Scientia Iranica D (2015) 22(3), 1018–1030

SCIENTIA IRANICA

Sharif University of Technology

Scientia Iranica

Transactions D: Computer Science & Engineering and Electrical Engineering

www.scientiairanica.com

Research Note

A multi-objective approach to model-driven performance bottlenecks mitigation

M. Amoozegar[a,*] and H. Nezamabadi-pour[b]

a. *Department of Information Technology, Institute of Science and High Technology and Environmental Sciences, Graduate University of Advanced Technology, Kerman, P.O. Box 76315-117, Iran.*

b. *Department of Electrical Engineering, Shahid Bahonar University of Kerman, Kerman, P.O. Box 76169-133, Iran.*

Received 25 December 2013; received in revised form 1 October 2014; accepted 7 March 2015

KEYWORDS

Bottleneck detection;
Multi-objective optimization;
Software performance engineering;
UML;
Gravitational search algorithm.

Abstract. Software Performance Engineering (SPE) evaluates the key performance factors such as response time and utilization in the entire life cycle of software development. One of the important issues of software performance is bottlenecks that have not been investigated much till now in the process of SPE. Meanwhile, Bottleneck detection and mitigation in software modeling stage is quality-centered and cost effective. Layered bottleneck is a type of bottleneck that occurs in systems with layered services and affects its utilization more than flat bottlenecks. The presented approach in this paper has selected Layered Queening Network (LQN) as an appropriate performance model to present and analyze the layered bottlenecks. The process of SPE from software model to performance model has been automatically implemented. Also, an optimization stage is added to find the best specification of software model in a way that the strength of the bottleneck, the response time and the cost will be minimized. To assess the proposed solution, two recently proposed multi-objective gravitational search algorithms are employed. To evaluate the effectiveness of the applied algorithms, two well-known multi-objective algorithms: NSGA-II and MOPSO are also applied to a case study, and a comprehensive comparison is presented.

1. Introduction

Software performance engineering integrates performance evaluation in the software development process from the early stages throughout the whole life cycle. Therefore, software model annotated with performance specification is transformed into a performance model. Recent research of SPE focused on feedback from analyzing performance results into software model.

Performance usually refers to response time; throughput and resource utilization. Many researches have been presented to improve these factors. Meanwhile SPE also covers the other performance issues such as bottlenecks. Although high utilization of resources is a main target, excessive utilization causes bottlenecks. A bottleneck is a single point of contention that limits the overall system performance [1].

Layered bottleneck is a type of bottleneck that occurs in systems with layered services. In this layered structure, each service in a layer is client for lower layer service, and server for upper layer service and resources are possessed simultaneously. Layered bottlenecks are more complicated than flat bottlenecks and the workload intensity of the system and the resources utilization are more affected. Therefore, one of the key factors in the software performance optimization is bottleneck detection and improvement.

Many bottleneck detection and improvement ap-

*. *Corresponding author. Tel.: 03433776611*
E-mail addresses: amoozegar@kgut.ac.ir (M. Amoozegar); nezam@kgut.ac.ir (H. Nezamabadi-pour)

proaches have been presented. Some approaches are based on simulation and monitoring. Ref. [2] presented an approach that supports automated monitoring, analyzing, and reporting by applying machine-learning in the context of staging. Therefore, this approach collects data and analyzes bottlenecks for a significant number of performance metrics.

Ref. [3] has presented an approach for simple and automatic detection of performance bottleneck based on the differential analysis method. This approach generates different binary variants obtained by patching individual or groups of instructions. Then the cost of an instruction group is evaluated. Differential Analysis is illustrated by the use of DECAN on a range of HPC applications to detect performance bottlenecks.

Many other approaches detect bottlenecks from runtime or source-code information [4-7]. Based on runtime information, a PERT (Program Evaluation and Review Technique) diagram is built, and the critical path of the diagram is identified as a performance bottleneck.

These methods are time consuming and very costly. The bottlenecks are detected in the last phase of software development so that their mitigation is very difficult.

Model based techniques for bottleneck detection evaluate software model. Therefore, in the early life cycle of software development, bottlenecks can be detected and mitigated by tuning the related performance specifications.

Based on [8], in performance model side, bottleneck identification and removal was introduced few decades ago [1] and has been continuously refined by more recent results [9]. But, there exist few works that detect such bottlenecks based on UML models. Garousi [10] presented a UML-driven technique to detect Performance bottlenecks of concurrent real-time systems. The control flow diagram and PERT diagram are built from the sequence diagrams and interaction overview diagrams to pinpoint performance bottlenecks.

This paper presents an automatic UML based bottleneck improvement solution that optimizes strength bottleneck [9] of the software model. Therefore, bottleneck detection and mitigation problem is modeled as an optimization problem that has considered more than one objective function (in this study three objective functions are considered together) and is solved with the Pareto-optimal concepts.

In recent years, some approaches [11,12] have been proposed that apply heuristic algorithms to evaluate the UML model for performance in terms of expected response time, throughputs and resource utilizations. Ref [13] has applied multi-objective evolutionary optimization to find the good value of the performance specifications of the software modeled with Palladio Component Model.

Gravitational Search Algorithm (GSA) is a relatively new optimization algorithm based on the laws of gravity and motion. In Ref. [14] we presented an approach to optimize the response time of the software model along with two constraints, cost and utilization, by single objective version of GSA. Single objective optimization algorithm provides one solution or one choice for the software architecture. Meanwhile, in multi-objective optimization algorithms, instead of a unique optimal solution, there is rather a set of alternative trade-offs. This research applies three multi-objective versions of GSA that are called NSGSA (Non-dominated Sorting Gravitational Search Algorithm) and MOGSA (Multi Objective Gravitational Search Algorithm) to find the best configurations of software model in a way that bottleneck strength be minimized. Bottleneck must be improved while cost and response time must also be controlled. Thus three objectives exist that must be minimized.

