SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2011; **41**:469–493
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.1032

Evolutionary deployment optimization for service-oriented clouds

Hiroshi Wada[1,*,†], Junichi Suzuki[2], Yuji Yamano[3] and Katsuya Oba[4]

[1]*National ICT Australia, Eveleigh, NSW 1430, Australia*
[2]*School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052, Australia*
[3]*Department of Computer Science, University of Massachusetts at Boston, Boston, MA 02125, U.S.A.*
[4]*OGIS International, Inc., San Mateo, CA 94404, U.S.A.*

SUMMARY

This paper focuses on service deployment optimization in cloud computing environments. In a cloud, an application is assumed to consist of multiple services. Each service in an application can be deployed as one or more service instances. Different service instances operate at different quality of service (QoS) levels depending on the amount of computing resources assigned to them. In order to satisfy given performance requirements, i.e. service level agreements (SLAs), each application is required to optimize its deployment configuration such as the number of service instances, the amount of computing resources to assign and the locations of service instances. Since this problem is NP-hard and often faces trade-offs among conflicting QoS objectives in SLAs, existing optimization methods often fail to solve it. E^3-R is a multiobjective genetic algorithm that seeks a set of Pareto-optimal deployment configurations that satisfy SLAs and exhibit the trade-offs among conflicting QoS objectives. By leveraging queueing theory, E^3-R estimates the performance of an application and aids defining SLAs in a probabilistic manner. Moreover, E^3-R automatically reduces the number of QoS objectives and improves the quality of solutions further. Experimental studies demonstrate that E^3-R efficiently obtains quality deployment configurations that satisfy given SLAs. Copyright © 2011 John Wiley & Sons, Ltd.

Received 27 June 2009; Revised 11 August 2010; Accepted 1 October 2010

KEY WORDS: search-based software engineering; genetic algorithm; queueing theory; service level agreement; service composition; cloud computing

1. INTRODUCTION

This paper envisions service-oriented applications in cloud computing environments. A service-oriented application consists of a set of *service*s and a *workflow*. Each service encapsulates the function of an application component. Each workflow defines how services interact with each other. When an application is uploaded to a cloud, the cloud deploys each service in the application as one or more service instances. Each service instance runs on a process or a thread, and operates based on a particular deployment plan; different service instances operate at different quality of service (QoS) levels. For example, Amazon Elastic Compute Cloud (EC2)[‡] offers eight different deployment plans that allow service instances to yield different QoS levels by providing different

*Correspondence to: Hiroshi Wada, National ICT Australia, Eveleigh, NSW 1430, Australia.
†E-mail: shu@cs.umb.edu
‡http://www.amazon.com/ec2.

amounts of resources at different prices[§]. If an application is intended to serve different categories of users (e.g. users with for-fee and free memberships), it is instantiated with multiple workflow instances, each of which is responsible for offering a specific QoS level to a particular user category.

Service level agreements (SLAs) are defined upon a workflow as its end-to-end QoS requirements such as throughput, latency, CPU utilization and costs (e.g. resource utilization fees). In order to satisfy differentiated SLAs (a set of SLAs defined for different user categories), application developers (or performance engineers) are required to optimize a deployment configuration of service instances for each user category by considering which deployment plans and how many service instances to use for each service. For example, a deployment configuration may be intended to improve the latency of a heavily accessed service by deploying its instance with an expensive deployment plan that allocates a large amount of resources. Another deployment configuration may deploy two service instances with two inexpensive deployment plans connected in parallel for improving the service's throughput.

This decision-making problem, called SLA-aware service deployment optimization (SSDO) problem, is a combinatorial optimization problem that searches the optimal combinations of service instances and deployment plans. Cloud computing environments are in strong and urgent need of solving the SSDO problem as in traditional Internet data centers [1].

There exist five challenging research issues in the SSDO problem. First it is known to be NP-hard [2], and so can take a significant amount of time, labor and costs to find the optimal deployment configurations from a huge search space (i.e. a huge number of possible combinations of service instances and deployment plans).

The second issue is that the SSDO problem requires a huge variety of QoS history data to evaluate combinations of service instances and deployment plans (e.g. response time of a service instance deployed on all types of deployment plans under various request arrival rates). Since there exist a large number of deployment configurations in the SSDO problem, it is not realistic (or nearly impossible) to prepare all possible QoS history data before searching the optimal deployment configurations.

The third issue is that the SSDO problem often faces trade-offs among conflicting QoS objectives in SLAs. For example, in order to reduce its latency, a service instance may be deployed with an expensive deployment plan; however, this is against another objective to reduce costs. Moreover, if the service's latency is excessively reduced for a user category, the other user categories may not be able to satisfy their latency requirements. Therefore, it is important to exhibit the optimal trade-offs among QoS objectives rather than finding a single optimal deployment configuration.

The fourth issue is that the SSDO problem tends to have a large number of objectives to optimize simultaneously. This issue is a significant obstacle to solve optimization problems since it increases the size of search space and makes it hard to find optimal trade-offs among QoS objectives.

The fifth issue is that traditional SLAs often consider QoS requirements as their average (e.g. average latency). Since average-based SLAs fail to consider the variance of QoS measures at runtime, applications can yield large QoS fluctuations and occasionally violate SLAs although they may still satisfy the SLAs on an average basis.

This paper proposes and evaluates an optimization algorithm that addresses the aforementioned research issues in the SSDO problem. The proposed algorithm, called E^3-R, is a multiobjective genetic algorithm (GA) that seeks a set of Pareto-optimal deployment configurations that satisfy the SLAs and exhibits the trade-offs among conflicting QoS objectives. Given multiple Pareto-optimal configurations, application developers can better understand the trade-offs among QoS objectives and make a well-informed decision to choose the best deployment configuration for them according to their requirements and preferences. E^3-R also leverages queueing theory to estimate the performance of various deployment configurations with a very limited amount of

[§] As of this writing, a deployment plan with a 1.0 GHz CPU and 1.7 GB memory costs $0.09 per hour. Another deployment plan with four 2.0 GHz CPU cores and 15.0 GB memory costs $0.7 per hour.

DOI: 10.1002/spe

QoS history data and supports probabilistic SLAs rather than average-based SLAs. Moreover, when QoS objectives are redundant in given SLAs, E^3-R automatically reduces them to expedite its optimization process. Experimental results demonstrate that E^3-R efficiently obtains quality deployment configurations that satisfy given SLAs by heuristically examining very limited regions in the entire search space.

2. RELATED WORK

This paper describes a set of extensions to the authors' prior work [3]. A key extension is a new objective reduction method that addresses the issue of high dimensionality of optimization objectives. It was out of scope in the prior work. In addition, this paper carries out a new set of experimental studies to evaluate E^3-R.

In search-based software engineering (SBSE), the SSDO problem is classified as a problem in the area of *search-based testing of QoS*, where heuristic search algorithms are studied to compose services and test SLAs in a QoS-aware manner [4]. A number of SBSE research efforts have investigated the SSDO problem [2, 5–8]. Most of them use the Simple Additive Weighting (SAW) method to define fitness functions [9]; however, it is not easy to determine weight values in a fitness function because objectives often have different value ranges and priorities. As a result, it is known that SAW does not work well for non-trivial optimization problems [10–12]. In addition, SAW is designed to seek a single optimal solution; it cannot reveal optimal trade-offs among conflicting QoS objectives.

Another major method used to solve the SSDO problem is linear programming [13–18]. However, it is not suitable for the SSDO problem because (1) it cannot reveal the trade-offs among QoS objectives, (2) it is not trivial to define the problem in a linear form, and (3) it is not scalable; its computation time grows exponentially along with the size of search space.

In order to solve large-scale SSDO problems, this paper leverages a multiobjective GA as an SBSE technique because, in general, GAs scale better than linear programming [2]. Several research efforts have investigated multiobjective GAs for the SSDO problem [19–21]. However, most of them do not support the assumptions that the current cloud computing platforms make. For example, they do not consider binding multiple service instances to a service. No existing work considers differentiated SLAs and supports service deployment configurations that model applications in cloud computing. E^3-R is the first attempt to investigate the SSDO problem in the context of cloud computing.

Moreover, as discussed in Section 1, E^3-R performs automatic objective reduction in its optimization process. Objective reduction is an emerging technique in the area of multiobjective GAs; a limited number of objective reduction methods have been studied [22–24]. They use covariance to determine the redundancy between objectives. However, covariance examines only linear dependence between objectives, and the objectives in practical problems such as the SSDO problem have non-linear dependence. In contrast to these existing work, E^3-R employs symmetrical uncertainty, which can examine non-linear dependence between objectives. (See Section 5 for more details.) In addition, E^3-R is the first algorithm that can dynamically reduce and increase the number of objectives at runtime. No existing work has studied objective reduction methods for the SSDO problem.

Existing research efforts in the SSDO problem assume that all QoS history data are available; however, this assumption is not realistic. Thus, E^3-R employs queueing theory to estimate the performance of various deployment configurations with a very limited number QoS history data. E^3-R is similar to [25] in that both obtains the probability distribution of QoS measures with the Monte Carlo method. However, E^3-R focuses on optimizing deployment configurations of service-oriented applications based on estimated performance, while [25] focuses on performance estimation of applications. Through queueing theoretic performance estimation and QoS aggregation with the Monte Carlo method, E^3-R aims to define and use probabilistic SLAs. This is the first attempt to support probabilistic SLAs in the SSDO problem.

DOI: 10.1002/spe

E^3-R is similar to [26] in that both study GAs for SLA-aware service deployment of service-oriented applications. However, the two works target different goals. Penta *et al*. [26] seeks deployment configurations that violate given SLAs. In contrast, E^3-R seeks the optimal deployment configurations that satisfy given SLAs. Aversano *et al*. [27] uses a GA to generate workflows that fulfill the required functional requirements. E^3-R assumes that workflows are given and any given workflows fulfill required functional requirements.

3. SERVICE DEPLOYMENT AND QOS MODELS IN E^3-R

In E^3-R a workflow consists of a set of services. An example workflow in Figure 1 consists of four services. It has a branch after Service 1 and executes Service 2 and Service 3 in parallel. (E^3-R assumes that branching probabilities are known from history data.) In order to process requests, each service is instantiated as a service instance(s) and deployed on a particular deployment plan(s). A set of service instances and deployment plans is collectively called a deployment configuration. In Figure 2, Service 1 is instantiated as three service instances and deployed on two instances of Deployment Plan 1 and one of Deployment Plan 3. E^3-R assumes that a deployment plan can operate at most one instance for each service since multiple service instances do not contribute to the performance improvement. For example, in terms of the performance, running two instances (processes) of web servers on a single physical machine is equivalent to running one instance on a single machine. On the other hand, a deployment plan can have instances of different services at a time. (Deployment Plan 2 in Figure 2 has two service instances.) Each deployment configuration can have arbitrary number of deployment plans and service instances to improve the service's throughput and latency.

