Multiobjective Optimization of SLA-aware Service Composition

Hiroshi Wada, Paskorn Champrasert and Junichi Suzuki
Department of Computer Science
University of Massachusetts, Boston
Boston, MA 02125-3393
{shu, paskorn, jxs}@cs.umb.edu

Katsuya Oba
OGIS International, Inc.
San Mateo, CA 94404
oba@ogis-international.com

Abstract

In Service Oriented Architecture, each application is often designed as a set of abstract services, which defines its functions. A concrete service(s) is selected at runtime for each abstract service to fulfill its function. Since different concrete services may operate at different Quality of Service (QoS) measures, application developers are required to select an appropriate set of concrete services that satisfies a given Service Level Agreement (SLA) when a number of concrete services are available for each abstract service. This problem, the QoS-aware service composition problem, is known NP-hard, which takes a significant amount of time and costs to find optimal solutions (optimal combinations of concrete services) from a huge number of possible solutions. This paper proposes an optimization framework, called E^3, to address the issue. By leveraging a multiobjective genetic algorithm, E^3 heuristically solves the QoS-aware service composition problem in a reasonably short time. The algorithm E^3 proposes can consider multiple SLAs simultaneously and produce a set of Pareto solutions, which have the equivalent quality to satisfy multiple SLAs.

1. Introduction

Service Oriented Architecture (SOA) is an emerging style of software architectures to build, integrate and maintain large-scale distributed systems [1, 2]. In SOA, each application is often designed with a set of *abstract service*s and a *business process*. Each abstract service encapsulates the function of an application component using its interface, and a *concrete service*(s) is selected (bound) at runtime to fulfill the function.

In SOA, a Service Level Agreement (SLA) is defined upon a business process as its end-to-end QoS constraints since a business process defines how abstract services interact to accomplish a certain business goal. Since different concrete services may operate at different QoS measures, application developers are required to select an appropriate set of concrete services that guarantees the fulfillment of a given SLA when a number of concrete services are available for each abstract service.

This problem, the QoS-aware service composition problem, is a combinatorial optimization problem which ensures the optimal mapping between each abstract service and available concrete services [3, 4]. Since the problem is known as NP-hard [5], it takes a significant amount of time and costs to find optimal solutions (optimal combinations of concrete services) from a huge number of possible solutions, and several heuristics have been proposed to find semi-optimal solutions in a reasonably short time [5–10]. However, existing heuristics assume simple service composition models. For example, they do not consider SLAs or consider only single SLA at a time. Also, they give single semi-optimal solution rather than a set of solutions that exhibits the trade-offs between different solutions.

This paper proposes an optimization framework, called E^3 (Evolutionary multiobjective sErvice composition optimizEr), to address the QoS-aware service composition problem. E^3 defines a service composition model and provides a multiobjective genetic algorithm, called E^3-MOGA, to solve the QoS-aware service composition problem. E^3-MOGA can consider multiple SLAs (e.g., Platinum, Gold and Silver service levels) simultaneously. Also, it provides a set of solutions of equivalent quality that give the option to assess the trade-offs between different solutions.

This paper is organized as follows. Section 2 shows the model of a service composition and its QoS measures that E^3 assumes. Section 3 describes the details of E^3-MOGA. Section 4 presents several simulation results to evaluate E^3-MOGA. Sections 5 and 6 conclude with some discussion on related work.

2. Service Composition

This section describes the model of a service composition and its QoS measures that E^3 assumes.

2.1. Business Process and Services

As Figure 1 and Tabl 1 illustrates, a business process consists of a series of *abstract service*s (e.g., `Abstract Service 1` and `Abstract Service 2`) and defines the order of executions of them. Although Figure 1 shows a sequence flow (a sequence of abstract services), a business process can split a flow into parallel flows to execute multiple abstract services in parallel (see Section 2.2).

Table 1: Elements in the Business Process Model in E^3

Business Process	A series of abstract services to execute
Abstract Service	An interface of a certain function
Concrete Service	An implementation of an abstract service
Business Process Instance	A set of Service Instances selected to complete a business process
Service Instance	A running process of a concrete service

An abstract service defines an interface of an application's component (e.g., homology search in bioinformatics), and a *concrete service*(s) realizes the interface (i.e., provides an actual implementation) at different QoS measures (e.g., `Concrete Service 1-1` and `Concrete Service 2-2`). When users (or business processes) use a certain abstract service, they can select concrete services to use depending on their requirements on QoS measures. For example, depending on the amount of allocated computing resources, a service operates at different QoS measures in a grid computing environment such as Amazon EC2[1]. Amazon EC2 allows anyone to deploy and operate applications on it. It provides three different deployment plans that allocate different amount of computing resources to applications' instances. For example, the low-end plan allocates a 1.0GHz processor and 1.7GB memory to an application's instance at $0.1 per hour. The high-end plan allocates four 2.0GHz processors and 15GB memory to an application's instance at $0.8 per hour. Different deployment plans in Amazon EC2 correspond to different concrete services in Figure 1.

