QUT Digital Repository:
http://eprints.qut.edu.au/

This is the accepted version of this conference paper. To be published as:

Ai, Lifeng and Tang, Maolin and Fidge, Colin J. (2010) *QoS-oriented sesource allocation and scheduling of multiple composite web services in a hybrid cloud using a random-key genetic algorithm.* In: 17th International Conference on Neural Information Processing (ICONIP 2010), 22-25 November 2010, Sydney. (In Press)

© Copyright 2010 Please consult the authors.

QoS-oriented Resource Allocation and Scheduling of Multiple Composite Web Services in a Hybrid Cloud Using a Random-Key Genetic Algorithm

Lifeng Ai, Maolin Tang and Colin Fidge

Queensland University of Technology
2 George Street, Brisbane, QLD 4001, Australia
{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 challenge. This is especially so in a hybrid cloud where there may be some free resources available from private clouds but some fee-paying 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 service. The other is scheduling the allocated resources when each resource may be used by more than one task and may be needed at different points of time. In addition, we must consider Quality-of-Service issues, such as execution time and running costs. Existing approaches to resource allocation and scheduling in public clouds and grid computing are not applicable to this new problem. This paper presents a random-key genetic algorithm that solves new resource allocation and scheduling problem. Experimental results demonstrate the effectiveness and scalability of the algorithm.

1 Introduction

"Cloud computing" [1] is an Internet-based computing paradigm driven by economies of scale, whereby a pool of computation resources, usually deployed as web services, are provided on demand over the Internet, in the same manner as public utilities. This paradigm provides an economical way for an enterprise to build new composite web services.

A composite web service is composed of two or more component web services, which can be provided by the private cloud owned by the enterprise, a number of public clouds, or both. Such a computing environment is called a *hybrid cloud* in this paper and is illustrated in Figure 1.

Component web services provided by the private cloud and the public clouds have the same functionality, but may have different Quality-of-Service (QoS) values, such as response time. In addition, in the private cloud a component web service may have a limited number of instances of the same component web services, each of which may have different QoS values. In public clouds, a component web service has a large number of instances of the same component web service. The QoS values of the instances of the same component web services in the same public cloud are identical. However, the QoS values of instances in different public clouds may vary.

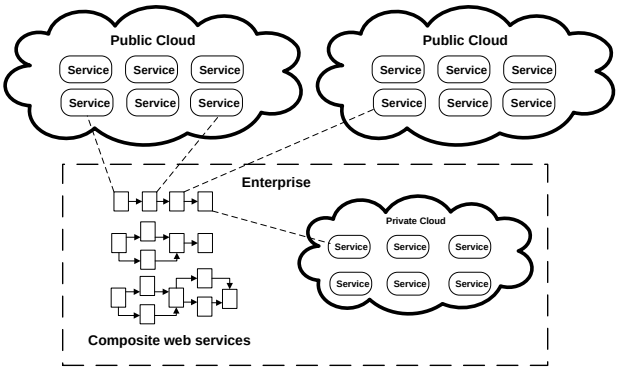

Fig. 1. Hybrid cloud

There may be many composite web services in an enterprise. Each of the component web services in the composite web services needs to be allocated an instance of the component web service. One instance of a component web service may be allocated to more than one component web service in the composite web services as long as it is used at different times. This is the so called component web service allocation problem.

In addition, in order to maximize the usability of the component web services in the private cloud and minimize the cost of using component web services in public clouds, allocated component web service instances can only be used for a fixed period of time. Thus, this involves scheduling those allocated component web service instances. This is the so called component web service scheduling problem.

There are two typical QoS-oriented component web service allocation and scheduling problems in the hybrid cloud. One is finding a resource allocation and scheduling plan which can minimize the total cost of the composite web services satisfying given response time constraints on each of the composite web services, called the *deadline constraint problem* in this paper; another is finding a resource allocation and scheduling plan which minimizes the total response times of the composite web services satisfying a total cost constraint, called the *cost constraint problem*. These two component web service allocation and scheduling problems can be categorized into resource allocation and scheduling problems. Although resource allocation and scheduling problems of various kinds have been studied in the context of public clouds and in grids [2–4], the two component web service allocation and scheduling problems are not the same, and existing solutions to those resource allocation and scheduling problems in public clouds and grids cannot be used immediately. This paper presents a random-key genetic algorithm which solves this new resource allocation and scheduling problem.