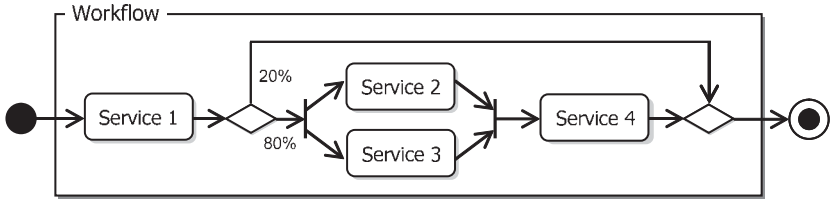

Figure 1. An example workflow.

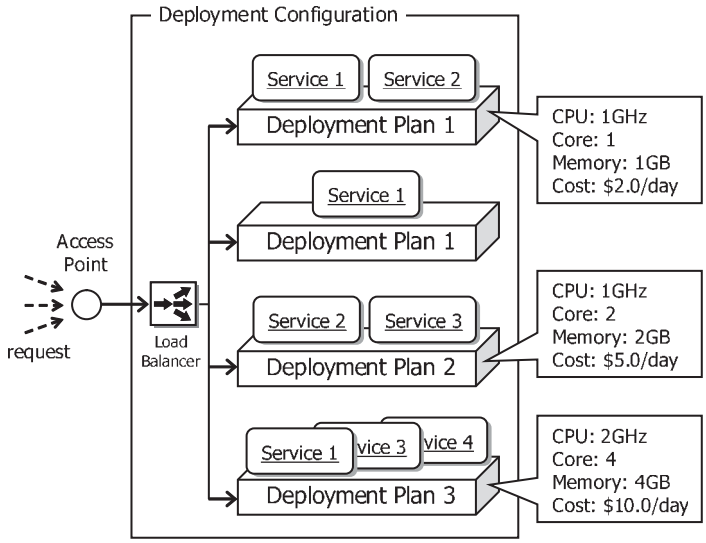

Figure 2. An example deployment configuration.

DOI: 10.1002/spe

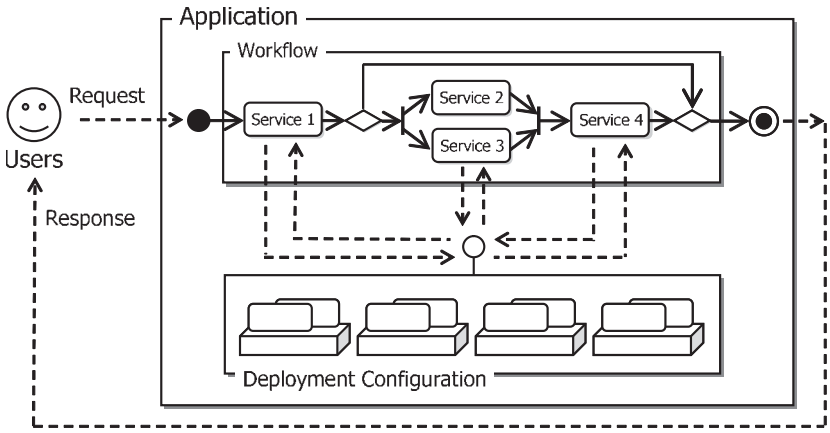

Figure 3. An example application.

Figure 3 illustrates how an application processes requests from users. When a user sends a request to an application, the application invokes a series of service instances in its deployment configuration according to its workflow and sends back a response to the user. In this example, once receiving a request from a user an application calls one of Service 1's instances, waits for a response from the service instance, calls Service 2 and Service 3's instances in parallel, waits for responses from them, then calls a Service 4's instance. A deployment configuration is assumed to have one access point equipped with a load balancer (Figure 2). Currently E^3-R supports two load-balancing algorithms, i.e. round robin and CPU load-balancing algorithm, to model applications, however E^3-R is designed to be extensible so that it can support other algorithms if load balancers in applications use other algorithms. When a deployment configuration uses a round robin load balancer, requests for a certain service are dispatched to corresponding service instances with equal probability. For example, in Figure 2, when a deployment configuration receives 1500 requests for Service 1 every second, each instance of Service 1 receives 500 requests every second since three instances are deployed in a deployment configuration. A CPU load-balancing algorithm dispatches requests to balance deployment plans' CPU usage.

Currently, SLAs are defined with the end-to-end throughput, latency, max CPU usage and cost of an application. In order to judge whether an application satisfies given SLAs, it is required to examine its end-to-end QoS by aggregating QoS measures of individual services.

E^3-R is designed to be practical and applicable to any cloud computing environments that differentiate resource provision, in turn QoS, for services; for example, Amazon EC2, FlexiScale[¶], GoGrid[‖] and Microsoft Azure[**]. Table I shows the mapping between the notions in E^3-R and four different cloud computing platforms. As the table illustrates E^3-R is generic enough to be applicable to various cloud computing platforms on the market.

3.1. Throughput and latency of service instances

Queueing theory is a well-established method for estimating performance of distributed systems and has been applied to a large number of research work [28, 29]. E^3-R estimates each service instance's throughput and latency by leveraging queueing theory, and it obtains the end-to-end throughput and latency by applying QoS aggregate functions.

Let μ_u be the mean unit service rate, which is the average number of requests processed per unit time by a service instance running on a unit CPU (e.g. 1 GHz CPU). μ_u is the inverse of the mean request processing time. E^3-R assumes that μ_u of all services in a workflow are known. Based

[¶]http://www.flexiscale.com.
[‖]http://www.gogrid.com.
[**]http://www.microsoft.com/windowsazure/.

DOI: 10.1002/spe

Table I. Mapping the notions in E^3-R and cloud computing platforms.

Platforms	Notions	
E^3-R	A deployment plan	A service instance
Amazon EC2	An EC2 instance	An application process operated in a server image
FlexiScale	A cloud server	An application process operated in a server image
GoGrid	A cloud server	An application process operated in a server image
Microsoft Azure	A compute instance	A web role or worker role instance

on μ_u, queueing theory can estimate throughput and the probability distribution of latency when a service instance runs on a deployment plan with various CPU configurations (e.g. one 2.5 GHz CPU core or four 1 GHz CPU cores) under various request arrival rates.

Assume that only one service instance runs on a deployment plan with n CPU cores, each of them is p times faster than a unit CPU. E^3-R models this service instance as an $M/D/N$ queue. (Poisson arrival, deterministic service time and multiple service centers.) Equation (1) is an approximation of the probability that the latency (waiting time W in a queue) is greater than α [30]:

$$\begin{aligned} Pr(W \geq \alpha) &= \left(\frac{np_n}{n-\rho}\right)\exp(-2(n-\rho)\alpha) \\ \text{where } \rho &= \lambda/(p\mu_u) \\ p_n &= p_0(\rho^n/n!) \\ p_0 &= \left[\sum_{k=0}^{n-1}\frac{\rho^k}{k!} + \frac{\rho^n}{(n-1)!(n-\rho)}\right]^{-1} \end{aligned} \tag{1}$$

λ is the mean request arrival rate, which is the average number of requests to a service instance per unit time. Since μ_u is known, $Pr(W \geq \alpha)$ can be calculated when λ is given. When $\lambda > np\mu_u$, however, the usage of all n CPU cores exceeds 100%. Therefore, λ must be reduced to λ' so that $\lambda' = np\mu_u$ holds. (e.g. by dropping $\lambda - \lambda'$ requests per unit time.) Hence, λ is the throughput of a service instance when $\lambda \leq np\mu_u$, while λ' (i.e. $np\mu_u$) is the throughput of a service instance when $\lambda > np\mu_u$ (λ' is also used instead of λ in Equation (1) to obtain the distribution of latency).

When multiple service instances run on one deployment plan, portions of the CPU power are assigned to each service instance based on their CPU usage. Assume that instances of two services, a and b, run on a deployment plan. μ_{ua} and μ_{ub} are their mean unit service rates, and λ_a and λ_b are their mean request arrival rates, respectively. For each CPU core, service a and b occupy $\rho_a = \lambda_a/(np\mu_{ua})$ and $\rho_b = \lambda_b/(np\mu_{ub})$ of CPU power. Therefore, for each CPU core, $1-\rho_b$ and $1-\rho_a$ are the available portions of CPU for services a and b, respectively. Since each service cannot use 100% of CPU power, their service rates are reduced to $p\mu_{ua}/(1-\rho_b)$ and $p\mu_{ub}/(1-\rho_a)$. These reduced service rates are applied to Equation (1) when calculating the service instances' latency. When $\rho_a + \rho_b > 1$, i.e. CPU usage exceeds 100%, service rates are reduced so that $\rho_a + \rho_b = 1$ holds.

Figure 4 illustrates an example. Instances of services a and b are deployed on a deployment plan with $n=1$ and $p=1$. λ_a is 3, μ_{ua} is 15, λ_b is 15 and μ_{ub} is 32. The CPU usages of a and b are $\frac{3}{15}=0.20$ and $\frac{15}{32}=0.47$, respectively. Since the CPU usage does not exceed the capacity ($0.20+0.47<1$), throughputs of a and b are the same as λ_a and λ_b. (All incoming requests are processed.) The portion of the CPU power available for a is $1-0.47=0.53$; therefore, a's mean service rate on this deployment plan is $0.53 \times 15 = 7.95$. The mean service rate of b is 25.6 as well. Then, the distributions of their latency are calculated based on their throughput and Equation (1).

To illustrate the validity of the performance model, Figures 5–7 compare the latency (cumulative distribution) of a testbed system and theoretical estimates. The testbed is an HTTP server that has the same number of threads as it has CPU cores (e.g. two threads when running on a two core CPU) and request arrival times follow a Poisson process. In this experiment, what developers know

DOI: 10.1002/spe

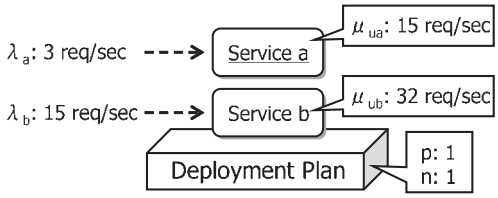

Figure 4. Multiple service instances.

