Generating Adaptation Rules of Software Systems: A Method Based on Genetic Algorithm

Yang Liu, Di Bai and Wenpin Jiao
Key Lab of High Confidence Software Technologies (Peking University), Ministry of Education, China
Institute of Software, School of EECS, Peking University, Beijing, 100871, China
Beijing, China
{liuyang13, baidi14, jwp}@sei.pku.edu.cn

ABSTRACT
Nowadays, applications are deployed and executed in open, uncertain and dynamic environments. Increasingly, applications are required to have the capability to automatically change its behaviors in response to changing environmental conditions. As the complexity of adaptive and autonomic systems grows, designing and managing the set of adaptation rules becomes increasingly challenging and may produce huge computation cost. If we dynamically generate adaptation rules at run time, it's difficult to deal with the changes quickly. The software system challenges the method for efficiently generating effective adaptation rules. This paper proposes an approach to combine genetic algorithm and linear regression to automatically generate adaptation rules for software systems. Unlike traditional rule-based adaptation methods, our solution enables a system to obtain a prediction function to determine the corresponding system configuration under any considered environmental condition. We have applied this genetic algorithm based approach to the dynamic reconfiguration of two different software systems. The experimental results show that our method is practical and highly-efficient in software reconfiguration under changing environmental conditions. Besides, the user's requirements can be well satisfied.

CCS Concepts
•**Theory of computation→Stochastic approximation**
• **Computing methodologies→Machine learning approaches**

Keywords
Self-adaptive system; evolutionary algorithm; genetic algorithm; intelligent control;

1. INTRODUCTION
As the complexity of software systems grows, applications need to be self-adaptive in response to changing requirements and dynamic environment. Self-adaptivity is the capability of the system to adjust its behavior in response to the environment. As a whole, the system is able to perceive and respond to the changes of open and dynamic environment, in order to meet user's goal or obtain better performance [1].

ICMLC 2018, February 26–28, 2018, Macau, China

ACM ISBN 978-1-4503-6353-2/18/02…$15.00
DOI: https://doi.org/10.1145/3195106.3195137

To achieve the goal of self-adaptivity, it is important for applications to be able to self-reconfigure in response to the changes. Rule-based approaches are very efficient for configuring adaptive systems, and they have the advantages of readability and elegance [2]. These methods usually use a predefined adaptation rule set to decide when and how to respond to dynamic changes. The software system challenges the method for efficiently generating effective adaptation rules.

Adaptation rules can be acquired artificially or automatically. To specify such a set of adaptation rules artificially, the person needs to have sufficient domain knowledge of the system under consideration. This specification task is not easy even for a domain expert, and is harder for common system users because it's very hard to take all factors of the system into account. To reduce the burden on developers, it is very necessary to help them generate adaptation rules automatically.

Some current works on rules' generation are based on probabilistic model checking [3, 4] or goal-based reasoning [5, 6, 7]. However, these methods always bring about huge computation cost and are not efficient to deal with complex self-adaptive systems. Due to the complexity of systems nowadays, it's infeasible to traverse all configurations to find the best adaptation rules.

To resolve this problem, we proposed a method that transforms the generation of adaptation rules into a goal-based optimization problem. Then we apply a genetic algorithm to solve the optimization problem. Genetic algorithms (GAs) are the earliest, most well-known, and most widely-used evolutionary algorithm. Evolutionary algorithm (EA) refers to an algorithm that optimizes a problem's solution over many iterations, one iteration of an EA is called a generation in keeping with its biological foundation [8].

A genetic algorithm is a stochastic search-based technique for finding solutions to optimization and search-based problems [9]. It can make the solution of the problem tends to be optimal through finite iterations, and reduce the computation cost through stochastic search. In practice, genetic algorithms can be surprisingly fast in efficiently searching complex, highly nonlinear, and multidimensional search spaces [10].

The automatic generation of adaptation rules can be divided into two categories: offline and online. Offline methods have a common problem: they can't take all possible changes in the environment into consideration, so it's hard for them to tame the uncertainty. The online methods also have their disadvantage: most of them bring about huge computation cost and not efficient to deal with complex self-adaptive systems. Although GA is very effective, it still cost time to evolve the individuals. As the complexity of problem grows, GA will take more time. Due to the

above reason, it's hard for the system to deal with the changes in environment in time.

To tackle this problem, we use the output of GA (i.e. a lot of individuals) as training set to learn a prediction function which can determine how to configure the system under different environmental conditions. We exploit linear regression to derive the prediction function, since it's a widely-used technique to predict numerical data. The input of prediction function is the environmental condition; the output is the configuration of software system. We can obtain the prediction function offline and use it to reconfigure the system. When the system meets an unexpected condition at runtime, the prediction function can also give a proper system configuration in a short time. So, by combining genetic algorithm and linear regression, we avoid the defects of both online and offline adaptation rule generation methods.

In general, our method for adaptation rules generation is realized through a two-phase process as follows:

- Phase 1. Generation of training set. In this phase, we use GA to obtain some individuals which tend to be optimal. Each individual in the GA can be treated as an adaptation rule, which indicates the system configuration under different environmental conditions. This stochastic search process will continue until it reaches the termination criterion. There are different criteria to stop a GA. We can stop the GA after the best individual fitness does not change appreciably for a certain number of generations; or we can stop it after a preset number of iterations. In this paper, we use a combination of the above criteria.
- Phase 2. Generation of prediction function. In this phase, we build prediction model using a widely-used learning technique, called linear regression. The individuals generated in phase 1 are used as training set of the linear regression algorithm.