2 Problem Description

Inputs

1. A set of composite services $W = \{W_1, W_2, \ldots, W_n\}$ modelled by directed acyclic graphs (DAGs), where n is the number of composite services. Each composite 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 service W_i, where n_i is the number of abstract web services contained in composite service W_i.
2. A set of candidate cloud services $S_{i,j}$ for each abstract web service $o_{i,j}$, such that $S_{i,j} = S^u_{i,j} \cup S^v_{i,j}$. Here, $S^v_{i,j}$ denotes the set of candidate private cloud services for abstract web service $o_{i,j}$, such that $S^v_{i,j} = (S^v_{i,j,1}, S^v_{i,j,2}, \ldots, S^v_{i,j,\ell})$, where ℓ is the number of candidate private cloud services. Set $S^u_{i,j}$ denotes the candidate public cloud services for abstract web service $o_{i,j}$, such that $S^u_{i,j} = (S^u_{i,j,1}, S^u_{i,j,2}, \ldots, S^u_{i,j,m})$, where m is the number of candidate public cloud services.
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.

The deadline constraint problem

Output An allocation and scheduling plan $X = \{X_i \mid i = 1, 2, \ldots, n\}$, such that the total cost $Cost(X)$ is minimal. Let X_i denote an allocation and scheduling plan for composite service W_i, such that $X_i = ((M_{i,1}, F_{i,1}), (M_{i,2}, F_{i,2}), \ldots, (M_{i,n_i}, F_{i,n_i}))$, where $M_{i,j}$ represents the selected cloud service for abstract web service $o_{i,j}$ and $F_{i,j}$ represents the finishing time of an instance of abstract web service $o_{i,j}$. Equation 1 gives the definition of total cost for a composite service.

$$Cost(X) = \sum_{i=1}^{n} \sum_{j=1}^{n_i} Cost(M_{i,j}) \tag{1}$$

Constraints Provision of web services is subject to both precedence and resourcing constraints, and all composite services are subject to deadline constraints.

$$F_{i,k} \leq F_{i,j} - d_{i,j}, \text{ where } j = 1, \ldots, n_i \text{ and } k \in Pre_{i,j} \tag{2}$$

$$\sum_{j \in A(t)} r_{j,m} \leq 1, \text{ where } m \in S^v_{i,j} \tag{3}$$

$$F_{i,n_i} \leq d_i \tag{4}$$

In Equation 2, set $Pre_{i,j}$ denotes all abstract web services which must execute before the abstract web service $o_{i,j}$. Equation 2 imposes the precedence relationships between abstract web services.

In Equation 3, set $A(t)$ denotes the abstract web services being used at time t. We let $r_{j,m} = 1$ if abstract web service j requires private cloud service m to be processed and $r_{j,m} = 0$ otherwise. Equation 3 states that each private cloud service can only process one abstract web service at a time.

In Equation 4, d_i denotes the deadline promised to the customer for composite service W_i. F_{i,n_i} is the finishing time of the last abstract service of composite service W_i, that is, the execution time of W_i. Equation 4 imposes the deadline constraint for each composite service.

The cost constraint problem

Output An allocation and scheduling plan $X = \{X_i \mid i = 1, 2, \ldots, n\}$, such that the total response time $Time(X)$ is minimal.

$$Time(X) = \sum_{i=1}^{n} (F_{i,n_i}) \tag{5}$$

Here, Equation 5 gives the definition of total response times for all composite services.

Constraints Provision of web services is subject to both precedence (defined by Equation 2) and resourcing constraints (defined by Equation 3), and the cost is subject to a total cost constraint defined by Equation 6, in which $Cost_{constraint}$ denotes the total cost constraint.

$$Cost(X) \leq Cost_{constraint} \tag{6}$$

3 A Random-Key Genetic Algorithm for Hybrid Cloud Services

We use a random-key genetic algorithm (GA) to solve the hybrid cloud allocation and scheduling problem. Its main features include: 1) a random-key representation to facilitate the maintaining of individuals' feasibility, which is considered to be difficult in a sequencing problem as studied here; 2) and a modified decoding procedure based on the traditional decoding procedure [5] to handle unconstrained resources, i.e., the public cloud services, so as to make more efficient use of them. In the following parts of this section, we introduce our GA in detail.