The privileges of the presented approach are:

1. The software model that is specified with performance property automatically analyzed from two important performance issues; bottleneck and response time. Bottlenecks, especially software bottlenecks, are affected by the software and hardware resources. By the presented approach, the optimum number of resources will be determined, before the software development starts. This process is very cost effective especially when the role of the software resources, e.g. threads, in the development process has been considered.
2. The bottleneck improvement is considered as the main objective during the SPE process. The software architecture who is not aware of the performance engineering can easily optimize the software model.
3. NSGSA and MOGSA are applied in SPE scope and bottleneck improvement problem with respect to response time and cost.

A case study has been selected and whole steps of SPE, from software modeling to optimization, have been applied to it. NSGA-II and MOPSO are also used for optimization. Important metrics such as coverage metric, diversity and spacing evaluate the performance of these algorithms. Comprehensive comparison of the results has been provided that show the NSGSA has the best behavior.

The rest of the paper is structured as follows: Basic concepts are discussed in Section 2. Section 3 describes the bottleneck detection and the measurement strategy. Formal definition of an optimization problem is presented in Section 4. Introduction of the

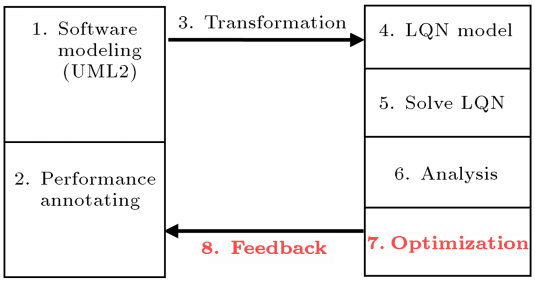

Figure 1. Software performance engineering process.

Case study, simulation and the analysis of the results are illustrated in Sections 5 and 6. The final section is related to a conclusion and future work.

2. Background

2.1. Software performance engineering

Software Performance Engineering (SPE), firstly presented in [15], is a systematic, quantitative approach to construct software systems that meet performance objectives. It is based on the careful and methodical evaluation of performance attributes throughout the lifecycle, from requirements and specification to implementation and maintenance [16].

Figure 1 shows the steps of SPE. The first step is the designing of software model in UML2 with activity, component and deployment diagram. In the second step, the MARTE [17] profile extends software model to annotate performance requirements and specify performance property of resources and activities using stereotypes and tagged values. Steps three and four are related to software model transform into CBML by means of existing algorithms and tools that have been provided in our already research [18]. CBML is an XML based language designed for describing layered queuing models with embedded components, and also the component sub-models [19,20], the LQN model will be solved with LQNS solver [21] in step 5. LQNS uses analytical mean value queuing approximations to solve the queues at all entries. Analysis is done in step 6 with SPEX which supports experiment instrumentation [22]. This tool can execute parameterized experiments using LQNS, and is very useful for repeating a parameterized run. In the final step, a search is performed through the space of the problem and the best value for performance parameter annotated in software model will be presented finally. All of these steps are used for an E-shop software case study.

A key factor in the successful performance analysis is the automation. Over the last decade, many researches have been directed towards integrating performance analysis into the software development process [23]. In this way, a software architecture who is not proficient in performance domain can easily design the software with high performance.

2.2. Bottleneck

A bottleneck is a single point of contention that limits the overall system performance [1].

LQN models [24] extend the traditional queuing network models by considering both software and hardware contention, and the impact of layers on service time. LQN model has layered structure. Each layer is the server of the upper layers and a client for the lower layers.

When a software task has been fully utilized, while the resources that it uses are underutilized, a software bottleneck has accrued [1]. Hardware bottlenecks usually occur at a CPU, a disk, or other devices. Software bottlenecking is quite different from hardware bottleneck. When a software task is highly utilized, it will "push back" on its clients which makes them appear to be saturated, too. This is while the used resources by this task are underutilized. In the systems with deeply layered structure, software blocking extends to other layers quickly [25]. A detailed discussion of software bottlenecking and their features can be found in [1].

2.3. Multi-objective optimization

Some problems have more than one objective function to be optimized and sometimes these objectives are in conflict with each other. Multi-objective optimization algorithms are presented to solve these problems. Since these algorithms provide more than one solution, the trade-off between objectives is necessary to find the best solution.

Suppose that $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ is the vector of decision variable, $f_i : R^n \to R, i = 1, \cdots, k$ are the objective functions and g_i, $h_j : R^n \to R$, $i = 1, \cdots, m$, $j = 1, \cdots, p$ are the constraints of the problem. Pareto domain based algorithms that are used in this paper, present this problem and definition as follows [26]:

Minimize:

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_k(\mathbf{x})].$$

Subject to:

$$g_i(\mathbf{x}) \leq 0, \qquad i = 1, 2, \cdots, m,$$

$$h_j(\mathbf{x}) = 0, \qquad j = 1, 2, \cdots, p.$$

Definition 1: Given two vectors $\mathbf{x}$, $\mathbf{y} \in R^k$, we say that $\mathbf{x} \leq \mathbf{y}$ if $x_i \leq y_i$, $i = 1, \cdots, k$ and that $\mathbf{x}$ dominates $\mathbf{y}$ (or presented by $\mathbf{x} \prec \mathbf{y}$) if $\mathbf{x} \leq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$.

Definition 2: we say $\mathbf{x} \in \chi \subset R^k$ is non-dominated with respect to χ, if there does not exist another $\mathbf{x}' \in \chi$ such that$\mathbf{f}(\mathbf{x}') \prec \mathbf{f}(\mathbf{x})$.

Selected non-dominated solutions are stored in an external archive. The external archive is updated at

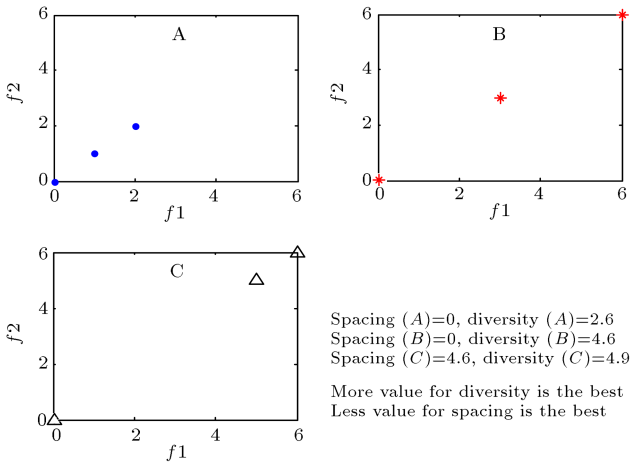

Figure 2. Diversity and spacing concept.

each iteration of the algorithm. When the running algorithm is terminated, external archive will present final results.

