Towards Evolutionary Testing of Component-based DRE System Deployments in the Cloud

T. Manjula Peiris and James H. Hill
Dept. of Computer and Information Science
Indiana University-Purdue University Indianapolis
Indianapolis, IN USA
Email: {tmpeiris, hillj}@cs.iupui.edu

***Abstract*—Enterprise distributed real-time and embedded (DRE) systems consists of a large number of software components. These software components must be properly deployed and configured on many different hardware nodes to ensure the system meets its desired quality-of-service (QoS) requirements, *e.g.*, end-to-end response time, latency, and throughput. Existing approaches for identifying candidate deployment and configurations (D&Cs) of enterprise DRE systems, however, are either simulation or heuristic bin-packing approaches that do not take into account QoS requirements.**

This work-in-progress paper provides three contributions to deployment and configuring enterprise DRE systems. First, it presents how evolutionary testing and test clouds can be used to identify optimal D&Cs. Secondly, it discusses our approach for evaluating our proposed approach. Our initial results show that there are two kinds of fitness functions that must be supported by our evolutionary testing framework: closeness, which is the relative distance a candidate is from its QoS requirement, and correctness, which is the the probability of meeting a QoS requirement.

***Keywords*-enterprise DRE systems, evolutionary testing, QoS, components, cloud computing**

I. Introduction

Emerging enterprise distributed real-time and embedded (DRE) systems are now being developed using component-based middleware [6]. This is as opposed to low-level monolithic and stove-piped applications. One key advantage of using high-level middleware abstractions is that it allows clean separation-of-concerns [6]. For example, deployment and configurations (D&Cs), application life cycle management, and enabling/disabling of non-functional concerns can be addressed independently of each other. This can therefore result in highly adaptive and scalable enterprise DRE systems that are easier to develop and maintain [12].

Although component-based middleware has its advantages, next-generation enterprise DRE systems are increasing in both size (*e.g.*, number of software/hardware components) and complexity (*i.e.*, envisioned operational scenarios and target execution environments) [8]. One growing challenge associated with emerging component-based DRE systems therefore is deploying and configuring its components to ensure that the system meets its desired quality-of-service (QoS) requirements, such as end-to-end response time, latency, and throughput.

This work was sponsored in part by the Australian Defense Science and Technology Organization (DSTO).

Different deployment and configurations (D&Cs) of an enterprise DRE system, however, will yield different QoS results. For example, deploying software components that communicate often on the same hardware component (*i.e.*, host) *may* yield lower end-to-end response times due to low network latencies than placing such software components on different hosts where network communication is a necessity. Likewise, the number of D&Cs increase exponentially as the number of software and hardware components increase [1]. It therefore is hard—if not *impossible*—to evaluate all D&Cs of emerging DRE systems in a timely manner.

Previous studies have investigated the use of deployment techniques, *e.g.*, bin-packing, mixed with Evolutionary testing(ET) [4], [11] to search the deployment solution space. Such studies, however, are primarily analytical- or simulation-based studies [2]. Moreover, a literature review conducted by Afzal et al. [15] showcases different research efforts of applying ET to validating non-functional system properties, such as end-to-end execution time. To best of the author's knowledge, there is no existing research that applies ET to searching for optimal deployments that meet a given QoS requirement in a production environment, *i.e.*, on the target architecture and in the target execution environment, using large-scale software systems.

This work-in-progress paper therefore discusses our current results on constructing an environment named *eCloud* to support ET of component-based DRE systems in clouds that emulate production environments [13]. DRE system testers use eCloud by defining the behavior and workload of each component in their domain. Testers then specify how to instrument the application and define a single domain-specific QoS expression for optimization based on collected metrics. eCloud then automatically searches the solution space by evaluating different deployments on its target architecture. Our current findings show that there are two classes of fitness functions that eCloud must support for validating enterprise DRE system QoS properties: *closeness*, which measures the relative distance a candidate is from its QoS requirement; and *correctness*, which measures the probability of meeting a QoS requirement.

Paper organization. The remainder of this paper is organized as follows: Section II-B provides a brief overview of evolutionary testing; Section III presents the initial design of eCloud for finding the best component-node mapping for a given QoS requirement; Section IV presents work related to the proposed approach and Section V presents concluding remarks and future research directions.

II. Enabling Technologies of eCloud

This section gives a brief overview of the technologies used to realize eCloud.

