Adaptive Deployment in Ad-Hoc Systems Using Emergent Component Ensembles: Vision Paper

Lubomír Bulej[1,2] Tomáš Bureš[1,2] Vojtěch Horký[1] Jaroslav Keznikl[1,2]

[1]Charles University in Prague
Faculty of Mathematics and Physics
Malostranské náměstí 25
118 00 Prague 1, Czech Republic

[2]Academy of Sciences of the Czech Republic
Institute of Computer Science
Pod Vodárenskou věží 2
182 07 Prague 8, Czech Republic

{bulej,bures,horky,keznikl}@d3s.mff.cuni.cz

ABSTRACT

Mobile cloud computing in the context of ad-hoc clouds brings new challenges when offloading computation from mobile devices. The management of application deployment needs to ensure that the offloading provides users with the expected benefits, but it suddenly needs to cope with a highly dynamic environment which lacks a central authority and in which computational nodes appear and disappear.

We propose an approach to the management of ad-hoc systems in such dynamic environment using component ensembles that connect mobile devices with more powerful computation nodes. Our approach aims to address the challenges of scalability and robustness of such systems without the need for central authority, relying instead on simple patterns that lead to reasonable adaptation decisions based on limited and imprecise information.

Categories and Subject Descriptors

[**Computer systems organization**]: Other architectures — Self-organizing autonomic computing, Distributed architectures — Cloud computing; [**Software and its engineering**]: Extra-functional properties — Software performance

Keywords

ad-hoc cloud, ensembles, adaptive deployment

1. INTRODUCTION

Increasing capabilities of handheld devices and improvements in mobile network infrastructures pave the way for mobile cloud computing [1], an architectural solution where mobile devices offload computation to the cloud to gain advantage for example in increased computing power or reduced battery usage. Another motivation is the emergence of ad-hoc clouds [2], whose computing power comes from pooled resources of nearby general-purpose computing devices rather than from dedicated servers. Our work, carried out in the scope of the ASCENS project [3], aims to combine and extend the two trends by blurring the traditionally strict [4] boundary between the client devices and the cloud infrastructure. We envision an ad-hoc cloud formed as a multitude of dynamically emerging groups of computational devices that share their computing power. The groups will typically involve a number of mobile devices along with locally situated general-purpose computers, potentially connected to a remotely situated dedicated cloud infrastructure.

Important specifics of our ad-hoc cloud concept are that it has a dynamic, mostly uncontrollable architecture with fluctuating computing power and partially limited resources, e.g. the available battery charge for mobile devices. The promise of the ad-hoc cloud is in maintaining the usual benefits associated with offloading computation to a cheap, flexible and resilient environment—however, the ad-hoc cloud applications must dynamically adapt their deployment to deliver the expected user experience in such specific conditions.

The challenge in application adaptation is related to the open character of the ad-hoc cloud. Where a common cloud application can react to increased utilization by requesting additional computing resources, an application in an ad-hoc cloud must act in presence of other adapting applications that share the same resources. Even the scheduling solutions for computational grids, which do cope with shared resources, assume a degree of centralization knowledge and control over the grid that is not available in the ad-hoc cloud [5]. We therefore believe that the application of adaptation solutions in ad-hoc clouds will not assume the shape of complex algorithms that compute close-to-optimal deployment under dynamic conditions, but rather the shape of relatively simple patterns that lead to reasonable adaptation decisions relying on limited and imprecise information.

In this vision paper, we focus on the problem of adaptive deployment planning in ad-hoc clouds and outline an adaptation approach based on a component system with emergent component ensembles [6, 3]. We assume an external mechanism would be responsible for the actual migration [7]. We discuss the potential benefits and scalability of this approach.

2. MOTIVATING EXAMPLE

Besides smart phones, we consider tablet computers to be a perfect target for mobile computing in ad-hoc clouds—modern applications take advantage of their relatively high computational power and users tend to use them both for

ICPE'13, April 21–24, 2013, Prague, Czech Republic.

work and entertainment. However, they are still constrained by the limited battery life.

