Combining Separation of Concerns and Performance in Distributed Software Reconfiguration

Maverick Chardet

Abstract

Distributed software and infrastructures continue to become more and more complex and dynamic. Therefore, there is a need for models assisting their management, including their reconfiguration. Existing reconfiguration models are either specific to a subset of reconfigurations or are unable to provide both good performance and high separation of concerns between the actors interacting with them. In this article, we present our plans to extend Madeus, a non-specific and efficient deployment model, to support reconfiguration and to provide a good separation of concerns.

Combining Separation of Concerns and Performance in Distributed Software Reconfiguration

Maverick Chardet

December 12, 2018

1 Introduction

Distributed software refers to software composed of multiple interacting components, possibly running on different inter-connected machines. Reconfiguring distributed software consists in modifying its current state. Reconfiguration can be desired for many reasons: optimization of the quality of service or cost, real-time energy management, dynamic evolution of the services provided (*e.g.*, dynamic update, improvement of the functionalities of an application), etc.

Autonomic computing is a way to automatize the reconfiguration. A common way of conceiving autonomic computing is to use a MAPE-K loop [1]. The system is *Monitored* (M), and the information gathered is *Analyzed* (A) to decide if a reconfiguration needs to be performed. If so, a reconfiguration is *Planned* (P) and then *Executed* (E). The models and information used by the four phases are grouped in the common *Knowledge* (K).

A reconfiguration model enables both the user to define an assembly of components and (P) to define reconfigurations on this assembly using a dynamic Assembly Description Language (ADL). In this paper, we focus on the reconfiguration model itself and on its ADL (in K). We also focus on the reconfiguration engine (in E) which performs reconfigurations defined in this ADL.

A reconfiguration model can be analyzed against different properties. Its *genericity* refers its non-specificity to given hardware or software. *Expressiveness* is a measure of the variety of types of reconfiguration it handles. Its *performance* refers to how fast it can perform a reconfiguration (*e.g.*, by parallelizing some tasks). *Scalability* designates how well performance is maintained as the size of the system increases. Moreover, a reconfiguration model can provide more or less *abstraction*, *i.e.*, hide complexity to the user. A high level of abstraction usually comes with good safety properties, as the actions that the user can do are well-defined. Finally, *separation of concerns* designates to what extent each actor interacting with the system in any way only does what they are supposed to do (*i.e.*, what is in their area of expertise).

Most of the existing reconfiguration frameworks do not provide at the same time (1) high abstraction preserving performance and separation of concerns and (2) good genericity and expressiveness. We aim at providing a reconfiguration model allowing to fulfill these two objectives. Section 2 presents the state of the art, while Section 3 gives an overview of our approach and the work accomplished to date. Our plans to evaluate this work are presented in Section 4. Finally, Section 5 concludes and presents future work.

2 State of the Art

Most of the existing frameworks supporting reconfiguration fall in one of the following two categories. The first one regroups those which target specific kinds of reconfiguration: they show a good level of abstraction and separation of concerns but at the cost of a low expressiveness. For example, Amazon EC2 and Kubernetes [2] support scaling and reboot-on-crash, CoRDAGe [3] supports addition and removal of components. The second category comprises the frameworks which provide low abstraction for reconfiguration, hence allowing to perform virtually any kind of reconfiguration. However, this comes at the cost of a low separation of concerns (the reconfiguration developer must understand the underlying software and they are responsible for performance concerns) as well as no guarantees that the reconfiguration is going to perform well. Examples of such frameworks are Fractal [4], GCM [5] and FraSCAti [6].

Aeolus [7] is a reconfiguration model which is not specific to any type of reconfiguration and abstracts away the management of the different states of a component (*i.e.*, its life-cycle) from the reconfiguration developer. Each component of a distributed software is represented by a finite-state machine, each state corresponding to a step in the life-cycle of the component. Each state has a list of its dependencies and services it provides, hence increasing parallelism thanks to the fine-grained dependencies. Plus, some proofs can be made on this model, such as reachability analysis (*e.g.*, to prove that a reconfiguration can end). However, Aeolus (a) does not allow parallelism within a component, (b) does not provide an operational semantics (hence requiring human action before a deployment can be scheduled) and (c) shows poor separation of concerns between the developer of components and the developer of reconfiguration (the latter must understand the underlying software to define a reconfiguration).

Madeus [8] is a deployment model inspired by Aeolus. It addresses points (a) and (b) in the case of deployment (that we see as a specific case of reconfiguration). Our goal is to extend Madeus to support reconfiguration, while preserving (a) and (b) and addressing (c).