E^3 currently assumes that each concrete service has three QoS attributes: throughput, latency and cost (usage fee). Cost is fixed, but throughput and latency can vary at runtime. E^3 assumes that probability distributions of QoS measures are known from history data. For example, `Concrete Service 2-1` operates without any problem, i.e., throughput is 2000 (req/sec) and latency is 30(ms), at 80% of probability, but its performance degrades at 18% and it is not available at 2% of probability.

For each abstract service in a business process, a *business process instance* (a running business process) must have at least one instance of a corresponding concrete service to complete the business process. A business process instance can have multiple service instances in parallel to improve the availability and performance (e.g., throughput). In case of Amazon EC2, users can deploy arbitrary number of application's instances at a time. For example, users can deploy two and three instances of an application with low-end and high-end deployment plans respectively. In Figure 1, a business process instance have multiple instances of `Concrete Service 1-1`, `2-2` and `1-4` to realize `Abstract Service 1`.

Given a definition of a business process and a set of concrete services, E^3 determines (1) which concrete services to use, and (2) how many instances of each concrete services to use, in order to create business process instances that satisfies a certain SLA. For example, E^3 automatically finds multiple business process instances of which throughput is over 12000 (req/sec), latency is less than 100 (ms) and cost is less than 1000 ($).

2.2. QoS Model

Computing overall QoS measures of a business process instance requires to aggregate QoS measures of services instances that the business process instance contains. First, E^3 calculates the expected QoS measures of each service instance since each instance has multiple levels of QoS measures with certain probabilities. Let $X = (x_0, ..., x_n)$ be a set of possible QoS measures and $P_{x_i} = Pr\{X = x_i\}$ ($\sum p_{x_i} = 1$) be the corresponding probabilities. Expected QoS measures of a service instance are calculated as $E(X) = \sum x_i p_{xi}$.

Then, E^3 calculates overall QoS measures of a business process instance. Depending on the structure of a business process (Fig 2) and QoS attributes, different aggregate function is applied (Table 2).

A set of service instances for an abstract service is deployed in *redundant parallel* and instances are used with equal probability (Figure 2). An overall throughput is the summation of service instances' throughput, an overall latency is the average of service instances' latency, and an overall cost is the summation of service instances' cost.

In a sequence of abstract services, corresponding service instances are executed one by one. Therefore, an overall throughput is the minimal abstract service's throughput (i.e., bottleneck), an overall latency is the summation of abstract services' latency, and an overall cost is the summation of service instances' cost. For example, in Figure 2, an overall QoS measures of a sequence consisting of `Abstract Service 2` and `Abstract Service 3` is calculated as an aggregation of `Abstract Service 2` and `Abstract Service 3`'s overall QoS measures.

In a parallel of sequences, sequences of abstract services are joined (synchronized) at the end. Therefore, an overall throughput is the minimal sequence's throughput, an overall