We start our vision with an example, where we consider a user travelling in a train or a bus, who wants to do productive work using her tablet computer or review travel plans and accommodation. Her tablet registers the presence of an offload server machine located in the bus itself, and to save battery, it offloads most computationally intensive tasks to that machine. Later, when the bus approaches the destination, the offload server notifies her tablet that its service will soon become unavailable and tasks will start moving back to the tablet. When the bus enters the terminal, the tablet will discover another offload server, provided by the terminal authority, and move some of its tasks to the newly found machine.

Many similar examples can be found, and they would follow similar pattern. Abstracting away from the details, we can try to capture such examples formally under one general umbrella. Assume a mobile device M (tablet in our example) and two stationary devices S and T (offload servers in our example). M executes application A, which is internally split into two parts: a frontend Af, responsible for the interaction with a user, and a backend Ab, responsible for the computationally intensive tasks.

In our scenario, M discovers S and assesses that offloading the computationally intensive Ab to S could save M's battery. After some time, S signals that it is going to be unavailable, but M discovers that there are other devices available. Ab is thus migrated to the one that appears most suitable for running Ab—device T in our scenario.

The challenge is in predicting which deployment scenario will—in the context of ad-hoc cloud—deliver the expected user experience. We assume that each application will have a simple performance model that, given specifics of the execution environment and other constraints, will provide a rough estimate of the expected user experience (e.g. what frame rate could be achieved given CPU and GPU budget). The application deployment would be then planned dynamically, taking into account the expected user experience estimated by the model, possibly corrected for measured accuracy of the model from past deployments.

3. COMPONENT ENSEMBLES

Although the scenario of the running example is relatively straightforward, it is relatively difficult to realize due to the inherent dynamicity of the whole ecosystem (i.e. all applications and devices). Further, the combination of the dynamicity and the autonomy of the applications and devices imply the absence of the notion of a global state. In fact, every information about the ecosystem has the form of a "belief" – i.e. an information valid to only a certain extent.

To cope with these issues, we suggest in this paper to take advantage of a component system based on emergent component ensembles [6, 3] and use it to represent situation in such a dynamic ecosystem and to manage the belief about it. To this end, we outline in the rest of the section the basic principles of component ensembles and explain their use in addressing the adaptation in Section 4.

Emergent component ensembles are based on the idea of implicit communication via implicit bindings. Specifically, an ensemble is a dynamically formed group of components, where a component constitutes knowledge (i.e., data) and processes (i.e., active threads operating upon the knowledge). The membership of a component in an ensemble is determined dynamically (the task of the component system runtime framework) according to the membership condition of the ensemble specified upon the knowledge of the components. In an ensemble one component plays the role of the ensemble's coordinator while others play the role of members. A single component can be member and/or coordinator of multiple ensembles at the same time; thus an ensemble forms an independent logical overlay over components.

The interaction among the components forming an ensemble takes the form of knowledge exchange, carried out implicitly (by the runtime framework); i.e., the runtime framework transfers knowledge from one component of the ensemble to another independently on the components' execution.

The benefit of such ensembles is that they allow for capturing communication (i.e., exchange of knowledge) among a (potentially) large, declaratively defined set of components in a concise way.

4. ADAPTATION ARCHITECTURE

To address the aforementioned challenges, we present a generic architecture of adaptation logic, based on emergent component ensembles. The proposed *adaptation architecture* follows the following basic principles.

To ensure separation of concerns, the adaptation logic forms a separate overlay architecture mirroring the architecture of the adapted application.

Additionally, the entities important for deciding adaptation, i.e., (a) computation nodes (and their NFPs), (b) individual adapted applications (and their NFP preferences), and (c) the applications' components (and their NFPs), are explicitly reflected in the adaptation architecture.

4.1 Adaptation architecture components

The adaptation architecture (Figure 1) is formed by following components:

Planner. Each adapted application, and particularly its NFP preferences, are represented by the Planner component. Specifically, the Planner selects a (potentially optimal) deployment of the application, given the alternatives for deploying each of the application's components. We assume an external mechanism [7] to interpret the deployment plan provided by the Planner and perform the adaptation (e.g., by migrating a component). The alternatives comprise important NFP-related data (NFPData) indicating the (potential) performance of the corresponding application component in that particular deployment (e.g., FPS, energy, etc.). The Planner also advertises definitions of Monitors for individual application components (MonitorDef); see Device.