3 Approach and Results

Overview of the approach Our approach is to propose a new reconfiguration model extending Madeus to support reconfiguration. This model should

(1) not be specific to any kind of software, hardware or reconfiguration (*i.e.*, it has good genericity and expressiveness), (2) show good performance by allowing to express a high level of parallelism, (3) abstract away performance and coordination-related concerns from the reconfiguration developer, (4) show good separation of concerns between the developer of components and the developer of reconfiguration, and (5) have a formal semantics allowing to make formal proofs (such as termination).

Preliminary results A preliminary feasibility study regarding the extension of Madeus for reconfiguration has been done. We enabled the developers of components to define multiple *behaviors* (*i.e.*, high-level reconfiguration actions for the component, such as *deploy* or *change-configuration*) to each component. This is done by assigning to each behavior a set of transitions of the component's life-cycle state-machine allowing to perform the desired action. We also provided an algorithm to generate an abstraction of the state-machine of each component to the developer of reconfiguration, abstracting away all the details that they do not need to know. This allows a greater separation of concerns between them and the developer of components. Finally, we modified the description language of Madeus to handle reconfiguration and take advantage of the behaviors.

4 Evaluation

We plan to implement a proof-of-concept for a reconfiguration engine using our model. The evaluation will be made with reproducibility in mind on synthetic and real-world use-cases to demonstrate the gains in performance, abstraction, separation of concerns and expressiveness compared to existing solutions. We target current and expected reconfiguration patterns. Particularly, we would like to address specific patterns for future architectures such as fog computing[9]. Such use-cases include migration of a database from a centralized to a decentralized instance, rolling updates with limited resources and reconfiguration of a decentralized OpenStack[1].

We will compare our model to solutions including Aeolus for its performance, Ansible for its expressiveness and wide adoption by the industry, along with platform or technology-specific tools such as AWS CloudFormation or Kubernetes for their good abstraction and separation of concerns.

To evaluate performance, we will compare the run-time of the same reconfiguration performed by our engine and by the other solutions on the experimental platform Grid'5000[2]. The effect on performance of parameters such as the size of the cluster and the locality of a VM image that need to be downloaded will be evaluated as well. To evaluate separation of concerns, the metrics will capture the knowledge that each actor needs to have about the system outside of their area of expertise (*e.g.*, number of references to internal implementation elements

[1] `http://beyondtheclouds.github.io/`
[2] `https://www.grid5000.fr/`

in the definition of the reconfiguration by the developer of reconfiguration). Finally, the number of lines of code, weighed by their complexity, that a developer must write to express a reconfiguration will be considered as an indicator of the level of abstraction of the model.

5 Conclusion

In this article, we have presented our approach to developing a new reconfiguration model for distributed software that combines efficiency and good separation of concerns. We have also planned how to evaluate this model in terms of performance, separation of concerns and level of abstraction.

In addition to formalizing the model, future work includes providing high-level abstractions when it does not hurt performance (*e.g.*, resource management, hierarchical components). We also plan to study the possibility of decentralizing the reconfiguration engine to increase scalability and exploring the topic of concurrent reconfigurations to increase performance.

References

[1] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, jan 2003.

[2] E. A. Brewer, "Kubernetes and the path to cloud native," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, (New York, NY, USA), pp. 167–167, ACM, 2015.

[3] L. Cudennec, *CoRDAGe : Un service générique de co-déploiement et redéploiement d'applications sur grilles*. PhD thesis, jan 2009.

[4] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "The FRACTAL component model and its support in Java," *Software: Practice and Experience*, vol. 36, pp. 1257–1284, sep 2006.

[5] F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, and C. Pérez, "GCM: a grid extension to Fractal for autonomous distributed components," *annals of telecommunications - annales des télécommunications*, vol. 64, pp. 5–24, feb 2009.

[6] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J.-B. Stefani, "Reconfigurable SCA Applications with the FraSCAti Platform," in *2009 IEEE International Conference on Services Computing*, pp. 268–275, IEEE, 2009.

[7] R. Di Cosmo, J. Mauro, S. Zacchiroli, and G. Zavattaro, "Aeolus: a Component Model for the Cloud," *Information and Computation*, pp. 100–121, 2014.

[8] M. Chardet, H. Coullon, D. Pertin, and C. Pérez, "Madeus: A formal deployment model," in *4PAD 2018 - 5th International Symposium on Formal Approaches to Parallel and Distributed Systems (hosted at HPCS 2018)*, (Orléans, France), pp. 1–8, July 2018.

[9] R. Mahmud and R. Buyya, "Fog Computing: A Taxonomy, Survey and Future Directions," nov 2016.