[1] Amazon Elastic Compute Cloud, www.amazon.com/ec2

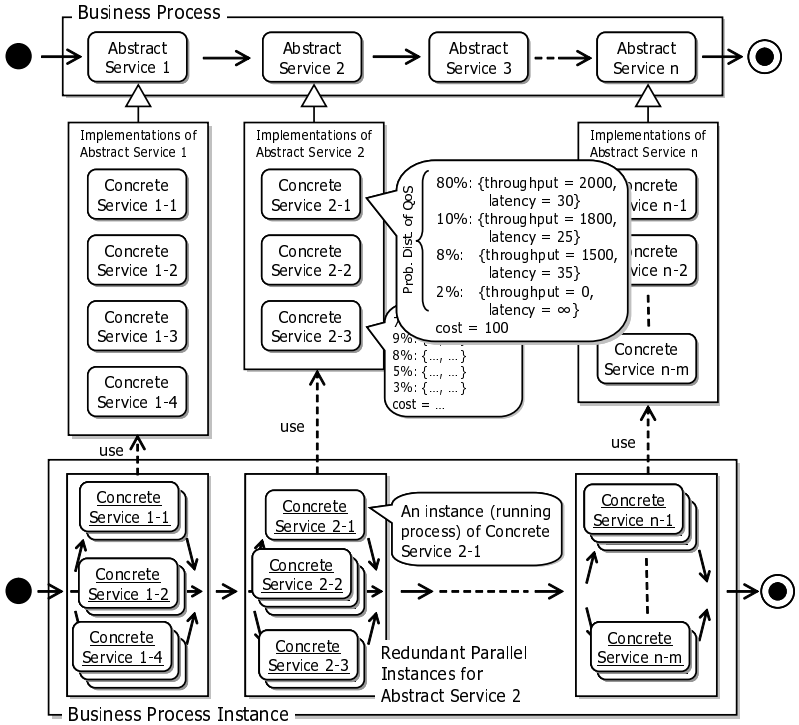

Figure 1: Business Process Model

Table 2: Aggregate Functions

QoS Attr.	Redundant Parallel	Sequence	Parallel
Throughtput (T)	$\sum_{i \in service\ instances} T_i$	$Min_{a \in abstract\ services} T_a$	$Min_{s \in sequences} T_s$
Latency (L)	$Avg_{i \in service\ instances} L_i$	$\sum_{a \in abstract\ services} L_a$	$Max_{s \in sequences} L_s$
Cost (C)	$\sum_{i \in service\ instances} C_i$	$\sum_{a \in abstract\ services} C_a$	$\sum_{s \in sequences} C_s$

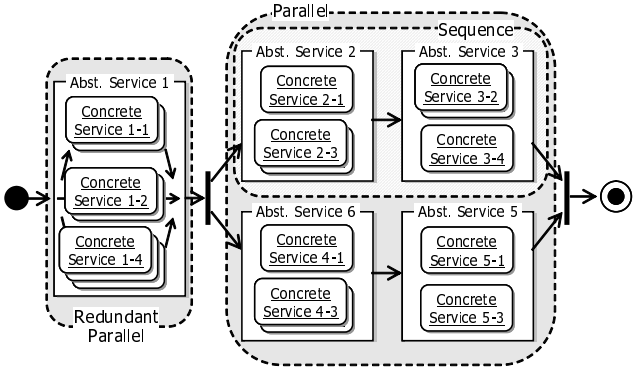

Figure 2: Structure of Business Process

latency is the minimal sequence's latency, and an overall cost is the summation of service instances' cost.

Since any business processes can be decomposed to a collection of redundant parallel, sequence and parallel, overall QoS measures of a business process is calculated by applying aggregate functions in Table 2.

3. Multiobjective Optimization for Service Composition

When a problem has a number of possibly conflicting objectives (goals) to be optimized simultaneously, there is no single optimal solution but rather a whole set of alternative solutions of equivalent quality, which is called Pareto solutions. For example, in the QoS-aware service composition problem minimizing cost and maximizing throughput are clearly conflicting and, therefore, there is no single optimum to be found. Multiobjective GAs can yield a whole set of Pareto solutions, which are all optimal in some sense, and give the option to assess the trade-offs between different solutions. For example, application developers can choose different service compositions: one yields low-throughput with low-cost, high-throughput with high-cost, or intermediate results.

3.1. Optimization Objectives

E^3-MOGA is a multiobjective genetic algorithm designed to solve the QoS-aware service composition problem. An individual is designed to represent a solution of the QoS-aware service composition problem (i.e., a service

composition or a business process instance) through a set of genes. For example, Figure 3 shows a set of genes that specifies a service composition of a business process consisting of three abstract services, i.e., `Abstract Service 1`, `Abstract Service 2` and `Abstract Service 3`, and each abstract service has four or three concrete services. The service composition in Figure 3 consists of two instances of `Concrete Service 1-1`, one instance of `Concrete Service 1-2`, three instance of `Concrete Service 2-1`, two instances of `Concrete Service 3-2` and one instance of `Concrete Service 3-3`.

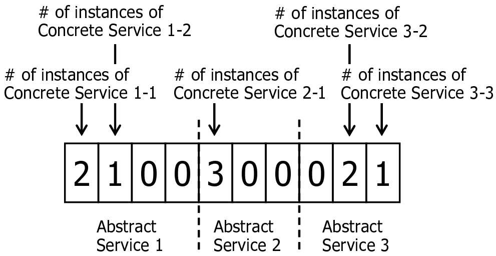

Figure 3: Genes Representing a Business Process Instance

In fact, E^3 currently supports three different service levels, i.e., Platinum, Gold and Silver, simultaneously and an individual is designed to represent three service compositions for each service level. An individual is evaluated and a set of overall QoS measures, i.e., throughput, latency and cost of each service level and the total cost, is given as its objective values (Figure 4).