A. *Overview of Evolutionary Testing*

Evolutionary Testing (ET) [10] is a search-based software engineering (SBSE) [5] activity rooted in biological evolution. In ET, metaheuristic search techniques, *e.g.*, genetic algorithms, are used to select, or generate, test data. The process starts with an initial set of test data that is typically generated randomly. This test data is then evaluated using a fitness function. The value of the evaluation is used to drive genetic operations, such as selection, crossover, and mutation, to generate the next set of test data. Finally, this process is repeated until a solution that satisfy the testing criterion is found, or predetermined stopping condition is met (*e.g.*, maximum number of iterations).

B. *Overview of CUTS and UNITE*

CUTS [7] is a system execution modeling tool for conducting system integration test that validate QoS requirements on the target architecture during early phases of the software lifecycle. CUTS has following facilities to carry out DRE system integration testing and QoS analysis:

1) Domain-Specific Modeling Languages (DSMLs) [9] to model component behavior and workload at high-levels of abstraction;
2) Generative programming techniques to generate source code for software components from constructed models that conform to the target architecture (*i.e.*, look and feel like the real software components);
3) Emulation techniques to execute the test in realistic environment, such as Emulab [13], that has the same characteristics as the production environment; and
4) Facilities for instrumenting the system in its production environment, and analyzing domain-specific QoS properties.

CUTS main tool for QoS analysis is UNITE. UNITE uses system execution traces collected during the system execution phase for the QoS analysis process. UNITE uses a concept called log formats, which are templates representing different log messages in the system execution trace. A log format has static parts that remain constant among different instances and variable parts that are populated with different values in different log messages. UNITE uses dataflow models and relational database theory to analyze QoS properties. UNITE also constructs QoS performance graphs that illustrate data trends throughout the lifetime of the system (*i.e.*, how a QoS property changes with respect to time).

In addition UNITE uses log format variables to capture the DRE system state at any point of time during the system execution. Using such variables, it is possible to describe QoS requirements as a state-based specification. Listing 1 shows an example of such a QoS requirement expressed using log format variables.

```
Cv : LF1.cmp_name = "Planner"
Ev : (LF2.recvTime − LF1.sendTime) < 50
```

Listing 1. An example QoS expression.

As shown in the above listing C_v specifies the context where the QoS requirement applies. `LF1` and `LF2` represents log formats and `cmp_name`, `recvTime`, and `sendTime` represents log format variables in their corresponding log format. In this example, the QoS requirement is applied to an entity named `Planner`.

E_v specifies the actual QoS inequality that needs to be satisfied in the context C_v. In the above example, E_v states that the event latency need to be less than 50 time units.

III. The Design and Functionality of eCloud

This section discusses the current design and functionality of eCloud.

A. *Deployment Model for ET*

The main goal of eCloud is to locate optimal deployments of a component-based DRE system that satisfies a domain-specific QoS requirement (or expression). In existing work, White et al. [16] formally define a DRE deployment as a 7-tuple:

$$D = < C, N, s(\vec{T}), r(\vec{T}), p(\vec{T}), co(\vec{T}), e(\vec{T}) >$$

where C is the set of components and N is the set of nodes. $\vec{T}$ is a vector representing the component-node mapping (*i.e.*, i^{th} position of $\vec{T}$ represents i^{th} component and the value of i^{th} position represents the node). For example, $\{2, 2, 1\}$ is an example of $\vec{T}$ for three components and two hosts. In this example, component 1 and component 2 are mapped to node 2, and component 3 is mapped to node 1.

$s(\vec{T})$ is a function that returns the total number of deadlines that components will miss. $r(\vec{T})$ is a function that returns the total number of nodes with over consumed resources. $p(\vec{T})$ is a function that returns the violations of spatial-attribute based constraints such as components that are not deployed within specific distance from a certain point.

$co(\vec{T})$ is a function that returns the number of violations of component co-locations constraints. Finally, $e(\vec{T})$ is an objective function that calculates the power consumption of a specific deployment.

For eCloud's deployment model that supports ET, we leverage White's model. We, however, reduce the complexity of the model. For example, we assume there are no resource constraints and no component co-location constraints, *e.g.*, requiring two components to *always* be collocated. This assumption will eliminate $r(\vec{T})$ and $co(\vec{T})$. Furthermore, $p(\vec{T})$

is a domain specific constraint defined because of their application domain (*i.e.*, satellites). Because we do not have such constraints, we can safely remove this constraint from eCloud's model. $e(\vec{T})$ and $s(\vec{T})$ are fitness functions they have defined in their domain. In eCloud model we are replacing those fitness functions with a single fitness function according to the eCloud requirements.

