The Journal of Systems and Software 138 (2018) 37–51

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Empirical validation of cyber-foraging architectural tactics for surrogate provisioning

Fahimeh Alizadeh Moghaddam[a,b,*], Giuseppe Procaccianti[a], Grace A. Lewis[a], Patricia Lago[a]

[a] Software and Services Research Group, Vrije Universiteit Amsterdam, The Netherlands
[b] Systems and Network Engineering Research Group, University of Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 15 June 2016
Revised 14 November 2017
Accepted 21 November 2017
Available online 22 November 2017

Keywords:
Software engineering
Software architecture
Cyber-foraging
Energy efficiency
Resilience
Software sustainability

ABSTRACT

Background: Cyber-foraging architectural tactics are used to build mobile applications that leverage proximate, intermediate cloud surrogates for computation offload and data staging. Compared to direct access to cloud resources, the use of intermediate surrogates improves system qualities such as response time, energy efficiency, and resilience. However, the state-of-the-art mostly focuses on introducing new architectural tactics rather than quantitatively comparing the existing tactics, which can help software architects and software engineers with new insights on each tactic.
Aim: Our work aims at empirically evaluating the architectural tactics for surrogate provisioning, specifically with respect to resilience and energy efficiency.
Method: We follow a systematic experimentation framework to collect relevant data on Static Surrogate Provisioning and Dynamic Surrogate Provisioning tactics. Our experimentation approach can be reused for validation of other cyber-foraging tactics. We perform statistical analysis to support our hypotheses, as compared to baseline measurements with no cyber-foraging tactics deployed.
Results: Our findings show that Static Surrogate Provisioning tactics provide higher resilience than Dynamic Surrogate Provisioning tactics for runtime environmental changes. Both surrogate provisioning tactics perform with no significant difference with respect to their energy efficiency. We observe that the overhead of the runtime optimization algorithm is similar for both tactic types.
Conclusions: The presented quantitative evidence on the impact of different tactics empowers software architects and software engineers with the ability to make more conscious design decisions. This contribution, as a starting point, emphasizes the use of quantifiable metrics to make better-informed trade-offs between desired quality attributes. Our next step is to focus on the impact of runtime programmable infrastructure on the quality of cyber-foraging systems.

1. Introduction

In 2014, the number of mobile users exceeded the number of desktop users globally, which was about 1.7 billion users (Bosomworth, 2015). Consequently, many computation tasks are migrated to handheld devices as mobile apps. Statistics provided by “The Statistics Portal” forecast approximately 269 billion mobile app downloads for 2017, which is around 20% more than the previous year (Statistica., 2013). Although handheld devices are often selected as the main target for consumers and app developers, they are still limited in resources in terms of computational power and battery life.

The importance of extended device battery life has motivated software architects to introduce *Mobile Cloud Computing* solutions, in which the cloud takes charge of compute- and data-intensive tasks. Although these solutions significantly help to address resource limitations, a number of prerequisites need to be met. For example, a reliable Internet connection must exist between the handheld device and the cloud, which is not necessarily guaranteed in resource-scarce environments. Resource-scarce environments usually lack stable environmental conditions. *Cyber-foraging* has been introduced to enable resource-limited devices to benefit from available external resources in such environments with dynamic conditions.

A number of cyber-foraging tactics have been identified and categorized in Lewis et al. (2014; 2016) to help software architects select the best tactics to meet system requirements. In this

* Corresponding author at: De Boelelaan 1081a, Vrije Universiteit Amsterdam, The Netherlands.

E-mail addresses: f.alizadehmoghaddam@vu.nl (F. Alizadeh Moghaddam), g.procaccianti@vu.nl (G. Procaccianti), g.a.lewis@vu.nl (G.A. Lewis), p.lago@vu.nl (P. Lago).

https://doi.org/10.1016/j.jss.2017.11.047
0164-1212/© 2017 Elsevier Inc. All rights reserved.

work we particularly focus on the "Surrogate Provisioning" tactics from an experimentation point of view. We study to what extent the cyber-foraging architectural tactics for surrogate provisioning impact system resilience and energy efficiency. Our findings guide software architects and software engineers to trace the impact of their design decisions with scientific insights concluded from quantifiable metrics. Our main contributions are:

- we provide a detailed description of cyber-foraging tactics for surrogate provisioning;
- we present a runtime optimization algorithm to support surrogate provisioning tactics and describe a proof-of-concept implementation;
- we show the systematic design and execution of our experimentation approach applied to surrogate provisioning, which can be reused for validating other cyber-foraging architectural tactics;
- we report on the execution and the results of our empirical experimentation aimed at quantifying the impact of the cyber-foraging tactics for surrogate provisioning on resilience and energy efficiency in a controlled environment;
- we provide an evaluation of the cyber-foraging tactics for surrogate provisioning, emphasizing trade-offs with respect to different system qualities.

This paper is organized as follows: Section 2 presents an overview of the cyber-foraging architectural tactics. Section 3 focuses on the surrogate provisioning tactics and how online optimization algorithms play a role in the system. In Section 4 we describe the scope of the experimentation using the goal, research questions, and metrics. Section 5 provides details of the planning steps from different perspectives such as context selection, variable selection, hypothesis formulation, subject selection, experiment design, and instrumentation. The steps taken to execute the experiments are explained in Section 6. In Sections 7 and 8 we present and discuss our results. Section 9 discusses the implications of our findings for software architecture. In Section 10 we describe the possible threats to validity and their mitigation. Section 11 discusses related work. Finally, Section 12 concludes the paper and outlines the research direction for our future work.

2. Background

Cyber-foraging is a mechanism that leverages cloud servers, or local servers called surrogates, to augment the computation and storage capabilities of resource-limited mobile devices while extending their battery life (Satyanarayanan, 2001). There are two main forms of cyber-foraging (Flinn, 2012; Lewis and Lago, 2015a; Sharifi et al., 2012). One is computation offload, which is the offload of expensive computation in order to extend battery life and increase computational power. The second is data staging to improve data transfers between mobile devices and the cloud by temporarily staging data in transit on intermediate, proximate nodes. While cyber-foraging can take place between mobile devices and cloud resources, our focus is on systems that use intermediate, proximate surrogates.

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both (Bass et al., 2012). Software architectures are created because a system's qualities, expressed as functional and non-functional requirements, can be analyzed and predicted by studying its architecture.

One of the main challenges of building cyber-foraging systems is the dynamic nature of the environments that they operate in. For example, the connection to an external resource may not be available when needed or may become unavailable during a computation offload or data staging operation. As another example, multiple external resources may be available for a cyber-foraging system but not all have the required capabilities. Adding capabilities to deal with the dynamicity of the environment has to be balanced against resource consumption on the mobile device so as to not defeat the benefits of cyber-foraging. Being able to reason about the behavior of a cyber-foraging system in light of this uncertainty is key to meeting all its desired qualities, which is why software architectures are especially important for cyber-foraging systems.

Given the potential complexity of cyber-foraging systems, it would be of great value for software architects to have a set of reusable software architectures and design decisions that can guide the development of these types of systems, the rationale behind these decisions, and the external context/environment in which they were made; this is called *architectural knowledge* (Kruchten et al., 2006; Lago and Avgeriou, 2006). One way to capture architectural knowledge is in the form of *software architecture strategies*.

We define a *software architecture strategy* as the set of architectural design decisions that are made in a particular external context/environment to achieve particular system qualities. Software architecture strategies are codified as architectural tactics that can be reused in the development of software systems. We define *architectural tactics* as design decisions that influence the achievement of a system quality (i.e., quality attribute) (Bass et al., 2012).

Software architecture strategies for cyber-foraging systems are therefore the set of architectural design decisions, codified as reusable tactics, that can be used in the development of cyber-foraging systems to achieve particular system qualities such as resource optimization, fault tolerance, scalability and security, while conserving resources on the mobile device (Lewis, 2016).

In previous work we conducted a systematic literature review (SLR) on architectures for cyber-foraging systems (Lewis et al., 2014; Lewis and Lago, 2015a). The common design decisions present in the cyber-foraging systems identified in the SLR were codified into functional and non-functional architectural tactics (Lewis and Lago, 2015a; 2015b). Functional tactics are broad and basic in nature and correspond to the architectural elements that are necessary to meet cyber-foraging functional requirements. Non-functional tactics are more specific and correspond to architecture decisions made to promote certain quality attributes. Non-functional tactics have to be used in conjunction with functional tactics.

A cyber-foraging system must have at a minimum the following combination of functional tactics:

- Computation Offload and/or Data Staging tactics to provide cyber-foraging functionality.
- A Surrogate Provisioning tactic to provision a surrogate with the offloaded computation or data staging capabilities.
- A Surrogate Discovery tactic so that the mobile device can locate a surrogate at runtime.

Then, based on additional functional and non-functional requirements, such as fault tolerance, resource optimization, scalability/elasticity, and security, complementary tactics are selected.

The work in this paper focuses on surrogate provisioning tactics. We compare the different surrogate provisioning tactics from an architectural point of view with respect to their resilience and energy efficiency.