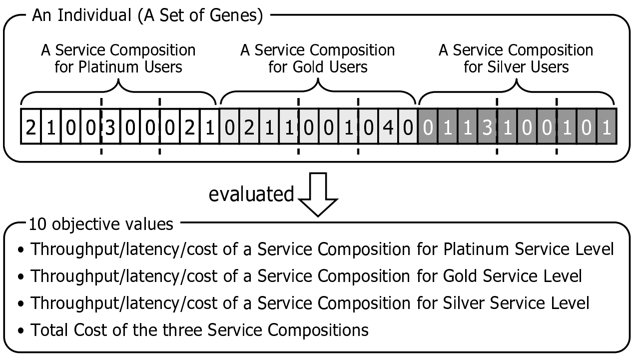

Figure 4: An Example of an Individual

3.2. E^3-MOGA

Listing 1 is a pseudo code of E^3-MOGA. In E^3-MOGA, a fitness value of an individual is calculated based on its *domination rank* and *density*, which are calculated based on objective values that the individual provides. The domination rank of an individual indicates how the individual outperforms the other individuals. The density of an individual indicates how many individuals provide similar objective values as the individual. The domination rank encourages a fitness value, while the density discourages a fitness value. E^3-MOGA maintains a set of individuals with higher fitness values, which is called *elite population*, and they evolve their genes across generations through the use of genetic operations (i.e., crossover and mutation).

Listing 1: E^3-MOGA

```
g = 0
P^0 = Randomly generated μ individuals, Q^0 = ∅
repeat until g = g_max {
  repeat until |Q^g| = μ {
    p_1 = RWSelection(P^g), p_2 = RWSelection(P^g)
    q   = Crossover(p_1, p_2)
    q   = Mutation(q)
    Q^g = Q^g ∪ q if q ∉ Q^g
  }
  P^{g+1} = Top μ of Sort(P^g ∪ Q^g)
  g=g+1
}

Crossover(p_1, p_2){
  for i = 1,...,n {
    center_i = (p_1[i] + p_2[i])/2
    q[i] = center_i + (Fitness(p_1) − Fitness(p_2))|p_1[i] − p_2[i]|/2 / (F(p_1) + F(p_2))
  }
  return q
}
```

Table 3: Variables and Functions in E^3-MOGA

g_{max}	Maximum number of generations
μ	Population size
P^g	A set of individuals $\{p_k \vert 1 \leq k \leq \mu\}$ in elite population at g-th generation
$p[i]$	i-th gene of individual p, $(0 \leq i \leq n)$
Q^g	A set of offspring that is generated at g-th generation
Sort(P)	A function sorting P based on Fitness(p_k)
RWSelection(P)	A function performing a roulette wheel selection on P based on Fitness(p_k)
Fitness(p_k)	A function returning DRanking(p_k) / Density(p_k)
DRanking(p_k)	A function returning p_k's domination rank
Density(p_k)	A function returning the density of p_k
Mutation(p_k)	A function randomly changing one of p_k's genes

A domination rank indicates the excellence of an individual in the objective space [11]. Let a set of objective values of an individual i be $\overrightarrow{o^i} = \left(\overrightarrow{o^i_{max}}, \overrightarrow{o^i_{min}}\right) = \left(o^{i,1}_{max}, ..., o^{i,n}_{max}, o^{i,1}_{min}, ..., o^{i,m}_{min}\right)$ where $\overrightarrow{o^i_{max}}$ and $\overrightarrow{o^i_{min}}$ are sets of objective values to be maximized and minimized respectively. An individual i is said to *dominate* an individual j, if both of the following conditions are true: (1) $\{\forall\left(o^{i,k}_{max}, o^{j,k}_{max}\right) | o^{j,k}_{max} \geq o^{j,k}_{max}\}$ and $\{\forall\left(o^{i,k}_{min}, o^{j,k}_{min}\right) | o^{j,k}_{min} \leq o^{j,k}_{min}\}$, and (2) $\{\exists\left(o^{i,k}_{max}, o^{j,k}_{max}\right) | o^{j,k}_{max} > o^{j,k}_{max}\}$ or $\{\exists\left(o^{i,k}_{min}, o^{j,k}_{min}\right) | o^{j,k}_{min} < o^{j,k}_{min}\}$. Moreover, E^3-MOGA considers constraints of objectives. For example, a constraint specifies that throughput

must be over 8000 (req/sec). When an individual cannot satisfy constraints, the individual is said to be *infeasible*. E^3-MOGA considers a degree of violation of constraints for infeasible individuals, in addition to objective values. An individual i is said to *constraint-dominate* an individual j, if any of the following conditions are true: (1) individual i is feasible and j is not, (2) both i and j are feasible and i dominates j with respect to their objective values, or (3) both i and j are infeasible but i dominates j with respect to their constraint violations. (Constraint violations are objective values to be minimized.)

An individual's domination rank is assigned based on the ascending order of the number of constraint-dominating individuals. An individual i's domination rank is given as $\mu - ndom_i$ where $ndom_i$ is the number of individuals that are not constraint-dominated by the individual i. Therefore, individuals that constraint-dominated by a number of other individuals have lower domination ranks, while individuals that are not constraint-dominated by others, called *non-dominated individual*s, are at the highest domination rank. Then, E^3-MOGA gives higher fitness to individuals with higher domination ranks (Table 3).