Monitor. Each application component, particularly each of its deployment alternatives, is reflected by the Monitor component, which is responsible for obtaining the NFPData for that particular alternative. Monitor operates in one of the two modes, depending on the actual deployment of the corresponding application component.

- Monitor is in the *running mode* if it resides on the same computation node as the corresponding application component, i.e. it reflects the actual deployment. NFPData is obtained by performance measurement and analysis of the running application component.
- Monitor is in the *mock mode* if it resides on a different computation node, i.e. it represents a potential deployment alternative. NFPData is obtained from

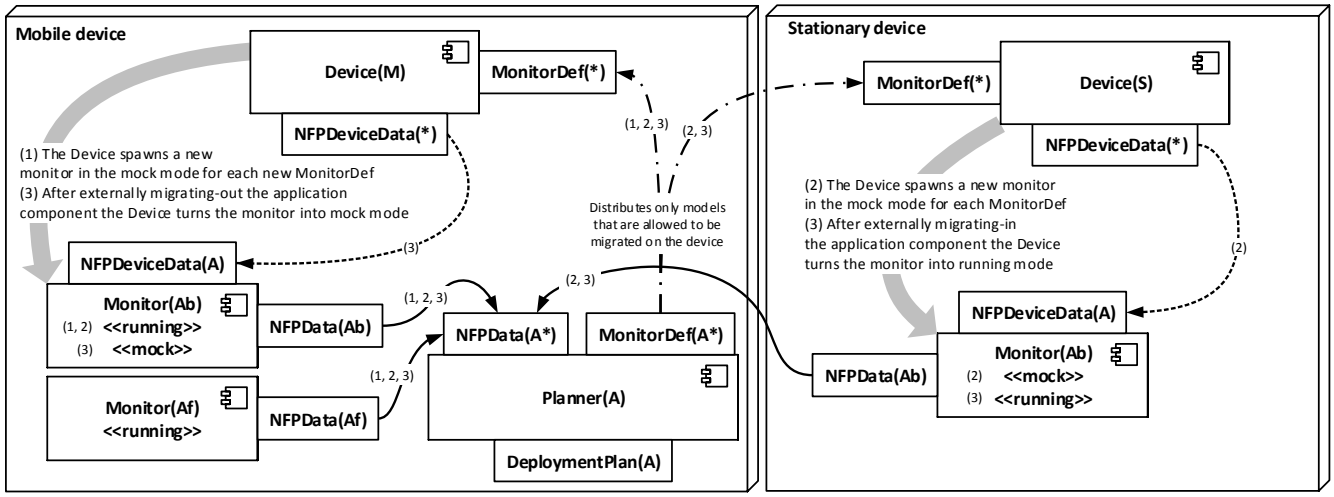

Figure 1: Adaptation architecture of the running example: phases 1 (M isolated), 2 (S discovered), and 3 (Ab migrated to S). Phases 1,2,3 are in the figure denoted by (1), (2), (3).

the included performance dependency model of the corresponding application component (e.g., the function $CPU \times GPU \rightarrow FPS$). In other words, Monitor predicts – based on the model – the performance of the application component if it would be deployed on that computation node. The model might depend on particular machine-specific performance data (NFPDeviceData, e.g., available CPU speed, etc.); see Device.

Device. Each computation node is reflected by the Device component. Specifically, a Device component ensures management of the Monitors (e.g., it instantiates Monitors advertised by newly discovered Planners) and it provides NFPDeviceData for Monitors in the mock mode.