3. Surrogate provisioning tactics

3.1. Tactics description

To be able to use a surrogate for cyber-foraging, it has to be provisioned with the offloaded computation and/or the computational elements that implement the offloaded computation or en-

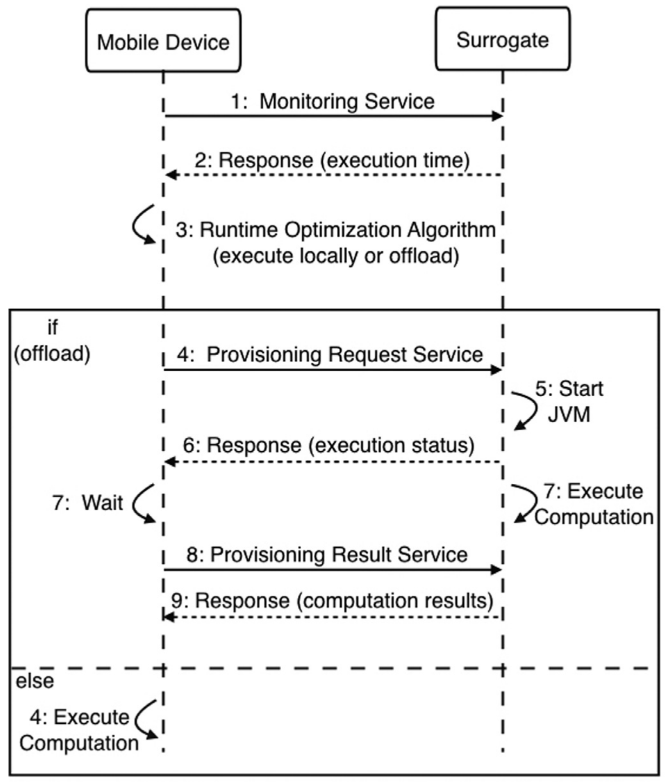

Mobile Device
Surrogate
1: Monitoring Service
2: Response (execution time)
3: Runtime Optimization Algorithm (execute locally or offload)
if (offload)
4: Provisioning Request Service
5: Start JVM
6: Response (execution status)
7: Wait
7: Execute Computation
8: Provisioning Result Service
9: Response (computation results)
else
4: Execute Computation

Fig. 1. The sequence diagram for Static Surrogate Provisioning in case of computation offloading.

able data staging. There are two main types of tactics for surrogate provisioning (Lewis and Lago, 2015a):

- *Static Surrogate Provisioning*: Surrogates are pre-provisioned with the capabilities that are requested by mobile clients. Fig. 1 shows the sequence diagram of how a pre-provisioned surrogate interacts with the mobile device. The mobile app decides whether to request remote execution or execute the computation locally. To make that decision it first collects data on the network connection and the surrogate status through a *monitoring service*. A *runtime optimization algorithm* outlines the optimum offloading plan based on the input data. If the plan is not to offload, the computation will be executed locally in the mobile device. If the plan is to offload, a *provisioning request service* is called that starts a JVM in the surrogate and notifies the mobile app with the resulting status. The mobile app waits for a specific period and then requests the results of the computation through a *provisioning result service*.
- *Dynamic Surrogate Provisioning*: Surrogates are provisioned at runtime with the computation capabilities. Surrogates can receive the offloaded computation from either the mobile device or the cloud. Fig. 2 shows the steps that take place in Dynamic Surrogate Provisioning. Similar to Static Surrogate Provisioning, the mobile app must first collect data through a *monitoring service* but the monitoring is more complex as the status data of the cloud repository is also required. A runtime optimization algorithm can suggest either different provisioning sources (the cloud or the mobile device) or local execution. If the plan is to offload, the mobile app calls a *provisioning request service*. In the case of provisioning from the mobile device, the mobile device *sends the computation capability* to the surrogate itself while in provisioning from the cloud, the mobile device only informs the surrogate with the location of the offloaded computation in the form of a URL. Therefore, the surrogate will be able to download the computation from a cloud repository and install the computation inside an execution container (JVM as shown in the figure). Again, the mobile app can retrieve the results by calling a *provisioning result service*.

In our previous study, we propose a decision model to select the best fitted architectural tactics according to functional and non-functional requirements in cyber-foraging (Lewis et al., 2016). Fig. 3 shows the decision model specified for surrogate provisioning tactics. As the figure shows, there are pros and cons for each surrogate provisioning tactic. For instance, Static Surrogate Provisioning *simplifies the deployment process*. Therefore, it is a good match for applications with a small set of computations or data processing operations that can be pre-loaded on the surrogate. Static Surrogate Provisioning performs the best in cyber-foraging applications, in which multiple surrogates offer the same capabilities. This reduces *flexibility* because surrogates are limited by the pre-installed capabilities. Another disadvantage with Static Surrogate Provisioning is a reduction on *maintainability* because changes to capabilities must be propagated to all surrogates. Differently, Dynamic Surrogate Provisioning offers *greater flexibility* because capabilities are not limited by what is already installed, making it a good match for when there is a large set of capabilities that can execute on a surrogate. Various capabilities that reside on the mobile device can be offloaded to a surrogate at runtime. However, Dynamic Surrogate Provisioning has a negative impact on *provisioning time* compared to Static Surrogate Provisioning because capabilities have to be downloaded first from either a cloud repository or the mobile device. In the case of provisioning from the cloud, the capabilities must exist in a repository in the cloud, and connectivity between the surrogate and the repository is required to download the capabilities, which affects *availability* negatively. This improves *maintainability* because changes to capabilities only need to be propagated to the cloud repository. In contrast, in the case of provisioning from the mobile device, *maintainability* is reduced because changes to offloadable capabilities must be propagated to all mobile devices. Depending on the size of the capability to be transferred, *bandwidth efficiency* could be negatively affected. Consequently, *energy efficiency* is decreased on the mobile device because of the battery power required on the mobile device to send the capability to the designated surrogate.

3.2. Runtime optimization algorithm for surrogate provisioning

In this paper, we propose an algorithm for runtime selection of a task execution environment aided by surrogate provisioning tactics. The objective of our optimization algorithm is to minimize response time, which is the period of time it takes for the mobile device to receive the results, either by offloading or by local execution. To do so, response time (RT) is estimated for different scenarios using Eqs. (1) and (2), which are adopted from Chang and Hung (2011). RT_{offload} is used for both types of surrogate provisioning (Static and Dynamic). However, data size differs for different tactics.

$$RT_{\text{local}} = T_{\text{local}} \tag{1}$$

$$RT_{\text{offload}} = T_{\text{surrogate}} + \frac{\text{data size}}{BW_{network}} + T_{delay} * \left(1 + \frac{\text{data size}}{\text{TCP window size}}\right) \tag{2}$$

T_{local} and $T_{\text{surrogate}}$ show the time it takes for the mobile device and the surrogate to execute the computation task. The local execution time is known in advance. The surrogate execution time is calculated in the algorithm at runtime. We calculate the

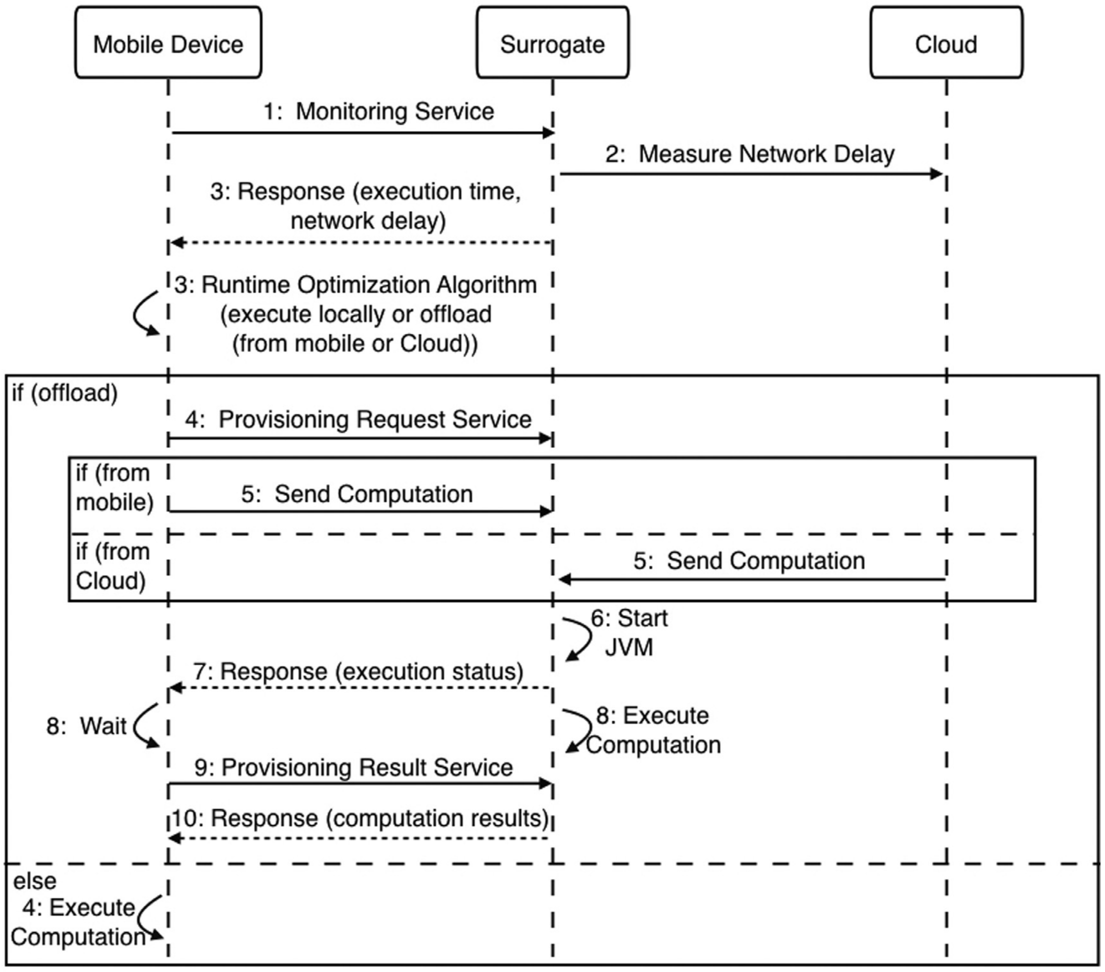

Mobile Device
Surrogate
Cloud
1: Monitoring Service
2: Measure Network Delay
3: Response (execution time, network delay)
3: Runtime Optimization Algorithm (execute locally or offload (from mobile or Cloud))
if (offload)
4: Provisioning Request Service
if (from mobile)
5: Send Computation
if (from Cloud)
5: Send Computation
6: Start JVM
7: Response (execution status)
8: Wait
8: Execute Computation
9: Provisioning Result Service
10: Response (computation results)
else
4: Execute Computation

Fig. 2. The sequence diagram for Dynamic Surrogate Provisioning in case of computation offloading.

network connection overhead using $BW_{network}$, which is the wireless network bandwidth between the mobile device and the surrogate, and T_{delay}, which shows the network delay. $BW_{network}$ is obtained offline using the *iperf*[1] application and then hard-coded on the mobile device. T_{delay} is measured at runtime using *ping* messages. We use the default value of TCP window size[2] on Android, which is 65,536 bytes (64KB).

The pseudo-code in Algorithms 1 and 2 specifies the steps taken by our algorithm in cases of static and Dynamic Surrogate Provisioning. The algorithm relies on Eqs. (1) and (2) to calculate the response time values. As shown in the pseudo-code, the optimization algorithm for Dynamic Surrogate Provisioning performs a number of extra steps, in which it calculates the time-overhead to install the computation on the surrogate.

4. Experiment definition

With our experimentation we aim to empirically evaluate the surrogate provisioning tactics, and provide software architects and software engineers with reproducible scientific insights. The systematic design of our experimentation can be adopted by other researchers as a viable, reusable approach. Our experimentation process follows the well-known framework introduced by Basili et al. (1986). It consists of four phases: 1) Definition, 2) Planning, 3) Execution, and 4) Analysis. For the first phase (Definition), we selected the Goal-Question-Metric (GQM) paradigm.