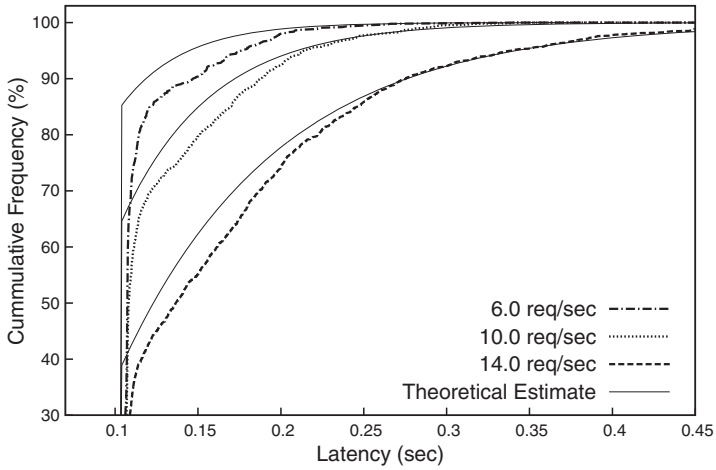

Figure 5. Latency on two 1.66 GHz cores.

is only the service rate of two services A and B on a 1.66 GHz CPU, i.e. 9.17 and 6.25 requests per second, respectively. Figure 5 compares latency of service A running on a CPU consisting of two 1.66 GHz cores under various request rates and theoretical estimates. (Dotted lines are measured latency and solid lines are theoretical estimates.) Figure 6 compares latency of service A running on a 2.2 GHz CPU and theoretical estimates. Figure 7 illustrates the case when services A and B are deployed on a deployment plan with a CPU consisting of two 2.2 GHz cores. Services A and B receive requests at 12 and 4 requests per second, respectively. As these results illustrate $M/D/N$ queues approximate the latency of services under any CPU configurations and request arrival rates accurately.

3.2. End-to-End QoS aggregation

In order to examine whether a deployment configuration can satisfy an end-to-end throughput requirement defined in SLAs, E^3-R first distributes the required throughput to each service instance by leveraging a load-balancing algorithm specified in a model. For example, in Figure 2, when required throughput is 1500 requests per second, a round robin algorithm distributes 500 requests to each instance of Service 1. Each instance of Service 2, 3 and 4 receives 750, 750 and 1500 requests per second, respectively. Then, E^3-R examines each deployment plan whether its CPU usage exceeds 100%. Based on the CPU usages, distributions of throughput and latency of each service instance are obtained. Throughput of a service is determined as the summation of its instances' throughput (e.g. service A's throughput is the summation of the throughput of all service A's instances).

Then, the end-to-end throughput of a deployment plan is determined by applying QoS aggregate functions in Table II according to the structure of a workflow. For example, in Figure 1, the throughput of the part where Services 2 and 3 are connected in parallel is the minimum throughput of Services 2 or 3. Then, the minimum throughput among Service 1, the parallel part of Service 2 and 3, and Service 4 is selected as the end-to-end throughput.

DOI: 10.1002/spe

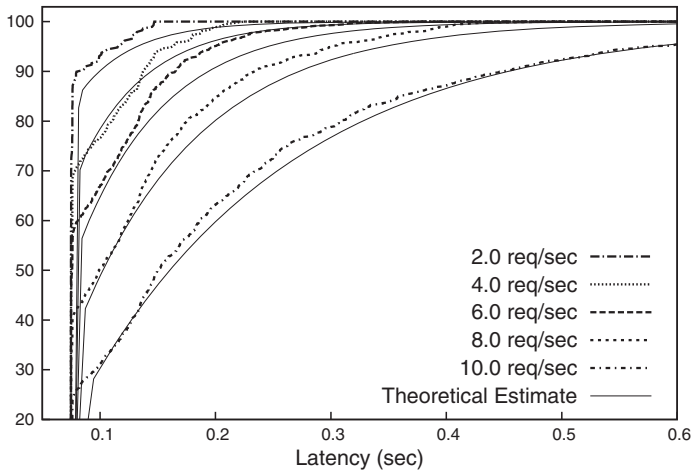

Figure 6. Latency on a 2.2 GHz CPU.

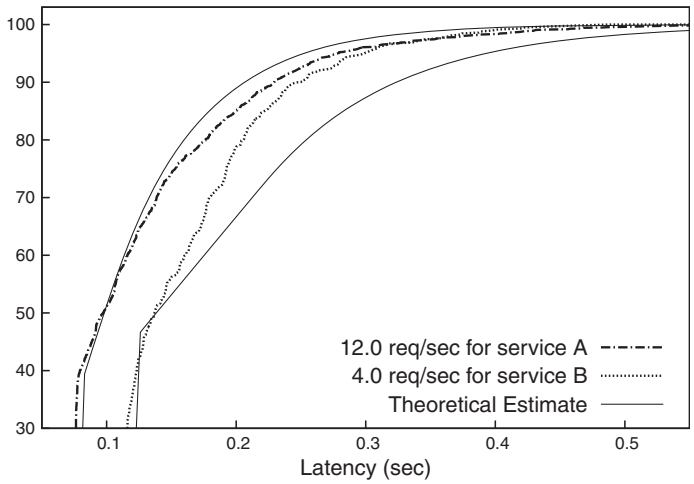

Figure 7. Latency on two 2.2 GHz cores.

Table II. Aggregate functions.

QoS objectives	Service	Sequence	Branch	Parallel
Throughtput (T)	Obtained by Queueing Theory	$\min_{s \in services} T_s$	$\sum_{b \in branches} T_b$	$\min_{seq \in sequences} T_{seq}$
Latency (L)	Obtained by Queueing Theory	$\sum_{s \in services} L_s$	N/A	$\max_{seq \in sequences} L_{seq}$
CPU Usage (U)	$\max_{p \in deployment\ plans} U_p$	$\max_{s \in services} U_s$	$\max_{b \in branches} U_b$	$\max_{seq \in sequences} U_{seq}$
Cost (C)	$\sum_{p \in deployment\ plans} C_p$	$\sum_{s \in services} C_s$	$\sum_{b \in branches} C_b$	$\sum_{seq \in sequences} C_{seq}$

In order to obtain the probability distribution of the end-to-end latency E^3-R employs Monte Carlo method since a simple aggregation (e.g. summation of each service's average latency) cannot reveal the probability distribution and lead to a too *pessimistic* estimation [25]. In order to obtain the end-to-end latency of each request E^3-R simulates the process in a workflow by (1) selecting services to execute (a path in a workflow) according to branching probabilities (all services are executed if a workflow has no branches.), (2) for each service selecting one of the instances according to their throughput (an instance with larger throughput has a higher chance to be selected), (3) for each service instances taking a sample value of its latency according to the

DOI: 10.1002/spe

probability distribution of its latency, and (4) aggregating a set of sample latencies by applying aggregate functions in Table II. By repeating this process many times, i.e. simulating many requests to an application, E^3-R approximates the probability distribution of the end-to-end latency.

Since E^3-R obtains the probability distribution of the end-to-end latency, it allows developers to define SLAs in a probabilistic manner, e.g. using best-case, worst-case, 99 percentile, average, and the most frequent latency as QoS objectives. It also allows for using multiple types of latency in SLAs, e.g. SLAs can define both average and worst-case latency as QoS objectives.

CPU usage is another QoS objective that E^3-R supports. It is the maximum of CPU usage among all deployment plans in a deployment configuration. It represents the characteristics of a deployment configuration since a deployment plan with the maximum CPU usage is a bottleneck of a deployment configuration.

The cost of a deployment configuration is also a QoS objective that E^3-R supports. It is defined as the summation of all deployment plans' costs. Therefore, the more number and the more expensive deployment plans an application uses, the higher its cost becomes.

4. MULTIOBJECTIVE OPTIMIZATION OF SERVICE DEPLOYMENT WITH E^3-R

As a GA, E^3-R maintains a population of individuals, each of which represents a set of service deployment configurations for user categories and encodes it as genes. E^3-R evolves and optimizes individuals in generations by repeatedly applying genetic operators to them. E^3-R currently assumes three user categories: platinum, gold and silver users; however, E^3-R is designed to be able to consider any number of user categories.

Figure 8 shows an example individual. An individual consists of three sets of genes, each of which represents a deployment configuration for each user category. A deployment configuration consists of deployment plans and service instances. An example in Figure 8 assumes that four types of deployment plans are available and three services are defined in a workflow. A deployment plan is encoded as a set of four genes; the first gene indicates the type of a deployment plan (i.e. 0 to 3 represents the index of the type) and the second to fourth genes indicate whether an instance of a certain service is deployed on it. (i.e. 1 indicates that an instance is deployed.) Therefore, the first four genes in the example, i.e. `2011`, represent a deployment plan of the third type that instances of the second and third services are deployed on. Since a deployment configuration can have arbitrary number of deployment plans, the length of genes varies depending on the number of deployment plans.

4.1. Genetic operators in E^3-R

Since the number of genes in an individual varies, crossover and mutation, i.e. two most important genetic operators, are designed to deal with the variable length of genes in E^3-R. The crossover operator in E^3-R performs one-point crossover on genes for each user category (Figure 9). A crossover point is randomly selected from points dividing deployment plans. For example, in

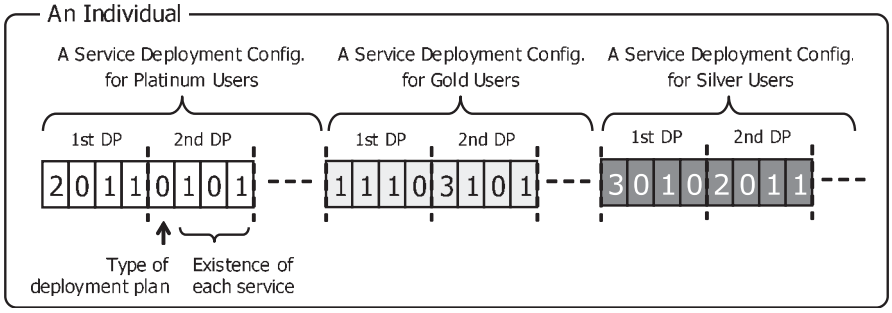

Figure 8. An example individual and optimization objectives.

DOI: 10.1002/spe

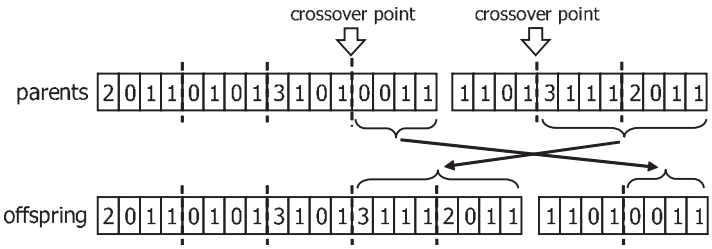

Figure 9. Crossover operator in E^3-R.