4.2 Adaptation architecture ensembles

The expectation is that the number of available computation nodes, as well as the number of Monitors, changes dynamically. Therefore, the communication among the components exploits the concept of emergent component ensembles. The architecture involves the following ensembles (Figure 1):

Planner and Device(s). Each Planner is a coordinator of an ensemble that distributes MonitorDefs (including the performance dependency model) of application components to Devices representing currently available computation nodes (including the one the Planner is running on). The Planner is able to constraint which MonitorDefs should be distributed to which Devices (effectively constraining the potential migration destinations for a particular application component). A simplified example of a definition of this ensemble is in Figure 2. It specifies that only reachable devices within 2 network hops are to be considered and that this check is to be performed every 15 seconds. The distribution of the MonitorDefs is performed by adding the MonitorDef to the target component's knowledge.

Planner and Monitor(s). Each Planner is a coordinator of an ensemble that aggregates NFPData from all Monitors corresponding to the components of the application reflected by the Planner. Thus, this ensemble aggregates all the deployment alternatives for the application.

```
1 ensemble PlannerToDevice:
2   coordinator: Planner
3   member: Device
4   membership: HopDistance(Planner.device, Device) ≤ 2
5   knowledge exchange:
6     Device.monitorDef[Planner.app] := Planner.monitorDef
7   scheduling: periodic( 15s )
```

Figure 2: Example of an ensemble definition.

Device and Monitor(s). Each Device component is a coordinator of an ensemble that distributes NFPDeviceData to the Monitors in the mock mode residing on the corresponding computation node.

4.3 Adaptation architecture in action

In this section, we illustrate on the motivation example the adaptation architecture interaction at runtime.

At first (phase 1, Figure 1), the ensemble distributes the MonitorDefs of both Af and Ab from Planner of A to the Device component of the mobile device (M), which subsequently spawns Monitors for both components and sets them to the running mode. The Monitors start measuring NFPData of the running components which are then aggregated back to the Planner. So far no deployment alternatives are discovered.

After the stationary device (S) is discovered (phase 2, Figure 1), the ensemble propagates MonitorDefs of the components that could be (potentially) migrated (i.e., Ab) to its Device component, which spawns a new Monitor. Since Ab is deployed on a different Device this Monitor runs in the mock mode. Thus, the Device component of the stationary device feeds the Monitor with NFPDeviceData allocated for A. Based on this NFPDeviceData and the performance dependency model of Ab the Monitor produces NFPData reflecting the expected performance of Ab on S. Consequently, another ensemble aggregates all the currently produced NFPData for Af and Ab to the Planner. The Planner thus eventually discovers that there are two deployment alternatives for Ab (i.e., one actually running on M and one modeled on S) and finally decides to deploy Ab on the stationary device.

After Ab is migrated to the stationary device (phase 3, Figure 1), the Monitor on S is set to the running mode, while the Monitor on M is set to the mock mode and the whole monitoring and planning process repeats.

In the case of discovering further stationary devices, new Monitors in the mock mode are spawned which eventually results in new deployment alternatives aggregated in the Planner (similarly, if devices disappear).

5. BENEFITS

Scalability and robustness. By exploiting the features of the ensembles, the adaptation architecture scales well with the number of computation nodes, applications, and components per application. In fact, the adaptation architecture does not require any changes when increasing the number of nodes/applications/components. Furthermore, it is very robust with respect to emergence of computation nodes.

Transparent trade-off management. Due to the declarative nature of ensembles, it is possible to easily manage the trade-offs between the benefit of migration and the effort necessary for monitoring and planning. For instance, Monitors do not have to be spawned on all available nodes but only on a subset; e.g., only the nodes in the same subnet.

Respecting interests of all involved parties without central authority. Although each application is planned autonomously, it is possible (without any centralized authority) to take into account the interests of the other applications and of host devices by regulation of the NFPDeviceData and management of the application's Monitors — e.g., the NFPDeviceData may reflect only a portion of device's resources.

Flexible NFP data acquisition. The NFPData produced by Monitors may contain any information important for deciding adaptation as along as it is obtainable via measurements and/or performance dependency model, e.g., latency between Ab and Af, expected up-time of the computation node (for detecting shutdowns), etc. Moreover, a Monitor can decide between accepting NFPDeviceData given by Device and measuring its own, e.g., Monitor(Ab) can either individually measure latency to Af or rely on the network latency information given by Device(S). Although the performance dependency model employed by a Monitor will usually provide only a rough approximation of the expected performance, it can be potentially improved by actual measurements.