To verify the generality of our method, we apply our method on an unmanned underwater vehicle system [11] and an auction website [12]. We generate self-adaptive rules for them.

In summary, we make the following contributions:

1) We adapt GA and linear regression algorithms to generate adaptation rules, and we evaluate the rules to show they have ideal performance under dynamic environment and improve the adaptability of the system.
2) We propose a model for self-adaptive systems which describes the factors of a self-adaptive system and the relationship between them. This model is helpful to demonstrate our method.
3) We apply our method on two different systems. The experimental results show that our method is practical and highly-efficient in software reconfiguration under changing environmental conditions. Besides, our method has a certain degree of generality.

The outline of the rest of this paper is as follows. In Section 2, we introduce some background knowledge of our work. We give the motivation example in Section 3. Section 4 provides a detailed explanation of our approach. In Section 5, we apply the proposed method on two running examples and evaluate the effect of our approach. Section 6 discusses some related work and Section 7 finally concludes our work.

2. BACKGROUND

This paper concentrates on the generation of adaptation rules for software systems. Self-adaptive system embodies a closed-loop mechanism to control their dynamic behavior [1]. This loop, called the *adaptation loop*, consists of four key processes: monitoring, detecting, deciding, acting.

To demonstrate the structure of self-adaptive systems, we propose a conceptual model, which describes the main factors of a self-adaptive system and the relationship between them.

2.1 Adaptation Loop

This feedback loop, called the *adaptation loop*, consists of several processes, as well as sensors and effectors, as depicted in Figure 1. The adaptation processes, which exist at the operating phase can be summarized as follows [1]:

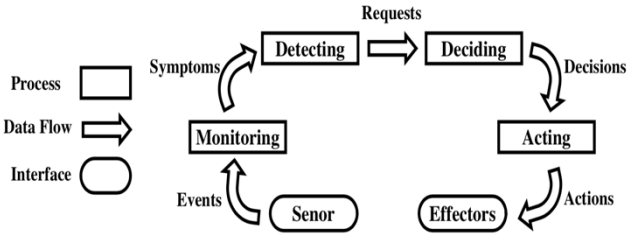


Figure 1. Feedback Loop in Self-Adaptive Software

—The **monitoring process** is used for collecting data from sensors. Then this data can be converted to behavioral patterns and symptoms. Threshold checking, as well as other methods, can be used to realize this process.

—The **detecting process** is used for analyzing the data collected by the monitoring process, the process helps detect *when* a response is required. It can also identify *where* the source of a transition to a new state is. The new state may deviate from desired states.

—The **deciding process** is used to determine *what* needs to be changed, and *how* to change it. This depends on certain criteria to compare different ways to achieve the best outcome.

—The **acting process** is used for applying the actions which are determined by the former process. By predefined workflows, this process can manage non-primitive actions or mapping actions to what is provided by effectors or certain dynamic adaptation techniques.

Except the above adaptation processes, sensors and effectors are also important parts of self-adaptive systems. Sensors can monitor software entities to generate a collection of data which reflect the system states; effectors depend on some mechanisms to apply changes. In other words, effectors realize adaptation actions.

2.2 Model of Self-adaptive Systems

A self-adaptive system can be described as a tuple as follows (shown in Figure 2):

<Environment, Rule, Software, Goal>

The self-adaptive system is deployed in the **environment** which consists of a lot of resource variants. The values of these variants represent the state of the environment.

In response to the changes, self-adaptive **software** evaluates its own behavior. Behavior should be changed when the evaluation indicates that it is not accomplishing what the software is intended

to do, in other word, the **goals** are not satisfied. The self-adaptive software can be described through feature model [13]. Feature is a software characteristic specified or implied by requirements documentation (for example, functionality, performance, attributes, or design constraints) [13].

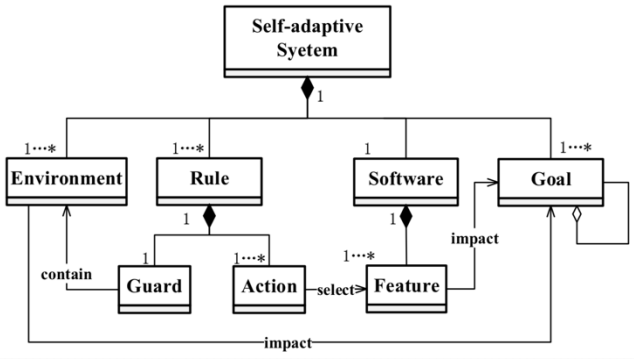


Figure 2. Conceptual Model of Self-Adaptive System

As a bridge between the environment and software, the adaptation **rules** play an essential role in self-adaptive systems. The rules which are driven by the changes in the environment, decide which adaptation actions should be taken and changes the system configuration. A good adaptation rule set can ensure the goals of a system to be satisfied well.

3. MOTIVATION EXAMPLE

We describe a UUV system (based on [11]) that we use as one of the cases to evaluate our proposed method in Section 5 and to illustrate the detailed description of our method in the next section. UUVs are increasingly used for a wide range of tasks. Here we look at UUVs used for oceanic surveillance, e.g., to monitor pollution of an area. The UUV must operate in a limited and disturbing environment: correct sensing may be difficult to achieve, communications may be noisy, etc., requiring the UUV system to be adaptive.

In addition, due to the stringent UUV requirements that the system should not have an impact on the ocean area, there is a need for guarantees. Besides, because vehicles are expensive equipment, they should not be lost during the mission.

The adaptive UUV system in our study for performing surveillance and data collection tasks is equipped with five on-board sensors that can measure the same properties of the marine environment (e.g., water current or salinity). Each sensor performs a scan with a certain speed and accuracy while consuming a certain amount of energy (see Table 1). A scan is performed every second.