The GQM is a top-down conceptual decomposition of the goal into questions and metrics, that provides the traceability of mea-

Algorithm 1 Runtime optimization algorithm for Static Surrogate Provisioning.

```
while $TRUE$ do
  if $newComputationRequest$ not Null then
    $RT_{local}$ ←estimate the local execution time ($T_{local}$)
    Calculate $RT_{offload}$ (the output of equation 2):
    $T_{surrogate}$ ←estimate the remote execution time
    $BW_{network}$ ←estimate the bandwidth of the connection between the mobile device and the surrogate
    $T_{delay}$ ←measure the network delay of the connection between the mobile device and the surrogate
    if $RT_{local} < RT_{offload}$ then
      Execute the computation locally and update $RT_{local}$ based on the execution results
    else
      Start the offloading process and update the $RT_{offload}$ variables based on the execution results
    end if
  end if
end while
```

[1] It is a known tool to measure network performance: https://iperf.fr/.

[2] The Transmission Control Protocol (TCP) is one of the main standard network protocols that complements the Internet Protocol (IP).

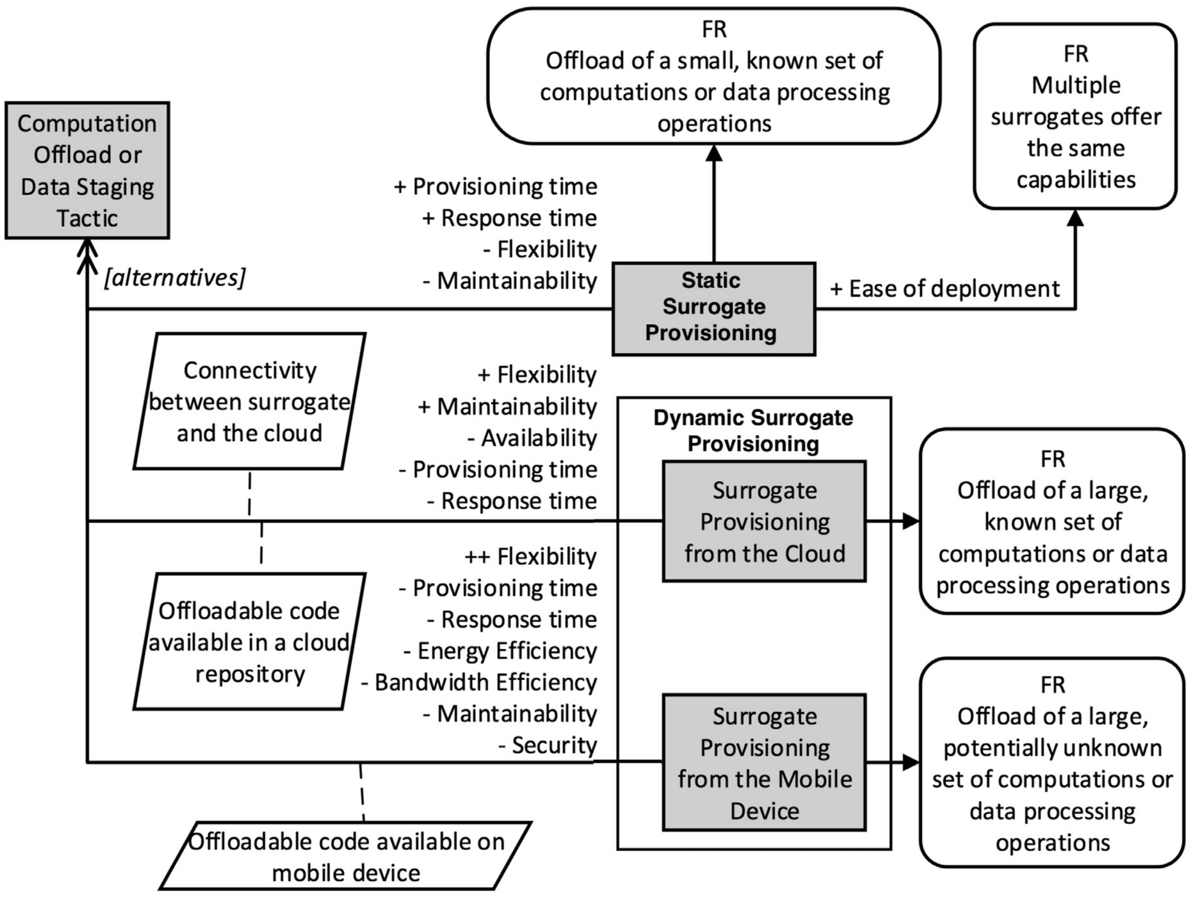

Computation Offload or Data Staging Tactic
[alternatives]
FR Offload of a small, known set of computations or data processing operations
FR Multiple surrogates offer the same capabilities
+ Provisioning time
+ Response time
- Flexibility
- Maintainability
Static Surrogate Provisioning
+ Ease of deployment
Connectivity between surrogate and the cloud
+ Flexibility
+ Maintainability
- Availability
- Provisioning time
- Response time
Dynamic Surrogate Provisioning
Surrogate Provisioning from the Cloud
FR Offload of a large, known set of computations or data processing operations
Offloadable code available in a cloud repository
++ Flexibility
- Provisioning time
- Response time
- Energy Efficiency
- Bandwidth Efficiency
- Maintainability
- Security
Surrogate Provisioning from the Mobile Device
FR Offload of a large, potentially unknown set of computations or data processing operations
Offloadable code available on mobile device

Fig. 3. The decision model for surrogate provisioning tactics adapted from our previous study (Lewis et al., 2016).

Algorithm 2 Runtime optimization algorithm for Dynamic Surrogate Provisioning.

```
while $TRUE$ do
    if $newComputationRequest$ not Null then
        $RT_{local}$ ←estimate the local execution time ($T_{local}$)
        Calculate two values of $RT_{offload}$ for two data sources (the output of equation 2):
        $T_{surrogate}$ ←estimate the remote execution time
        $BW_{network}$ ←estimate the bandwidth of the connection between the mobile device and the surrogate
        $BW_{network-cloud}$ ←estimate the bandwidth of the connection between the surrogate and the designated cloud server
        $T_{delay}$ ←measure the network delay of the connection between the mobile device and the surrogate
        $T_{delay-cloud}$ ←measure the network delay of the connection between the surrogate and the designated cloud server
        if $Min(RT_{local}, RT_{offload}$ from the mobile, $RT_{offload}$ from the cloud)$== RT_{local}$ then
            Execute the computation locally and update $RT_{local}$ based on the execution results
        else
            Start the offloading process and update the $RT_{offload}$ variables based on the execution results
        end if
    end if
end while
```

surement data in the goal achievement (Basili, 1992). Our goal is formulated as follows:

"Analyze **architectural tactics for surrogate provisioning** for the purpose of **evaluation** with respect to **resilience and energy efficiency** from the viewpoint of **software architects** and **software engineers** in the context of **cyber-foraging applications**"

Our objects are surrogate provisioning tactics. We focus on two quality attributes: resilience and energy efficiency. We evaluate these tactics from the point of view of a software architect or software engineer: this means that our results will be helpful when making design decisions related to these quality attributes and their possible trade-offs. Our results apply to the general field of cyber-foraging applications, although they might provide useful insights for a broader range of software systems.

We compare the tactics based on a number of predefined metrics. To proceed, we define the following questions, which elaborate our goal in a quantifiable way.

RQ1: What is the difference in terms of resilience between the surrogate provisioning tactics?

Cyber-foraging tactics are known to provide systems with dynamic behavior to account for unavailable resources. However, the extent to which systems can benefit from this dynamicity has not been studied. We answer this question by quantifying the resilience of a system (Almeida and Vieira, 2011). Before and during the execution of the mobile applications we introduce a change to the system, which requires an online decision. The changes vary from low battery level to bad network connection. Fig. 4 shows the phases that a resilient system goes through when a change

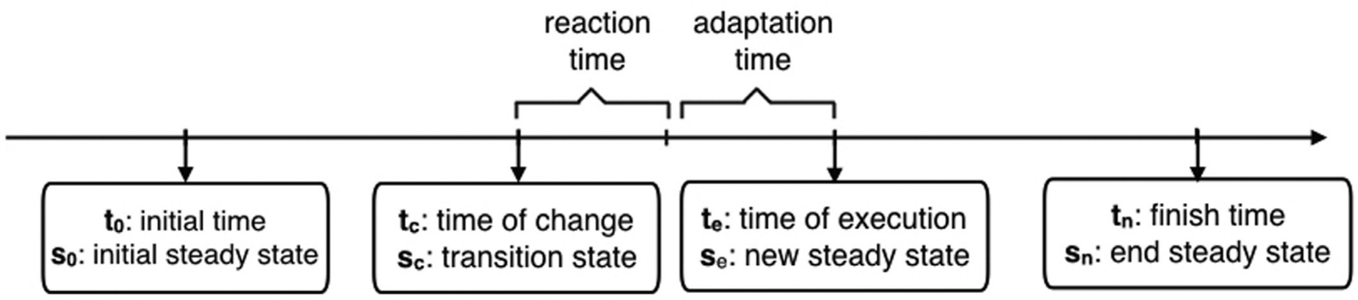


Fig. 4. The states of a resilient system when introducing a system or environmental change.