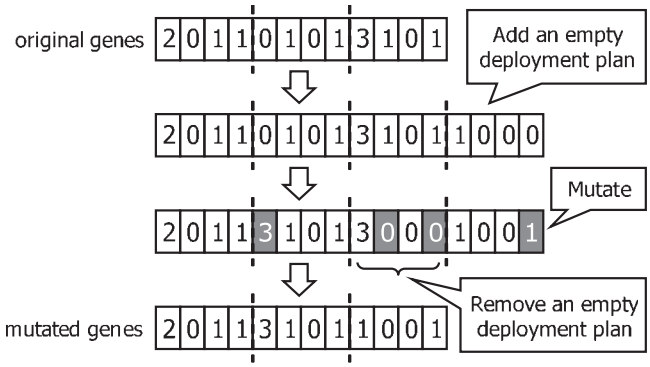

Figure 10. Mutation operator in E^3-R.

Figure 8, a crossover point must be between $4i$th and $4i+1$th genes, e.g. 4th and 5th or 8th and 9th genes, since each deployment plan is encoded as a set of four genes.

A mutation operator in E^3-R is designed to change the value of genes and the length of genes in an individual. In order to provide an opportunity to add a new deployment plan to a deployment configuration, the mutation operator first adds an *empty* deployment plan, i.e. a deployment plan that has no service instances on it, before mutating genes (Figure 10). (The type of a deployment plan is randomly selected.) Mutations occur on genes with the probability (i.e. mutation rate) of $1/n$ where n is the number of genes in an individual. When a mutation occurs on a gene specifying the type of a deployment plan, its value is randomly altered to a different type of deployment plan. When a mutation occurs on a gene indicating the existence of a service instance, its value is changed from zero to one or one to zero. (i.e. non-existent to existent or existent to non-existent.) Therefore, the mutation operator may turn a newly added empty deployment plan into a non-empty one and may turn existing non-empty deployment plans into empty ones. After performing mutations, the mutation operator examines each deployment plan and removes empty deployment plans from a deployment configuration. This way, the number of genes (the number of deployment plans) in an individual may change.

4.2. Optimization process in E^3-R

Listing 1 shows the optimization process in E^3-R. P^g denotes a population at the generation g. At each generation, two parents, p_α and p_β, are selected with binary tournaments. They reproduce two offsprings by performing the crossover operator. (The crossover rate, $Pr_{crossover}$, specifies the probability to perform a crossover.) Then, the offspring's genes are mutated. (The mutation rate, $Pr_{mutation}$, specifies the probability to perform a mutation.) This reproduction process is repeated until the number of offspring ($|Q^g|$) reaches u. From a union of P^g and Q^g E^3-R selects the top u individuals with respect to their fitness values. A fitness value indicates a quality (or goodness) of an individual; it is calculated with a fitness function in `AssignFitnessToIndividuals()`. E^3-R repeats this process for $g_{\max}$ times.

DOI: 10.1002/spe

```
1   g ← ∅ // initialized as zero
    P^∅ ← A population of randomly generated u individuals
3   repeat until g == g_max {
      AssignFitnessValuesToIndividuals(P^g)
5     Q^g ← ∅
      repeat until |Q^g| == u {
7       // Parent selection via binary tournament
        p_a, p_b ← Randomly selected two individuals from P^g
9       p_α ← Either p_a or p_b with a higher fitness value
        p_a, p_b ← Randomly selected two individuals from P^g
11      p_β ← Either p_a or p_b with higher fitness value

13      if Pr_crossover >= RandomValue then
          q_1, q_2 ← Crossover(p_α, p_β) // Reproduction via crossover
15      else
          q_1, q_2 ← p_α, p_β
17
        // Mutate offspring
19      q_1 ← Mutation(q_1) if Pr_mutation >= RandomValue
        q_2 ← Mutation(q_2) if Pr_mutation >= RandomValue
21
        Add q_1 to Q^g if Q^g does not contain q_1.
23      Add q_2 to Q^g if Q^g does not contain q_2.
      }
25    AssignFitnessValuesToIndividuals(P^g ∪ Q^g)
      P^{g+1} ← Top u of P^g ∪ Q^g in terms of their fitness values
27    g ← g+1
    }
29
    AssignFitnessValuesToIndividuals(P){
31    DominationRanking(P)
      foreach p in P {
33      if p is feasible then
          // Fitness function for a feasible individual
35        f ← p's domination value × p's sparsity
        else
37        // Fitness function for an infeasible individual
          f ← ∅ - p's SLA violation / p's domination value
39
        p's fitness value ← f
41    }
    }
```

Listing 1. Evolution Process in E^3-R.

E^3-R is designed to seek individuals that satisfy given SLAs and exhibit wider trade-offs among QoS objectives. In order to fulfill both the requirements, E^3-R distinguishes *feasible* individuals, which satisfy SLAs, and *infeasible* individuals, which do not. E^3-R uses two different fitness functions for feasible and infeasible individuals. (See `AssignFitnessToIndividuals()`.) The fitness function for feasible individuals is designed to encourage them to improve their QoS values in all objectives and maintain diversity in their QoS values. The fitness function for infeasible individuals is designed to encourage them to reduce SLA violations and turn into feasible individuals. Feasible individuals have positive fitness values while infeasible ones have negative fitness values. E^3-R considers higher-fitness individuals as higher quality. Therefore, feasible individuals always have higher chance of being selected as parents for reproduction compared with infeasible individuals.

4.3. Domination ranking

In the design of fitness functions, E^3-R employs the notion of *domination ranking* [31]. An individual i is said to *dominate* an individual j if any of the following conditions hold.

1. Individual i is feasible and j is not.
2. Both i and j are feasible, and i outperforms j in terms of their QoS values.
3. Both i and j are infeasible, and i outperforms j in terms of their SLA violations.

DOI: 10.1002/spe

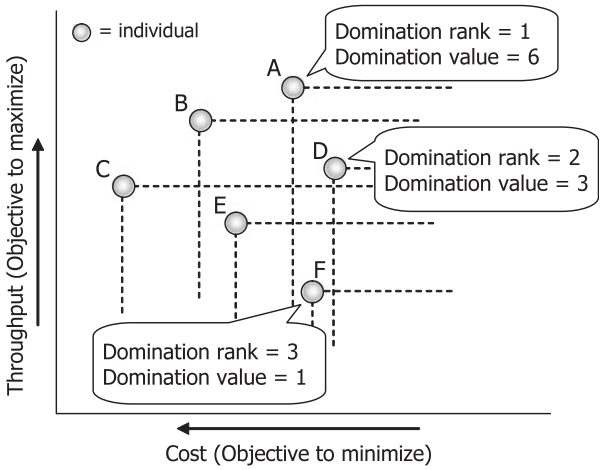

Figure 11. An example domination ranking.

In the second condition, an individual i is said to *outperform* an individual j if i's QoS values are better than, or equal to, j's in all QoS objectives, and i's QoS values are better than j's in at least one QoS objective. In the third condition, an individual i is said to *outperform* an individual j if i's SLA violations are smaller than, or equal to, j's in all violated QoS objectives in SLAs, and i's SLA violations are smaller than j's in at least one of violated QoS objectives. An SLA violation is measured as an absolute difference between an actual QoS measure and a required QoS level in SLAs.

Figure 11 shows an example of the second condition in domination ranking. It examines domination relations among six feasible individuals in terms of their QoS measures. In this example, individuals A, B and C are at the first rank, called *non-dominated*, since no individual dominates them. Individuals D and E are at the second rank since they are dominated by only top ranked individuals. Similarly, individual F is at the third rank.

4.4. Fitness function for feasible individuals

For each feasible individual, E^3-R assigns a fitness value based on its *domination value* and *sparsity*. (See `AssignFitnessToIndividuals()` in Listing 1.) These metrics contribute to increase a fitness value. An individual's domination value indicates the number of other individuals in the same or lower domination ranks than the individual in question. Therefore, non-dominated individuals have the highest domination values (Figure 11).

Sparsity represents the diversity of individuals; how individuals spread uniformly in the objective space. Maintaining the diversity of individuals is an important consideration in E^3-R to reveal wider variety of trade-offs in objectives. For each individual, E^3-R calculates the Euclidian distance to the closest neighbor individual in the objective space and determines the distance as its sparsity. (In this calculation, QoS values are normalized in case QoS objectives have different scales.)

4.5. Fitness function for infeasible individuals

For each infeasible individual, E^3-R assigns a fitness value based on its *total SLA violation* and domination value. (See `AssignFitnessToIndividuals()` in Listing 1.) The total SLA violation is calculated as the summation of violations against QoS requirements in SLAs. (In this calculation, violation values are normalized in case QoS requirements have different scales.) The total SLA violation contributes to decrease a fitness value while a domination value contributes to increase it.

DOI: 10.1002/spe

5. OBJECTIVE REDUCTION IN E^3-R

In multiobjective problems it is known that most individuals tend to become non-dominated when a problem has more than three objectives [32]. An individual must outperform another individual in terms of all objectives in order to dominate it (Section 4.3). As the number of objectives increases, it becomes difficult for individuals to dominate others and, in consequence, most individuals become non-dominated. Therefore, all individuals have the same or similar domination values and it weakens the pressure to evolve individuals. This issue is a major challenge to apply multiobjective GAs to problems with many objectives such as the SSDO problem. (In experimental studies in Section 6, a problem has 13 objectives.) In order to address this issue, this paper proposes an objective reduction method that identifies redundant objectives and *reduces*, i.e. does not use, them when performing a domination ranking. This objective reduction makes it easy for individuals to dominate others. Therefore, they are properly evolved.

5.1. The design of objective reduction algorithm

Redundant objectives are objectives that have a relationship such that improving one of the objectives leads to the improvement of others. For example, systems with lower latency tend to have higher throughput and vice versa. Therefore, even if throughput is not optimized explicitly, it can be optimized implicitly by optimizing latency. This relationship is the opposite of a trade-off relationship such as the relationship between latency and cost in the SSDO problem. This way, reducing (or ignoring) some of the redundant objectives can effectively reduce the number of the objectives without losing the pressure to optimize all objectives.

Although objective reduction is an effective technique, it is non-trivial to manually select a set of objectives to reduce since the redundancy of objectives are not always obvious and it requires deep knowledge of problems to select appropriate ones to reduce. Moreover, redundant objectives are usually not completely redundant. For example, throughput can be increased by connecting number of service instances in parallel, but it does not lower the latency. Therefore, if latency is completely ignored in an optimization process, it may not be optimized at all.

In order to address these difficulties, E^3-R calculates the redundancy of each objective every generation and automatically reduces ones identified as redundant. In addition, even if objectives have been reduced, E^3-R could *restore* them, i.e. use them again, in a domination ranking if they become non-redundant. This automatic objective reduction/restoration method eliminates the need for a manual objective reduction and lowers the risk to reduce non-redundant objectives.