Table 1. Parameters of Sensors of the UUV.

UUV on-board sensor	*Energy cons. J/s*	*Scan Speed m/s*	*Accuracy %*
Sensor1	170	2.6	97
Sensor2	135	3.6	89
Sensor3	118	2.6	83
Sensor4	100	3.0	74
Sensor5	78	3.6	49

The UUV system has two environmental resource variables [11]:

- U1: The average scan speed per second. A segment of surface over a distance of S should be examined by the UUV within a given time t;
- U2: The average energy consumption per second. To perform the mission, a given amount of energy E is available within a given time t;

The UUV system has to satisfy the following requirement:

- R1: Subject to U1 and U2, the accuracy of measurements should be maximized.

In order to fulfill these requirements, the sensor can be switched on and off dynamically during the mission. We assume that only one sensor is active at a time, however, we use a set of sensors for each adaptation period. For example, to perform a mission with energy consumption of 135 J/s we may either use Sensor2 100% of the time, or use Sensor1 50% of the time (using 170*0.5=85 J/s) and Sensor4 50% of the time (using 100*0.5=50 J/s).

The environmental resource variables U1 and U2 are crucial to the success of the monitoring task, but they may change at run time due to unpredictable events in the environment.

The general adaptation problem we have to solve is the following: In order to ensure that the needs of our users are met and the solutions are optimized, regardless of the possible fluctuations in the system parameters, requirement changes, and unpredictable environmental changes.

The UUV scenario offers one concrete instance of this general problem. Defining and developing an adaptive solution for this general problem introduces several key challenges. First, the appropriate adaptation sensors (measured variables) must be carefully chosen. Second, the software system must be modeled. Third, the appropriate adaptation mechanism that satisfies user's goals, while taming uncertainty, must be developed. Fourth, the system must incorporate an optimization approach to optimize the solution.

4. PROPOSED APPROACH

4.1 Overview

In this section, we will give an overview of our method to show how to generate adaptation rules of software system. Figure 3 shows the general process of our method.

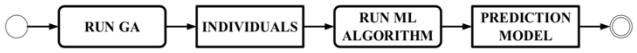


Figure 3. General Process of our method

As aforementioned in Section 1, our method consists of two phases. In the first phase, we exploit a genetic algorithm to get a training set as the input of phase 2. In this phase, we obtain some individuals which tend to be optimal. Each individual in the GA can be treated as an adaptation rule, which indicates the system configuration under different environmental conditions. Genetic algorithms are based on a stochastic search technique that includes a population of individuals, where each individual encodes a candidate solution in a chromosome [10].

In the second phase, we use linear regression, a widely-adopted machine learning algorithm, to train the data collected in the former phase. After this phase, we obtain a prediction function which has the same functionality with adaptation rules. Both the prediction function and adaptation rules decide the configurations of a system, by monitoring the changing environment. The input

of the function is the environmental conditions, and the output is the system configuration. Linear regression is a mature technique to predict numerical data, we apply it to our method because it's simple and effective. This algorithm may not be the only choice and can be replaced by other machine learning algorithms, but it's not in the scope of this paper.

Next, we will demonstrate the genetic algorithm which is the one of the main contribution of our work.

4.2 Genetic Algorithm Design

Representation. Developers must choose a suitable representation for encoding candidate solutions as individuals in a genetic algorithm. For example, in our method, every individual in the population encodes an adaptation rule.

To apply the genetic algorithm to generate training set for linear regression, we consider the entire training set as the population. So, the samples in the training set need to be encoded as individuals. An individual should contain the information of both the environment and software. Considering a self-adaptive system with *n* resource variables in its environment and *m* system configuration variables, the individual can be represented as follows:

$$\text{Individual} = (r_1, r_2, ..., r_n, f_1, f_2, ..., f_m)$$

In the UUV scenario, we encode the environment and the configuration of the UUV system as follows:

$$(u_1, u_2, x_1, x_2, x_3, x_4, x_5)$$

Where: x_j (with $j \in [1;5]$) is the portion of time (in decimals) the sensor j should be used during system operation; u_1 and u_2 are scanning speed and energy consumption of the system respectively.

The individual is subject to the following equations:

$$\begin{cases} E_1 \times x_1 + E_2 \times x_2 + \cdots + E_5 \times x_5 = u_1 \\ V_1 \times x_1 + V_2 \times x_2 + \cdots + V_5 \times x_5 = u_2 \\ x_1 + x_2 + \cdots + x_5 = 1 \end{cases}$$

Where: E_j (with $j \in [1;5]$) is the energy consumed by sensor j; V_j is the scanning speed of sensor j.

By using the above constrains, we can erase x_3 to x_5 because we can represent them by x_1 and x_2. So, the individual in the UUV scenario can be encoded as:

$$(u_1, u_2, x_1, x_2)$$

GA Operators. The crossover and mutation operators are specific to the encoding scheme used. The default crossover and mutation operator is designed for fixed-length binary string representation [10]. If a different representation scheme is used to encode candidate solutions, then specialized crossover and mutation operators need to be developed and used. For example, each individual in the UUV scenario encodes a candidate solution that comprises integer, and floating-point values. As a result, our method uses encoding-specific crossover and mutation operators to directly manipulate system configuration. The crossover operator exchanges the portion of time the sensor should be used between two parents. Likewise, the mutation operator randomly adds/subtracts the portion of time the sensor should be used.