A density indicates the diversity of individuals in the objective space. Maintaining the diversity of individuals is an important consideration in MOGA to obtain individuals uniformly distributed over the objective space. Without taking preventive measures, the population tends to form relatively few clusters and cannot yield a whole set of potential solutions. E^3-MOGA follows the simple hyper-grid based scheme from PESA [12] to assign density for each individual. The objective space is divided into certain size of K-dimensional cells. (K is the number of objectives.) The number of individuals in each cell is defined as the density of the cell, and the density of an individual is equal to the density of the cell where the individual is located. Then, E^3-MOGA gives higher fitness to individuals with lower densities to improve the diversity of individuals (Table 3).

Individuals evolve their genes across generations through genetic operations. Once two parents are selected from the elite population by using roulette wheel selection based on fitness values [13], they crossover to generate an offspring. As in Listing 1, the offspring's gene values are calculated based on the proportion of fitness values of its two parents, i.e., gene values are assigned close to a parent that provides higher fitness value. After that, one of offspring's genes is randomly changed. After /mu of offspring are created, they are added to the elite population. Then, the top /mu individuals with respect to their fitness values are preserved as the next generation's elite population. E^3-MOGA repeats this process until g_{max}-th generation, and gives a set of feasible and non-dominated solutions as Pareto solutions.

4. Simulation Evaluation

This section describes simulation configurations and a set of simulation results.

4.1. Simulation Configurations

This simulation study simulates a business process consists of four abstract services as shown in Figure 5. Each abstract service has three concrete services at different QoS measures: one for high-end (i.e., high performance but high cost), one for low-end (i.e., low performance and low cost) and intermediate one (Table 4).

Table 5 shows a set of SLAs. Platinum and Gold service levels define the worst case throughput and latency. Silver service level defines only the worst case throughput. Platinum and Gold service levels have no limit on the cost (budget), but Silver service level has a severe constraint. Moreover, there is a constraint on the total cost of three service compositions. E^3 is expected to find solutions (i.e., service compositions for Platinum, Gold and Silver service levels) that satisfy all constraints.

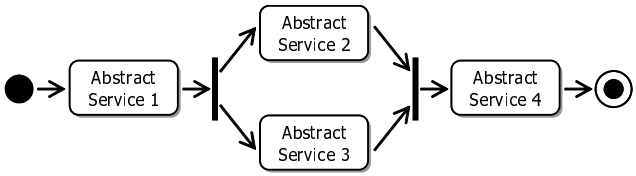

Figure 5: A Business Process in a Simulation Study

4.2. Simulation Results

In this simulation study, the population size (*mu*) is 300 and the maximum generation (g_{max}) is 1000. Figure 6 to 17 show simulation results.

Figure 6, 7 and 8 show throughput of service compositions for Platinum, Gold and Silver service levels respectively. Since initial service compositions are randomly generated and use excessive number of service instances, they achieve unnecessarily high throughput. E^3-MOGA, therefore, reduces the number of service instances as well as the cost during the first 25 generations drastically. The worst case of each level's throughput reaches to its lower bound at around 170th generation.

Figure 9, 10 and 11 show latency of service compositions for each service level. Silver service level's latency increases since it has no constraint. The worst cases of Platinum and Gold service levels' throughput reach to their upper bounds at around 170th generation.

Figure 12, 13 and 14 show cost of service compositions for each service level. The average of Silver service level's cost reaches to its upper bound at around 40th generation. After that, E^3 evolves individuals to find better solutions and the average keeps decreasing. The worst case of Silver service level's cost reaches to their upper bounds at around