Therefore, multi-objective optimization algorithms provide a set of solutions that is required to be well-spread and uniformly covering wide areas of the Pareto front. To evaluate the performance of these algorithm, two important metrics have been defined. The spread of the solution set is measured by using the diversity metric [27]. The second metric is spacing that demonstrates how uniformly the solutions in the objective space are distributed [28]. These metrics are measured based on the described formula in Section 6. Figure 2 shows the concept of these metric for three solution sets A, B and C in a two-objective problem (f_1 and f_2).

3. Bottleneck detection and measurement strategy

Bottleneck detection and mitigation can be considered as one of the principal targets of SPE as mentioned above. Layered bottleneck is a type of bottleneck that is related to layered services. In LQN model, total service time of each server or resource, includes its service time and holding time and service time of other resources are used in nested form. Therefore, when a service includes other services and they are waiting, they possess resources in the form of nested simultaneous.

A directed graph can be used to model dependencies between recourses. A node presents each resource A, and an arc connects A to their dependent resources. Details and related equations are described in [9]. Basic parameters and equations of A to measure bottleneck are:

- m_A = multiplicity of A;
- X_A = service time of A;
- W_A = waiting time for requests to A;
- R_A = response time = $W_A + X_A$;
- f_A = throughput of resource A;
- $U_A = f_A X_A$ = utilization of A;
- sat_A = the saturation level of $A = U_A/m_A$.

Ref. [1] presents two observations and two definitions:

Observation 1: When a task is a software bottleneck, it is saturated but its servers are not.

Observation 2: A saturated task or processor tends to saturate tasks which use it by rendezvous, so bottlenecks tend to spread saturation in a rendezvous net back along the request arcs.

Definition 3: A software bottleneck occurs when a task (or a set of task threads) exhibits a high utilization, which is also high relative to the utilization of each of its servers, either direct or indirect.

Definition 4: A measure for the bottleneck, called the bottleneck strength, is the ratio of its utilization to that of its most highly utilized server, direct or indirect.

Based on these observations and definitions, a "bottleneck strength" measure, that has been defined in [1] and completed in [9], is presented as:

$$B\text{Strength}_A = \frac{\text{sat}_A}{\text{sat}_{\text{Shadow}(A)}},$$

$$\text{Shadow}(A) = \max_{a \in \text{Below}(A)} \text{sat}_a. \quad (1)$$

$B\text{Strength}_A$ is the strength of the bottleneck at A and, Below (A) is the set of resources that A depends on, directly or indirectly.

A layered bottleneck accrues when a statured resource bounds the throughput of the system.

Sat* shows the resource saturation threshold that must be initialized before the bottleneck detection. These conditions must be checked to detect the bottlenecks [9].

If sat value of one or more resources (for example a) in Processors is more than sat* then:

$$B = \max_{a \in \text{Processors}} \text{sat}_a. \quad (2)$$

Else, sat value of one or more other resources is more than sat*, B is any resource which satisfies both of:

$$\text{sat}_B > \text{sat}^*,$$

$$B = \max_{A \notin \text{Processors}} B\text{strength}_A. \quad (3)$$

Therefore, both software and hardware resources are considered. Also, LQN provides a simple manner to model and analyze the bottlenecks.

4. Problem modeling

In this section, the problem formulation, solution space and objective functions are precisely described. Finding the best values for performance attribute of these recourses is very important. Design options or performance specifications are annotated in software model by MARTE profile and describe a particle. Therefore, suppose that s is the number of servers and t is the number of multi-threaded object or task; the decision vector is defined by $\mathbf{M}$ whose elements are, respectively, chosen from the set $\{\text{cpus}_i, i = 1 \cdots s, \text{cpu}_i, i = 1 \cdots s, \text{thrm}_j \text{ and } j = 1 \cdots t\}$, where:

1. cpus: CPU speed of each server in deployment diagram annotated with speedfactor tag value of Resource stereotype;
2. cpum: CPU multiplicity of each server in deployment diagram annotated with Resmult tag of Resource stereotype;
3. thrm: Multiplicity of thread for each active object in UML diagram annotated with poolsize tag of PaRunTInstance stereotype.

The most important software resources are threads that have important effects on the performance of the software. Determining the number of threads in the software modeling stage facilitates and accelerates the software developing process. This resource is equivalent to the software task in LQN.

Also, we can add other important parameters to evaluate performance carefully, but we restrict adding the number of parameters to control the complexity of the problem.

Objective functions are as follows. The first objective is $F_1(M)$ that minimizes the response time. Response time is annotated with respTime tags in activity diagram and is measured by LQNS tools after model transformation.

The second objective is Total cost that controls and restricts the number of resources indirectly. Although more recourse decrees response time and bottleneck, the total cost must be limited. The total cost is defined as:

$$F2 = \sum_{k=1}^{s} H * \text{cpus}_k * \text{cpum}_k, \tag{4}$$

where H is the speed coefficient.

The third objective is the bottleneck strength that must be minimized. This objective function is calculated from Eq. (1). In this stage, the bottlenecks and their strength are determined according to the defined instruction in Section 3. The maximum value of BStrength is considered as the strength of the bottlenecks.

In the optimization process, the number of software and hardware resources which are known as the performance specifications, in the software model, have been determined automatically. Two objectives, response time and strength bottleneck, increase the capacity of the system, but total cost, F_2, equilibrates and controls the used resources.

