Towards Predictive Self-optimization by Situation Recognition

Sebastian Götz, René Schöne, Claas Wilke, Julian Mendez and Uwe Aßmann

Fakultät Informatik, Technische Universität Dresden
sebastian.goetz@acm.org, s3970849@mail.zih.tu-dresden.de,
{claas.wilke, julian.mendez, uwe.assmann}@tu-dresden.de

Abstract: Energy efficiency of software is an increasingly important topic. To achieve energy efficiency, a system should automatically optimize itself to provide the best possible utility to the user for the least possible cost in terms of energy consumption. To reach this goal, the system has to continuously decide whether and how to adapt itself, which takes time and consumes energy by itself. During this time, the system could be in an inefficient state and waste energy. We envision the application of predictive situation recognition to initiate decision making before it is actually needed. Thus, the time of the system being in an inefficient state is reduced, leading to a more energy-efficient reconfiguration.

1 Introduction

Various approaches to self-optimizing software systems have been proposed in literature [GWCA12, CC05]. The basic principle of a feedback loop is common to all approaches. This loop comprises four steps: (1) the system continuously monitors itself and its environment, (2) it analyzes the collected data and (3) decides for or against as well as (4) performs reconfiguration. These four steps (monitor, analyze, plan/decide, act) have been introduced in literature by Oreizy et al. [OGT+99]. In our previous work [GWS+10, GWCA12], we proposed a reification of this feedback loop for energy efficiency.

For example, an audio processing system that improves the quality of audio files (e.g., noise reduction, loudness normalization, etc.) for multiple concurrent users can make good use of self-optimization in terms of energy efficiency. One strategy to save energy is to automatically choose the best server to process a user request based on the current state of the system. Imagine two servers: one is very good in terms of energy consumption but provides poor performance, and the other one is bad in terms of energy consumption but offers very good performance. Let us assume that the efficient server is currently fully utilized, whereas the inefficient server is optimally utilized but is able to process additional requests. If a user sends a request to process an audio file and agrees with waiting for the result for some time, a self-optimizing system would schedule this user request on the efficient server at a later point in time instead of the apparent choice to process the file on the currently free, but inefficient server.

The goal of self-optimizing systems is to automatically reconfigure to the most efficient state possible [ST09a]. Thus, the time of the system being in an inefficient state should be as short as possible. Figure 1 depicts the four steps of the feedback loop and highlights the actual problem: the first two steps are continuously performed until a situation, which requires reconfiguration is detected. Whenever such a situation is detected, the last two steps are performed. For the time of these two steps the system is in an inefficient state. If the system aims at optimizing energy efficiency, the time of being in an inefficient state conforms to wasted energy. A peculiarity in this regard is the question *when* the plan/decide step is performed. In our previous work [GWCA12], this step is performed when a user invokes functionality of the system. That is whenever a user interacts with the system, the optimal configuration is computed and the system gets adjusted accordingly. The approach of Chang and Collet [CC05] initiates the decision step whenever a condition, specified at design time, is sensed to be violated. The problem of both approaches is that they initiate the time-consuming decision step at a point in time, when an immediate reconfiguration is required. That is the system will be in an inefficient configuration for the time of decision making as well as the time required to reconfigure the system.

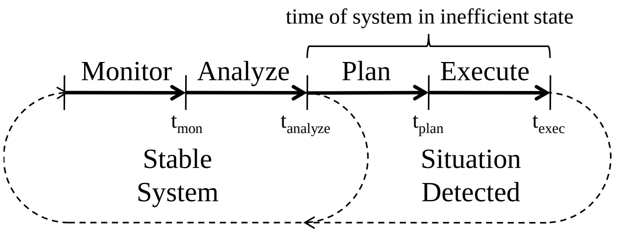

Figure 1: Time Behavior of Feedback Loop

Thus, an intelligent approach to decision making should be performed before the system gets into an inefficient state. This requires a classification of situations which demand for system reconfiguration, as it allows for the application of situation recognition to proactively detect the imminent demand for the decision step [End00]. Thereby, the time between the actual occurrence of a situation and when reconfiguration starts can, in the best case, be decreased by the time required for the decision step. In consequence, the time spans of the system being in an inefficient state are reduced.

Our envisioned solution uses predictive situation recognition [End00] based on description logic reasoning. For this purpose we developed an automatic translation of software engineering artifacts (models, code, etc.) to the web ontology language (OWL) called OWLizer[1]. Keeping an ontology synchronous to the running software system allows the reasoner to recognize predefined situations and, by investigating the history stored in the ontology, allows for the prediction of imminent situations.