4.3 Fitness Function Design

Self-adaptive systems usually contain three key elements: monitoring, decision-making, and reconfiguration. Monitoring enables applications to understand their environment and to detect conditions that will ensure reconfiguration; decisions determine which reconfiguration response should be triggered under a specific set of monitored conditions; reconfiguration enables the application to make changes to meet its requirements [14]. In our method, the user's requirements are presented by the fitness function of genetic algorithm.

In general, a single fitness function may be not sufficient to quantify all the possible effects of a particular reconfiguration plan in balancing multiple goals [15]. Instead, developers should define a set of fitness sub-functions to evaluate the reconfiguration plan based on the optimization dimensions specified by the user. Each fitness sub-function measures a single goal. Each adapted sub-function should have an associated coefficient to determine the importance or relevance of that sub-function to other functions. A weighted sum can be used to combine the values obtained from each fitness sub-function into one scalar value [16]. The final fitness function can be represented as follows:

$$fitness(x) = \sum_{i=1}^{s} w_i f_i(x) \qquad (1)$$

where *fitness*(*x*) is the fitness of individual *x*, *s* is the number of sub-functions, w_i is the weight of the *i-th* sub-function, f_i is the value of the *i-th* sub-function.

In the UUV scenario, since maximizing the accuracy of measurements is the only objective (i.e. user's requirement), we don't need to define fitness sub-functions. The fitness function is defined as follows:

$$Acc_1 \times x_1 + Acc_2 \times x_2 + \cdots + Acc_5 \times x_5$$

where: Acc_j is the accuracy of sensor *j*; and x_j (with $j \in [1;5]$) is the portion of time (in decimals) the sensor *j* should be used during system operation.

By the way, the decision-making problem in the UUV scenario has been transformed into an optimization problem:

Maximize *Accuracy*:

$$max[Acc_1 \times x_1 + Acc_2 \times x_2 + \cdots + Acc_5 \times x_5]$$

Subject to:

$$\begin{cases} E_1 \times x_1 + E_2 \times x_2 + \cdots + E_5 \times x_5 = u_1 \\ V_1 \times x_1 + V_2 \times x_2 + \cdots + V_5 \times x_5 = u_2 \\ x_1 + x_2 + \cdots + x_5 = 1 \end{cases}$$

Where: x_j (with $j \in [1;5]$) is the portion of time (in decimals) the sensor *j* should be used during system operation; Acc_j is the accuracy of sensor *j*; E_j is the energy consumed by sensor *j*; V_j is the scanning speed of sensor *j* (for the concrete values of Acc_j, E_j, V_j, see Table 1); and u_1 and u_2 are scanning speed and energy consumption of the system respectively.

4.4 Linear Regression

Linear regression is a widely-used machine learning algorithm to predict numeric values. The goal when using linear regression is to predict a numeric target value. One way to do this is to write out an equation for the target value with respect to the inputs. This is known as a regression equation. Once you've found the

regression weights, forecasting new values given a set of inputs is easy. All you have to do is multiply the inputs by the regression weights and add them together to get a forecast. This algorithm is easy to interpret results, and computationally inexpensive. So, it's suitable to decide what adaptation actions should be taken to achieve the best outcome. The general process of linear regression is described as follows:

- Collect: Collect the data for training.
- Prepare: We'll need numeric values for regression. Nominal values should be mapped to binary values.
- Train: Find the regression weights.
- Test: We can measure the R^2, or correlation of the predicted value and data, to measure the success of our models.
- Use: With regression, we can forecast a numeric value for a certain input.

Since the individuals generated by genetic algorithm are used as training set of the linear regression algorithm, we hope that the individuals can cover environmental state space as comprehensive as possible. We achieve this target by gridding the environmental state space.

To a certain environment variable *e*, the interval between *e* will be:

$$interval_e = \frac{e_{max} - e_{min}}{\sqrt[m]{N} - 1} \quad (2)$$

Where: e_{max} is the maximum value of *e*; e_{min} is the minimum value of *e*; *m* is the number of environmental variables; *N* is the size of training set.

For example, in the UUV scenario, if the size of training set is 10000, to the environment variable u_1 (i.e. average scan speed per second, $u_1 \in [2.6,3.6]$), the interval of u_1 will be (3.6-2.6)/(100-1) $\approx$ 0.01.

After running the linear regression algorithm, we produce a prediction function. The input of the function is the environmental conditions, and the output is the system configuration. The prediction function can be used to generate system configurations under the changing environment. In the UUV scenario, the prediction function takes the environmental condition (u_1, u_2) as its input.

5. EXPERIMENTAL EVALUATION

We empirically evaluate our method with two cases. First, we describe the experimental setting of the UUV case. Second, we show the software adaptation performed by our method when the environmental conditions of the system are changed at runtime. The second case with RUBiS is described and evaluated subsequently.

5.1 Experimental Setting: UUV Case

We use the UUV system described in Section 3 as a primary case to evaluate our method. The system is implemented in a Java simulation environment. The initial parameters of the sensors are specified in Table 1.

Adaptation is performed every 100 surface measurements of the UUV system: 1 k = 100 measurements, and a measurement is performed each second. At each adaptation step the application calculates the average measured value of the goal (i.e., scanning accuracy) during the past 100 measurements.

Developers must set up the GA for the problem they solve. Typical parameters include the population size, crossover and selection type, mutation rates, and so on [17]. Table 2 gives the different parameters and values used for our method. In particular, we perform "elitism" for each generation, which makes sure that the best individuals in a GA are retained in the population from one generation to the next.