Listing 2 shows the objective reduction method in E^3-R. The function `reduction` is invoked at the end of each generation (i.e. at the end of a loop in Listing 1). P^{g+1} denotes a population at the generation $g+1$, which is produced at the end of the generation g. The function first calculates the redundancy of each objective. (The details of the calculation are discussed in Section 5.2.) If all individuals are feasible and more than P_{non} percent of individuals are non-dominated in P^{g+1}, E^3-R attempts to reduce redundant objectives. For each objective o, E^3-R identifies it as a redundant objective if the summation of its redundancy (Sum_o) is positive. Then, an objective is reduced if it has been identified as redundant for $G_{trigger}$ generations ($Pos_o \geq G_{trigger}$). In addition, E^3-R examines whether any of reduced objectives are non-redundant anymore and attempts to restore them. E^3-R identifies an objective as a non-redundant if the summation of its redundancy is negative. An objective is reduced if it has been identified as non-redundant for $G_{trigger}$ generations. Note that `reduction` does nothing for the first $G_{trigger}$ generations of an optimization process and for $G_{trigger}$ generations after reducing or restoring objectives since it is highly possible that an optimization is proceeding and the redundancy of objectives is not stable during the period.

```
   INPUT: Population P^{g+1}
2
   // these variables are initialized at the beginning of each E^3-R run
4  g_skip ← G_trigger
   for each o∈ all objectives {Sum_o, Pos_o, Neg_o ← ∅} // initialized as zero
6
```


DOI: 10.1002/spe

```
    reduction(P^{g+1}){
 8    if g_skip > ∅ then return
      g_skip ← g_skip - 1
10
      {<o, redundancy>} ← calculate the redundancy of each objective from P^{g+1}
12    for each <o, redundancy> in {<o, redundancy>} {
        Sum_o ← Sum_o + redundancy // keep the sum of redundancy of each objective
14      // count the number of times o is identified as redundant or non-redundant
        if redundancy > ∅ then Pos_o ← Pos_o + 1
16      if redundancy < ∅ then Neg_o ← Neg_o + 1
      }
18
      if all individuals in P^{g+1} are feasible &&
20       more than P_non% of individuals in P^{g+1} are non-dominated {
         attemptToReduce({<o, redundancy>})
22    }

24    attemptToRestore({<o, redundancy>})

26    if any objective are newly reduced or restored {
        g_skip ← G_trigger
28      for each o∈ all objectives {Sum_o, Pos_o, Neg_o ← ∅}
      }
30  }

32  attemptToReduce({<o, redundancy>}){
      for each <o, redundancy> in {<o, redundancy>}
34      if Sum_o > ∅ && Pos_o >= G_trigger then reduce o
    }
36
    attemptToRestore(<o,redundancy>){
38    for each <o,redundancy> in <o,redundancy> {
        if o is reduced && Sum_o < ∅ && Neg_o >= G_trigger then restore o
40    }
    }
```

Listing 2. Objective Reduction/Restoration in E^3-R.

5.2. Reduction calculation

E^3-R employs symmetrical uncertainty to measure the redundancy of each objective [33]. The symmetrical uncertainty of two random variables denotes the mutual dependence of the two variables. For example, if the symmetrical uncertainty of X and Y is one, they are completely dependent (or identical) and knowing X determines the value of Y and vice versa. At the other extreme, if the symmetrical uncertainty of X and Y is zero, then they are completely independent and knowing that X does not give any information about the value of Y. In the SSDO problem each QoS objective corresponds to a random variable that consists of individuals' objective values. Given a set of individuals the mutual dependence of any two QoS objectives is measured as the symmetrical uncertainty.

The symmetrical uncertainty of X and Y, $U(X, Y)$, is calculated as in Equation (2). $p(x, y)$ is the joint probability distribution function of X and Y. $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y, respectively:

$$
\begin{aligned}
U(X,Y) &= 2\frac{I(X;Y)}{H(X)+H(Y)} \\
\text{where } I(X;Y) &= \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\left(\frac{p(x,y)}{p(x),p(y)}\right) \\
H(X) &= \sum_{x\in X} p(x)\log p(x)
\end{aligned}
\tag{2}
$$

The symmetrical uncertainty alone cannot distinguish whether two variables are conflicting or redundant since it only measures their dependence. (The symmetrical uncertainty is close to one either when two variables are strongly conflicting or strongly redundant.) Therefore, E^3-R calculates the covariance of two variables to examine whether they are conflicting (Equation (3)).

DOI: 10.1002/spe

Covariance is a measure of how much two variables change together. If the covariance is positive, two variables are positively correlated. That is, when one of them is above its expected value, then the other variable tends to be above its expected value as well. If the covariance is negative, two variables are negatively correlated.

$$Cov(X,Y) = E((X - E(X))(Y - E(Y))) \tag{3}$$

Note that the positive covariance represents either a conflict or a redundancy depending on the nature of random variables. When two random variables correspond to two QoS objectives to minimize (or maximize), the positive covariance denotes they are redundant whereas the negative covariance denotes they are conflicting. For example, the latency and cost are conflicting and both are objectives to minimize. Since reducing latency usually increases cost, which means their covariance is negative. On the other hand, when two random variables correspond to QoS objectives to minimize and maximize, respectively, the positive covariance denotes they are conflicting while the negative covariance denotes they are redundant. For example, throughput and cost are conflicting and they are objectives to maximize and minimize, respectively. Since increasing throughput usually increases cost, which means the covariance of them is positive. This way, it is determined whether two QoS objectives are conflicting or redundant.

Equation (4) shows how to calculate the redundancy. The redundancy of two QoS objectives is determined as their symmetrical uncertainty. However, when two QoS objectives are conflicting, E^3-R multiplies the symmetrical uncertainty by a negative one to obtain the redundancy:

$$Redundancy(X,Y) = \begin{cases} U(X,Y), & X \text{ and } Y \text{are redundant} \\ -U(X,Y), & X \text{ and } Y \text{are conflicting} \end{cases} \tag{4}$$

After calculating the redundancies of all pairs of QoS objectives, E^3-R sums them up for each QoS objective (Listing 3). For example, if a problem has 10 QoS objectives, the redundancy of each QoS objective is a summation of the redundancy between an objective in question and the other nine QoS objectives.

```
1   INPUT: Population P^{g+1}
    OUTPUT: Pairs of objective and its redundancy R={<o,redundancy>}
3
    R ← ∅
5   for each o1 ∈ all objectives {
      for each o2 ∈ all objectives {
7       redundancy_{o1,o2} ← U(o1, o2) based on individuals in P^{g+1}
        covariance_{o1,o2} ← Cov(o1, o2) based on individuals in P^{g+1}
9
        if both o1 and o2 are objectives to maximize or minimize {
11        if covariance_{o1,o2} < 0 then redundancy_{o1,o2} ← -1 × redundancy_{o1,o2}
        }
13
        if o1 and o2 are objectives to maximize and minimize {
15        if covariance_{o1,o2} > 0 then redundancy_{o1,o2} ← -1 × redundancy_{o1,o2}
        }
17
        r_{o1} ← r_{o1} + redundancy_{o1,o2}
19    }
      R ← R ∪ <o1, r_{o1}>
21  }
```

Listing 3. Redundancy Calculation in E^3-R.

6. EXPERIMENTAL STUDIES

This section evaluates E^3-R through experimental studies. Sections 6.2 and 6.3 show the quality of solutions that E^3-R yields. Section 6.5 shows the efficiency of E^3-R. Moreover, Section 6.6 evaluates E^3-R's versatility with different problem settings. Throughout experimental studies, E^3-R

DOI: 10.1002/spe

Table III. Deployment plans.

Name	CPU Core Speed (GHz)	# of cores	Cost ($)
High	1.0	4	50
Mid	1.2	2	30
Low	1.5	1	10

Table IV. Service level agreements (SLAs).

	SLAs				
User category	Throughput (req/s)	95%ile Latency (s)	CPU usage (%)	Cost ($)	Total cost ($)
Platinum	50	0.5	50	N/A	2000
Gold	90	1.0	70	N/A	
Silver	150	10.0	80	400	

is compared with E^3-R without its objective reduction method (denoted in figures as E^3-*R without reduction*) and NSGA-II, which is one of the most well-known multiobjective GAs [34].

6.1. Experimental setup and search space analysis

Each experimental study simulates a workflow shown in Figure 1. When running on a deployment plan with one 1.0 GHz CPU, Services 1, 2, 3 and 4 process 28, 25, 23 and 20 requests per second, respectively. In Table III are available the deployment plans.

Table IV shows SLAs used in experimental studies. The SLAs define each user category's throughput, 95 percentile latency, maximum CPU usage and cost. This experimental study assumes that 25 of platinum, 90 of gold and 750 of silver users access to an application simultaneously and each user in each category is allowed to send 2, 1 and 0.2 requests per second, respectively. Therefore, required throughput of platinum, gold and silver users is 50, 90, and 150 requests per second. In addition, platinum and gold users have no cost (or budget) limits whereas silver users have. In addition, there is a limit on the total cost incurred by all the three user categories.

A deployment plan is represented as one of $D\cdot(2^S-1)$ combinations of genes where D is the number of deployment plan types and S is the number of services in a workflow. When a deployment configuration has M deployment plans, there are ${}_{D\cdot(2^S-1)}H_M = (M+D\cdot(2^S-1)-1)!/(M!\cdot(D\cdot(2^S-1)!-1))$ combinations of genes. Therefore, the SSDO problem has a search space of $\Sigma_{i=1}^{N}(i+D\cdot(2^S-1)-1)!/(i!\cdot(D\cdot(2^S-1)-1)!)$ where N is the maximum number of deployment plans in a deployment configuration.

In this problem, D is 3, S is 4 and N is 200 since the maximum total cost is \$2000 and the most inexpensive deployment plan costs \$10. Therefore, the problem has approximately 3.7×10^{49} deployment configurations. Since E^3-R is configured to run for 500 generations with 100 individuals, it examines 50 100 individuals at most, which is a very limited (only 2.2×10^{-44}%) regions in the entire search space.

In order to illustrate the difficulty of this SSDO problem, 5 000 000 individuals are randomly created and their sums of QoS violations are examined. Figure 12 shows the cumulative distribution of summation of QoS violations. For each objective, violations are normalized in case they have different scales. Since the problem has 10 objectives, the maximum summation of violations is 10. All 5 000 000 individuals have positive violations; it means that all of them are infeasible. As Figure 12 shows, it is hard to find individuals with less than 0.5 of QoS violations and, therefore, this problem is practically impossible to solve with a random search or a brute force search.

Table V shows the parameters of each algorithm.