4.1. Optimization algorithms

The selected algorithms to optimize the mentioned problem are divided into 3 categories: The first category is related to multi-objective algorithms based on the gravitational search algorithm or GSA. GSA has been inspired from the law of gravity and mass interactions that uses the theory of Newtonian physics. Searcher agents are the collection of masses that make an isolated system. Every mass can be informed of the position of other masses. Different masses transfer their information to other masses using the gravitational force. Detailed and related equations are given in [29].

Non-dominated sorting Gravitational search algorithm or NSGSA [30] and Multi-objective GSA or MOGSA [31] are selected for the first category.

NSGSA update the gravitational accelerations by using the non-dominated sorting approach. Some elitism mechanisms exist in these algorithms which have been provided by an external archive of the Pareto optimal solutions. Non-dominated solutions are added to the external archive. The length of the archive is limited, therefore, once the length of the external archive is violated, one member must be removed. The deviation of the members crowding distances from the average is calculated to select a member that must be removed. Sign and reordering mutations are two proposed operators that keep diversity within the moving masses. Figure 3 shows the pseudo code of this algorithm.

MOGSA also uses an external archive to reserve the non-dominated solutions and update it the same as Simple Multi-Objective PSO (SMOPSO) [32]. The used mechanism to control archive is elimination par-

```
Initialization ()
t = 1
While t < t max
    Evaluate fitness of each particle
    Update external archive ()
    Non-dominated sorting ()
    Update the list of moving particles ()
    Update the mass of moving particles ()
    Update particles acceleration ()
    Update particles velocity ()
    Update and mutate particles position ()
    t = t + 1
EndWhile
Report Pareto optimal solutions stored in archive
```

Figure 3. Pseudo-code of NSGSA.

ticle from crowded areas. For this purpose, objective functions and solution space are divided into hyper-rectangles. When archive overflow accrues, one particle from the most crowded hyper-rectangle is randomly selected and removed. In MOGSA, the distance between each particle to its nearest neighbor is calculated and its mass is updated. Distribution of archived particles is done by using the niching technique. In this algorithm, the archived particles apply gravitational force.

NSGA-II is a multi-objective extended version of the genetic algorithm that is based on non-dominated sorting solutions [33]. This algorithm is chosen because of its similarity with NSGSA.

SMPOS is a multi-objective version of PSO. This algorithm is chosen because of its similarity with MOGSA.

Therefore, two multi-objective versions of the new gravitational search algorithm are selected to evaluate their behavior in the bottleneck improvement scope.

The second reason for these selections is providing the possibility of compression gravitational based algorithms with similar genetic based and PSO based multi-objective algorithms.

5. Case study

E-shop is a software system that provides user interface to purchase and payment items. Details of this system are presented in component, deployment and activity diagrams (Figures 4-7).

This system is modeled with UML2 in Enterprise Architect tools. Figure 4 shows a component diagram which models functionality of system in high level, Figure 5 shows a deployment diagram which specifies hardware framework, and Figure 6 shows an activity diagram that presents the behavior of the system.

After modeling functionality of the software, we must specify non-functional features, for example performance. Therefore, annotate model with MARTE profile. Tables 1, 2 and 3 show needed performance annotations and related elements of the model.

The software model was transformed into CBML

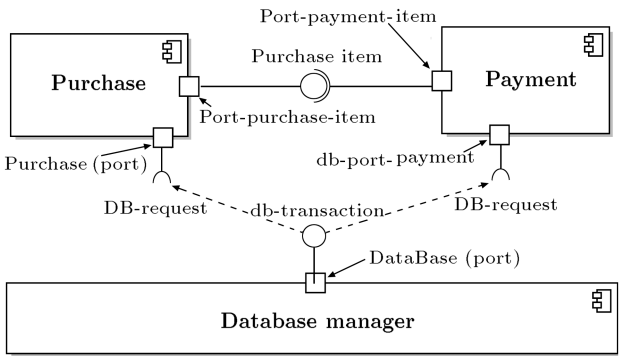

Figure 4. Component diagram of E-shop system.

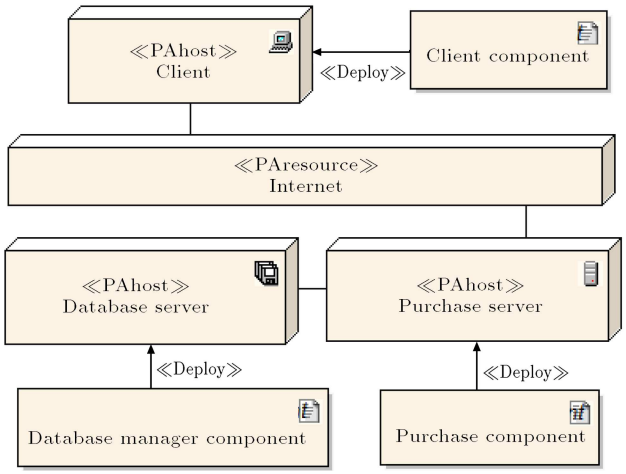

Figure 5. Deployment diagram of E-shop system.

Table 1. Deployment diagram annotation.

Tag value	Client	Purchase server	Database server
Resmult	1	1	1
SchedPolicyKind	Ref	Fcfs	Fcfs
speedFactor	1	1	1
Replica	1	1	1

by our tool [18]. At the first step, XMI, that is an XML-based document of software model, is generated; then, it is transformed into LQN model and is rewrited based on SPEX tool format. SPEX provides a framework for iterative run of LQN model that will be integrated with optimization code algorithms in the next step.

6. Simulation results and comparisons

Each particle is codded as a vector with 5 dimensions, M = [multiplicity of purchase server, multiplicity of database server, multiplicity of thread for processing request task, multiplicity of thread for check availability task, and multiplicity of thread for Database manager task]. The lower and upper boundaries of each dimension are set to 1 and 20.

Four mentioned algorithms are configured according to the parameters, and their values are enlisted in Table 4. The problem section shows the shared parameter between all optimization algorithms.