Table 2. GA Parameters

Parameter	*Value*
Crossover type	Single-point crossover
Mutation rate	2%
Selection type	Roulette-wheel selection
Elitism	Replace worst 10
Population size	200
Initialization	Random(5N-N)
Ternination condition	Combined criteria

The meanings of some important parameters are illustrated as follows:

— **Mutation rate**. Mutation in biology is relatively rare, at least in so far as it noticeably affects offspring. In most GA implementations, mutation is also rare (on the order of 2%). However, it's hard to decide the correct setting of a GA mutation rate in general.

— **Elitism** is a way of making sure that the best individuals in a GA are retained in the population from one generation to the next. There is the danger that we might lose some of our best individuals from one generation to the next. We can implement elitism by producing *N* children and replacing the worst children with the best E individuals of the previous generation. Figure 4 shows a simple genetic algorithm modified for elitism.

```
Parents ← {randomly generated population}
While not (termination criterion)
    Calculate the fitness of each parent in the population
    Elites ← Best E parents
    Children ← Ø
    While |Children| < |Parents|
        Use fitness to probabilistically select a pair of parents for mating
        Mate the parents to create children c1 and c2
        Children ← Children ∪ {c1, c2}
    Loop
    Randomly mutate some of the children
    Parents ← Children ∪ Elites
    Parents ← Best N Parents
Next generation
```

Figure 4. A Genetic Algorithm Modified for Elitism.

— **Initialization**. Suppose we want to run a GA with *N* individuals. One initialization approach is to generate more than *N* individuals, and simply keep the best *N* as our initial population. For example, this research work [18] generates *5N* random individuals and keeps the best *N* as the initial population.

— **Termination condition** decided when to stop a GA. We can stop the GA after the best individual fitness does not change appreciably for a certain number of generations. Or we can stop it after a preset number of iterations. In this paper, we use a combination of the above criteria.

5.2 Adaptation Results

Through the experiment, we want to answer two research questions:

- RQ1: For any environment state, can we provide corresponding system configuration with high fitness value?
- RQ2: Can we obtain the system configuration in response to the changes in time?

Since the UUV scenario can be seen as a linear programming problem, we use linprog [19], which is a linear programming solver in Matlab, to get the exact solution of each environmental condition. Since we transform the decision-making problem into a constrained optimization problem, there will be no feasible solution for certain conditions. Figure 5 shows the feasible region of the UUV scenario. The x-axis represents the scanning speed of the vehicle and the y-axis represents the energy consumption per second. The blue area means there is no feasible solution for the corresponding environmental condition. For example, there is no system configuration which cost 100 J/s with the scanning speed 2.8 m/s.

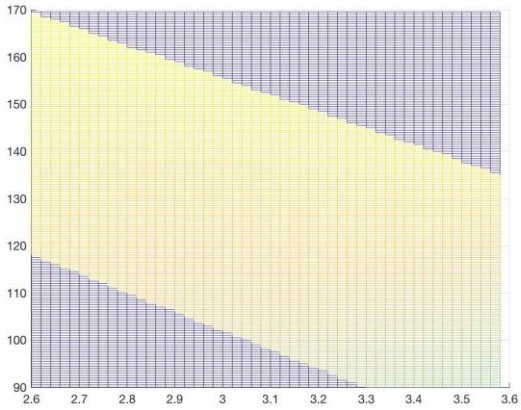
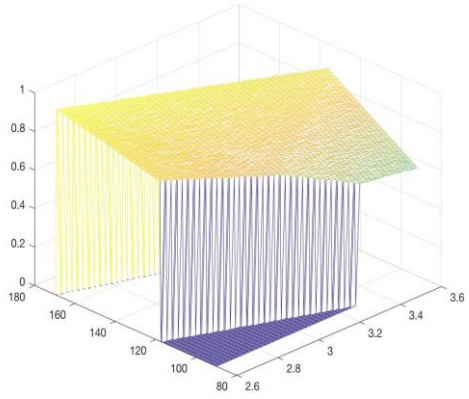


Figure 5. The Feasible Region of the UUV Scenario.

In order to verify the effectiveness of our method, we use the exact value computed by linprog as ground truth, and compare the adaptation results with the ground truth (shown as Figure 6).



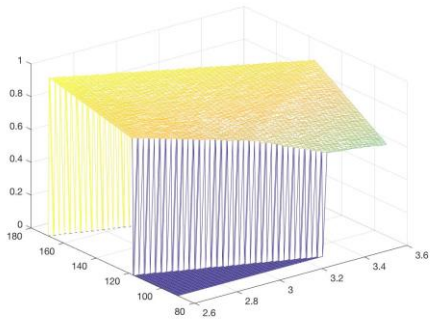


Figure 6. The Feasible Region of the UUV Scenario.

The x-axis and y-axis are of the same meaning as aforementioned before; the z-axis represents the scanning accuracy of the UUV system, which ranges from 0 to 1.

The left part in Figure 6 is the ground truth, and the right part is the results obtained by our method. We can see that the two part of Figure 6 are very similar. This means the solution of our method is close enough to the optimal one.

We define the error as follows:

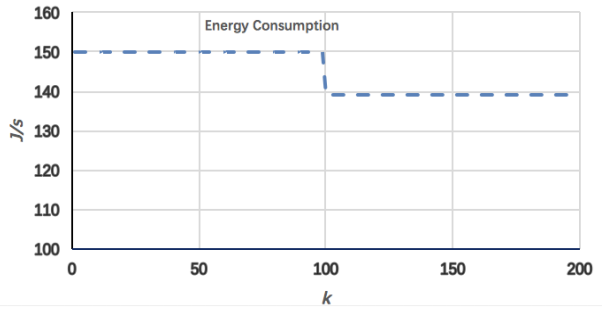
$$\Delta = \frac{Groundtruth(i) - GALR(i)}{Groundtruth(i)} \qquad (3)$$

Where: *Groundtruth*(*i*) is the scanning accuracy of the exact method under the environmental condition *i*; *GALR*(*i*) is the scanning accuracy of our method under the environmental condition *i*. In fact, the average error is 1.529% in the UUV scenario. So, the RQ1 has been well answered.