Scalable extensions. Being declarative, the ensembles allow the design to scale with respect to potential extensions of the basic architecture. For instance the Planner itself can be subject to migration in case the application does not have any frontend. Additionally, when understanding the Planner as an entity controlling the NFPs of the application, it is possible to foresee the existence of multiple Planners per application, thus hierarchically decomposing the adaptation.

6. RELATED WORK

In our previous work [8] we proposed to use Stochastic Performance Logic (SPL) [9] to express rules for adaptation in component systems based on real and predicted performance of individual components. The rules controlling the adaptation are similar to the decision logic of the Planner that compares deployment alternatives for Ab.

The issue and challenges of dynamic deployment adaptation has been formulated [4] and addressed [10] previously. However, in spite of the variety of solutions to the individual challenges (e.g., parameters of decision, migration to stationary only or also to mobile devices, acquisition of NFP-related data, etc.), in the majority of the approaches a predetermined solution is used. On the other hand, our adaptation architecture is dynamic enough to allow for combining multiple solutions simultaneously and selecting among them dynamically. It also provides general means to address the remaining challenges (e.g., scheduling of NFP data acquisition or estimation of cost for running before real execution).

A significant body of work has been devised in the related area of Mobile Cloud Computing (MCC) [1]. Although many of the challenges and solutions can be adopted in ad-hoc clouds, there is a significant difference in perceiving the role of the mobile device, i.e., MCC considers the mobile device as separate from the cloud while ad-hoc clouds do not distinguish among the role of the devices.

We assume an external mechanism responsible for the deployment/migration-related aspects of our approach since it has been intensively researched separately, e.g., in [7].

7. CONCLUSION

In this paper, we have presented our vision on addressing the problem of planning deployment adaptation in ad-hoc clouds. In particular, we have described a generic architecture for deployment adaptation logic that is based on the concept of emergent component ensembles. We have also discussed the potential benefits and scalability of this architecture.

8. ACKNOWLEDGEMENTS

This work has been supported by EU project 257414 ASCENS and GACR project P202/10/J042 FERDINAND.

9. REFERENCES

[1] L. Guan *et al.*, "A survey of research on mobile cloud computing," in *Proc. ICIS'11*, pp. 387–392, IEEE CS, 2011.

[2] G. N. C. Kirby *et al.*, "An approach to ad hoc cloud computing," *CoRR*, vol. abs/1002.4738, 2010.

[3] M. Hölzl *et al.*, "Engineering Ensembles: A White Paper of the ASCENS Project." ASCENS Deliverable JD1.1, 2011. Online: `http://www.ascens-ist.eu`.

[4] B.-G. Chun and P. Maniatis, "Dynamically partitioning applications between weak devices and clouds," in *Proc. MCS'10*, pp. 1–5, ACM, 2010.

[5] C. Jiang *et al.*, "A survey of job scheduling in grids," in *Advances in Data and Web Management*, vol. 4505 of *LNCS*, pp. 419–427, Springer, 2007.

[6] J. Keznikl *et al.*, "Towards Dependable Emergent Ensembles of Components: The DEECo Component Model," in *Proc. WICSA/ECSA'12*, IEEE, 2012.

[7] S.-H. Hung *et al.*, "Executing mobile applications on the cloud: Framework and issues," *Computers & Mathematics with Applications*, vol. 63, no. 2, 2012.

[8] L. Bulej *et al.*, "Performance awareness in component systems: Vision paper," in *Proc. COMPSAC'12 Workshops*, pp. 514–519, IEEE CS, 2012.

[9] L. Bulej *et al.*, "Capturing Performance Assumptions using Stochastic Performance Logic," in *Proc. ICPE'12*, pp. 311–322, ACM, 2012.

[10] M. Sharifi, S. Kafaie, and O. Kashefi, "A survey and taxonomy of cyber foraging of mobile devices," *IEEE Commun. Surveys Tuts.*, vol. 14, 2012.