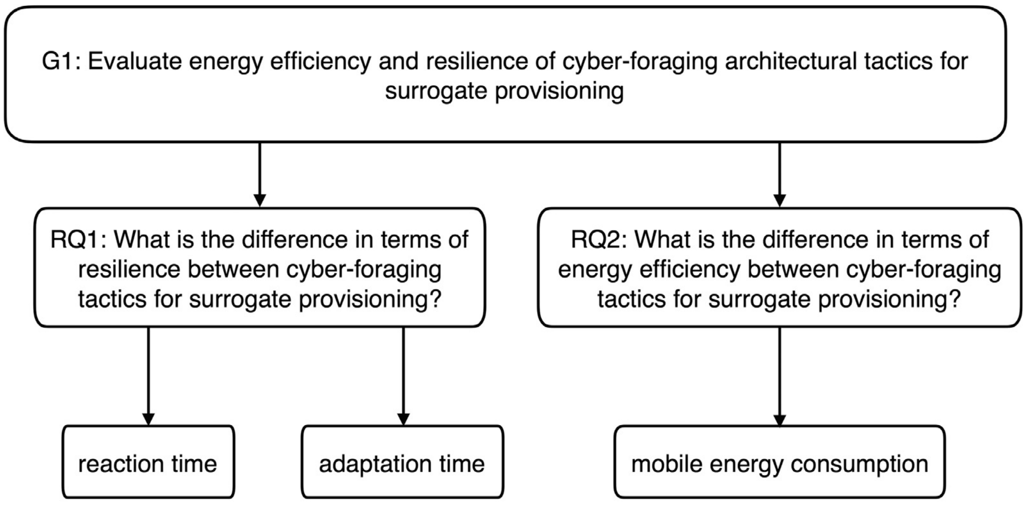

G1: Evaluate energy efficiency and resilience of cyber-foraging architectural tactics for surrogate provisioning
RQ1: What is the difference in terms of resilience between cyber-foraging tactics for surrogate provisioning?
RQ2: What is the difference in terms of energy efficiency between cyber-foraging tactics for surrogate provisioning?
reaction time
adaptation time
mobile energy consumption

Fig. 5. The GQM graph summarizing the relation between the goal of our experiment, the research questions, and the metrics.

occurs. At first the system is in its “Initial steady state” until the time of the change. With change, a two-phase transition state starts, which includes the “Reaction” and “Adaptation” phases. At the end of the adaptation phase, the system returns to a steady state. The faster that the system passes through the transition phase to the steady state, the higher resilience the system provides (Almeida and Vieira, 2011). We compare the resilience of the system when using different surrogate provisioning tactics by analyzing the reaction and the adaptation times.

RQ2: What is the difference in terms of energy efficiency between the surrogate provisioning tactics?

We analyze the energy efficiency of the surrogate provisioning tactics at runtime. We calculate the energy consumption of the mobile device using the measured average power consumption during the execution of a synthetic application. Basically, the synthetic application consists of a computation task, which can either be performed locally or offloaded to a surrogate.

We use the following metrics to quantitatively answer our questions:

- **Mobile energy consumption**: the number of joules consumed to execute the computation task.
- **Reaction time**: the time the system takes to detect a change and decide on a suitable setup.
- **Adaptation time**: the time the system takes to adapt to the new setup.

The directed GQM graph in Fig. 5 indicates how our goal is covered by providing quantified answers to the questions using the metrics. *RQ1* is dependent on the metrics “Reaction time” and “Adaptation time” which enable us to quantify resilience of a system. *RQ2* requires the metric “Mobile energy consumption” to investigate how the surrogate provisioning tactics impact the energy consumption of the mobile application.

5. Experiment planning

In this section we describe our experimentation in terms of 1) Variable selection, 2) Hypothesis formulation, 3) Subject selection, 4) Experiment design and 5) Instrumentation.

5.1. Variable selection

The metrics identified in the GQM tree (See Fig. 5) are our *dependent variables*: Mobile energy consumption, Reaction time, and Adaptation time. We use the dependent variables to answer the research questions and draw conclusions. We also consider two additional dependent variables, which help to calculate the mobile energy consumption metric:

- Average power consumption: The power consumption of the mobile device is measured at runtime in 1 s intervals using a system profiler. We calculate the average value of the power consumption values for each trial.
- Response time: It specifies the duration from the moment that the computation is requested until the results are collected on the mobile device. It includes the time to execute the computation task and the time to offload the task to the surrogate if offloading is decided.

In contrast, the *independent variables* are those that are controllable in the experiment. In our case, the deployment of a surrogate provisioning tactic is the main independent variable that we select as a factor. We define two distinct treatments:

- **Treatment 1: Static Surrogate Provisioning**: In this treatment, the surrogate is pre-provisioned with the offloaded computation.
- **Treatment 2: Dynamic Surrogate Provisioning**: In this treatment, the surrogate is provisioned at runtime. Depending on environmental conditions such as the quality of the network connection, the capabilities can be downloaded from the mobile device itself or from a remote host in the cloud.

The runtime optimization algorithm implemented for the treatments decides at runtime whether the computation should be executed locally or remotely. In addition, we performed a number of baseline measurements to provide a reference set of values for our metrics. These measurements were performed on the same mobile device used for our experiment, running the computation task *locally*.

Other independent variables, such as hardware and software configurations are related to the execution environment. These variables have been kept constant in our experimentation (see Section 5.5) to avoid confounding factors. Another independent variable we considered is the network connection because in cyber-foraging scenarios these connections might not always be available and reliable. For this reason, in our experimentation we introduced a 10% probability of having a faulty connection, in which case our adaptation algorithm has to select local computation.

5.2. Hypotheses formulation

In this section, we formulate the aforementioned research questions into *Null* and *Alternative* hypotheses.

- *RQ1: What is the difference in terms of resilience between surrogate provisioning tactics?*
  As explained earlier, the resilience of the system is dependent on the duration of the transition phase when introducing a change. We observe the difference in resilience of the two cyber-foraging systems based on Eq. (3).

  $$\Delta T = T_d - T_s \tag{3}$$

  T_s, in seconds, is the sum of the reaction and adaptation time of the system using a static provisioning tactic, and T_d shows the same for a dynamic provisioning tactic.
  The null hypothesis in Eq. (4) suggests that both surrogate provisioning tactics provide the same level of resilience. In contrast, the alternate hypothesis in Eq. (5) states that the resilience of the cyber-foraging system with Dynamic Surrogate Provisioning is lower (i.e. the transition time is higher) compared to the Static Surrogate Provisioning.

  $$H1_0 : \Delta T \approx 0 \tag{4}$$

  $$H1_a : \Delta T > 0 \tag{5}$$

- *RQ2: What is the difference in terms of energy efficiency between surrogate provisioning tactics?*
  Eq. (6) shows the difference in energy efficiency of the surrogate provisioning tactics. Energy efficiency is measured as the energy consumed (in Joules) when performing a task. EE_d is the energy efficiency of the system with Dynamic Surrogate Provisioning and EE_s is the energy efficiency of the system with Static Surrogate Provisioning. The null hypothesis in Eq. (7) indicates that both surrogate provisioning tactics have the same level of energy efficiency. In contrast, the alternate hypothesis in Eq. (8) implies that energy efficiency differs between the surrogate provisioning tactics.

  $$\Delta EE = EE_d - EE_s \tag{6}$$

  $$H2_0 : \Delta EE \approx 0 \tag{7}$$

  $$H2_a : \Delta EE \neq 0 \tag{8}$$

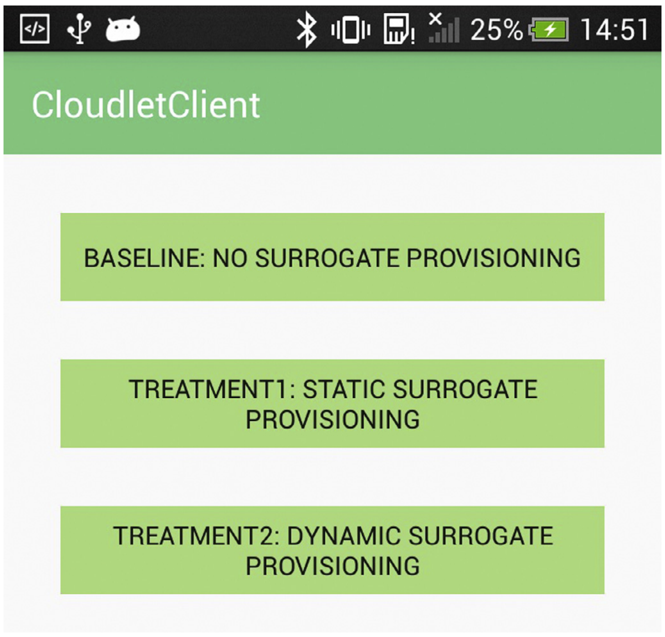

25% 14:51
CloudletClient
BASELINE: NO SURROGATE PROVISIONING
TREATMENT1: STATIC SURROGATE PROVISIONING
TREATMENT2: DYNAMIC SURROGATE PROVISIONING

Fig. 6. Our mobile application, providing three execution scenarios: No surrogate provisioning, Static Surrogate Provisioning, and Dynamic Surrogate Provisioning.

5.3. Subject selection

The main objects of our experiments are the architectural tactics for surrogate provisioning. We select a synthetic application as our subject, using convenience sampling. This defines our study as a **quasi-experiment** as our sample is not randomly selected. Therefore, we can not guarantee that our sample is representative of all mobile applications. However, we mitigate this concern by implementing the typical behavior for a real mobile application in resource-scarce environments (i.e., low coverage and hostile environments (Lewis and Lago, 2015c)). We further discuss this concern in Section 10.

Our mobile application executes a specific computation-intensive task. It converts input colored images to grayscale. It imitates the real life resource-hungry computations that utilize the available resources in the mobile device notably. Hence, our experimentation still provides valuable information with regards to the cyber-foraging applications.

We implemented our Android mobile application to support three different scenarios, as shown in Fig. 6. Despite the architectural differences in each scenario, they all perform a specific computation task, known as our subject:

- In the *baseline* scenario, the computation task can only be executed on the mobile device. The mobile app will not have the flexibility to offload the computation task to external devices. No cyber-foraging architectural tactics are implemented in this scenario.
- The *static* treatment implements the cyber-foraging architectural tactic for Static Surrogate Provisioning. This tactic gives

