1 Background

Self-Adaptive Software (SAS) is a special category of software that can change its own behaviors and structure in response to the dynamic and uncertain environment at runtime (e.g., changing workload) [6]. In the past decade, SAS has been widely applied in a range of application domains, including pervasive computing, robotics, service computing and cloud computing. The primary goal of SAS is to continually optimize some important, and possibly conflicted, non-functional attributes (e.g., performance, reliability, cost and energy), thus their requirements can be better complied. Typically, SAS consists of two parts: an adaptable software and an adaptation engine that controls the adaptable software. Such fact does require the target software to be adaptable, i.e., it needs to have a set of well-defined interfaces for sensing and effecting. This is because in software engineering, there are many instances where the source code of the software is hidden or it cannot be changed at runtime. In addition, as stated by Welsh and Sawyer [10], making a software adaptable may be the key condition of a solution, especially when the non-functional attributes and their priorities changes according to the environment.

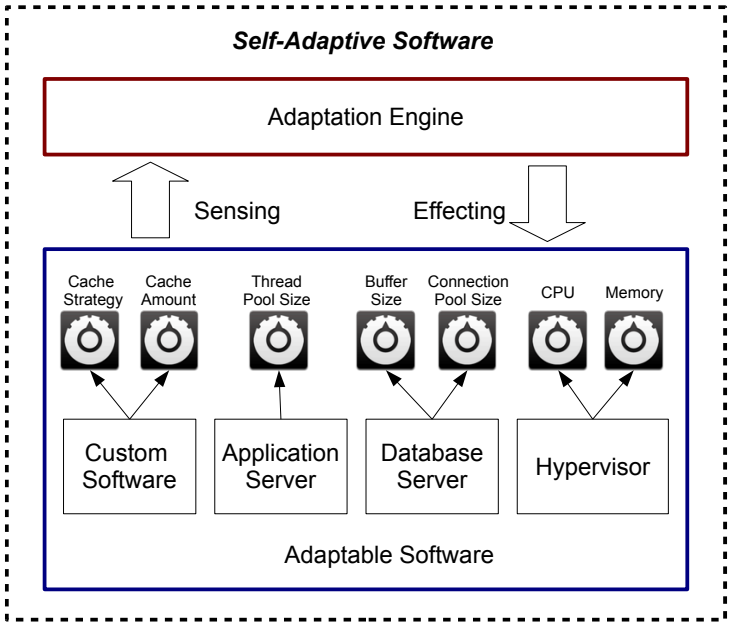

Figure 1: An exampled architecture of SAS.

Figure 1 shows a typical example of the architecture for SAS. A custom software is deployed and running on top of some other commercial off-the-shelf or free software, such as the application server, database server and hypervisor. Such software stack as a whole is actually an adaptable software, which consists of a set of well-defined interfaces. As we can see from the Figure 1, each of the software in the software stack could have one or more control knobs. For examples, the **Cache Strategy** in custom software can be configured as different modes, such as *No-Cache*, *Per-Application-Caching*, *Per-Session-Caching* or *Per-Request-Caching*, leading to different structure of its internal components; the application server can be adjusted to support a certain size of thread pool; and the hypervisor can be allocated with different amount of memory. Depending on the changing environment, all these control knobs and their interplay can cause different and significant impact to the non-functional attributes of the adaptable software. The decision of those control knobs can only be effectively made at runtime, since the environment changes and the sensitivity of non-

functional attributes to those control knobs are often not known at design time. As the environment changes on-the-fly, optimizing the SAS architecture aims to search the right combination of control values that achieves the best non-functional performance. Although in Figure 1, we only illustrate an exampled architecture with 7 control knobs, it can still imply a complicated situation: suppose that there are 4 options for **Cache Strategy** and 20 options for each of the rest control knobs; and that there are no dependencies or mutual exclusions exist between them, then this has already given us a search space of 2.56×10^8 possible adaptation decisions. Consequently, optimizing SAS architecture at runtime requires novel algorithm to efficiently and effectively perform the computational search.

2 How to effectively optimize SAS architecture at runtime and mitigate the effects of requirements uncertainty? (working with Ke)

2.1 The problem 1

Majority of the work for optimization SAS architecture has been relying on weighted aggregation (usually weighted sum) of non-functional objectives [4] and [5], in which the weights are specified by users. However, the the well-know drawback of this approach is that it may be difficulty for a software engineer to specify weights and the result only provide partial information of the trade-off surface. There are few instances where evolutionary multi-objective algorithms (mainly NSGA-II) are used, but they are specific to an application domain of SAS, e.g., cloud [3], [9]. Evolutionary multi-objective optimization for SAS architecture in general has been limited. As a result, there are plenty of opportunities for exploring more advance evolutionary multi-objective optimization to this problem.