3.1 Random-Key Genetic Encoding

We encode each individual (or chromosome) as a list of N real numbers, where N is the number of abstract web services involved in all composite web services. Each gene in the chromosome corresponds to an abstract web service. The value of a gene consists of two parts: 1) an integral part, indicating which service is assigned to the abstract web service, among a set of candidate web services that could be used by the abstract web service, and 2) a fraction generated randomly in range $[0, 1)$, used as a sort key, determining the order of the web service in the schedule. The sort key of a gene represents its corresponding abstract web service's priority to occupy a cloud service, when there are multiple abstract web services competing for the same cloud service. The abstract web service with the lowest priority value has the highest priority to occupy the cloud service among all competing services.

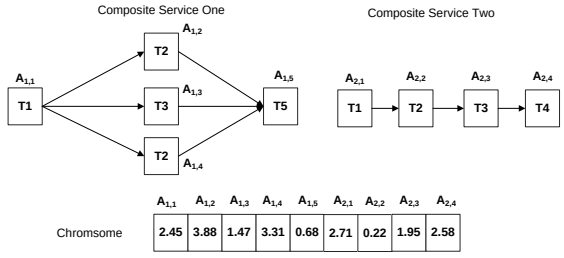

Fig. 2. Example of our random-key encoding method

In on our method, a problem with two composite service (see Figure 2, in which one composite service has five abstract services and the other has four abstract services) can be encoded as a real number list with nine (the number of abstract web services). In the sample chromosome, the value of gene $A_{1,1}$ is 2.45, the integer part of which means that abstract web service $A_{1,1}$ uses the cloud service with index 2 in the set of abstract web service $A_{1,1}$'s candidate cloud services, and the fractional part means that the priority of abstract web service $A_{1,1}$ to use the cloud service is 0.45. Abstract web service $A_{2,1}$ has the same functionality (namely task $T1$) as abstract web service $A_{1,1}$ but has gene value 2.71, which means it also uses cloud service 2 and its priority is 0.71. According to this scenario, abstract web service $A_{1,1}$ will use the cloud service first when both of them want to use it at the same time.

3.2 Decoding

We modified the classical decoding algorithm proposed by Bierwirth and Mattfeld [5] to sequence the abstract web services which use public cloud services. This variation is imperative because existing algorithms cannot cope with public cloud services, which have no well-defined bounds, unlike limited private cloud services. A common assumption made by existing algorithms is that a service (resource) can only be used by one abstract web service at any time. For a public cloud service, however, this is not the case, because a 'single' service can be used by multiple abstract web services at the same time thanks to the virualisation technology used by public cloud service providers.

The notations used in the procedure include: 1) o_{ij}, the j^{th} abstract service of composite service i; 2) σ_{ij}, the earliest starting time of abstract service o_{ij}; 3) p_{ij}, the processing time of abstract service o_{ij}; 4) and τ_{ij}, the earliest completion time of abstract service o_{ij}.

1. Construct the set of all abstract services that are ready to be scheduled, $A := \{o_{ij} \mid 1 \leq i \leq N\}$.
2. If there exist abstract services in A using public cloud services:
   (a) Randomly select an abstract service o^*_{ij} from A which uses a public cloud service.
   (b) Append abstract service o^*_{ij} to the schedule and calculate its starting and completion time.

(c) If a successor o^*_{ij+1} of the abstract service o^*_{ij} exists, insert it into A, i.e., $A := A \cup \{o^*_{ij+1}\}$.

3. Determine $\tau^* = min\{\tau_{ij} | o_{ij} \in A\}$ and the cloud service m^* on which τ^* could be realised.
4. Build a set B from all abstract services in A which are processed on m^* and then determine $\sigma^* = min\{\sigma_{ij} \mid o_{ij} \in B\}$.
5. Build a set C in accordance with parameter θ such that $C := \{o_{ij} \in B \mid \sigma_{ij} \leq \theta\tau^* + (1-\theta)\sigma^*, 0 \leq \theta \leq 1\}$.
6. Select the abstract service o^*_{ij} from C which has the highest priority (*determined by the sort keys of current individual being decoded*) among abstract services in C and delete it from A, i.e., $A := A - \{o^*_{ij}\}$.
7. Append abstract service o^*_{ij} to the schedule and calculate its starting and completion time.
8. If a successor o^*_{ij+1} of the abstract service o^*_{ij} exists, insert it into A, i.e., $A := A \cup \{o^*_{ij+1}\}$
9. If $A \neq \phi$, go to Step 2. Otherwise, terminate.