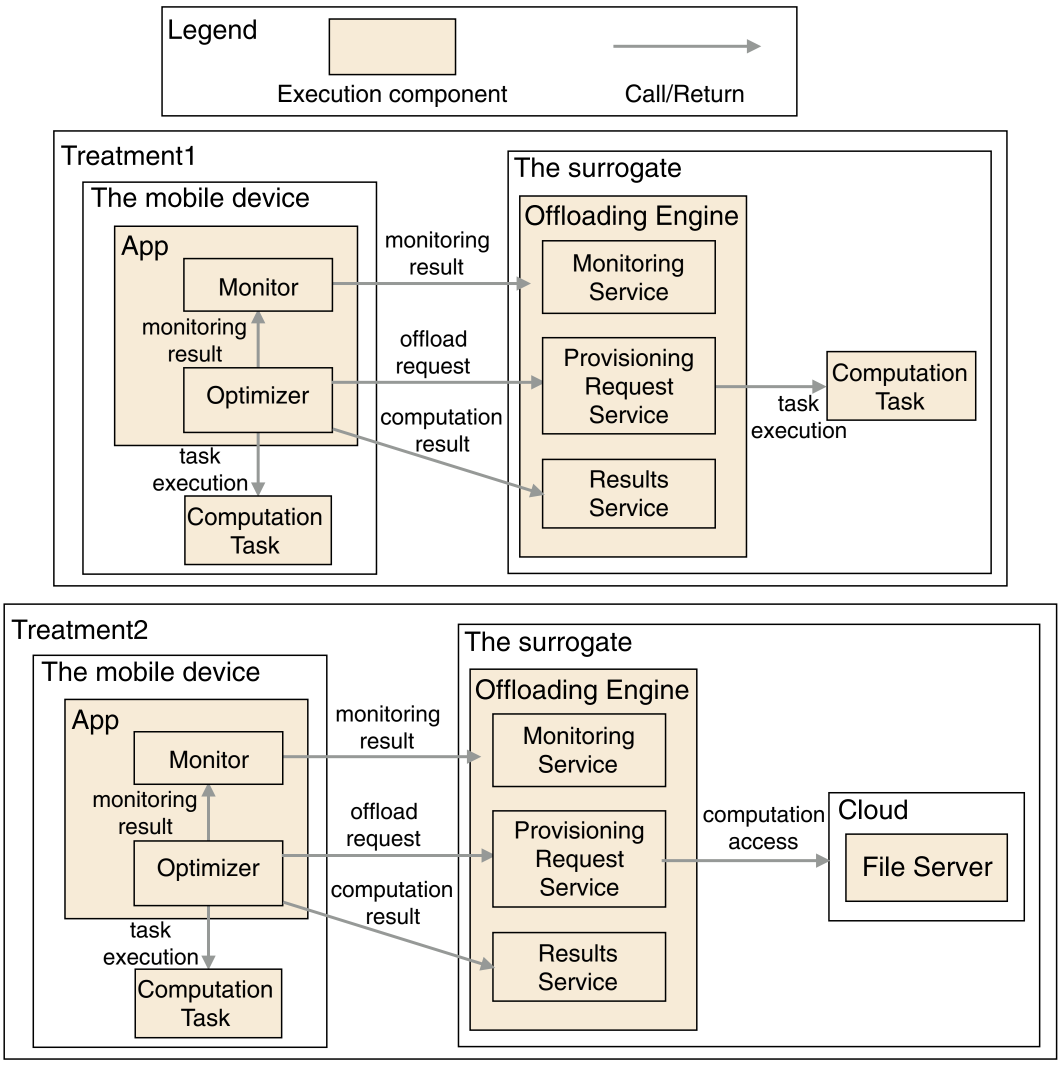


Fig. 7. The architecture of the cyber-foraging mobile application implementing the Static (Treatment 1) and the Dynamic (Treatment 2) Surrogate Provisioning tactics.

more flexibility to the mobile app compared to the baseline scenario, as follows: depending on the battery level of the mobile device and other availability factors, the mobile app might decide to offload the computation task to a nearby surrogate. In this treatment, the surrogate capability is static, i.e., the surrogate is provisioned at design time with the specific computation task. So, if at runtime the mobile app decides to offload the computation, it only needs to invoke the execution on the surrogate.

- The *dynamic* treatment implements the cyber-foraging architectural tactic for Dynamic Surrogate Provisioning. This tactic provides the highest flexibility to the mobile app in terms of the variation of the offloaded computation. However, the surrogate must be provisioned with the capability at runtime. The Dynamic Surrogate Provisioning can take place from two different resources, the mobile device itself or a remote cloud server. The mobile app must decide if the computation should be offloaded and if so, from which resource the surrogate should be provisioned.

In order to implement the two treatments, we set up a number of web services on the surrogate. Fig. 7 shows how the services interact with different components of the mobile app for each treatment. In particular, the **Optimizer** of the mobile app, which receives the task execution request from the users, is the component that starts the optimization process. It retrieves monitoring data from the **Monitor** component and accordingly selects the task execution environment, which is either *The mobile device* or *The surrogate*, where the **Computation Task** is executed. For the surrogate the following describes each web service:

- *Monitoring service*: It collects status data such as computation execution time and remote host network delay, which can be invoked by the mobile app.
- *Provisioning request service*: It receives the offload request from the mobile app. In the case of the static treatment, the computation task is already installed on the surrogate and the mobile app will only send the necessary execution parameters to start the computation. Differently, in the dynamic treatment, the mobile app sends the surrogate provisioning parameters along with the execution parameters. In the case of surrogate provisioning from the cloud, the provisioning parameter is a URL to a cloud-based **File Server** that provides the computation capability. However, in case of the surrogate provisioning from the mobile device, the provisioning parameter is the computation task itself that will be installed on the surrogate.
- *Results service*: It provides the results of the computation to the mobile app. The mobile app will wait for a certain amount of time and then invoke this service to collect the results.

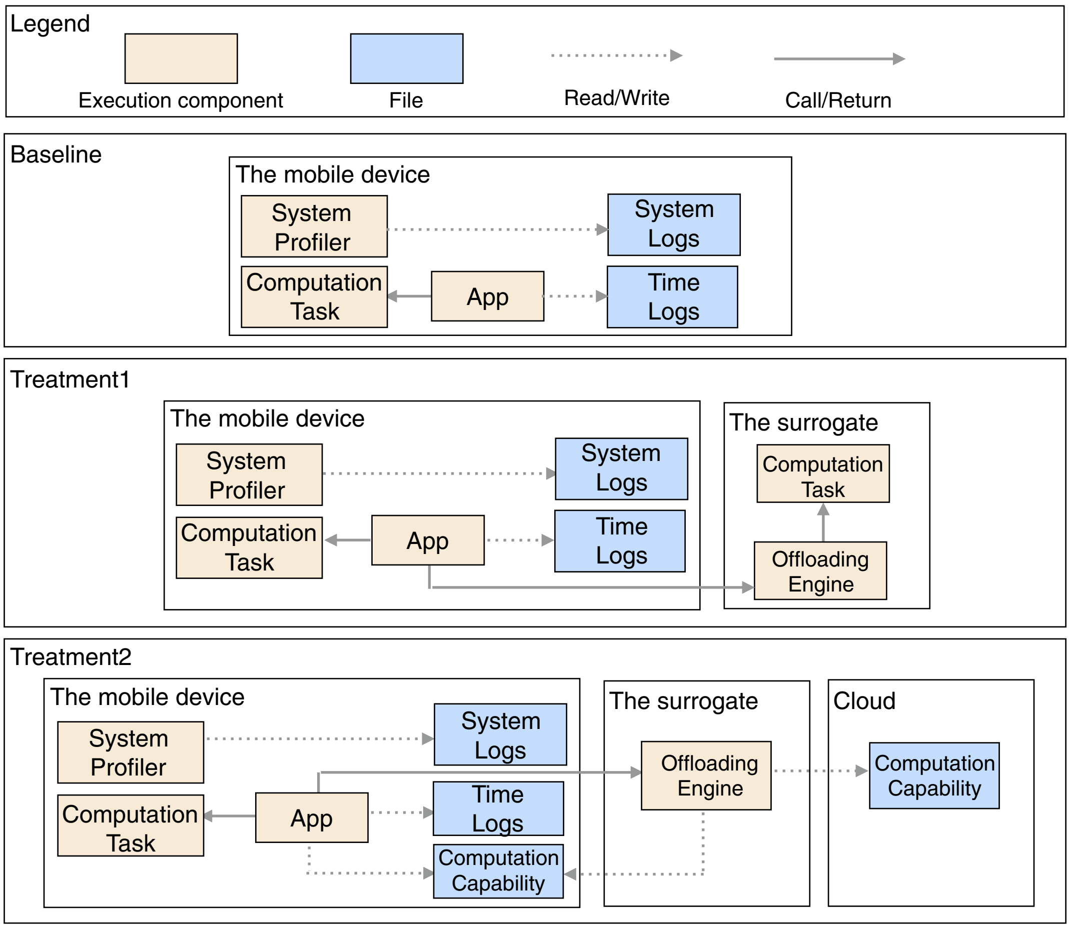


Fig. 8. The experiment architecture of different scenarios: the baseline and both treatments.

Table 1
The trial set for the experiment.

	Factor: Surrogate provisioning	
Baseline	Static	Dynamic
Control (30 trials)	Group A (30 trials)	Group B (30 trials)

A cyber-foraging mobile application that implements the surrogate provisioning tactics will follow the steps in Figs. 1 and 2 to interact with the web services residing in the surrogate.

5.4. Experiment design

The experiment factor in our experimentation is the use of cyber-foraging architectural tactics for surrogate provisioning. Our factor has two treatments—Dynamic and Static—that identify the two main experimental groups. In addition, a baseline scenario with no cyber-foraging tactics deployed has been evaluated as a control group. Each group is composed of 30 trials (i.e., experimental runs) (Table 1).

5.5. Instrumentation

All the experiments were executed in the Green Lab of Vrije Universiteit Amsterdam.[3] Fig. 8 displays the high-level architecture of our experimentation for the different scenarios. The number of components for each scenario varies depending on the required *flexibility*. For our experimentation we used a number of hardware and software tools:

Hardware

- Test server (HP DL360 G5): Hosts the web services of our surrogate and has a LAMP server running on its Ubuntu Server 12.04 operating system.
- Mobile device (HTC One X): Runs our surrogate client program. It is based on Android OS 4.2.2.
- Linksys X2000 wireless-N router: Provides a wireless connection between the mobile device and the surrogate.

Software

- LAMP stack: Hosts the web services provided by the surrogate.
- JVM: Runs the byte-codes offloaded by the mobile device. The JVM is needed on both the surrogate for remote execution and the mobile device for local execution.
- PowerTutor: Open-source application that logs the power consumption of different system components on a mobile device (Zhang et al., 2010).

6. Experiment execution

In this section we describe the operational phase of our experimentation.

6.1. Data collection

For each scenario (*Baseline, Static Surrogate Provisioning, Dynamic Surrogate Provisioning*) we performed 30 trials. During each

[3] http://www.s2group.cs.vu.nl/green-lab/.

Table 2
General overview of the dataset.

Independent variable	Treatment	Min	Median	Mean	Max
	Baseline	61.13	94.60	96.05	152.80
Energy consumption (J)	Static	7.73	8.09	12.70	43.69
	Dynamic	7.80	8.16	9.88	38.79
	Baseline	0.424	0.518	0.518	0.604
Power consumption (W)	Static	0.347	0.363	0.364	0.378
	Dynamic	0.348	0.362	0.362	0.381
	Baseline	121.9	181.1	185.7	286.7
Response time (s)	Static	22.04	22.17	34.89	120.70
	Dynamic	22.21	22.37	27.29	107.90

trial, we logged the power consumption of the mobile device using PowerTutor. Also, we recorded the timestamps of different actions in the mobile application.