2.2 Our proposal 1

Our proposal is to explore better evolutionary multi-objective algorithm to the problem of SAS architecture optimization. In particular, we aim to produce knee solutions for the problem. We might start from the scenarios where objective number is less than 3, and we can explore many objective optimization in the future. In particular, we have the following research questions:

- How to better handle the mixed control variables (e.g., categorical, discrete and integer variables) in the adaptable software?
- How to design/select the operator and how to produce individual given mixed control variables?
- Considering knee points in the optimization process v.s. considering them after the optimization, which one is better?
- Whether the proposal approach is better than existing weighted sum optimizations and NSGA-II?

2.3 The problem 2

For any SAS, stakeholders often specify a set of requirements on the non-functional attributes, representing their satisfactions/expectation on the SAS, e.g., *the response time of the SAS should be less than 2 seconds*. These requirements would be used as (usually soft) constraints on the adaptation engine of SAS and they may be changed at runtime, as requested by the stakeholders. However, the requirements exhibit *uncertainty*, by which we refer to the rationality of requirements is uncertain, hence they may be unrealistic and misleading. This is because the requirements given by stakeholders are highly subjective and depending on their domain knowledge/assumptions, hence inevitably

prone to uncertainty in the rationality of those requirements. In addition, engineers may also provide requirements for certain non-functional attributes that cannot be easily monitored. For instance, the maximum memory consumed by the adaptable software and its best possible performance are properties that may be available from the software's source code, but not easily obtainable through runtime monitoring. Consequently, engineers may influence the requirements based on their past experience, which again introduces uncertainty. The problem becomes even more complex when the non-functional attributes requires trade-offs, i.e., they are conflicting.

Optimization for SAS architecture can be formulated in two ways: constraint satisfaction and objective optimization. However, the requirements (constraints) play an integral roles in both formalizations, hence they may affect the optimization in the following ways if their uncertainty is not properly handled:

- If the given requirements are too good, the optimization of SAS architecture may be struggle to find feasible decisions and thus difficult to converge to optimal result(s), as the requirements introduce too much pressure in the search process.
- If the given requirements are too bad, the requirements may not produce enough search pressure and thus losing the point of having those requirements.

Therefore, how to handle the uncertainty in stakeholders' requirements is a significant problem and it has been identified as one of the key research challenges for engineering SAS [2] [5].

Most of the work (e.g., the FUSION [4] and [5]) on SAS architecture optimization assume that the requirements are always be rationally given, which is unrealistic. On the other hand, existing efforts for dealing with requirements uncertainty has been focusing on *top-down* manner. Specifically, traditional work try to address the problem by designing a methodology that can help stakeholders when specifying requirements [7] at design time. However, this requires a lot assumptions, e.g., each non-functional attribute can be considered independently. In addition, such an approach cannot deal with the uncertainty caused by requirement changes at runtime. More recently, the community has been using Model@runtime paradigm to 'reflect' the requirements at runtime [8]. In those approaches, the requirements are continually reasoned at runtime based on monitoring of the SAS. The aim is to correct those irrational requirements, which might be too good or too bad, after a period of time; and to handle requirements changes. Nevertheless, the drawback is obvious: it may need quite a few adaptation cycles for the requirements to stabilize and the initial quality of requirement can significantly affect the process. All these approakch are called *top-down* because they try to reason about the requirements before they are used in the subsequent optimization process.

2.4 Our proposal 2

Instead of tackling the problem in the traditional *top-down* manner, we try to resolve it via a *bottom-up* way: we aim to develop an underlying optimization algorithm that is resilient to the requirements uncertainty. That is to say, even when the given requirements are too good or too bad, we anticipate that the algorithm can still produce optimal or near-optimal results. By doing so, we free the stakeholders and engineers from the tedious assumptions and complicated requirements elicitation/negotiations process, which could in turn, save development effort. In addition, tackling the problem from a *bottom-up* manner eliminates the need of another complex runtime process (i.e., Model@runtime). At this stage, we aim to answer three research questions:

- Assuming that rational requirements are given, how to consider those requirements in SAS architecture optimization?
- Optimizing with rational requirements v.s. optimizing without (i.e., the approach from proposal 1), which one is better?

- Considering rational requirements in the optimization process v.s. considering them after the optimization, which one is better?
- Removing the assumption of rational requirements. Now the question becomes: how to formulate the problem and design the corresponding optimization algorithm that can mitigate requirements uncertainty?
- How does the algorithm perform when compared with others under irrational requirements?

2.5 The problem 3

Existing work for optimizing SAS architecture has assumed static optimization, i.e., there is an isolated optimization problem at each time step during the entire runtime life cycle. Given the runtime nature of the SAS architecture optimization problem, changes can occur anytime during the optimization, thus turning the classic static optimization into dynamic optimization might yield additional benefits. However, new challenges and considerations arise.