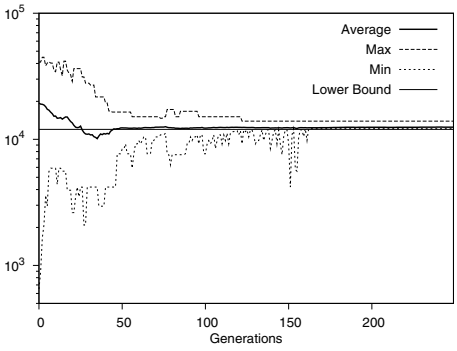

Figure 6: Platinum Level's Throughput

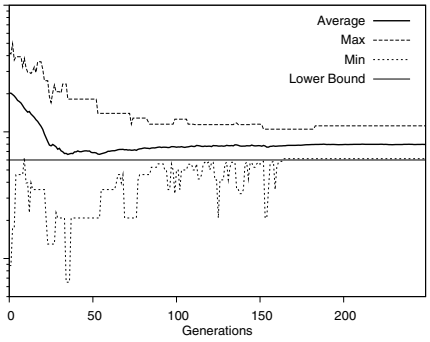

Figure 7: Gold Level's Throughput

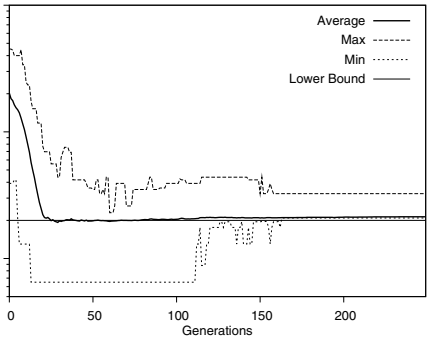

Figure 8: Silver Level's Throughput

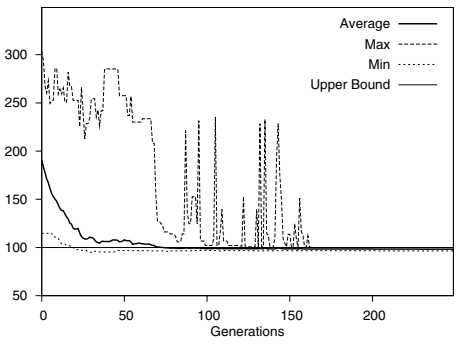

Figure 9: Platinum Level's Latency

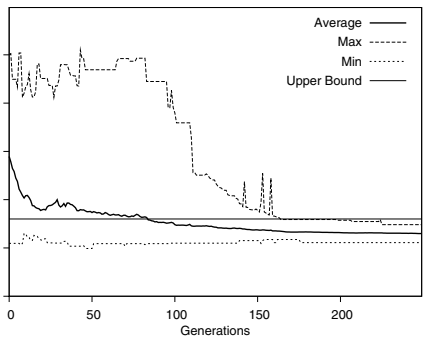

Figure 10: Gold Level's Latency

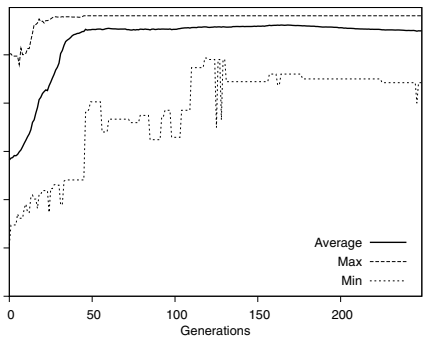

Figure 11: Silver Level's Latency

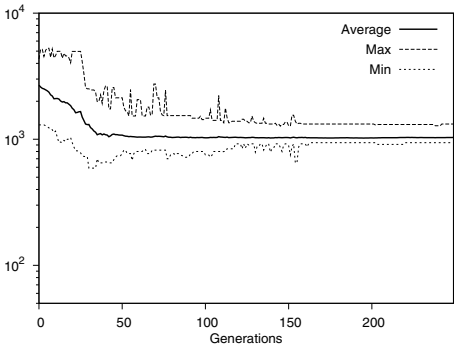

Figure 12: Platinum Level's Cost

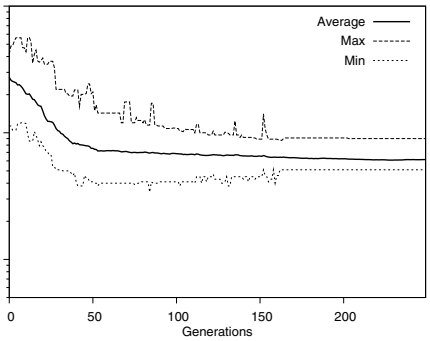

Figure 13: Gold Level's Cost

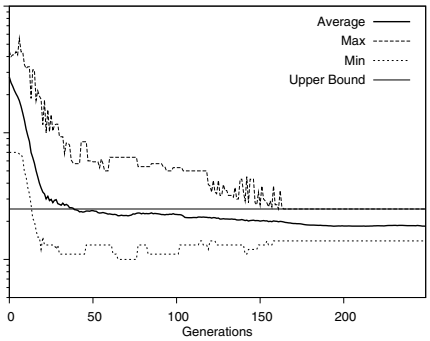

Figure 14: Silver Level's Cost

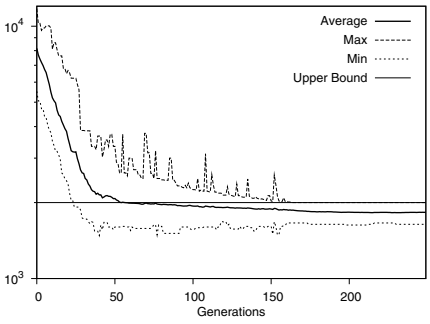

Figure 15: Total Cost

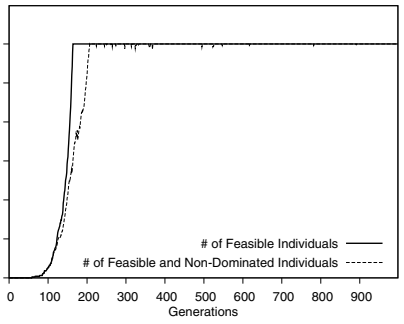

Figure 16: # of Feasible Individuals

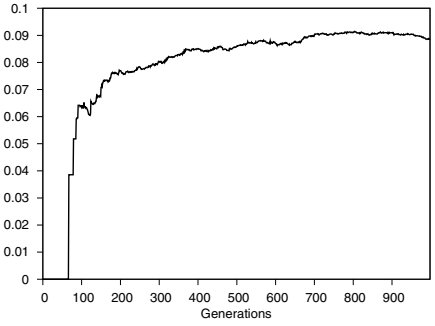

Figure 17: Avg of CV of Objective Values

Table 4: QoS Measures of Concrete Services

Abst. Service	Conc. Service	Probability Distribution of QoS measures Prob. (%)	Throughput	Latency	Cost
1	1	0.85	9000	60	90
		0.05	10000	50	
		0.05	6000	80	
		0.05	0	0	
	2	0.80	5500	60	50
		0.15	4000	100	
		0.05	0	0	
	3	0.30	2000	200	10
		0.30	3000	180	
		0.20	1500	250	
		0.20	0	0	
2	1	0.70	2000	20	50
		0.30	2300	18	
	2	0.90	4000	15	100
		0.05	6000	13	
		0.05	3000	20	
	3	0.70	4000	25	70
		0.20	3000	23	
		0.05	2500	30	
		0.05	0	0	
3	1	0.70	1500	30	30
		0.30	2000	20	
	2	0.80	3000	12	80
		0.10	5000	20	
		0.10	500	80	
	3	0.50	1000	60	10
		0.30	500	50	
		0.20	0	0	
4	1	0.75	2500	50	20
		0.25	3000	55	
	2	0.90	6000	15	70
		0.05	4000	20	
		0.05	3000	20	
	3	0.85	1000	90	5
		0.05	500	120	
		0.05	100	150	
		0.05	0	0	

Table 5: QoS Constraints

Service Level	Constraints			
	Throughput (Lower Bound)	Latency (Upper Bound)	Cost (Upper Bound)	Total Cost (Upper Bound)
Platinum	120000	100	-	2000
Gold	6000	130	-	
Silver	2000	-	250	