A major difference between our algorithm and existing ones is the introduction of an extra step (Step 2), which processes abstract web services consuming public cloud services. The basic idea behind our algorithm is that we can schedule abstract services which use public clouds in the set of abstract services which are ready to be processed (namely the abstract services in A), because scheduling public abstract services will not delay other abstract services. After these abstract services have been scheduled, the remaining abstract services in A can be scheduled in the usual way.

3.3 Fitness Function

For the deadline constraint problem, the fitness function is defined by Equation 7.

$$Fitness(X) = \begin{cases} F^{Cost}_{Max}/Cost(X), & \text{if } V(X) \leq 1; \\ 1/V(X), & \text{otherwise.} \end{cases} \tag{7}$$

$$V(X) = \prod_{i=1}^{n}(V_i) \tag{8}$$

$$V_i = \begin{cases} F_{i,n_i}/d_i, & \text{if } F_{i,n_i} > d_i; \\ 1, & \text{otherwise.} \end{cases} \tag{9}$$

In Equation 7, $V(X)$ denotes the total constraint violation degree for composite services. $V(X) \leq 1$ denotes there is no constraint violation. $V(X) > 1$ denotes some constraints are violated, and the larger the value of $V(X)$, the higher the degree of constraint violation. F^{Time}_{Max} is the maximal total cost $F_{obj}(X)$ among all feasible individuals in current generation. $F^{Cost}_{Max}/F_{obj}(X)$ is used to scale the fitness value of all feasible solutions in range $[1, \infty)$. Using Equation 7, 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 total cost for a solution, the better fitness the solution will have.

For the cost constraint problem, the fitness function is defined by Equation 10, in which F_{Max}^{Time} denotes the maximal total response time $F_{obj}(X)$ among all feasible individuals in the current generation.

$$Fitness(X) = \begin{cases} F_{Max}^{Time}/Time(X), & \text{if } V(X) \leq 1; \\ 1/V(X), & \text{otherwise.} \end{cases} \tag{10}$$

$$V(X) = \begin{cases} 1, & \text{if } Cost(X) \leq Cost_{constraint}; \\ Cost_{constraint}/Cost(X), & \text{otherwise.} \end{cases} \tag{11}$$

3.4 Genetic Operators and Elitist Selection

Our algorithm adopts the uniform crossover. The mutation is used in a broader sense than usual — we introduce a small number of randomly generated new individuals into the next generation, rather than conducting gene-by-gene mutation as is done usually. Elitist selection that preserves the best certain number of individuals is also adopted to increase performance.

4 Experimental Results

Simulation experiments were conducted to evaluate the effectiveness and scalability of our algorithm. The experimental settings for our GA are as follows: both the population size and the maximal generations were 200; the percent of the elitist selection was 2%; 83% of new individuals in the next generation are generated by crossover operator; 15% of new individuals in the next generation are randomly generated (a broad sense of mutation). These parameters were obtained through doing trials on randomly generated test problems.

To evaluate the scalability of our GA, we applied it to a number of test problems with different sizes. The problem size of a multiple composite service resource allocation and scheduling problem depends on three factors: the number of composite web services, the total number of abstract services, and the number of candidate services for each abstract service. Therefore we construct three sets of problem, each of which was designed to evaluate how one of the three factors would affect the computation time of the GA.

4.1 Test example one

We constructed five test problems with different numbers of composite web services, ranging from 5 to 25. In each problem, all composite web services contain 10 abstract web services, with 5 candidate cloud web services for each abstract web services. For each test problem, we set a deadline constraint for each composite services involved in the problem to construct a deadline constraint problem. A deadline constraint D using the method proposed by Yu and Buyya [3] where $D = (D_{max} + D_{min})/2$. Here, D_{max} is the largest average execution time for a composite service, obtained using the cost

optimisation algorithm [3] to solve the cost optimisation problem without deadline constraint. D_{min} is the smallest average execution time for a composite service, obtained using the execution time optimisation algorithm [3] for the time optimisation problem without cost constraint. Considering the stochastic nature of GAs, we ran our algorithm 10 times for each of the test problems. In each run, our algorithm always found a feasible solution. How the number of composite services affected the computation time of our GA is depicted by Plot 1 of Figure 3. It can be seen that the computation time of our GA increased from 25.57 seconds to 205.31 seconds when the number of composite web services increased from 5 to 25. The computation time increases almost linearly as the number of composite web services increases.

