is no constraint violation. Conversely, $V(X) > 1$ means some constraints are violated, and the larger the value of $V(X)$, the higher the degree of constraint violation. F_{Max}^{Cost} is the worst $F_{obj}(X)$, namely the maximal total cost, among all feasible individuals in a current generation. Ratio $F_{Max}^{Cost}/F_{obj}(X)$ is used to scale the fitness value of all feasible solutions into range $[1, \infty)$. Using Equations 1 to 3, we can guarantee that the fitness of all feasible solutions in a generation are better than the fitness of all infeasible solutions. In addition, the lower the total cost for a feasible solution, the better fitness the solution will have. The higher number of constraints that are violated by an infeasible solution, the worse fitness the solution will have.

3.3 Algorithm Description

Algorithm 1 summarises our Cooperative Coevolutionary Genetic Algorithm. Step 1 initialises all the subpopulations. Steps 2 to 7 evaluate the fitness of each individual in the initial subpopulations. This is done in two steps. The first step combines the individual $indiv[i][j]$ ($indiv[i][j]$ denotes the j th individual in the ith subpopulation in the CCGA) with the jth individual from each of the other subpopulations to form a complete solution c to the problem, and the second step calculates the fitness value of the solution c using the fitness function defined by Equation 1.

Steps 8 to 18 are the co-evolution rounds for the N subpopulations. In each round, the N subpopulations evolve one by one from the 1st to the Nth. When evolving a subpopulation $SubPop[i]$, where $1 \leq i \leq N$, we use the same selection, crossover and mutation operators as used in our previously-described random-key genetic algorithm (RGA) [1]. However, the fitness evaluation used in the CCGA is different from that used in the RGA. In the CCGA, we use the aforementioned collaborator selection strategy and the credit assignment method to evaluate the fitness of an individual. The cooperative co-evolution process is repeated until certain termination criteria are satisfied, specific to the application (e.g., a certain number of rounds or a fixed time limit).

4 Experimental Results

Experiments were conducted to evaluate the scalability and effectiveness of our CCGA for the resource allocation and scheduling problem by comparing it with

Algorithm 1: Our cooperative coevolutionary genetic algorithm

```
1  Construct N sets of initial populations, SubPop[i], i = 1, 2, ..., N
2  for i ← 1 to N do
3  |  foreach individual indiv[i][j] of the subpopulation SubPop[i] do
4  |  |  c ← SelectPartnersBySamePosition(j)
5  |  |  indiv[i][j].Fitness ← FitnessFunc (c)
6  |  end
7  end
8  while termination condition is not true do
9  |  for i ← 1 to N do
10 |  |  Select fit individuals in SubPop[i] for reproduction
11 |  |  Apply the crossover operator to generate new offspring for SubPop[i]
12 |  |  apply the mutation operator to offspring
13 |  |  foreach individual indiv[i][j] of the subpopulation SubPop[i] do
14 |  |  |  c ← SelectPartnersByBestFitness
15 |  |  |  indiv[i][j].Fitness ← FitnessFunc (c)
16 |  |  end
17 |  end
18 end
```

our previous RGA [1]. Both algorithms were implemented in Microsoft Visual C♯, and the experiments were conducted on a desktop computer with a 2.33 GHz Intel Core 2 Duo CPU and a 1.95 GB RAM. The population sizes of the RGA and the CCGA were 200 and 100, respectively. The probabilities for crossover and mutation in both the RGA and the CCGA were 0.85 and 0.15, respectively. The termination condition used in the RGA was "no improvement in 40 consecutive generations", while the termination condition used in the CCGA was "no improvement in 20 consecutive generations". These parameters were obtained through trials on randomly generated test problems. The parameters that led to the best performance in the trials were selected.

The scalability and effectiveness of the CCGA and RGA were tested on a number of problem instances with different sizes. Problem size is determined by three factors: the number of composite web services involved in the problem, the number of abstract web services in each composite web service, and the number of candidate cloud services for each abstract service. We constructed three types of problems, each designed to evaluate how one of the three factors affects the computation time and solution quality of the algorithms.