170th generation. Platinum and Gold service levels do not have constraints on their cost, but Platinum service level's constraints on throughput and latency are more severe than Gold service level's ones (Table 4). Thus, a service composition for Platinum service level tends to use large number of instances of high-end concrete services, and it results in higher cost compare with Gold service level. Figure 15 shows the total cost of three service compositions. The average of the total cost reaches to its upper bound at around 50th generation. After that, E^3 evolves individuals to find better solutions and the average keeps decreasing. The worst case of the total cost reaches to their upper bounds at around 160th generation.

Figure 16 shows the number of (1) feasible individuals and (2) feasible and non-dominated individuals. All individuals become feasible at around 170th generation, and become feasible and non-dominated at around 230th generation. At that time all individuals represent optimal solutions, however they may form few clusters. E^3-MOGA is designed to obtain individuals uniformly distributed over the objective space. Figure 17 shows the average of the coefficient of variations of each objective values. (The more diverse objective values are, the higher the coefficient of variation is.) As shown in Figure 17, even after all individuals being feasible and non-dominated, E^3 evolves individuals to obtain diverse solutions.

Figure 18 shows examples of feasible and non-dominated individuals obtained at 1000th generation. As shown in the Figure, a service composition for Platinum service level tends to use high-end concrete services while a service composition for Silver service level tends to use low-end concrete services. Moreover, the three individuals are distributed, and it shows that E^3 gives a set of diverse individuals that exhibits the trade-offs. The first one in Figure 18 optimizes Platinum service level's performance, the second one optimizes Gold service level's performance, and the third one optimizes the total cost. If developers want to improve QoS measures as long as budget allows, they can select the first or the second individuals depending on which service levels, i.e., Platinum or Gold, to focus. Developers can select the third one if they want to save budget as long as SLAs are satisfied.