We also use the test problems to test the cost constraint problem. The cost constraint was set using the cost optimisation algorithm [3] and the execution time optimisation algorithm [3] for the cost optimisation problem without deadline constraint and time optimisation problem without cost constraint, respectively, equals the average value of the total cost for all composite services found by the two algorithms. Our algorithm also found feasible solutions for these problems, and the computation tread is the same with the tread of the algorithm for deadline constraint problem.

4.2 Test example two

We used five test problems with different numbers of abstract services in each composite service, ranging from 5 to 25 in different tests. Each test has 10 composite web services. The number of candidate cloud web services for each abstract web services is also 5. The deadline constraint and cost constraint were set in the same way just mentioned. The experimental results for different numbers of abstract web services is shown in Plot2. From the plot we can see that the computation time of our genetic algorithm increased from 29.12 seconds to 148.15, as the number of abstract services in each composite service increased from 5 to 25. The computation time increase trend is close to linear. For all these deadline and cost constraint problems, our GA always found a feasible solution.

4.3 Test example three

Five test problems were used with different numbers of candidate cloud services for each abstract service, ranging from 5 to 25 in different tests. The number of composite and abstract web services was fixed. Plot3 reveals that the computation time remained steady as the number of candidate web services for every abstract service increased, indicating that the number of candidate services for each abstract web service has little impact on the computation time of our genetic algorithm. Our GA always found a feasible solution for each test case with deadline or cost constraints.

From the above results,we can make the following conclusions: 1) our GA exhibits very good scalability. Its computation time increases almost linearly as the number of composite services, or the number of abstract services involved in a composite service increases. The computation time is not affected by the number of candidate services for each abstract one. 2) Our GA is effective. It can always find a feasible solution for deadline constraint and cost constraint problems with a reasonable constraint.

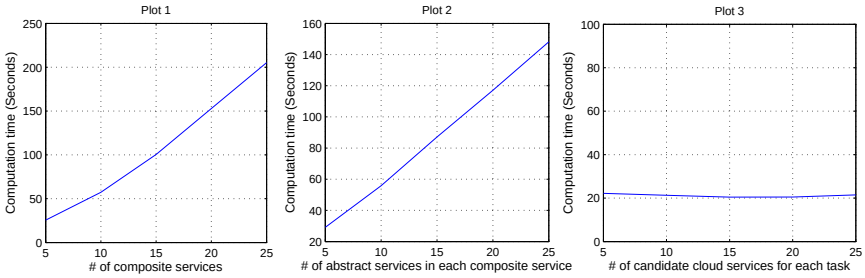

Fig. 3. Number of composite services, number of abstract web services in a composite web service, and number of candidate web service for each abstract web service, versus the computation time of our genetic algorithm

5 Conclusion

In this paper we have solved a new multiple composite web service resource allocation and scheduling problem in a hybrid cloud scenario where there may be limited local resources from private clouds and multiple available resources from public clouds. To the best of our knowledge, this problem has not been considered previously. In particular, we have presented a random-key genetic algorithm for the resource allocation and scheduling problem. This algorithm handles both problems simultaneously. It has been tested empirically and the experimental results have demonstrated good scalability and effectiveness.

References

1. Foster, I., Zhao, Y., Raicu, I., and Lu, S.. Cloud computing and grid computing 360-degree compared. *Proceedings of the 2008 IEEE Grid Computing Environments Workshop*. Novemeber 2008. p.1-10.
2. Rizos, S., Z. Henan, Eleni T., and Marios D.. Scheduling Workflows with Budget Constraints. *Proceedings of the Integrated Research in GRID Computing CoreGRID Integration Workshop 2005*. Pisa, Italy, Springer-Verlag. April 2007. p.347-357.
3. Yu, J. and R. Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, January 2006. 14(3). p.217-230.
4. Yu, J., M. Kirley, and R. Buyya.. Multi-objective planning for workflow execution on Grids. *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, Washington, DC, USA*, IEEE Computer Society. September 2007. p.10-17.
5. Bierwirth, C. and D.C. Mattfeld, Production scheduling and rescheduling with genetic algorithms. *Evolutionary computation*. 1999. 7(1). p.1-17.
6. Bean, J. C.. Genetic Algorithms and Random Keys for Sequencing and Optimization. *INFORMS Journal on Computing*. 1994. 6(2). p.154-160.