As we know, traditional online adaptation methods (e.g. control theory, genetic algorithm) should compute and get the solution at runtime. As the complexity of problem grows, they will bring about huge computation cost and time cost. During the computing period, the utility of the system will have a decline. The user's requirement may not be satisfied.

To illustrate that our method can respond to the changes quickly and avoid the decline of system utility, we compare the runtime effect of our method with the SimCA [11] adaptation method.

Figure 7 shows the changes on the UUV system configured according to Table 1. At k = 100 we change the available energy from 150 J/s to 139 J/s, at k = 160 we change the distance to be scanned from 2.7 m/s to 2.9 m/s.



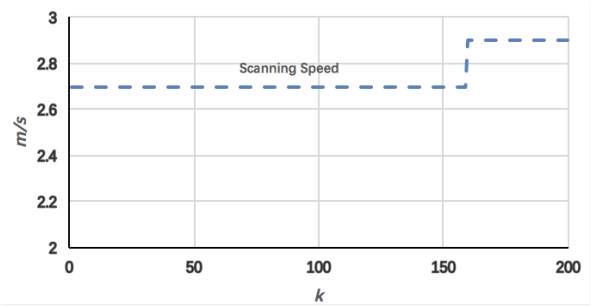


Figure 7. Different runtime changes in UUV System.

Figure 8 shows the scanning accuracy curve over time. It's oblivious that SimCA (blue line) need some time to converge to the nearest achievable value. However, our method (red line) can generate a new system configuration immediately in response to the changes. So, our method is more stable and the RQ2 can be answered.

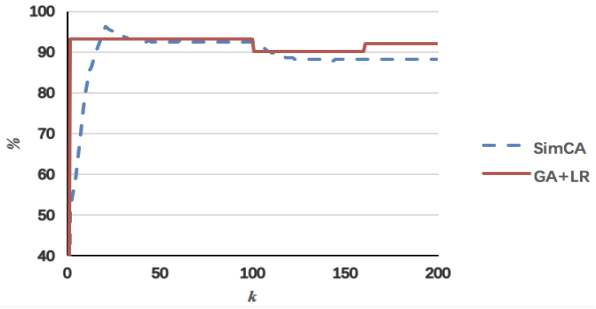


Figure 8. Adaptation of UUV according to different runtime changes.

5.3 Experimental Scenario 2: RUBiS

To show the generality of our method, we evaluate the approach with a second case: RUBiS [12].

RUBiS is a widely-used benchmark that implements an auction website, similar to eBay. It has been used in many research work, e.g. in [20, 21, 22]. We use a simple recommendation engine to expand RUBiS, the users will be recommended for some items which they may have interest in. Such a recommendation engine adds significant value to the user experience, thereby increasing the owner's revenue.

In the RUBiS scenario, we encode the environment and the configuration of the system as follows:

(resource, load, recommendation)

Where: resource (resource $\in$ [0;800%]) is the amount of CPU resources allocated to RUBiS; load (load $\in$ [100;800]) is the application's load — the number of connected users; and recommendation (recommendation $\in$ [0;100%]) is the recommendation rate for each request.

As aforementioned before, a single fitness function may be not sufficient to quantify all the possible effects of a particular reconfiguration plan in balancing multiple goals. In this scenario, there are 2 sub-functions measure different objectives: increasing the throughput of web server; and reducing the number of requests out of time. We assume that each sub-function has the same coefficient in this experiment.

We report two experiments, each focused on different time-varying aspect. First, we change the resources available to the application. Second, we change the application load (i.e. the number of connected users).

Figure 9 and Figure 10 are structured as follows. The bottom x-axis represents the time elapsed since the beginning of the experiment. Each figure presents a single experiment.

We compare our method with Plato, a genetic-algorithm based decision-making process [14]. Unlike our approach, Plato does not have a linear regression process which generate a predict model offline.

In Figure 9, the fitness values with Plato method (benchmark) are shown in blue points and its y-axis is depicted on the left side of the plot, however the fitness values with our method are shown in red point. At k = 10, we change the allocated CPU resource from 200% to 400%, at k = 20 we change the available CPU resource from 400% to 600%. Figure 9 shows that our method performs better and take shorter time to converge to a high fitness value when environmental condition changes during our experiment.

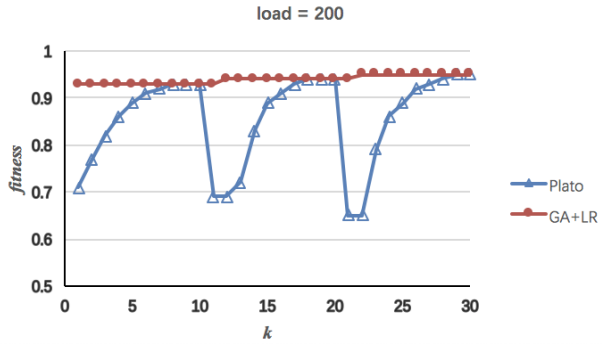


Figure 9. Adaptation of RUBiS, Varying the Resource Allocation.

In the second experiment, we set the CPU allocation to 400% and keep it constant, then we vary the number of users who are accessing RUBiS. In a real data center, this may happen due to a flash crowd.