DOI: 10.1002/spe

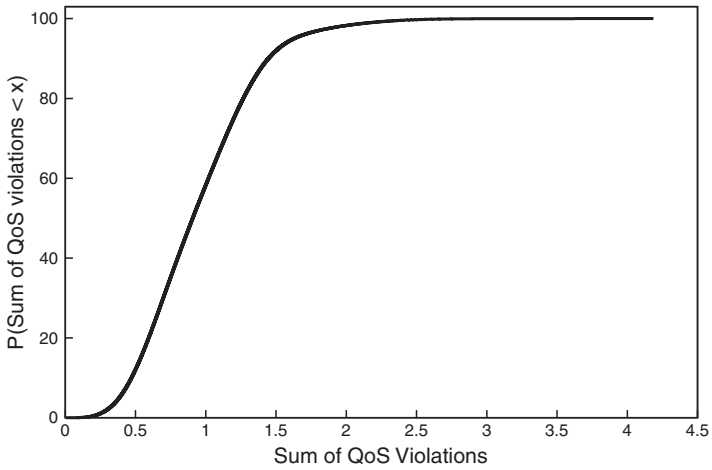

Figure 12. Probability of QoS violations.

Table V. Parameters of algorithms.

Parameters	E^3-R	NSGA-II
Population size (u)	100	
Offspring population size	100	
Maximum generation ($g_{\max}$)	500	
Parent selection operator	Binary tournament	
Crossover operator	One described in Section 4.1	
Crossover rate	0.9	
Mutation operator	One described in Section 4.1	
Mutation rate	1.0	
$G_{trigger}$	5	N/A
P_{non}	0.8	N/A

6.2. *Trade-off analysis*

The first experimental study reveals the maximum and minimum objective values that each algorithm yields. The results are obtained through 50 independent runs. Larger maximum and smaller minimum objective values offer a wider variety of deployment configurations. It allows application developers to understand the trade-offs among QoS objectives and make a well-informed decision to choose the best deployment configuration.

Figures 13(a)–16 show the maximum and minimum objective values that feasible individuals yield for each user category. Table VI shows the mean objective values obtained at 500th generation with the standard deviations in parenthesis. Throughputs are not shown here since feasible solutions' throughputs are exactly same as those in SLAs. (e.g. Throughput of platinum users is exactly 50 requests per second in all feasible solutions.)

Figure 13(a)–(c) shows the latency that feasible individuals yield for each user category. The initial individuals are randomly generated and all of them are infeasible at the beginning of each simulation run. It takes approximately 20 generations to find feasible individuals, which satisfy SLAs in all user categories. (By evaluating only 2000 individuals GAs can find an individual better than any of 5 000 000 individuals evaluated in a random search.) After that, feasible individuals are evolved to reveal wider trade-offs, i.e. lower and higher latency. As illustrated E^3-R reveals much wider trade-offs compared with E^3-R without reduction and NSGA-II. Since E^3-R tends to reduce (or ignores) latency of gold or silver users from its optimization objectives, it yields smaller maximum latency (Table VI) and smaller trade-offs for gold and silver users. However, E^3-R can optimize those objectives (i.e. yielding smaller minimum latency) compared with other algorithms.

DOI: 10.1002/spe

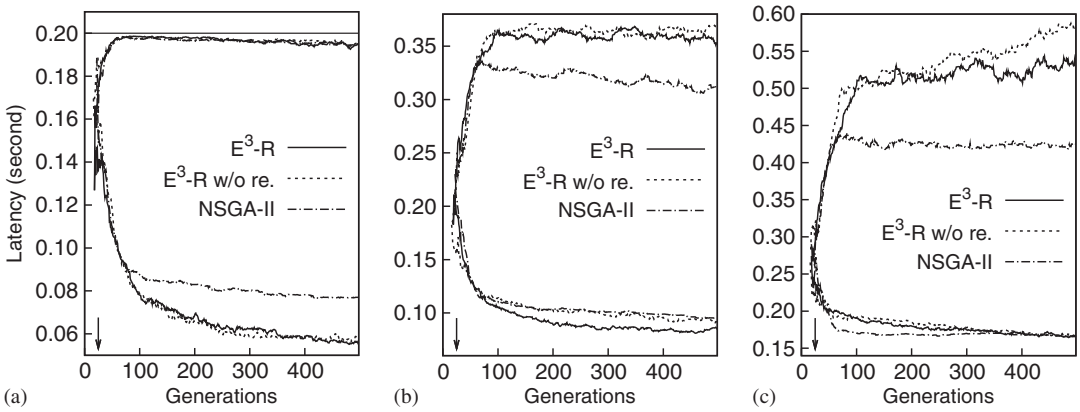

Figure 13. Latency: (a) platinum; (b) gold; and (c) silver.

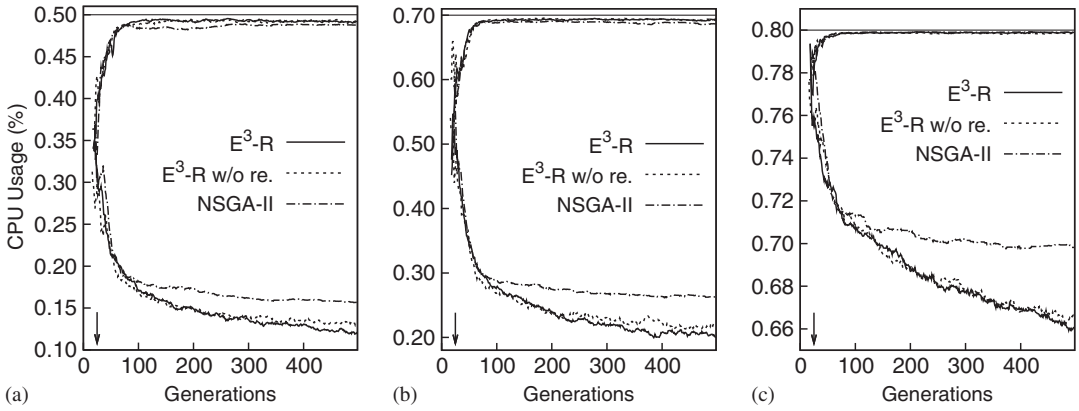

Figure 14. CPU usage: (a) platinum; (b) gold; and (c) silver.

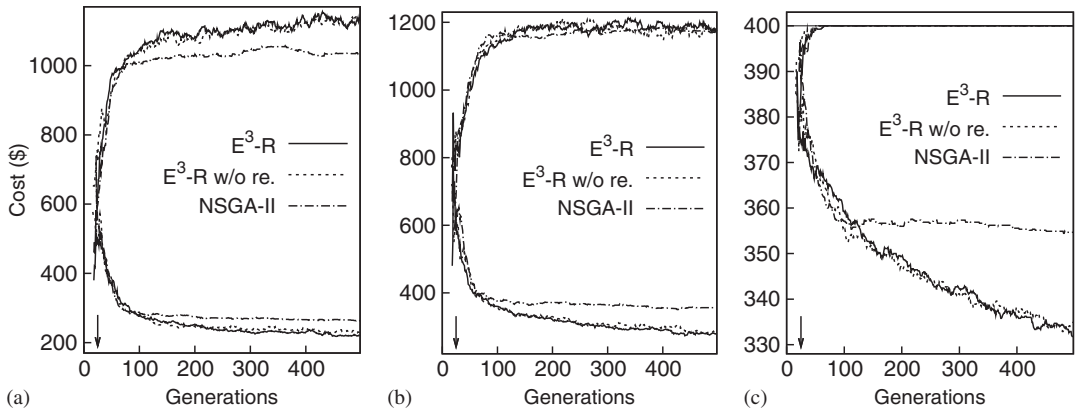

Figure 15. Cost (Usage fee): (a) platinum; (b) gold; and (c) silver.

Figure 14(a)–(c) shows the maximum of deployment plans' CPU usage. As well as latency, E^3-R reveals much wider trade-offs compared with NSGA-II. In addition, E^3-R yields larger maximum and smaller minimum objective values compare with E^3-R without reduction for most of the user categories (Table VI).

DOI: 10.1002/spe

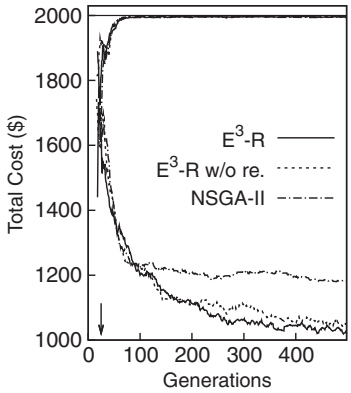

Figure 16. Total cost.

Table VI. Objective values E^3-R, E^3-R without objective reduction and NSGA-II yield.

			E^3-R	E^3-R w/o reduction	NSGA-II
Latency (s)	Platinum	Max	**0.1951** (8.064E−4)	0.1950 (6.609E−4)	0.1945 (9.843E−4)
		Min	**0.05550** (1.512E−3)	0.05662 (1.770E−3)	0.07600 (3.399E−3)
	Gold	Max	0.3505 (5.687E−3)	**0.3695** (4.294E−3)	0.3109 (6.634E−3)
		Min	**0.08442** (4.183E−3)	0.09221 (4.536E−3)	0.09533 (4.344E−3)
	Silver	Max	0.5284 (0.02083)	**0.5786** (0.02151)	0.4248 (0.01574)
		Min	**0.1666** (3.164E−3)	0.1675 (2.997E−3)	**0.1666** (4.949E−3)
CPU Usage (%)	Platinum	Max	**0.4925** (7.8360E−4)	0.4910 (1.0910E−3)	0.4876 (1.3222E−3)
		Min	**0.1206** (2.1697E−3)	0.1288 (2.4557E−3)	0.1564 (2.8957E−3)
	Gold	Max	0.6925 (7.8274E−4)	**0.6943** (5.1576E−4)	0.6864 (1.3193E−3)
		Min	**0.1967** (4.3958E−3)	0.2191 (3.5881E−3)	0.2631 (4.3457E−3)
	Silver	Max	0.7988 (2.2740E−4)	0.7986 (2.7233E−4)	**0.7990** (2.7997E−4)
		Min	**0.6615** (5.1150E−3)	0.6662 (3.8423E−3)	0.6982 (3.2374E−3)
Cost ($)	Platinum	Max	1133 (9.294)	**1134** (12.46)	1038 (12.90)
		Min	**222.4** (4.479)	233.6 (3.786)	263.3 (5.587)
	Gold	Max	**1197** (12.26)	1169 (11.81)	1166 (11.22)
		Min	**279.8** (5.150)	286.7 (5.019)	356.4 (5.288)
	Silver	Max	**400.0** (0.000)	**400.0** (0.000)	**400.0** (0.000)
		Min	**332.7** (2.988)	335.1 (1.129)	354.4 (1.900)
	Total	Max	**1998** (0.6508)	1996 (0.9715)	1995 (0.9749)
		Min	**1014** (11.51)	1049 (9.055)	1183 (12.43)

Figure 15(a)–(c) shows the cost that feasible individuals incurred by each user category. Figure 16 shows the total cost. As in Table IV gold users' SLA on throughput is more demanding than that of platinum users. Therefore, a deployment configuration for gold users tends to have many high-end deployment plans, and it results in higher cost. Similar to CPU usages, E^3-R yields larger maximum and smaller minimum objective values compared with E^3-R without reduction for most user categories (Table VI).