Using the collected power values, we calculate the energy consumption of the mobile device based on Eq. (9). P_{avg} is the average power consumption of the mobile device during runtime. $T_{response}$ shows the execution time of the task as measured by the mobile device, including (when applicable) the time required for offloading to the surrogate and receiving the results.

$$\text{Energy consumption(J)} = P_{avg} * T_{response} \tag{9}$$

With regards to resilience measurements, we recorded the reaction time and the adaptation time of the cyber-foraging system when a change occurs. We do so by mapping the logged timestamps to the different execution phases.

- *Reaction time*: the period of time it takes for the optimization algorithm to decide on the execution platform, i.e., locally on the phone or remotely on the surrogate.
- *Adaptation time*: if the optimization algorithm decides for local execution, then the adaptation time will be 0. Otherwise, in case of offloaded execution, the adaptation time is measured from when the decision is made until the receipt of the notification from the *"Provisioning Request Service."*

6.2. Data analysis

During our data analysis process, we used the Shapiro–Wilk test to determine whether the normality assumption holds for our data. For hypothesis testing, we used the Wilcoxon signed rank test to determine mean differences between our samples. In addition, we report the effect sizes for our treatments using Hedges'g and Cliff's δ. We use the significance level of 0.05 in all our tests ($\alpha = 0.05$). All the R scripts and the plots are available online.[4]

7. Results

Before we present the results of hypothesis testing, we provide an overview of our observations of the collected data. The summary of the data set is shown in Table 2, which reports descriptive statistics on our response variables: 1) Energy consumption, 2) Power consumption, 3) and Response time. In the table, the *Treatment* column indicates the different treatments used (*static* vs. *Dynamic* Surrogate Provisioning) and the baseline (local execution) presented as a reference.

7.1. General observations

For all three variables, the baseline values follow a normal distribution (as tested by means of the Shapiro–Wilk test).

[4] http://goo.gl/vcalyH.

Table 3
Group means and effect sizes of the transition time values of the treatments.

Median transition time (Static)	0.097
Median transition time (Dynamic)	0.266
Hedges'g	0.634 (medium)
Cliff's δ	0.933 (large)

With regards to our treatments, the data is normally distributed for the power consumption variable. However, the response time values of both treatments do not have a normal distribution. There is an evident significant difference in the execution time between local and the remote computation. Therefore, the non-normality of the values might be caused by the optimization algorithm, which decides on alternative platforms to execute the computation task. However, we checked the distribution of the data separately for each platform and we still detected a non-normal distribution of the response time values. Only the response time values for each treatment were normally distributed with respect to the execution platform.

The distribution of the energy consumption values is only normal for Static Surrogate Provisioning. Also, it is interesting to note that the maximum energy consumed in our treatments (43.69 J and 38.79 J) is still lower than the minimum energy consumption of the baseline (61.13 J). This difference in energy consumption can be explained by the difference in power consumption and response time. Furthermore, Table 2 shows that the local execution when deploying cyber-foraging techniques (our treatments) has better performance than the baseline that does not have the overhead of the optimization algorithm.

7.2. Hypothesis testing

1. ***H1 (Resilience):*** we compare the resilience of both surrogate provisioning tactics based on their transition time (See Eq. (10)). The shorter the transition time, the higher the resilience of the system.

   $$\text{Transition time} = \text{Reaction time} + \text{Adaptation time} \tag{10}$$

   We first check the normality of the transition time values for our treatments. The distribution of the transition data is not normal for both treatments (Shapiro–Wilk's p-value for Treatment 1 $= .0006378$ and for Treatment 2 $= 1.995e-10$)
   Given the non-normal distribution of the values, we make use of the non-parametric *Wilcoxon signed rank* test. It shows that there is a significant difference between the median values of the two treatment samples ($\text{p}-\text{value} = 5.47e-10$). As shown in the box-plot in Fig. 9, Static Surrogate Provisioning benefits from higher resilience compared to Dynamic Surrogate Provisioning. In Table 3 we report the group medians and the effect size of the treatment. In this case, group median is reported instead of mean, as it is a more robust indicator of central tendency in presence of non-normally distributed values.
2. ***H2 (Energy efficiency)***: as explained earlier, we calculate the energy efficiency of each trial based on the amount of energy consumed to perform one computation task. The execution of the computation task is platform independent, and energy consumption measures are done on the mobile device regardless where the task is executed (i.e., locally or remotely). Because the energy consumption values for different treatments do not have a normal distribution, we again use *Wilcoxon signed rank*. We cannot reject our null hypothesis ($\text{p}-\text{value} = .1646$) which indicates there is no significant difference between our treatments.

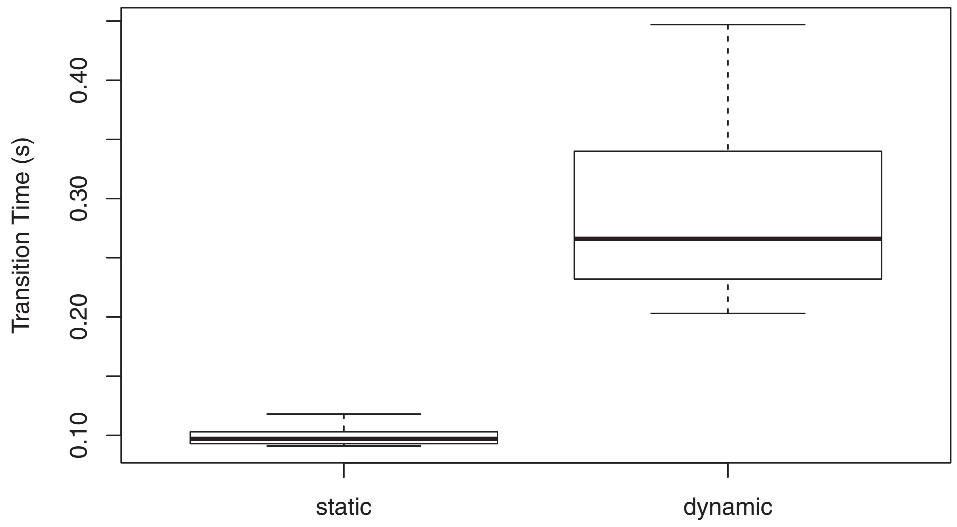


Fig. 9. The box-plot of the transition time values of static and Dynamic Surrogate Provisioning. Outliers are excluded from the plot.

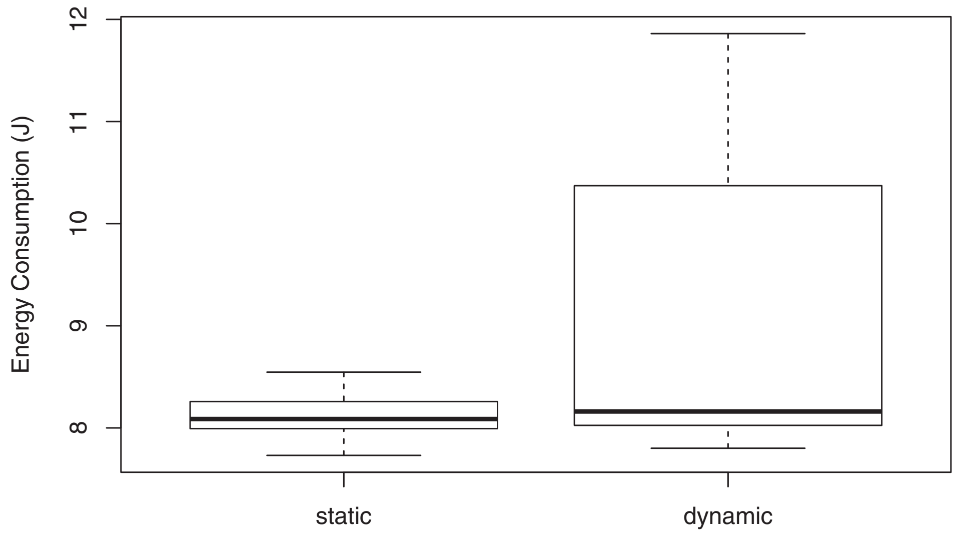


Fig. 10. The box-plot of the energy consumption values of static and Dynamic Surrogate Provisioning. Outliers are excluded from the plot.

Table 4
Group medians and effect sizes results on the energy consumption values of the treatments.

Median energy consumption (Static)	8.087
Median energy consumption (Dynamic)	8.161
Hedges's *g*	−0.296 (small)
Cliff's δ	0.21 (small)

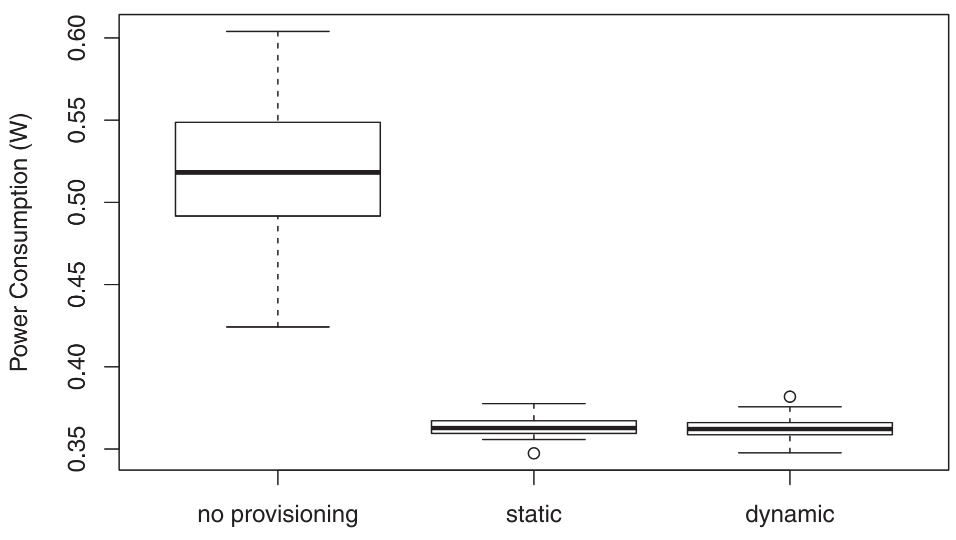
In the box-plot of Fig. 10 we report the energy consumption values for static and Dynamic Surrogate Provisioning. In Table 4 we report the group medians and the effect size of the treatment. Also in this case, group median is reported instead of mean, as it is a more robust indicator of central tendency in presence of non-normally distributed values.