Based on these observations, eCloud uses a 4-tuple model, which is defined as follows:

$$D =< C, N, Q, f(\vec{T}) >$$

where C is the set of components and N is the set of nodes. $\vec{T}$ is a vector representing the component-node mapping as described in White et al.'s model.

Q—a context aware inequality similar to what is illustrated in Listing 1. This context aware inequality represents the user provided QoS requirement.

$f(\vec{T})$ is the fitness function (objective function) which will be based on Q. Structure of $f(\vec{T})$ depends on whether the optimization is carried out for correctness or a closeness situation. Section III-B describes the structure of this fitness function in detail.

B. Fitness Functions for Evaluating QoS Requirements

In hard real time systems, it is not acceptable to violate a QoS requirement, such as missing a deadline. When evaluating QoS requirements of hard real-time systems in eCloud, we propose a *correctness fitness function* to evaluate a given deployment. Here C_v and E_v have the same semantics as described in Listing 1. The correctness fitness function is defined as a probability $P = |E_v \cup C_v|/|C_v|$ where $|C_v|$ is the number of states satisfying context C_v and $|E_v \cup C_v|$ is the number of states satisfying both E_v and C_v together. The goal of this correctness fitness function is to evolve towards a candidate deployment that has a fitness value of $P = 1$, or close to 1, for the QoS requirement under evaluation.

In soft real-time systems, missing QoS deadlines is acceptable given that the gap between the achieved QoS value and the required QoS value is not disturbing system functionality and systemic QoS. When evaluating QoS requirements of soft real-time systems, we propose a *closeness fitness function* that evaluates how close a particular candidate is to achieving a QoS requirement. This is opposed to calculating a probability based on the success or failure.

Cheon et al. [3] proposed an approach for simple object-oriented programs where simple Java expressions are mapped into fitness functions. In their approach, a value called *branch distance* is calculated. Branch distance specifies how far an object instance is from satisfying a particular branch statement in the source code. eCloud adapts this idea when evaluating soft real-time DRE system QoS properties.

We define the closeness fitness function similar to how we define the correctness fitness function. Using the example provided in Listing 1, let E be the expected value of a particular QoS property. As shown in this example, the value of the context should be less than 50 time units. Let the value for this QoS equation in a particular state satisfying C_v is s_i. The goal of the ET when using the closeness fitness function is to maximize the following equation:

$$\sum_{i=1}^{n}(E - s_i)$$

In the equation above, n is the total number of states satisfying the context C_v.

C. Application of eCloud

Based on either one of the fitness functions defined above, the ET process in eCloud will locate the best component-node mapping for a given QoS requirement. This process will be carried out via the following steps:

1) A population of $\vec{T}$ is created. Each member in the population is assigned a random $\vec{T}$. As described above the number of items in the vector is equivalent to number of available components. The positions in the vector will contain values in the range of 1 to maximum number of available nodes.
2) Using CUTS, a test for each member in the population (*i.e*, deployment) is executed on a testbed.
3) System execution traces are collected using CUTS's logging facilities and stored in a database for offline processing and analysis.
4) CUTS analysis facilities are then used to analyze the fitness of each $\vec{T}$ according to one of the fitness functions $f(\vec{T})$ defined above.
5) Based on the fitness value for each member in the current population, standard evolutionary operators (*e.g.*, selection, crossover and mutation) are used to generate the next population.
6) Steps 2–4 is carried out repeatedly until a the user-defined stopping criteria is met. Example stopping criteria is maximum number of iterations or reaching a predetermined fitness value for all the members of a population.
7) Finally, $\vec{T}$ from the final population with the highest fitness is selected as the solution.

IV. Related Work

Afzal et al. [15] conducted a survey showcasing different research efforts on using ET to evaluate non-functional system properties. In the surveyed literature, ET was applied to simple applications or in simulation environments. Our work extends existing research by applying ET to applications in a production environment. Moreover, our work focuses on a generalized approach for applying ET to evaluate any QoS property without *a priori* knowledge of how to evaluate it in a cloud (or production environment).

White et al. [16] present a hybrid approach called *ScatterD* based on both evolutionary algorithms and bin-packing techniques to identify optimal deployments of a DRE system. Their main goal, however, is identifying deployment topologies that minimize power consumption. White et al. also define a second sub-objective function in which realtime deadline constraints of a critical path are taken into consideration. Our approach is similar to ScatterD in that they both apply ET to the deployment problem. Our approach differs in that (1) we apply the ET process in a cloud environment (*i.e.*, a production environment) and (2) the definition process of our fitness functions is more generalized, and can be applied to different QoS properties and application contexts.