Figure 17 shows the ratio of feasible and non-dominated individuals over generations. All individuals in NSGA-II and E^3-R without reduction become non-dominated at 80th and 70th generation, respectively. It eliminates the pressure to evolve individuals. Compared with them, a few individuals in E^3-R remain dominated and it keeps the pressure to evolve individuals. And it turns into better objective values.

6.3. Hypervolume analysis

Section 6.2 shows the trade-offs that each algorithm reveals. However, trade-offs do not show the quality of *balanced* solutions. For example, a solution with relatively small latency and relatively

DOI: 10.1002/spe

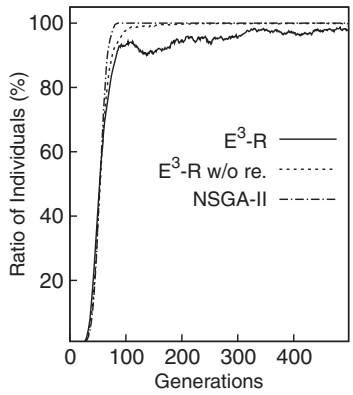

Figure 17. Feasible and non-dominated individuals.

Table VII. $v(A, B)$ Measures of E^3-R, E^3-R w/o reduction and NSGA-II.

$A \backslash B$	E^3-R	E^3-R w/o reduction	NSGA-II
E^3-R	—	0.01552 (0.012125)	0.02449 (0.01295)
E^3-R w/o reduction	0.01083 (0.01008)	—	0.02363 (0.01455)
NSGA-II	0.001700 (0.003338)	0.002115 (0.003730)	—

small cost is a well-balanced one and is often a practical solution to real-world problems rather than *extreme* solutions, which are optimized in only a few objectives.

To show the quality of solutions including balanced and extreme solutions Table VII compares algorithms in v measure [35]. $v(A, B)$ ($\in [1, 0]$) denotes the ratio of the volume of hyperspace where solution set A dominates but solution set B does not ($H_{A \backslash B}$) to the minimum hyperspace-containing solution sets A and B ($H_{A \cup B}$). Larger $v(A, B)$ and smaller $v(B, A)$ mean that solution set A exclusively dominates larger hyperspace and outperforms solution set B. $v(A, B)$ is calculated by Monte Carlo sampling and counting the fraction of samples that are dominated exclusively by A. 50 000 samples are used as proposed in [35].

Table VII shows the mean and the standard deviation of v measures over 50 independent runs. Since standard deviations are relatively large in these results, it is not obvious whether the differences of mean values are statistically significant. Thus, this experimental study first uses the Welch's t-test, which is an adaptation of Student's t-test intended for use with two samples having possibly unequal variances, to test the null hypothesis that means of two data sets are equal. When comparing two data sets v(E^3-R, NSGA-II) and v(NSGA-II, E^3-R), the mean of their difference is 0.02279 and 95% confidence interval of their difference is from 0.02226451 to 0.02331549. The two-tailed P value of the two sets is less than 0.0000001, which means that the probability of observing the difference as large as or larger than 0.02279 is less than 0.00001% if the null hypothesis was true. Hence, it can be concluded that v(E^3-R, NSGA-II) and v(NSGA-II, E^3-R) are significantly different. Similarly, other two combinations, i.e. v(E^3-R without reduction, NSGA-II) and v(NSGA-II, E^3-R without reduction), and v(E^3-R, E^3-R without reduction) and v(E^3-R without reduction, E^3-R), are significantly different. Therefore, it is possible to compare the algorithms in their mean v measures.

As in Table VII E^3-R and E^3-R without reduction largely outperforms NSGA-II. Moreover, E^3-R outperforms E^3-R without reduction. Therefore, E^3-R yields not only wider trade-offs discussed in Section 6.2 but also balanced solutions compared with E^3-R.

6.4. Processing time analysis

Table VIII shows the processing time of each algorithm measured with a Sun Java SE 6.0 VM running on a Windows Vista PC with an Intel Core 2 Quad 2.8 GHz CPU and 2 GB memory space.

DOI: 10.1002/spe

Table VIII. Processing time of E^3-R, E^3-R without reduction and NSGA-II.

Algorithms	Processing time (s)
E^3-R	192.6 (4.577)
E^3-R without reduction	177.3 (3.391)
NSGA-II	213.4 (3.938)

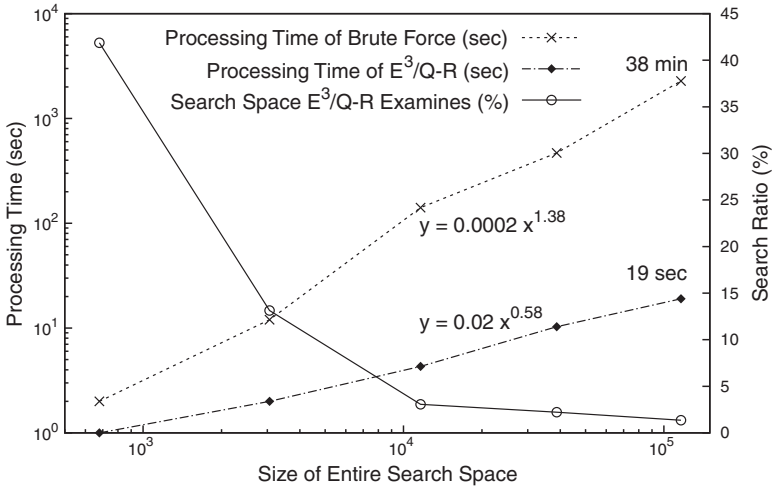

Figure 18. Ratio of search space examined.

Each algorithm runs for 500 generations with 100 individuals. The results are means and standard deviations over 50 independent runs.

The processing time of E^3-R is 8.5% longer than that of E^3-R without reduction since E^3-R calculates the redundancy of objectives at every generation. However, even after including the overhead of the redundancy calculation, the processing time of E^3-R is 9.7% shorter than that of NSGA-II. NSGA-II's crowding distance calculation is a relatively heavy process compared with E^3-R's sparsity calculation and it leads to the longer processing time.

According to Sections 6.2, 6.3 and 6.4, it is concluded that the additional overhead introduced by the redundancy calculation is reasonably small and E^3-R yields higher quality of deployment configurations in a shorter time compared with NSGA-II.

6.5. Efficiency analysis

The SSDO problem is known to be NP-hard and a brute force search is the only method that guarantees the finding of true optimal solutions. However, in relatively small problems E^3-R can find true optimal solutions efficiently compared with a brute force search. Figure 18 shows results of another experimental study that investigates the efficiency of E^3-R. This experimental study sets up several problems of which search space is 679 to 1.7×10^5 and obtains a set of the true optimal solutions for each problem through brute force searches. E^3-R also obtains the true optimal solutions for these problems.

Figure 18 shows the processing time of brute force searches and E^3-R that was measured with a Sun Java SE 6.0 VM running on a Windows Vista PC with an Intel Core 2 Quad 2.8 GHz CPU and 2 GB memory space. Although the processing time of both algorithms grows linearly, the growth constant of E^3-R is smaller than that of brute force, i.e. $2.0e^{-4}$ and $2.0e^{-2}$. This is because the average ratio of search space that E^3-R examines to find the true optimal solutions decreases as the search space increases. It results that E^3-R can find optimal solutions in a very short time even in problems with large search space. For example, in a problem with the search space of 10^5,

DOI: 10.1002/spe

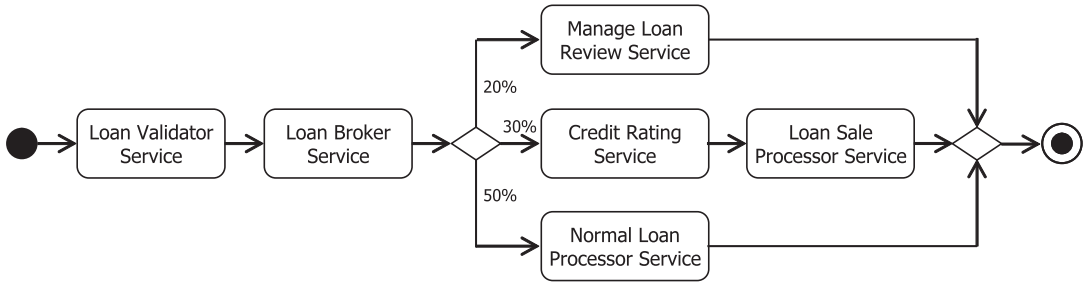

Figure 19. A workflow of a loan processing application.

Table IX. Service level agreements (SLAs).

	SLAs				
User category	Throughput (req/s)	95% Latency (s)	CPU usage (%)	Cost ($)	Total cost ($)
VIP	10	1.0	50	N/A	500
Regular	30	2.5	70	N/A	

E^3-R takes only 19 s to find the true optimal solutions while a brute force search takes 38 min. This result shows that E^3-R efficiently finds optimal solutions.

6.6. Experimental study in a real-world example

In order to show its versatility E^3-R is also evaluated in an application modeled from a real-world loan processing application. The application is one of the most representative service-oriented applications [36]. Different versions of the application are implemented on several middleware products such as BEA AquaLogic Service Bus[††].

The experimental study simulates a workflow shown in Figure 19. A loan validator service first receives a loan application, i.e. a message from a client, and examines whether it is complete. Then, a broker service routes the loan application according to a rule. When an application contains a request for a rate less than 5%, a management approval is required and the application is routed to a manage loan review service. When an application contains a request of greater than $25 million, it is examined for sale to a secondary loan company. The applicant's credit rating information is retrieved by a credit rating service and the application is forwarded to a loan sale processor service. Otherwise, an application is forwarded to a normal loan processor service. The service rate of each service on a deployment plan of one 1.0 GHz CPU is assigned according to the measurement results in [29]. The probability of routing messages is configured as 20, 30, and 50%, respectively.

Table IX shows SLAs. This experimental study uses the smaller number of user categories, i.e. VIP and regular, than the previous study to examine the efficiently of E^3-R in a different setup.

Table X shows the mean objective values obtained at 500th generation with the standard deviations in parenthesis. Table XI compares algorithms in v measure. Since the application in this experimental study has a small number of objectives, the difference between E^3-R and E^3-R without reduction is smaller compared with the results yielded in the previous experimental study. However, E^3-R yields the best or comparable objective values compared with other algorithms (Table X) and outperforms other algorithm in terms of v measure (Table XI). Therefore, it can be concluded that E^3-R is versatile enough to efficiently work in different problems.