8. Discussion

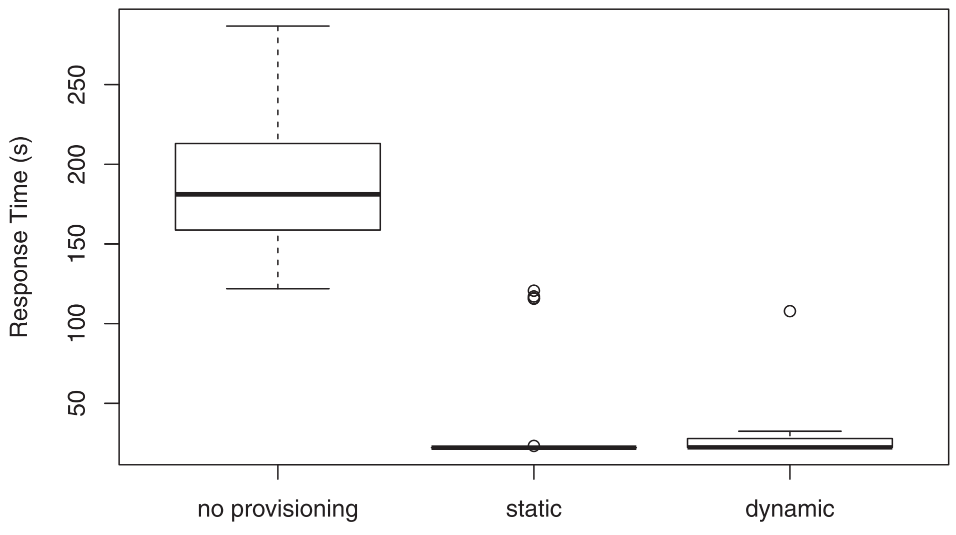
From Table 2 we can already notice that there is a significant difference in the energy consumption, power consumption, and response time between the baseline and the surrogate provisioning tactics. The box-plots in Fig. 11 clearly show this difference.

The outliers of Static and Dynamic surrogate provisioning box-plots in Fig. 11(b) represent cases when the optimization algorithm has decided not to offload the computation task to the surrogate. Surprisingly, in those cases the local execution of the task does not increase power consumption in the same way as in the baseline. The power consumption of the mobile device in presence of cyber-foraging tactics is significantly lower than a non cyber-foraging system. A possible explanation for this effect is related to the temperature of the mobile device, known to affect the variance of the battery discharge curve (Zhang et al., 2010). During our baseline measurements, the computational task was only performed locally. This resulted in a high CPU load for a longer amount of time, which in turn may have raised the temperature of the device and consequently its battery usage (Zhang et al., 2010). We tried to collect temperature data from our logs to confirm our analysis, but unfortunately the amount of data we collected was not sufficient to draw a final conclusion. For this reason, we are preparing a replication of our experiment where we will also collect temperature data.

As discussed in Section 3, both Static and Dynamic Surrogate Provisioning tactics benefit from the increased *flexibility* that the runtime optimization algorithm introduces. Our expectation was that using Dynamic Surrogate Provisioning, the optimization algo-



(a) Power Consumption



(b) Response Time

Fig. 11. The box-plots of the power consumption values and the response time of the baseline compared with our treatments: "Static Surrogate Provisioning" and "Dynamic Surrogate Provisioning".

rithm itself would add an overhead in terms of resource usage. This overhead would result in a difference in the variable "Reaction Time" which includes the execution time of the algorithm. However, we found no significant difference (Wilcoxon signed rank test p-value = .6349) in Reaction Time among different cyber-foraging architectural tactics. This indicates that in our experimentation scenario, performing the online choice does not introduce a significant overhead. Although this result cannot be generalized to all cyber-foraging scenarios, we can conclude that Dynamic Surrogate Provisioning increases *flexibility* while not negatively influencing *performance*.

9. Reflection

Cyber-foraging architectural tactics offer reusable design decisions to accommodate certain types of functionality (in our case computation and data offload) while ensuring required system qualities (such as energy efficiency and resilience). We specifically focus on surrogate provisioning tactics to perform an empirical evaluation. Our work is one of its kind because it measures the adoption of such tactics, and it can lead to a reduction of the learning curve. Having the reusability requirement of the tactics in mind, our work, as an initial step, enriches the knowledge on the tactics with the help of quantitative insights. Therefore, software architects and software engineers can make more conscious and better-informed decisions on selecting the tactics.

In this study, we adopt a systematic experimentation framework to objectively collect and analyze data on the impact of surrogate provisioning tactics. As for data collection, we describe the design procedure of our experimentation step by step, which can benefit other researchers to repeat our experimentation approach, and carry out similar experimentations to study other types of tactics. As for data analysis, we perform a series of statistical tests to extract insights out of data and to validate our hypotheses. Our analysis procedure is meant to be reused by other researchers to observe findings from different measurements.

The quantitative analysis shows that Dynamic and Static Provisioning deliver the promised flexibility with surprisingly negligible costs in terms of resilience and energy efficiency. We investigate resilience, a key quality attribute of sustainable systems, from the transition time perspective. We introduce a specific change that both treatments are resilient to at runtime, namely low battery level. We show that a system adopting Static Surrogate Provisioning is able to take quick action in the presence of a change in its runtime environment. From another angle, a system with Dynamic Surrogate Provisioning can be responsive to more diverse changes. For example if the change is an error in the computation capability of the surrogate, a system with Dynamic Surrogate Provisioning can recover seamlessly by downloading the computation capability from different resources. In general, Dynamic Surrogate Provisioning has a greater self-organizing degree than Static Surrogate Provisioning. Dynamic Static Provisioning has more *flexibility* in terms of lower number of pre-assigned configurations. Consequently, it corresponds to longer transition phases for the system during runtime.

In our study, we report that both surrogate provisioning tactics perform well with no significant difference to fulfill the main objective, which is extending the limited resources lifespan. We measure the energy efficiency from the point of view of the mobile device as an example of resource-scarce environments. However, for software architects the energy efficiency of cyber-foraging systems in its entirety plays an important role as well. The cyber-foraging architectural tactics involve different components, which are utilized differently. Increasing the *adaptability* degree by adding more operational components, might influence the energy efficiency of the entire system negatively. Yet, such trade-offs need to be systematically evaluated, which is our plan for future work.

While cyber-foraging software is extremely novel (see also discussion in Section 10), its applicability and added value can be pervasive. As discussed in Lewis and Lago (2015c), cyber-foraging brings benefits to many contexts, from healthcare and emergency management to Internet of Things and wearable computing. According to the GeSI: Global e-Sustainability Initiative (2015), usage of mobile devices already accounts for nearly half of the ICT sectors emissions, and is expected to steadily grow. Accommodating needs while balancing system qualities and energy efficiency will require smart software solutions such as smart architectural tactics.

Cyber-foraging was developed with resource-scarce environments in mind, where battery life and connectivity are critical. However, our results offer a glimpse of its potential for environments where flexibility is necessary because the context continuously changes (such as smart city sensing) or some resources hosting data/computation can be charged more economically or effectively than others (thanks to, for example, advances in smart grid integration of renewables, or in micro-grid applications).

10. Threats to validity

Our aim is to illustrate the premises and the assumptions behind our experimentation. The classification of the threats follows that by Cook and Campbell (1979). As a general consideration, in this study we are mainly interested in *theory testing*, hence we focus on internal and construct validity, i.e., prove that our effects are representative of the theory and caused by the outcome, rather than conclusion and external validity.

10.1. Conclusion validity

Threats to conclusion validity affect the statistical significance of the findings. In our experimentation, we identify the following conclusion validity threats:

- *Reliability of measures.* Our measures of energy consumption were carried out by means of the PowerTutor software tool. We chose this approach instead of hardware power meters for two main reasons: first, it was deemed more practical; second, this allowed us to measure energy and temporal data on the same device. This removes the problem of multiple data sources with consequent data synchronization and data handling operations which are arguably error-prone. Regarding the accuracy of PowerTutor, according to its developers (Zhang et al., 2010) for 10 s intervals, it is accurate to within 0.8% on average with at most 2.5% error.
- *Low statistical power.* As discussed in Section 5, our sample is a single, synthetic software application. This small sample size obviously reduces the statistical power of our test. However, our target population is also small: cyber-foraging applications are extremely novel, hence scarce and not easily portable across multiple platforms. For this reason, our results are valuable in providing solid evidence on an emerging technique.

10.2. Internal validity

Threats to internal validity affect the interpretation of our findings with regards to the causality link between treatment and outcome.

- *Treatment implementation.* The effects we measured on the outcome might be affected by the specific implementation of the tactics. In order to mitigate this threat, the cyber-foraging tactics were implemented following the guidance of an expert in cyber-foraging whom worked on several practical applications of the tactics. In particular, we made sure that the difference between the Static and Dynamic tactic implementation was modular enough to isolate its effects with respect to the rest of the application.
- *Maturation.* This threat is related to the effects of time on our instrumentation during the measurements. Specifically, the battery of the phone depletes and the temperature of the components varies due to physical phenomena. To mitigate this threat, we performed a randomized application of the treatment, i.e., the application was executed in the Static Provisioning or Dynamic Provisioning version in a random order. This averaged out the effects of battery depletion and different temperatures across our repeated measurements.

10.3. Construct validity

Threats to construct validity affect the relationship between theory and observation. The only threat to construct validity we identify is related to the *definition of the constructs.* As argued previously, the theory behind cyber-foraging is extremely novel and the tactics we evaluated have been proposed in a limited amount of cases. For this reason, we cannot claim our implementation of the tactics can be taken as a reference for cyber-foraging theory. We mitigated this threat by involving the expert on cyber-foraging that defined and proposed the tactics we evaluated in this study. This ensures us that our implementation is a correct reification of the architectural tactics proposed in the theory.

10.4. External validity

Threats to external validity affect the generalization of our findings. We identified the following external validity threats:

- *Subject selection.* As discussed in Section 5, our study is a quasi-experiment with no randomized subject selection. This poses a clear problem of generalization. In fact, we cannot claim our results would generalize as such to a larger population of cyber-foraging applications.
- *Experimental setting.* Our instrumentation and experimental setting is based on a single mobile device and specific hardware technologies. Hence, our results might be affected by the specific experimental setting in which we operated. More evidence is needed to ensure our findings would also apply to other technologies and device families.