2 Envisioned Approach

As a basis for our approach to self-optimizing software system, we follow the models@run.time paradigm [MBJ+09]. The system is developed in a model-driven manner and a special model is kept synchronous to the runtime state of the system. This model is used to reason about the system. A peculiarity of our approach is the application of non-functional contracts to specify how implementations of software components behave. These contracts specify and qualify the dependencies between software components and hardware components, which are the direct consumers of energy.

To enable predictive situation recognition, we plan to synchronize the runtime model of our previous approach with an ontology using the tool OWLizer. The applicability of this synchronization has been shown in [Sch12]. The applicability of description logics for situation recognition has been shown in [ST09b]. Based on this, ontology situations can be recognized by standard reasoners. As a prerequisite, the types of situations to be recognized have to be classified. We identified two major classes of situations, which subdivide into two minor classes each:

- contract-concerned situations: (1) violation of contract clauses, which have been valid at the last decision made, (2) contract clauses, which have been invalid at the last decision made, but became valid

- user-concerned situations: (1) system overload, i.e., the system cannot serve more users, (2) increasing/decreasing number of concurrent users

Each of these situation types potentially demands for reconfiguration as they imply the likelihood of the existence of a better system configuration. For example, imagine a contract clause for an audio normalization algorithm, which specifies the requirement to have a CPU with less than 20% load to guarantee a processing time of at most 10% of the time a playback of the audio file would take. Notably, the processing time impacts energy consumption, as the longer the algorithm runs, the more energy will be consumed. The system decides to run this algorithm on a CPU with 10% load. Later, the load of this CPU increases to 50% due to some other processes, which are not under control of the self-optimizing software system. This situation is a contract clause violation. The guarantee of the contract (max. processing time) does not hold anymore, and another system configuration needs to be computed.

[1]http://st.inf.tu-dresden.de/owlizer

3 Conclusion

In this paper we first motivated the application of self-optimizing software systems to improve the energy efficiency of software at runtime. Then, we outlined the need to anticipate decision making in self-optimizing software systems to reduce their time in inefficient configurations. This is because the time of the system in an inefficient configuration implies a potential waste of energy. Finally, we proposed to utilize predictive situation recognition to detect imminent situations, which demand for reconfiguration. Thus, decision making can be anticipated and the inefficient time of the system can be reduced.

Acknowledgements

This research has been funded within the Collaborative Research Center 912 (HAEC), funded by the German Research Foundation (DFG) and within the research project ZESSY #080951806, by the European Social Fund (ESF) and Federal State of Saxony

References

[CC05] H. Chang and P. Collet. Fine-grained contract negotiation for hierarchical software components. In *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 28–35, 2005.

[End00] M. R. Endsley. Theoretical underpinnings of situation awareness: a critical review. In M. R. Endsley and D. J. Garland, editors, *Situation Awareness Analysis and Measurement*, pages 3–32. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 2000.

[GWCA12] Sebastian Götz, Claas Wilke, Sebastian Cech, and Uwe Aßmann. *Sustainable ICTs and Management Systems for Green Computing*, chapter Architecture and Mechanisms for Energy Auto Tuning, pages 45–73. IGI Global, June 2012.

[GWS+10] Sebastian Götz, Claas Wilke, Matthias Schmidt, Sebastian Cech, and Uwe Aßmann. Towards Energy Auto Tuning. In *Proceedings of First Annual International Conference on Green Information Technology (GREEN IT)*, pages 122–129. GSTF, 2010.

[MBJ+09] Brice Morin, Olivier Barais, Jean-Marc Jezequel, Franck Fleurey, and Arnor Solberg. Models@ Run.time to Support Dynamic Adaptation. *Computer*, 42(10):44–51, 2009.

[OGT+99] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14:54–62, May 1999.

[Sch12] René Schöne. Ontology-based Contract-Checking for Self-Optimizing Systems. Minor thesis, Technische Universität Dresden, December 2012.

[ST09a] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4:14:1–14:42, May 2009.

[ST09b] Thomas Springer and Anni-Yasmin Turhan. Employing Description Logics in Ambient Intelligence for Modeling and Reasoning about Complex Situations. *Journal of Ambient Intelligence and Smart Environments*, 1(3):235–259, 2009.