[††]http://www.bea.com/alsb.

DOI: 10.1002/spe

Table X. Objective values E^3-R, E^3-R without objective reduction and NSGA-II yield.

			E^3-R	E^3-R w/o reduction	NSGA-II
Latency (s)	VIP	Max	0.7518 (1.0510E−2)	**0.79489** (1.0298E−2)	0.63888 (1.0606E−2)
		Min	0.2278 (9.6648E−3)	0.21500 (9.0359E−3)	**0.19652** (7.4006E−3)
	Regular	Max	1.0792 (2.7236E−2)	**1.0968** (3.2295E−2)	0.77138 (1.6216E−2)
		Min	0.26496 (6.7983E−3)	0.28373 (7.1246E−3)	**0.20986** (2.0905E−3)
CPU Usage (%)	VIP	Max	0.49668 (3.6756E−4)	**0.49671** (4.0397E−4)	0.49448 (5.1799E−2)
		Min	**0.18385** (2.9254E−3)	0.18615 (2.7639E−3)	0.20034 (2.3621E−2)
	Regular	Max	0.69792 (3.0016E−4)	**0.69812** (2.1135E−4)	0.69684 (4.6177E−4)
		Min	**0.36911** (3.2829E−3)	0.37287 (3.4634E−3)	0.39902 (2.2067E−3)
Cost ($)	VIP	Max	275.11 (2.3013)	**282.22** (2.6460)	262 (2.0270)
		Min	**82.222** (0.93697)	**82.222** (0.93697)	85.778 (0.96808)
	Regular	Max	**397.56** (2.0645)	397.33 (1.6450)	395.78 (1.8187)
		Min	187.56 (1.6073)	**187.33** (1.6450)	206.89 (1.3267)
	Total	Max	**500** (0.0)	**500** (0.0)	**500** (0.0)
		Min	304 (2.2088)	**302** (2.2088)	331.56 (2.0836)

Table XI. $v(A, B)$ Measures of E^3-R, E^3-R w/o reduction and NSGA-II.

$A \backslash B$	E^3-R	E^3-R w/o reduction	NSGA-II
E^3-R	—	0.03013 (0.01329)	0.03531 (0.01473)
E^3-R w/o reduction	0.02576 (0.01698)	—	0.03600 (0.01904)
NSGA-II	0.02501 (0.01126)	0.02670 (0.01191)	—

7. CONCLUSION

This paper proposes and evaluates E^3-R, which is a multiobjective GA to solve the SSDO problem for service-oriented cloud applications. Since the SSDO problem is NP-hard and often faces trade-offs among conflicting QoS objectives, existing optimization methods often fail to solve it. E^3-R seeks a set of optimal deployment configurations that satisfy the SLAs and exhibits the trade-offs among conflicting QoS objectives. Moreover, E^3-R automatically reduces the number of QoS objectives and improves the quality of solutions further. Experimental studies demonstrate that E^3-R efficiently obtains quality deployment configurations that satisfy given SLAs in a reasonably short time.

Several extensions are planned as future work. E^3-R is designed to be applicable to any cloud computing platforms that differentiate resource provision. However, through various empirical evaluations the authors observed that different cloud platforms exhibit different performance characteristics and queueing theory does not always capture them accurately. Therefore, further empirical evaluations are planned to improve the quality of the performance estimation method. In addition, the current E^3-R does not consider the quality and cost (fee) of network connections among service instances. When an application consists of several instances distributed over physically different datacenters, latency, and data transmission cost among service instances affect the end-to-end QoS measures of the application. Therefore, E^3-R will be extended to capture the aspect of networks. This extension allows E^3-R to model applications in commercial cloud computing platforms more accurately and enhances the applicability of E^3-R to real-world problems.

ACKNOWLEDGEMENTS

This work is supported in part by OGIS International, Inc.

DOI: 10.1002/spe

REFERENCES

1. Menascé DA, Ngo P. Understanding cloud computing: Experimentation and capacity planning. *International Computer Measurement Group Conference*, Dallas, TX, 2009.
2. Canfora G, Penta MD, Esposito R, Villani ML. An approach for QoS-aware service composition based on genetic algorithms. *ACM International Conference on Genetic and Evolutionary Computation*, Ont., Canada, 2005; 1069–1075.
3. Wada H, Suzuki J, Oba K. Queuing theoretic and evolutionary deployment optimization with probabilistic SLAs for service oriented clouds. *IEEE International Workshop on Cloud Services*, Los Angeles, CA, 2009; 661–669.
4. Afzal W, Torkar R, Feldt R. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology* 2009; **51**(7):957–976.
5. Jaeger MC, Mühl G. QoS-based selection of services: the implementation of a genetic algorithm. *Workshop on Service-oriented Architectures and Service-oriented Computing*, Bern, Switzerland, 2007; 359–370.
6. Penta MD, Esposito R, Villani ML, Codato R, Colombo M, Nitto ED. WS Binder: a framework to enable dynamic binding of composite web services. *International Workshop on Service-oriented Software Engineering*, Shanghai, China, 2006; 74–80.
7. Berbner R, Spahn M, Repp N, Heckmann O, Steinmetz R. Heuristics for QoS-aware Web service composition. *International Conference on Web Services*, Chicago, IL, 2006; 72–82.
8. Canfora G, Penta MD. A lightweight approach for QoS-aware service composition. *International Conference on Service Oriented Computing*, New York, NY, 2004; 36–47.
9. Harman M, Clark J. Metrics are fitness functions too. *IEEE International Symposium on Software Metrics*, Chicago, IL, 2004; 58–69.
10. Coello CAC. Recent trends in evolutionary multiobjective optimization. *Evolutionary Multiobjective Optimization*, Jain L, Wu X, Abraham A, Jain L, Goldberg R (eds.). Springer: Berlin, 2005; 7–32.
11. Harman M. The current state and future of search based software engineering. *ACM/IEEE International Conference on Software Engineering*, Minneapolis, MN, 2007; 342–357.
12. Mark Harman SAM, Zhang Y. Search based software engineering: a comprehensive analysis and review of trends techniques and applications. *Technical Report TR-09-03*, Kings College London, Department of Computer Science, April 2009.
13. Anselmi J, Ardagna D, Cremonesi P. A QoS-based selection approach of autonomic grid services. *ACM Workshop on Service-oriented Computing Performance*, Monterey, CA, 2007; 1–8.
14. Yu T, Zhang Y, Lin KJ. Efficient algorithms for web services selection with end-to-end QoS constraints. *ACM Transactions on the Web* 2007; **1**(1):129–136.
15. Ardagna D, Pernici B. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering* 2007; **33**(6):369–384.
16. Cardellini V, Casalicchio E, Grassi V, Presti FL. Flow-based service selection for Web service composition supporting multiple QoS classes. *IEEE International Conference on Web Services*, Salt Lake City, UT, 2007; 743–750.
17. Qu Y, Lin C, Wang YZ, Shan Z. QoS-aware composite service selection in grids. *International Conference on Grid and Cooperative Computing*, Hunan, China, 2006; 458–465.
18. Zeng L, Benatallah B, Ngu A, Dumas M, Kalagnanam J, Chang H. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering* 2004; **20**(5):311–327.
19. Claro DB, Albers P, Hao J. Selecting Web services for optimal composition. *IEEE International Workshop on Semantic and Dynamic Web Processes*, Orland, FL, 2005; 32–45.
20. Taboada HA, Espiritu JF, Coit DW. MOMS-GA: A multi-objective multi-state genetic algorithm for system reliability optimization design problems. *IEEE Transactions on Reliability* 2008; **57**(1):182–191.
21. Martens A, Brosch F, Reussner R. Optimising multiple quality criteria of service-oriented software architectures. *International Workshop on Quality of Service-oriented Software Systems*, Amsterdam, Netherlands, 2009; 25–32.
22. Deb K, Saxena D. On finding Pareto-optimal solutions through dimensionality. Reduction for certain large-dimensional multi-objective optimization problems. *Technical Report*, Kanpur Genetic Algorithms Laboratory, July 2005.
23. Jaimes AL, Coello CAC, Chakraborty D. Objective reduction using a feature selection technique. *ACM International Conference on Genetic and Evolutionary Computation*, Atlanta, GA, 2008; 673–680.
24. Jaimes AL, Coello CAC, Barrientos JEU. Online objective reduction to deal with many-objective problems. *International Conference on Evolutionary Multi-criterion Optimization*, Nantes, France, 2009; 423–437.
25. Rosario S, Benveniste A, Haar S, Jard C. Probabilistic QoS and soft contracts for transaction-based Web services orchestrations. *IEEE Transactions on Services Computing* 2008; **1**(4):187–200.
26. Penta MD, Canfora G, Esposito G, Mazza V, Bruno M. Search-based testing of service level agreements. *ACM International Conference on Genetic and Evolutionary Computation*, London, England, 2007; 1090–1097.
27. Aversano L, Penta MD, Taneja K. A genetic programming approach to support the design of service compositions. *Computer Systems Science and Engineering* 2006; **21**(4):247–254.
28. Stewart C, Kelly T, Zhang A. Exploiting nonstationarity for performance prediction. *ACM European Conference on Computer Systems*, Lisbon, Portugal, 2007; 31–44.
29. Liu Y, Gorton I, Zhu L. Performance prediction of service-oriented applications based on an enterprise service bus. *IEEE International Computer Software and Applications Conference*, Beijing, China, 2007; 327–334.

DOI: 10.1002/spe

30. Chan WC, Lin YB. Waiting time distribution for the M/M/m queue. *IEE Proceedings Communications* 2003; **150**(3):159–162.
31. Srinivas N, Deb K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation* 1994; **2**(3):221–248.
32. Ishibuchi H, Tsukamoto N, Hitotsuyanagi Y, Nojima Y. Effectiveness of scalability improvement attempts on the performance of NSGA-II for many-objective problems. *ACM International Conference on Genetic and Evolutionary Computation*, Atlanta, GA, 2008; 649–656.
33. Witten IH, Frank E. *Data Mining*: *Practical Machine Learning Tools and Techniques*. Morgan Kaufmann: Los Altos, CA, 2005.
34. Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 2002; **6**(2):182–197.
35. Fieldsend J, Everson RM, Singh S. Using unconstrained elite archives for multiobjective optimization. *IEEE Transactions on Evolutionary Computing* 2003; **7**(3):305–323.
36. Hohpe G, Woolf B. *Enterprise Integration Patterns*: *Designing*, *Building*, *and Deploying Messaging Solutions*. Addison-Wesley Professional: Englewood Cliffs, NJ, 2003.

DOI: 10.1002/spe