These parameters include the number of iterations (Iteration), number of particles in the swarm (Swarm size) and archive size. In multi-objective optimization algorithms, an external archive is defined that keeps all the non-dominated (best) solutions. Special parameters of each algorithms are briefly described in Table 4. These parameters have been configured based on the most common values in the literatures. In the future research, adaptive version of these algorithms can be used to tune the parameters.

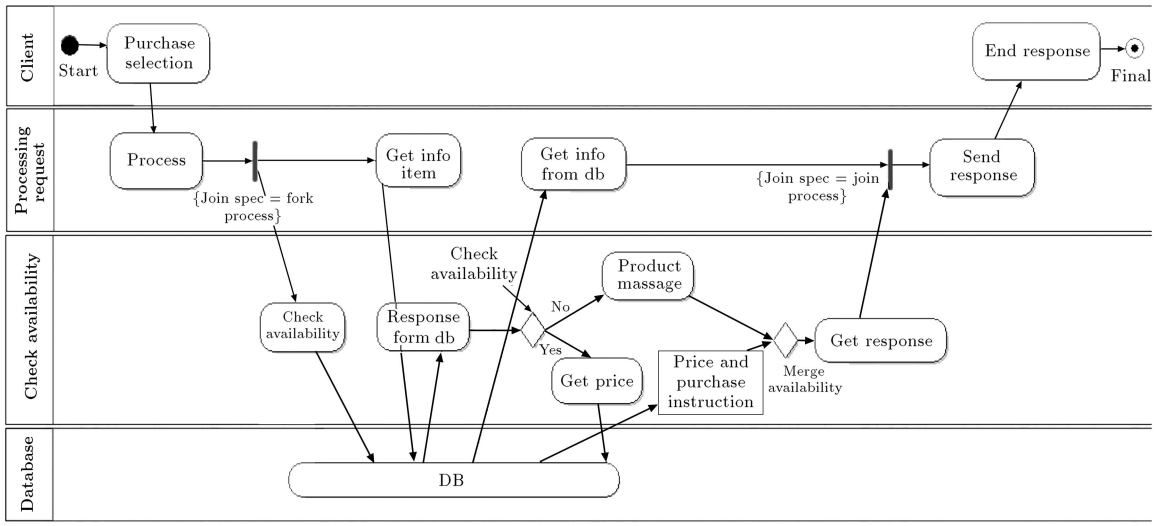

Figure 6. Activity diagram of E-shop system.

Table 2. Component diagram annotation.

Tag value	Element	Purchase		Database
PoolSize	Component	1		1
SchedPolicyKind	Component	Fcfs		Fcfs
		Port-db-purchase	Port-purchase-item	Port-get-request
Mapping	Port	0.5	0.5	0.5

Table 3. Activity diagram annotation.

Tag value	Purchase										Database manager				
	Processing request				Check availability										
	Process	Get info item	Get info from db	Send response	Check availability	Response from db	Product massage	Get price	Price and purchase instruction	Get response	Process query	Insert query	Select query	Update query	Send response
Poolsize	1				1						1				
Priority	0				0						0				
Host demand	1	1	1	1	1	1	1	1	1	1	1	1	2	3	1

In NSGSA and MOGSA, the gravitational constant, G, will take an initial value, $G0$, and it will be reduced with time according to:

$$G(i) = G0 * (1 - i/\text{Iteration}), \tag{5}$$

where i is the current iteration number.

Experiments are run in 2 states; in the first mode the archive size is set to 11, and in the other state the archive size is 19, and all experiments are repeated. The results of 20 independent runs of each algorithm are saved in a database. Thus, for each algorithm, 20 sets of non-dominated solutions are achieved.

Figure 7 shows the behavior of NSGSA in the Last iterations. Also, 6 members from 19 non-dominated solutions of Pareto front are shown in Table 5. This solutions are presented to software architecture who makes various trade-offs, or compromises among bottleneck strength, response time and cost, and chooses one solution.

The performance of used algorithms is evaluated using three important metrics. These metrics are selected because they do not require known Pareto optimal front. We review each metric and analyze the results.

Table 4. Parameters values of 4 algorithms.

Scope	Parameter name	Description	Value
Problem	Iteration	Maximum number of iterations	50
	N	Swarm size	30
	Rep	The size of external archive	11,19
NSGSA	Pr	Reordering mutation probability	0.4
	P_s	Sign mutation probability	0.9
	P_u	Uniform mutation probability	0.01
	P_{elitism}	Percent of elitism	0.5(0.5%)
	W_0	Initial value of inertial coefficient	0.9
	W_1	Final value of inertial coefficient	0.5
	β	Coefficient of search interval	2.5
NSGA-II	M.p	Mutation probability	0.3
	C.p	Crossover probability	0.8
MOGSA	G_0	Gravitational constant	8
		Mutation probability	0.5
	Divisions	Number of divisions in each dimension (to create the hypercube in the grid used to maintain diversity)	[3 2 2]
MOPSO	Divisions	Number of divisions in each dimension	[3 2 2]
	W	Inertia weight	0.5
	M.r	Mutation rate	0.5
	c_1	Acceleration coefficient (local)	1.5
	c_2	Acceleration coefficient (global)	2

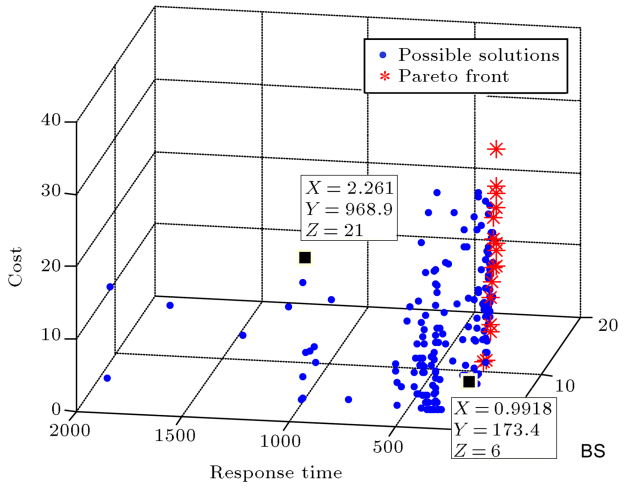

Figure 7. Obtained results and the behavior of NSGSA.