4.1 Experiments on the Number of Composite Web Services

This experiment evaluated how the number of composite web services affects the computation time and solution quality of the algorithms. In this experiment, we also compared the algorithms' convergence speeds. Considering the stochastic nature of the two algorithms, we ran both ten times on each of the randomly

generated test problems with a different number of composite web services. In this experiment, the number of composite web services in the test problems ranged from 5 to 25 with an increment of 5. The deadline constraints for the five test problems were 59.4, 58.5, 58.8, 59.2 and 59.8 minutes, respectively. Because of space limitations, the five test problems are not given in this paper, but they can be found elsewhere [1].

The experimental results are presented in Table 1. It can be seen that both algorithms always found a feasible solution to each of the test problems, but that the solutions found by the CCGA are consistently better than those found by the RGA. For example, for the test problem with five composite web services, the average cost of the solutions found by the RGA of ten times run was $103, while the average cost of the solutions found by the CCGA was only $79. Thus, $24 can be saved by using the CCGA on average.

Table 1. Comparison of the algorithms with different numbers of composite web services

No. of Composite Web Services	RGA		CCGA	
	Feasible Solution	Aver. Cost ($)	Feasible Solution	Ave. Cost ($)
5	Yes	103	Yes	79
10	Yes	171	Yes	129
15	Yes	326	Yes	251
20	Yes	486	Yes	311
25	Yes	557	Yes	400

The computation time of the two algorithms as the number of composite web services increases is shown in Figure 1. The computation time of the RGA increased close to linearly from 25.4 to 226.9 seconds, while the computation time of the CCGA increased super-linearly from 6.8 to 261.5 seconds as the number of composite web services increased from 5 to 25. Although the CCGA is not as scalable as the RGA there is little overall difference between the two algorithms for problems of this size, and a single web service would not normally comprise very large numbers of components.

4.2 Experiments on the Number of Abstract Web Services

This experiment evaluated how the number of abstract web services in each composite web service affects the computation time and solution quality of the algorithms. In this experiment, we randomly generated five test problems. The number of abstract web services in the five test problems ranged from 5 to 25 with an increment of 5. The deadline constraints for the test problems were 26.8, 59.1, 89.8, 117.6 and 153.1 minutes, respectively. The quality of the solutions found by the two algorithms for each of the test problems is shown in Table 2. Once again both algorithms always found feasible solutions, and the CCGA always found better solutions than the RGA.

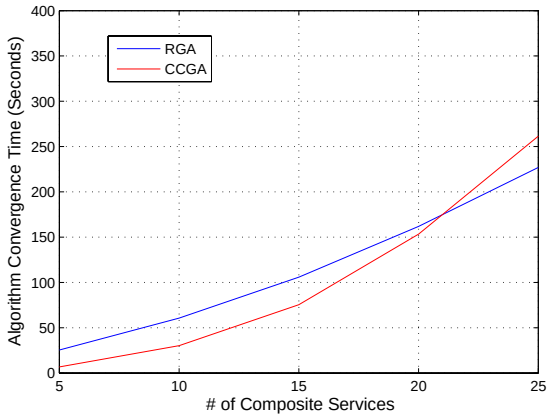

Fig. 1. Number of composite web services versus computation time for both algorithms

Table 2. Comparison of the algorithms with different numbers of abstract web services

No. of	RGA		CCGA	
Abstract Services	Feasible Solution	Ave. Cost ($)	Feasible Solution	Ave. Cost ($)
5	Yes	105	Yes	81
10	Yes	220	Yes	145
15	Yes	336	Yes	259
20	Yes	458	Yes	322
25	Yes	604	Yes	463

The computation times of the two algorithms as the number of abstract web services involved in each composite web service increases are displayed in Figure 2. The Random-key GA's computation time increased linearly from 29.8 to 152.3 seconds and the Cooperative Coevolutionary GA's computation time increased linearly from 14.8 to 72.1 seconds as the number of abstract web services involved in the each composite web service grew from 5 to 25. On this occasion the CCGA clearly outperformed the RGA.

4.3 Experiments on the Number of Candidate Cloud Services

This experiment examined how the number of candidate cloud services for each of the abstract web services affects the computation time and solution quality of the algorithms. In this experiment, we randomly generated five test problems. The number of candidate cloud services in the five test problems ranged from 5 to 25 with an increment of 5, and the deadline constraints for the test problems were 26.8, 26.8, 26.8, 26.8 and 26.8 minutes, respectively. Table 3 shows that yet again both algorithms always found feasible solutions, with those produced by the CCGA being better than those produced by the RGA.