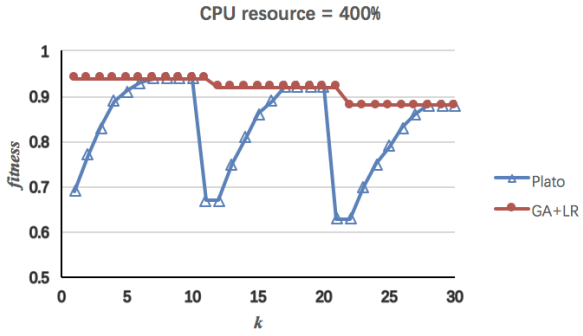


Figure 10. Adaptation of RUBiS, Varying the number of users.

Let us now discuss the results in Figure 10. At k = 10, we change the workload from 200 to 400, at k = 20 we change the workload from 400 to 800. We can make similar conclusions with the former experiment with Figure 9. It is noteworthy that when the load increased to 800 from 400, the utility decreased. We can explain this phenomenon because the CPU resources are not adequate to response so many requests from users.

5.4 Time Consuming Analysis

As shown in Figure 8, when we change the environmental conditions at k = 100, the SimCA method needs more than 20 time units to regain a stable utility value of the UUV system. Furthermore, at k = 160, SimCA does not even trigger system reconfiguration depending on the changes of environment. As shown in Figure 9 and Figure 10, in the RUBiS scenario, when we change the environmental conditions at k = 10 and k = 20, the Plato method takes about 10 time units to make the fitness converge to a stable value. This is because both SimCA and Plato are online decision-making methods, they need take some time to evolve an acceptable solution.

However, our method learns a prediction function off-line, so a good solution can be obtained directly at runtime within 1 time unit. This result proves that our method is less time-consuming and more highly-efficient in software reconfiguration under changing environmental conditions at runtime. More details can be seen in Table 3, where: "change point" is the moment we change the environmental conditions; "TCC", which stands for **T**ime **C**onsumed to **C**onverge, represents the time units that a method takes to converge to a certain value after the environment changes.

Table 3. Time Consumed in Online Decision-Making

Scenario	*Change Point*		*TCC*	
			SimCA	***GA+LR***
UUV	k = 100		23	1
	k = 160		0	1
			Plato	***GA+LR***
RUBiS	Varying the Resource Allocation	k = 10	8	1
		k = 20	9	1
	Varying the number of users	k = 10	7	1
		k = 20	8	1

6. RELATED WORK

Self-adaptation is playing a key role in the development of software systems [23, 24, 25]. The self-adaptive rules specify how to react to changes in environment. There are numbers of current research works which focus on the automatic generation of adaptation rules.

Control theory has proven to be a useful tool to introduce adaptability in software systems [12, 26, 27, 28, 29]. Although there have been many attempts to apply control theory to computing systems [30], the study is still in a preliminary stage.

Many researchers exploit probabilistic model checking to support non-functional adaptation [3, 4], this technique can perform online-evolution of adaptation rules. Goal-based reasoning [5, 6, 7] is another way to generate adaptation rules. The adaptation planner chooses software configurations that can optimize non-functional goals. The above researches have gained remarkable achievement. However, these methods always bring about huge computation cost and not efficient to deal with complex self-adaptive systems. As mentioned in [8], the genetic algorithm is a widely-used technique to resolve optimization problems and can be used in the field of self-adaptive systems. Besides, the genetic algorithm can decrease computation cost by reducing the search space.

Some researchers have applied genetic algorithm on the self-adaptive system [14, 17, 31, 32]. They use genetic algorithm to help the system make decision at runtime. The above online genetic algorithm methods also have their disadvantage: most of them bring about huge computation cost and not efficient to deal with complex self-adaptive systems. These methods can't respond the changes very quickly, because they still cost time to evolve the individuals.

The main novelty of our work is the design of a two-phase method. We use the output of GA as training set to learn a prediction function through linear regression. The prediction function is used to determine how to configure the system under different environmental conditions. We obtain the function offline and execute it at runtime to realize online decision-making. By combining genetic algorithm and linear regression, we avoid the defects of both online and offline adaptation rule generation methods.

Feature model is a promising tool to model variants and variation points [13, 33]. In this paper, we use the concept to model the self-adaptive software and propose a model of self-adaptive system based on it.

7. CONCLUTION AND DISCUSSION

Since it's hard to efficiently generate effective adaptation rules automatically, we try to resolve this problem in this paper. We propose a method which combines genetic algorithm and linear regression algorithm to generate adaptation rules. To evaluate the method, we apply our method on two different systems. The experimental results show that our method is practical and highly-efficient in software reconfiguration under changing environmental conditions. Besides, we propose a model for self-adaptive systems which describes the factors of a self-adaptive system and the relationship between them.

Our method has the following limitations: 1) this method can be used only when the adaptation rule generation problem can be transformed into an optimization problem, which needs to find the optimal solution or the sub-optimal solution from multiple alternatives by searching a huge space. 2) Meanwhile, if the system has multiple goals, in order to design the fitness function of genetic algorithm, these goals need to be integrated into one, through a certain function. This problem can be tackled by using NSGA (Nondonminated Sorting in Genetic Algorithms), but it's outside the scope of this article.

In future work, we want to extend our method with a nondonminated sorting genetic algorithm, which is a more realistic method that can find different Pareto-optimal solutions simultaneously. In this way, we don't have to combine multiple goals into one. Besides, we will apply our method to more applications to evaluate its generality.

8. ACKNOWLEDGMENTS

We would like to thank the anonymous referees, whose useful comments have greatly improved the presentation of this paper. This work is partially sponsored by the National Basic Research Program of China (973) (2015CB352200), and the National Natural Science Foundation of China (61620106007).