6.1. Coverage metric

The coverage metric [34] calculates the percentage of solutions in a certain approximation set which is dominated or equal to any solution in another competing approximation set. This metric is computed as:

$$C(A,B) = \frac{|\{b \in B/\exists a \in A : a \geq b\}|}{|B|}. \quad (6)$$

Table 5. Six selected members from 19 members of Pareto front.

Bottleneck Strength (BS)	Response time	Cost
1.34	51.97	38
1.21	53.25	33
1.29	52.77	24
0.81	69.31	18
0.80	102.93	9
0.99	173.40	6

If all points in A dominate or are equal to all points in B, then by definition $C = 1$. $C = 0$ implies the opposite. In general, $C(A,B)$ and $C(B,A)$ both have to be considered due to set intersections not being empty. If $C(A,B) = 1$ and $C(B,A) = 0$ then A is better than B.

Coverage metric is computed for each pair of algorithms. Note that the concept and value of $C(A,B)$ is different from $C(B,A)$. Therefore, in addition to computing $C(A,B)$, $C(B,A)$ is also computed. For example, cell in the NSGA-II row and the NSGSA column shows the result of C(NSGSAII, NSGSA) and

by replacing the row and column, the cell shows the result of C(NSGSA, NSGSAII).

Statistical analysis has been performed on the coverage metric of 20 independent runs of four algorithms; Table 6 shows the results when the size of archive is 11, and Table 7 shows the results when it is 19. STD is the standard deviation that is useful for stability evaluation of the results of the algorithm. A lower value of STD implies the more stability of the algorithm. By analyzing the results, these conclusions can be obtained:

1. NSGSA is certainly better than other algorithms. Mean and max of C(NSGSA, NSGA-II) > C(NSGA-II, NSGSA);
2. Mean and max of C (NSGSA, MOGSA) > C (NSGA-II, MOGSA) and C (NSGSA, MOPSO) > C (NSGA-II, MOPSO).

Table 6. Coverage metric; size of archive is 11.

Rep=11		NSGSAII	NSGSA	MOGSA	MOPSO
NSGA-II	Max	-	0.3636	0.8182	0.8182
	Min	-	0	0	0
	Mean	-	0.0550	0.3495	0.3702
	STD	-	0.0232	0.0214	0.0240
NSGSA	Max	0.7273	-	1.0000	0.9091
	Min	0	-	0	0.0909
	Mean	0.2127	-	0.4998	0.5095
	STD	0.0329	-	0.0183	0.0300
MOGSA	Max	0.2727	0.0909	-	0.8182
	Min	0	0	-	0.1818
	Mean	0.0616	0.0145	-	0.4970
	STD	0.0305	0.0168	-	0.0223
MOPSO	Max	0.3636	0.2727	1.0000	-
	Min	0	0	0.1818	-
	Mean	0.0766	0.0620	0.4882	-
	STD	0.0172	0.0305	0.0177	-

Table 7. Coverage metric; size of archive is 19.

Rep=19		NSGSAII	NSGSA	MOGSA	MOPSO
NSGA-II	Max	-	0.3684	0.8421	0.8421
	Min	-	0	0	0.0526
	Mean	-	0.0713	0.4333	0.4314
	STD	-	0.0153	0.0145	0.0186
NSGSA	Max	0.5263	-	0.7895	0.8421
	Min	0.0526	-	0.0526	0.1053
	Mean	0.2318	-	0.4895	0.5157
	STD	0.0117	-	0.0164	0.0181
MOGSA	Max	0.3684	0.2632	-	0.8947
	Min	0	0	-	0.1579
	Mean	0.1086	0.0513	-	0.5305
	STD	0.0215	0.0154	-	0.0150
MOPSO	Max	0.3684	0.2632	0.8421	-
	Min	0	0	0.1579	-
	Mean	0.1384	0.0876	0.5229	-
	STD	0.0172	0.0131	0.0168	-

Note that based on the STD values NSGSA are more stable than NSGA-II.

Comparison results also show much difference between NSGSA and NSGA-II with MOPSO and MOGSA. Therefore, non-dominated sorting based algorithms are better than others whose control policy of the archive is based on a grid.

The overall conclusions of coverage metric evaluation lead us to the following ranking: 1. NSGSA, 2. NSGA-II, 3. MOGSA, 4.MOPSO.

6.2. Diversity

Definition of diversity metric is as bellow [27]:

$$D = \sqrt{\sum_{i=1}^{n} \max\left(\|x'_i - y'_i\|\right)}, \tag{7}$$

where $\|x'_i - y'_i\|$ is the Euclidean distance between of the non-dominated solution x_i and the non-dominated solution y_i. For each non-dominated solution, x_i, in the external archive, the Euclidean distance with the other solutions (y_i) is calculated and the maximum value (farthest solution) is selected. A higher value indicates a better answer that has provided the more diverse solutions.

Since the scale of the objective function is different, the obtained values should be normalized between 0 and 1. Table 8 shows the diversity metric values of objectives. In both cases, with the size of 11 or 19 for archive, diversity of NSGSA is the best and MOPSO, NSGA-II and MOGSA are in subsequent positions, respectively. CV values show the stability of the result of NSGSA and NSGA-II are more stable and their CVs are much better than those of MOPSO and MOGSA.

In this problem, diversity in the solution space is as important as diversity in the objective space. For example, it may be possible, that the objective space is very divers while the actual solutions in problem space are very similar. Therefore, it is important to establish which type of diversity is aimed at or whether both are equally important in the problem domain [35]. In the context of the bottleneck improvement, diversity in the solution space is very important because the solution space contains the parameters of software model, and dissimilar values provide more choices for software engineer. One solution proposes the values for the parameters of model. Diverse solutions provide different proposals. The Software architecture can choose one solution based on the resource availability or other considerations. Population diversity is presented in Table 9. Since the scales of the parameters in solution space are the same, they do not have to be normalized. The results show that NSGA is the best algorithm and has provided more stable solutions. NSGA-II is close to NSGSA but with large difference, MOGSA and MOPSO are in rank three and four.