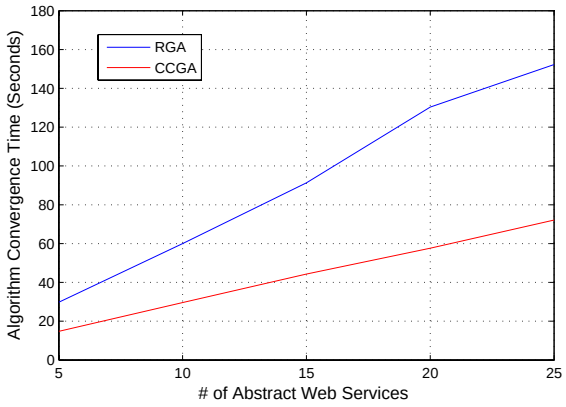

Fig. 2. Number of abstract web services versus computation time for both algorithms

Table 3. Comparison of the algorithms with different numbers of candidate cloud services for each abstract service

No. of Candidate	RGA		CCGA	
Web Services	Feasible Solution	Ave. Cost ($)	Feasible Solution	Ave. Cost ($)
5	Yes	144	Yes	130
10	Yes	142	Yes	131
15	Yes	140	Yes	130
20	Yes	141	Yes	130
25	Yes	142	Yes	130

Figure 3 shows the relationship between the number of candidate cloud services for each abstract web service and the algorithms' computation times. Increasing the number of candidate cloud services had no significant effect on either algorithm, and the computation time of the CCGA was again much better than that of the RGA.

5 Conclusion and Future Work

We have presented a Cooperative Coevolutionary Genetic Algorithm which solves the deadline-constrained cloud service allocation and scheduling problem for multiple composite web services on hybrid clouds. To evaluate the efficiency and scalability of the algorithm, we implemented it and compared it with our previously-published Random-key Genetic Algorithm for the same problem. Experimental results showed that the CCGA always found better solutions than the RGA, and that the CCGA scaled up well when the problem size increased.

The performance of the new algorithm depends on the collaborator selection strategy and the credit assignment method used. Therefore, in future work we

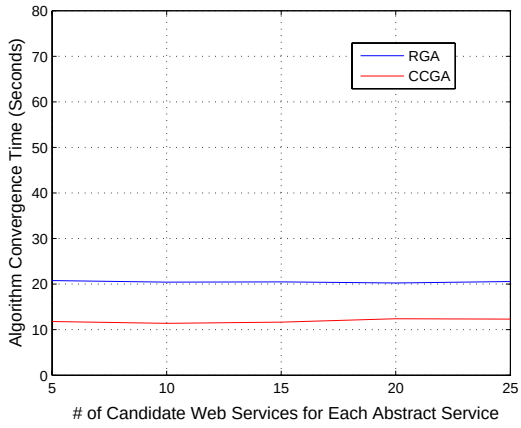

Fig. 3. Number of candidate cloud services versus computation time for both algorithms

will look at alternative collaborator selection and credit assignment methods to further improve the performance of the algorithm.

Acknowledgement

This research was carried out as part of the activities of, and funded by, the Cooperative Research Centre for Spatial Information (CRC-SI) through the Australian Government's CRC Programme (Department of Innovation, Industry, Science and Research).

References

1. Ai, L., Tang, M., Fidge, C.: QoS-oriented resource allocation and scheduling of multiple composite web services in a hybrid cloud using a random-key genetic algorithm. Australian Journal of Intelligent Information Processing Systems **12**(1) (2010) 29–34
2. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing **6**(2) (1994) 154–160
3. Potter, M.A., De Jong, K.A.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evolutionary Computation **8**(1) (2000) 1–29
4. Ray, T., Yao, X.: A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning. In: Proceeding of IEEE Congress on Evolutionary Computation. (2009) 983–989
5. Yang, Z., Tang, K., Yao, X.: Large scale evolutionary optimization using cooperative coevolution. Information Sciences **178**(15) (2008) 2985 – 2999
6. Wiegand, R.P., Liles, W.C., De Jong, K.A.: An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference. (2001) 1235–1242