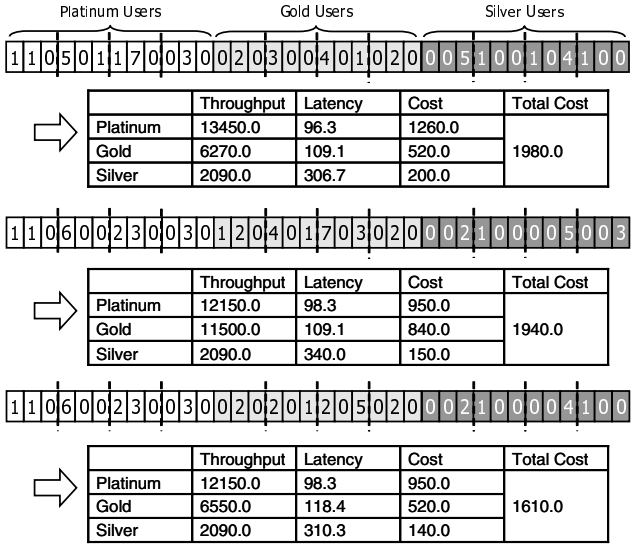

Figure 18: Feasible and Non-Dominated Individuals

5. Related Work

[5–7] leverage genetic algorithms to solve the QoS-aware service composition problem. Since they use single-objective genetic algorithms, they can provide only a single optimal solution. Also, they define aggregate functions to combine multiple objective values into a single objective value, and the quality of solutions highly depends on the design of the aggregate functions. For example, a weighted sum is widely used as an aggregate function, however it is

not easy to define weight values for each objective in a fairly manner since objectives have different value ranges and priorities. In contrast, E^3 leverages a multiobjective genetic algorithm. It gives multiple solutions of equivalent quality, i.e., the option to assess the trade-offs between different service compositions. Also, E^3-MOGA eliminates the need of aggregate functions and compare individuals through a set of objective values rather than aggregated values.

[8–10] leverage multiobjective genetic algorithms to solve the QoS-aware service composition problem. Although their approach is similar to E^3, their models are relatively simple. They do not consider the notion of SLA. Moreover, [9, 10] do not consider the notion of service instances and the probability distribution of QoS measures. In contrast, E^3 considers the notion of SLA, service instances and the probability distribution of QoS measures. The model in E^3 well reflects the realities of SOA applications especially in grid computing environments.

6. Conclusion

This paper proposes an optimization framework, called E^3, to address the QoS-aware service composition problem. E^3 defines a service composition model and provides a heuristic algorithm, called E^3-MOGA, to solve the QoS-aware service composition problem. E^3-MOGA considers multiple SLAs simultaneously and provides a set of solutions of equivalent quality.

Several extensions are planned as future work. E^3 will be extended to support QoS of connections among service instances. When a grid computing environment consists of several server farms distributed over the world, latency and data transmission cost among service instances varies depending on server farms where service instances are located. For example, latency between service instances in the same server farm is small, while latency between different server farms are large. Also, E^3 will be extended to support the notion of computing resources, such as hosts. When considering in-house applications, multiple service instances may be deployed on one host and it lowers their QoS measures compare with the case that a certain amount of computing resources are guaranteed for each service instances. This extension enables [3] to solve in-house SOA applications' optimization problem. For example, to find appropriate number of hosts to purchase and how many service instances to deploy on which hosts.

7. Acknowledgement

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

References

[1] M. Bichler and K. Lin. Service-Oriented Computing. *IEEE Computer*, 39(6), June 2006.

[2] M. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *IEEE Int'l Conf. on Web Information Systems Engineering*, December 2003.

[3] J. Anselmi, D. Ardagna, and P. Cremonesi. A QoS-based Selection Approach of Autonomic Grid Services. In *ACM High Performance Distributed Computing, Workshop on Service-Oriented Computing Performance*, June 2007.

[4] T. Yu and K. J. Lin. Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In *Int'l Conf. on Service-Oriented Computing*. Addison-Wesley Professional, Dec 2005.

[5] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An Approach for QoS-aware Service Composition based on Genetic Algorithms. In *Genetic and Evolutionary Computation Conference*, June 2005.

[6] M. C. Jaeger and G. Mühl. QoS-based Selection of Services: The Implementation of a Genetic Algorithm. In *Conference on Communication in Distributed Systems, Workshop on Service-Oriented Architectures and Service-Oriented Computing*, March 2007.

[7] Y. Gao, B. Zhang, J. Na, L. Yang, Y. Dai, and Q. Gong. Optimal Selection of Web Services with End-to-End Constraints. In *IEEE Int'l Conf. on Grid and Cooperative Computing*, October 2006.

[8] H. A. Taboada, J. F. Espiritu, and D. W. Coit. MOMS-GA: A Multi-Objective Multi-State Genetic Algorithm for System Reliability Optimization Design Problems. *IEEE Transactions on Reliability*, 57(1):182–191, March 2008.

[9] W. Chang, C. Wu, and C. Chang. Optimizing Dynamic Web Service Component Composition by Using Evolutionary Algorithms. In *IEEE/ACM Int'l Conf. on Web Intelligence*, Septemer 2005.

[10] D. B. Claro, P. Albers, and J. Hao. Selecting Web Services for Optimal Composition. In *IEEE Int'l Conf. on Web Services, Workshop on Semantic and Dynamic Web Processes*, July 2005.

[11] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[12] D. W. Corne, J. D. Knowles, and M. J. Oates. The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization. In *Parallel Problem Solving from Nature Conference*, 2000.

[13] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Professional, 1989.