2.6 Our proposal 3

Our proposal is to model SAS architecture optimization as a dynamic optimization problem and propose related approach to solve it. We have the following research questions:

- How to model the dynamics in the SAS architecture optimization problems? e.g., dynamics in workloads and the sensitivity of non-functional attributes to control variables?
- How to design the corresponding optimization algorithm to cope with dynamics? What dynamics handling strategy to use?
- Dynamic optimization v.s. static optimization, what are the added values?
  ,

3 When to trigger optimization and adaptation in SAS architecture? (working with Shuo)

3.1 The problem

Adaptations in SAS architecture occurs continually and it is often driven by the result of architecture optimization. This would therefore rise the questions of *when* or *how often* to adapt the SAS. It is again another dilemma: too frequent adaptations may generate large overhead as the optimization/adaptation process does not comes as free. On the other hand, too rare adaptations might not be useful for the SAS to response to the changes in environment. This problem is also one of the key research challenges in the field [1].

A generally recognized approach in the field is to trigger optimization/adaptation on-demand [4][5]. That is to say, the SAS will adapt once the requirements of non-functional attributes are violated (or some safe thresholds of requirements are violated). Such an approach can work in either reactive or proactive manner, for example, optimization/adaptation can be triggered when the violations have already occurred or they are predicted to occur very soon. However, this approach can be still problematic when:

- The violation is trivial.
- The payoff after the optimization is trivial under current circumstance.

3.2 Our proposal

We intend to adopt online machine learning and concept drift technique to capture the correlation between requirement violation, the benefits and cost after optimization/adaptation. Upon the occurrence of requirement violation, we aim to establish model(s) to assist more systematic reasoning about whether to trigger optimization in the SAS architecture. In this way, we anticipate that the proposed approach will reduce overhead in the adaptation engine and potentially improve stability of the SAS architecture. At this stage, we aim to answer three research questions:

- What are the important features that should be consider in the learned models?
- How to reason about and make trade-off between optimization/adaptation cost and payoff when optimization in runtime architecture? (I am discussing with Rami about using Technical Debt to this research question)
- Whether it is beneficial to continually track the changes in the models using concept drift (e.g., sudden changes in workload) and how to achieve such?

4 About evaluation

Currently there are quite a few existing benchmark software exist in the community of Performance Engineering, Requirement Engineering and Software Engineering for Self-Adaptive Software Systems. Among others, the most popular ones are Znn.com [1], RUBiS [2], TPC [3] and GridStix [4]. We plan to evaluate our approach based on one or more of the above. The evaluation metrics can be the achieved non-functional results and their compliance with respect to requirements. Additionally, the metrics (e.g., hypervolumn, GD and IGD) in multi-objective optimization community can be also applied.

In a later stage of dissemination, we intend to integrate our work with an open source software. Currently, I am thinking of choosing one from the Apache Software Foundation as it is well recognized as one of the largest open source software corporation.

References

[1] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit. *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, chapter A Design Space for Self-Adaptive Systems, pages 33–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[2] B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. *Software Engineering for Self-Adaptive Systems*, chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[3] D. El Kateb, F. Fouquet, G. Nain, J. A. Meira, M. Ackerman, and Y. Le Traon. Generic cloud platform multi-objective optimization leveraging models@run.time. In *Proceedings of the 29th*

[1] http://adaas.dei.uc.pt/adaas/resources/znn.com

[2] http://rubis.ow2.org/

[3] http://www.tpc.org/tpcw/

[4] Hughes, D., Greenwood, P., Coulson, G., Blair, G.: GridStix: supporting flood prediction using embedded hardware and next generation grid middleware. World of Wireless, Mobile and Multimedia Networks (2006)

Annual ACM Symposium on Applied Computing, SAC '14, pages 343–350, New York, NY, USA, 2014. ACM.

[4] A. Elkhodary, N. Esfahani, and S. Malek. Fusion: A framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 7–16, New York, NY, USA, 2010. ACM.

[5] N. Esfahani, E. Kouroshfar, and S. Malek. Taming uncertainty in self-adaptive software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 234–244, New York, NY, USA, 2011. ACM.

[6] R. Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. Göschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke. *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, chapter Software Engineering for Self-Adaptive Systems: A Second Research Roadmap, pages 1–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[7] V. Poladian, J. Sousa, D. Garlan, and M. Shaw. Dynamic configuration of resource-aware services. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 604–613, May 2004.

[8] H. Song, S. Barrett, A. Clarke, and S. Clarke. *Model-Driven Engineering Languages and Systems: 16th International Conference, MODELS 2013, Miami, FL, USA, September 29 – October 4, 2013. Proceedings*, chapter Self-adaptation with End-User Preferences: Using Run-Time Models and Constraint Solving, pages 555–571. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[9] H. Wada, J. Suzuki, Y. Yamano, and K. Oba. Evolutionary deployment optimization for service-oriented clouds. *Softw. Pract. Exper.*, 41(5):469–493, Apr. 2011.

[10] K. Welsh and P. Sawyer. *Requirements Engineering: Foundation for Software Quality: 14th International Working Conference, REFSQ 2008 Montpellier, France, June 16-17, 2008 Proceedings*, chapter When to Adapt? Identification of Problem Domains for Adaptive Systems, pages 198–203. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.