9. REFERENCES

[1] Salehie M, Tahvildari L. Self-adaptive software: Landscape and research challenges[J]. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2009, 4(2): 14.

[2] Zhao T, Zhao H, Zhang W, et al. User preference based autonomic generation of self-adaptive rules[C]//*Proceedings of the 6th Asia-Pacific Symposium on Internetware on Internetware*. ACM, 2014: 25-34.

[3] Filieri A, Tamburrelli G, Ghezzi C. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time[J]. *IEEE Transactions on Software Engineering*, 2016, 42(1): 75-99.

[4] Moreno G A, Cámara J, Garlan D, et al. Proactive self-adaptation under uncertainty: A probabilistic model checking approach[C]//*Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015: 1-12.

[5] Chen B, Peng X, Yu Y, et al. Self-adaptation through incremental generative model transformations at runtime[C]//*Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014: 676-687.

[6] Qian W, Peng X, Chen B, et al. Rationalism with a dose of empiricism: Case-based reasoning for requirements-driven self-adaptation[C]//*2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 2014: 113-122.

[7] Zhang Y, Guo J, Blais E, et al. Performance prediction of configurable software systems by fourier learning (T)[C]//*Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015: 365-373.

[8] Simon D. Evolutionary optimization algorithms[M]. John Wiley & Sons, 2013.

[9] Liu Y, Zhang W, Jiao W. A generative genetic algorithm for evolving adaptation rules of software systems[C]//*Proceedings of the 8th Asia-Pacific Symposium on Internetware*. ACM, 2016: 103-107.

[10] Koza J R. Genetic programming: on the programming of computers by means of natural selection[M]. MIT press, 1992.

[11] Shevtsov S, Weyns D. Keep it SIMPLEX: Satisfying multiple goals with guarantees in control-based self-adaptive systems[C]//*Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016: 229-241.

[12] Klein C, Maggio M, Årzén K E, et al. Brownout: Building more robust cloud applications[C]//*Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014: 700-711.

[13] Acher M, Collet P, Fleurey F, et al. Modeling context and dynamic adaptations with feature models[C]//*4th International Workshop Models@ run. time at Models* 2009 (MRT'09). 2009: 10.

[14] Ramirez A J, Knoester D B, Cheng B H C, et al. Applying genetic algorithms to decision making in autonomic computing systems[C]//*Proceedings of the 6th international conference on Autonomic computing*. ACM, 2009: 97-106.

[15] Deb K. Multi-objective optimization using evolutionary algorithms[M]. *John Wiley & Sons*, 2001.

[16] Montana D, Hussain T. Adaptive reconfiguration of data networks using genetic algorithms[J]. *Applied Soft Computing*, 2004, 4(4): 433-444.

[17] Ramirez A J, Cheng B H C, McKinley P K, et al. Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration[C]//*Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010: 225-234.

[18] Bhattacharya M. Reduced computation for evolutionary optimization in noisy environment[C]//*Proceedings of the 10th annual conference companion on Genetic and evolutionary computation*. ACM, 2008: 2117-2122.

[19] Henningsen A. linprog: Linear Programming[J]. *Optimization*. R package version 0.9-0, 2010.

[20] Chen Y, Iyer S, Liu X, et al. SLA decomposition: Translating service level objectives to system level thresholds[C]//*Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*. IEEE, 2007: 3-3.

[21] Shen Z, Subbiah S, Gu X, et al. Cloudscale: elastic resource scaling for multi-tenant cloud systems[C]//*Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM*, 2011: 5.

[22] Weyns D, Iftikhar M U, Malek S, et al. Claims and supporting evidence for self-adaptive systems: A literature study[C]//*Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2012: 89-98.

[23] Cheng B H C, De Lemos R, Giese H, et al. Software engineering for self-adaptive systems: A research roadmap[M]//*Software engineering for self-adaptive systems*. *Springer Berlin Heidelberg*, 2009: 1-26.

[24] Kephart J O. Research challenges of autonomic computing[C]//*Proceedings of the 27th international conference on Software engineering*. ACM, 2005: 15-22.

[25] Kramer J, Magee J. Self-managed systems: an architectural challenge[C]//*2007 Future of Software Engineering. IEEE Computer Society*, 2007: 259-268.

[26] Brun Y, Serugendo G D M, Gacek C, et al. Engineering Self-Adaptive Systems through Feedback Loops[J]. *Software engineering for self-adaptive systems*, 2009, 5525: 48-70.

[27] Diao Y, Hellerstein J L, Parekh S, et al. A control theory foundation for self-managing computing systems[J]. *IEEE journal on selected areas in communications*, 2005, 23(12): 2213-2222.

[28] Weyns D, Iftikhar M U, De La Iglesia D G, et al. A survey of formal methods in self-adaptive systems[C]//*Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*. ACM, 2012: 67-79.

[29] Filieri A, Ghezzi C, Leva A, et al. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements[C]//*Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society*, 2011: 283-292.

[30] Patikirikorala T, Colman A, Han J, et al. A systematic survey on the design of self-adaptive software systems using control engineering approaches[C]//*Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2012: 33-42.

[31] Pascual G G, Pinto M, Fuentes L. Run-time adaptation of mobile applications using genetic algorithms[C]//*Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2013 ICSE Workshop on*. IEEE, 2013: 73-82.

[32] Ramirez A J, Knoester D B, Cheng B H C, et al. Plato: a genetic algorithm approach to run-time reconfiguration in autonomic computing systems[J]. *Cluster Computing*, 2011, 14(3): 229-244.

[33] Fleurey F, Solberg A. A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems[J]. *Model Driven Engineering Languages and Systems*, 2009: 606-621.