Wada et al. [14] describes service deployment optimization approach for cloud computing environments. Their work focuses on finding the best D&C for a given Service Level Agreement (SLA) that defines the QoS requirements for a particular set of users. Similar to our approach, they have also analyzed historical QoS data (*i.e.*, system execution traces in our case) and used Genetic Algorithms (GA). In their model, multiple instances of same service can be created depending on the QoS demand. In contrast, our model focuses on finding out the best configuration for a set of single instances of the components where we have resource constraints in the DRE system. We believer, however, that our eCloud can be applied to the service deployment optimization for cloud computing environments problem.

V. Concluding Remarks

This work in progress paper presented initial work on eCloud, which is an ET based approach for finding the best deployment satisfying the QoS requirements of a component-based DRE system in a production environment. Based on our current work, the following are the lessons learned and future research directions:

- **Determining when to terminate the ET process**. Identifying the correct termination point in an ET process is an open research problem [15]. Currently, our approach is to run the process for a certain number of iterations, or terminating the process when the QoS property under evaluation reaches a certain value. This criteria may not always yield correct results. Future research therefore will focus on identifying the correct termination condition based on QoS requirement.
- **Comparing eCloud approach with other alternatives.** eCloud uses genetic algorithms as the ET process. There are other heuristic-bases search methods, such as particle swarm optimization that can be used for ET. Moreover non-ET approaches like integer programming and linear programming may be useful when optimizing requirements that have QoS trade-offs. Future research will therefore compare eCloud's results with the results of other approaches.

References

[1] S. Asaduzzamanand and M. Maheswaran. Towards a decentralized algorithm for mapping network and computational resources for distributed data-flow computations. In *International Symposium on High Performance Computing Systems*, 2007.

[2] D. Brian, W. Jules, D. C. Schmidt, K. Russell, and P. Jonathan. Deployment Optimization for Embedded Flight Avionics Systems. *CrossTalk Journal*, page To appear, 2011.

[3] Y. Cheon and M. Kim. A specification-based fitness function for evolutionary testing of object-oriented programs. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 1953–1954, New York, NY, USA, 2006. ACM.

[4] B. Dougherty, J. White, J. Balasubramanian, C. Thompson, and D. C. Schmidt. Deployment automation with blitz. In *International Conference on Software Engineering*, pages 271–274, 2009.

[5] M. Harman and A. Mansouri. Search based software engineering: Introduction to the special issue of the ieee transactions on software engineering. *IEEE Transactions on Software Engineering*, 36:737–741, 2010.

[6] G. T. Heineman and W. T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[7] J. H. Hill, D. C. Schmidt, J. Edmondson, and A. Gokhale. Tools for Continuously Evaluating Distributed System Qualities. *IEEE Software*, July/August 2010.

[8] S. E. Institute. Ultra-Large-Scale Systems: Software Challenge of the Future. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, June 2006.

[9] Á. Lédeczi, Á. Bakay, M. Maróti, P. Völgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing Domain-Specific Design Environments. *Computer*, 34(11):44–51, 2001.

[10] P. McMinn. Search-based software test data generation: a survey: Research articles. *Softw. Test. Verif. Reliab.*, 14:105–156, June 2004.

[11] D. D. Niz and R. Rajkumar. Partitioning bin-packing algorithms for distributed real-time systems. *International Journal of Embedded Systems*, 2:196–208, 2006.

[12] U. Rastofer and F. Bellosa. Component-based software engineering for distributed embedded real-time systems. *Iet Software/iee Proceedings - Software*, 148:99–103, 2001.

[13] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *SIGCOMM Comput. Commun. Rev.*, 33:65–81, April 2003.

[14] H. Wada, J. Suzuki, Y. Yamano, and K. Oba. Evolutionary deployment optimization for service-oriented clouds. *Softw. Pract. Exper.*, 41:469–493, April 2011.

[15] W. Wasif, R. Torkar, and R. Feldt. A Systematic Review of Search-based Testing for Non-functional System Properties. *Information and Software Technology*, 51(6):957–976, 2009.

[16] J. White, B. Dougherty, C. Thompson, and D. C. Schmidt. Scatterd: Spatial deployment optimization with hybrid heuristic / evolutionary algorithms. 2011.