6.3. Spacing

This metric demonstrates how uniformly the solutions in the objective space are distributed [28]. It is defined as:

$$S = \sqrt{\frac{1}{n-1} \sum \left(d_i - \bar{d}\right)^2},$$

Table 8. Diversity metric in objective space.

	Rep=11				Rep=19			
	NSGA-II	**NSGSA**	**MOGSA**	**MOPSO**	**NSGA-II**	**NSGSA**	**MOGSA**	**MOPSO**
Max	4.013	4.052	3.797	3.875	5.112	5.207	4.999	5.085
Min	3.601	3.646	3.683	3.629	4.772	4.691	4.779	4.804
Mean	3.814	3.910	3.729	3.770	4.951	5.058	4.881	4.940
STD	0.097	0.117	0.028	0.068	0.122	0.151	0.056	0.069
CV	0.025	0.030	0.008	0.018	0.025	0.030	0.011	0.014

Table 9. Diversity metric in problem space.

	Rep=11				Rep=19			
	NSGA-II	**NSGSA**	**MOGSA**	**MOPSO**	**NSGA-II**	**NSGSA**	**MOGSA**	**MOPSO**
Max	0.520	0.600	0.340	0.380	0.522	0.533	0.322	0.300
Min	0.380	0.420	0.160	0.180	0.278	0.322	0.167	0.144
Mean	0.466	0.513	0.222	0.243	0.391	0.392	0.232	0.227
STD	0.048	0.056	0.043	0.042	0.053	0.048	0.041	0.039
CV	0.103	0.110	0.196	0.172	0.136	0.123	0.176	0.172

Table 10. Spacing metric.

	Rep=11				Rep=19			
	NSGA-II	**NSGSA**	**MOGSA**	**MOPSO**	**NSGA-II**	**NSGSA**	**MOGSA**	**MOPSO**
Max	0.313	0.427	0.442	0.340	0.211	0.323	0.275	0.283
Min	0.179	0.224	0.225	0.094	0.090	0.069	0.184	0.123
Mean	0.234	0.328	0.318	0.264	0.142	0.238	0.217	0.200
STD	0.038	0.061	0.065	0.050	0.038	0.062	0.026	0.031
CV	0.161	0.185	0.205	0.191	0.264	0.262	0.121	0.154

$$d_i = \min \sum_{k=1}^{m} \left(\left| f_k^i(\mathbf{x}) - f_k^j(\mathbf{y}) \right| \right) i, \qquad j = 1, \cdots, n, \tag{8}$$

where $\bar{d}$ is the average of all d_i, m is the number of objective functions, and n is the number of points in a Pareto optimal set. A zero value indicates that all members of solutions are uniformly spaced.

The different objectives have different scales. At first, the obtained values are normalized between 0 and 1, then the spacing metrics are calculated. The final results are shown in Table 10. These results show that NSGA-II is the best. NSGSA is close to it but MOPSO and MOGSA are very different. In addition, CV values confirm the stability of the results of the first two algorithms. Although NSGSA focuses on spacing during the run, its spacing metric is not satisfied. Therefore, more study is needed about it.

7. Conclusion and future work

This paper presented an automatic UML based bottleneck improvement solution that optimizes strength bottleneck of the software model. Performance specifications of optimized model have been tuned so that strength bottleneck has been minimized. Response time and cost have also been considered. Therefor NSGSA and MOGSA, 2 multi-objective extension of GSA have explored problem space and presented best solutions. NSGA-II and MOPSO have also been applied and comprehensive comparisons and analysis of results have been presented.

Using this method, bottleneck detection and mitigation are done during the SPE process, and extra evaluation was not required. This approach can be extended considering other quality criteria, such as reliability. Also bottleneck detection and measurement strategy can be improved.

Acknowledgments

This research is supported under contract number 1/1719 by the Institute of Science and High Technology and Environmental Sciences, Graduate University of Advanced Technology, Kerman, Iran.

References

1. Neilson, J.E., Woodside, C.M., Petriu, D.C. and Majumdar, S. "Software bottlenecking in client-server systems and rendezvous networks", *Software Engineering, IEEE Transactions on*, **21**, pp. 776-782 (1995).
2. Jung, G., Swint, G., Parekh, J., Pu, C. and Sahai, A. "Detecting bottleneck in n-tier it applications through analysis",*Large Scale Management of Distributed Systems*, pp. 149-160 (2006).
3. Koliaï, S., Bendifallah, Z., Tribalat, M., Valensi, C., Acquaviva, J.-T. and Jalby, W. "Quantifying performance bottleneck cost through differential analysis", In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, pp. 263-272 (2013).
4. Dongarra, J.J. and Sorensen, D.C. "SCHEDULE: Tools for developing and analyzing parallel Fortran programs", Argonne National Lab., IL (USA) (1986).
5. Bennett, A.J. and Field, A. "Performance engineering with the UML profile for schedulability, performance and time: A case study", in *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, (MASCOTS 2004), *Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*, pp. 67-75 (2004).
6. De Rose, L.A. and Reed, D.A. "SvPablo: A multi-language architecture-independent performance analysis system", in *Parallel Processing, 1999. Proceedings. 1999 International Conference on*, pp. 311-318 (1999).
7. Schumann, M. "Automatic performance prediction to support cross development of parallel programs", in *Proceedings of the SIGMETRICS Symposium on Parallel and Distributed Tools*, pp. 88-97 (1996).
8. Arcelli, D. and Cortellessa, V. "Software model refactoring based on performance analysis: Better working on software or performance side?", *arXiv preprint arXiv:1302.5171* (2013).
9. Franks, G., Petriu, D., Woodside, M., Xu, J. and Tregunno, P. "Layered bottlenecks and their mitigation",

in *Quantitative Evaluation of Systems*, QEST 2006. *Third International Conference on*, pp. 103-114 (2006).

10. Garousi, V. "UML model-driven detection of performance bottlenecks in concurrent real-time software", in *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), International Symposium on*, pp. 317-324 (2010).

11. Martens, A., Koziolek, H., Becker, S. and Reussner, R. "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms", in *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, pp. 105-116 (2010).

12. Martens, A. and Koziolek, H. "Automatic, model-based software performance improvement for component-based software designs", *Electronic Notes in Theoretical Computer Science*, **253**, pp. 77-93 (2009).

13. Koziolek, A., Koziolek, H. and Reussner, R. "Peropteryx: Automated application of tactics in multi-objective software architecture optimization", in *Proceedings of the Joint ACM SIGSOFT Conference-QoSA and ACM SIGSOFT Symposium-ISARCS on Quality of Software Architectures-QoSA and Architecting Critical Systems-ISARCS*, pp. 33-42 (2011).

14. Amoozegar, M. and Nezamabadi-pour, H. "Software performance optimization based on constrained GSA", in *Artificial Intelligence and Signal Processing (AISP), 16th CSI International Symposium on*, pp. 134-139 (2012).

15. Smith, C.U., *Performance Engineering of Software Systems*, Addison-Wesley, **1**, p. 990 (1990).

16. Smith, C.U. and Williams, L.G., *Performance Solutions: A Practical Guide to Creating Responsive*, Scalable Software, **1**, Addison-Wesley Boston, MA (2002).

17. OMG "UML profile for MARTE: Modeling and analysis of real-time embedded systems", *Version 1.0 OMG Adopted Specification formal/2009-11-02* (November 2009).

18. Amoozegar, M. "A solution for performance evaluation of component-based software architecture", A Thesis Submitted in Partial Fulfillment of The Requirement for degree of Master of Science in Software Engineering (2008).

19. Wu, X. and Woodside, M. "Performance modeling from software components", *ACM SIGSOFT Software Engineering Notes*, **29**, pp. 290-301 (2004).

20. Franks, G., Maly, P., Woodside, M., Petriu, D.C. and Hubbard, A., *Layered Queueing Network Solver and Simulator User Manual*, Real-time and Distributed Systems Lab, Carleton University, Ottawa (2005).

21. Franks, G., Hubbard, A., Majumdar, S., Neilson, J., Petriu, D., Rolia, J., et al. "A toolset for performance engineering and software design of client-server systems", *Performance Evaluation*, **24**, pp. 117-136 (1995).

22. Hubbard, A. "SPEX: Software performance experiment driver", http://www.sce.carleton.ca/rads/lqn/lqn-documentation/spex.txt (August 1997).

23. Koziolek, H. "Performance evaluation of component-based software systems: A survey", *Performance Evaluation*, **67**, pp. 634-658 (2010).

24. Woodside, M., Neilson, J.E., Petriu, D.C. and Majumdar, S. "The stochastic rendezvous network model for performance of synchronous client-server-like distributed software", *Computers, IEEE Transactions on*, **44**, pp. 20-34 (1995).

25. Wu, X., *An Approach to Predicting Performance for Component Based Systems*, Carleton University (2003).

26. Reyes-Sierra, M. and Coello, C.A.C. "Multi-objective particle swarm optimizers: A survey of the state-of-the-art", *International Journal of Computational Intelligence Research*, **2**, pp. 287-308 (2006).

27. Hyun, C.J., Kim, Y. and Kim, Y.K. "A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines", *Computers & Operations Research*, **25**, pp. 675-690 (1998).

28. Schott, J.R., *Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization*, DTIC Document (1995).

29. Rashedi, E., Nezamabadi-pour, H. and Saryazdi, S. "GSA: a gravitational search algorithm", *Information Sciences*, **179**, pp. 2232-2248 (2009).

30. Nobahari, H., Nikusokhan, M. and Siarry, P. "Non-dominated sorting gravitational search algorithm", in *Proc. of the 2011 International Conference on Swarm Intelligence, ICSI*, pp. 1-10 (2011).

31. Hassanzadeh, H.R. and Rouhani, M. "A multi-objective gravitational search algorithm", in *Computational Intelligence, Communication Systems and Networks (CICSyN)*, Second International Conference on, pp. 7-12 (2010).

32. Cagnina, L., Esquivel, S. and Coello, C.A.C. "A particle swarm optimizer for multi-objective optimization", *Journal of Computer Science & Technology*, **5**, pp. 204-210 (2005).

33. Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. "A fast and elitist multiobjective genetic algorithm: NSGA-II", *Evolutionary Computation, IEEE Transactions on*, **6**, pp. 182-197 (2002).

34. Zitzler, E. and Thiele, L. "Multiobjective optimization using evolutionary algorithms-A comparative case study", in *Parallel Problem Solving from Nature-PPSN V*, pp. 292-301 (1998).

35. Burke, E. and Landa Silva, J. "The influence of the fitness evaluation method on the performance of multiobjective search algorithms", *European Journal of Operational Research*, **169**, pp. 875-897 (2006).

Biographies

Maryam Amoozegar received her BSc degree in Software Engineering from Kharazmi (Teacher Training) University of Tehran in 2003, and her MSc degree in Software Engineering from Iran University of Science and Technology in 2007. In 2012, she joined the Group of Computer and Information Technology at Institute of Science and High Technology and Environmental Sciences, Kerman, Iran, as a researcher. Her interested research area includes, software modeling, software quality evaluation and soft computing.

Hossein Nezamabadi-pour received his BSc degree in Electrical Engineering from Shahid Bahonar University of Kerman in 1998, and his MSc and PhD degrees in Electrical Engineering from Tarbait Moderres University, Iran, in 2000 and 2004, respectively. In 2004, he joined the Department of Electrical Engineering at Shahid Bahonar University of Kerman, Kerman, Iran, as an Assistant Professor, and was promoted to Full Professor in 2012. Dr. Nezamabadi-pour is the author and co-author of more than 300 peer reviewed journal and conference papers. His interests include image processing, pattern recognition, soft computing, and evolutionary computation.