11. Related work

There are many studies on introducing architectural tactics for cyber-foraging applications e.g. (Balan et al., 2007; Chun and Maniatis, 2009; Cuervo et al., 2010; Ra et al., 2011; Yang et al., 2008) and presenting reference architectures and frameworks that can be adopted by software engineers to realize cyber-foraging functionalities in different systems e.g. (Kristensen and Bouvin, 2008; Balan et al., 2003; Flinn et al., 2002; Zhang et al., 2011; Simanta et al., 2012). However, in this paper, we focus on studies that provide insights on existing architectural tactics and evaluate their effectiveness on system qualities. In this respect, some work has been done on the evaluation of tactics in different domains and from a different perspective. For instance, Wu and Kelly present a *qualitative* comparison of architectural tactics for system safety, which as a result can extend software design methodologies (Wu and Kelly, 2004). In another example, Harrison and Avgeriou model how different tactics can fit in different software architectural patterns from a compatibility perspective (Harrison and Avgeriou, 2010). Differently, our work assesses the impact of architectural tactics with respect to energy efficiency and resilience.

In particular, we are interested in architectural tactics in the domain of cyber-foraging with an emphasis on the impact on system qualities. Related work from this perspective mostly focuses on a *qualitative* evaluation of the tactics, which usually results in design guidelines for cyber-foraging applications. Agrawal and Prabhakar present Appification, which is a methodological framework to provide guidelines for architectural design, implementation and deployment of self-adaptive mobile apps (Agrawal and Prabhakar, 2015). According to the Appification framework, one should analyze the quality requirements of the application and choose the best fitting tactics. The framework, however, does not provide a *quantitative* evaluation of the tactics in such applications. Orsini et al. provide design guidelines for mobile-cloud computing applications and include a *qualitative* analysis. They classify computation offloading solutions from the literature based on their impact on a number of system quality requirements (Orsini et al., 2015). Liu et al. review application partitioning algorithms for mobile-cloud computing (Liu et al., 2015). They *qualitatively* discuss the implications of each algorithm in different usage scenarios. La and Kim present a taxonomy of computation offloading schemes, in which the schemes are evaluated in a *qualitative* manner based on five identified criteria for mobile-cloud applications (La and Kim, 2014). Their insights help in selecting the optimum offloading scheme for target apps. Shiraz et al. focus on application offloading frameworks in mobile-cloud computing (Shiraz et al., 2013). They introduce a thematic taxonomy to compare the existing frameworks. Abolfazli et al. survey cloud-based mobile augmentation approaches (Abolfazli et al., 2014). They introduce a comprehensive taxonomy and a number of decision-making flowcharts that can be used to build new approaches. Sharifi et al. review existing cyber-foraging solutions and present a categorization based on a number of factors such as the type of surrogate, the overhead of offloading, the granularity of offloading, and adopted metrics (Sharifi et al., 2012).

Differently, in our study, we conduct empirical experimentation to *quantitatively* evaluate cyber-foraging tactics in terms of their impact on system qualities. Our experimentation is an extension of our previous work on a decision model that helps software architects and software engineers select tactics to meet functional and non-functional requirements of cyber-foraging systems (Lewis et al., 2016). In our decision model, we review cyber-foraging tactics from several points of view such as quality impact, selection trade-offs, and dependencies between tactics. The work presented in our study complements the decision model by performing *quantitative* evaluation of tactics for energy efficiency and resilience. We place our focus on surrogate provisioning tactics, which are one of the required tactics to build a cyber-foraging system (Section 2).

12. Conclusion

This study presents an evaluation of the cyber-foraging tactics for static and Dynamic Surrogate Provisioning. Such tactics aim to provide *adaptability* at runtime. However, the actual impact of adopting the tactics on energy efficiency and resilience of the system is not evident in the literature. We performed an empirical experiment, following the experimentation framework devised by Basili et al. (1986), in order to analyze the cyber-foraging architectures systematically. We used the Green Lab of Vrije Universiteit Amsterdam to set up and carry out our experimentation.

Our results show a significantly higher resilience for Static Surrogate Provisioning than Dynamic Surrogate Provisioning. Also both architectural tactics improve energy efficiency compared to non-cyber-foraging architectures (our baseline measurements). However, none of the two tactics outperforms the other with respect to energy efficiency, which means that the overhead of the runtime optimization remains similar.

This paper is a first step toward providing guidance for software architects and software engineers to minimize their learning curve on the selection of the best fitting cyber-foraging architectural tactics. Our empirical experimentation helps making better-informed trade-offs between the desired quality attributes, i.e., flexibility, resilience, and energy efficiency. In our future work, we will further quantitatively evaluate such trade-offs. Also, we will consider other cyber-foraging architectural tactics, emphasizing on runtime programmable infrastructures.

Acknowledgement

This work has received partial funding from the Netherlands Enterprise Agency (rvo.nl), project GreenServe on energy efficient software and green cloud.

References

Abolfazli, S., Sanaei, Z., Ahmed, E., Gani, A., Buyya, R., 2014. Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges, IEEE. Commun. Surv. Tut. 16 (1), 337–368.
Agrawal, A., Prabhakar, T., 2015. Towards a framework for building adaptive app-based web applications using dynamic appification. In: European Conference on Software Architecture. Springer, pp. 37–44.
Almeida, R., Vieira, M., 2011. Benchmarking the resilience of self-adaptive software systems: perspectives and challenges. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS '11, ACM, New York, NY, USA, pp. 190–195.
Statistica., 2013. Number of mobile app downloads worldwide from 2009 to 2017 (in millions) http://www.statista.com/statistics/266488/forecast-of-mobile-app-downloads/ [Online; accessed 30-May-2016].
Balan, R.K., Gergle, D., Satyanarayanan, M., Herbsleb, J., 2007. Simplifying cyber foraging for mobile devices. In: Proceedings of the 5th International Conference on Mobile Systems. Applications and services, ACM, pp. 272–285.

Balan, R.K., Satyanarayanan, M., Park, S.Y., Okoshi, T., 2003. Tactics-based remote execution for mobile computing. In: Proceedings of the 1st International Conference on Mobile Systems, Applications and Services. ACM, pp. 273–286.
Basili, V.R., 1992. Software modeling and measurement: the goal/question/metric paradigm. Tech. rep.. University of Maryland at College Park, MD, USA.
Basili, V.R., Selby, R.W., Hutchens, D.H., 1986. Experimentation in software engineering. IEEE Trans. Software Eng. SE 12 (7), 733–743.
Bass, L., Clements, P., Kazman, R., 2012. Software Architecture in Practice, 3rd Edition Addison-Wesley.
Bosomworth, D., 2015. Mobile marketing statistics. Smart Insights Site.
Chang, Y.-S., Hung, S.-H., 2011. Developing collaborative applications with mobile cloud-a case study of speech recognition. J. Internet Serv. Inf. Secur. 1 (1), 18–36.
Chun, B.-G., Maniatis, P., 2009. Augmented smartphone applications through clone cloud execution. HotOS 9, 8–11.
Cook, T.D., Campbell, D.T., 1979. Quasi-Experimentation: Design & Analysis Issues for Field Settings. Houghton Mifflin Company, Boston.
Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., Bahl, P., 2010. Maui: making smartphones last longer with code offload. In: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services. ACM, pp. 49–62.
Flinn, J., 2012. Cyber Foraging: Bridging Mobile and Cloud Computing. In: Satyanarayanan, M. (Ed.), Synthesis Lectures on Mobile and Pervasive Computing. Morgan & Claypool Publishers.
Flinn, J., Park, S., Satyanarayanan, M., 2002. Balancing performance, energy, and quality in pervasive computing. In: Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on. IEEE, pp. 217–226.
GeSI: Global e-Sustainability Initiative, 2015. SMARTer2030 – ICT Solutions for 21st Century Challenges, 134.
Harrison, N.B., Avgeriou, P., 2010. How do architecture patterns and tactics interact? a model and annotation. J. Syst. Software 83 (10), 1735–1758.
Kristensen, M.D., Bouvin, N.O., 2008. Developing cyber foraging applications for portable devices. In: Portable Information Devices,2008 and the 2008 7th IEEE Conference on Polymers and Adhesives in Microelectronics and Photonics. PORTABLE-POLYTRONIC 2008. 2nd IEEE International Interdisciplinary Conference on. IEEE, pp. 1–6.
Kruchten, P., Lago, P., van Vliet, H., 2006. Building up and Reasoning about Architectural Knowledge. In: Hofmeister, C., Crnkovic, I., Reussner, R. (Eds.), Quality of Software Architectures. In: Lecture Notes in Computer Science, 4214. Springer Berlin, Heidelberg, pp. 43–58.
La, H.J., Kim, S.D., 2014. A taxonomy of offloading in mobile cloud computing. In: Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on. IEEE, pp. 147–153.
Lago, P., Avgeriou, P., 2006. First workshop on sharing and reusing architectural knowledge. SIGSOFT Software Eng. Notes 31 (5), 32–36.
Lewis, G., Lago, P., 2015a. Architectural tactics for cyber-foraging: results of a systematic literature review. J. Syst. Software 107, 158–186.
Lewis, G., Lago, P., Procaccianti, G., 2014. Architecture Strategies for Cyber-foraging: Preliminary Results from a Systematic Literature Review. In: Avgeriou, P., Zdun, U. (Eds.), Proceedings of the 8th European Conference on Software Architecture (ECSA 2014). Lecture Notes in Computer Science, 8627. Springer International Publishing, pp. 154–169.
Lewis, G. A., 2016. Software architecture strategies for cyber-foraging systems. PhD dissertation, Vrije Universiteit Amsterdam.
Lewis, G.A., Lago, P., 2015b. A catalog of architectural tactics for cyber-foraging. In: Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures. QoSA '15, ACM, New York, NY,USA, pp. 53–62.
Lewis, G.A., Lago, P., 2015c. Characterization of Cyber-foraging Usage Contexts. In: Weyns, D., Mirandola, R., Crnkovic, I. (Eds.), Software Architecture. LNCS, 9278. Springer International Publishing, pp. 195–211.
Lewis, G.A., Lago, P., Avgeriou, P., 2016. A decision model for cyber-foraging systems. In: Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA 2016). IEEE, pp. 51–60.
Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., Qureshi, A., 2015. Application partitioning algorithms in mobile cloud computing: taxonomy, review and future directions. J. Netw. Comput. Appl. 48, 99–117.
Orsini, G., Bade, D., Lamersdorf, W., 2015. Context-aware computation offloading for mobile cloud computing: requirements analysis, survey and design guideline. Proc. Comput. Sci. 56, 10–17.
Ra, M.-R., Sheth, A., Mummert, L., Pillai, P., Wetherall, D., Govindan, R., 2011. Odessa: enabling interactive perception applications on mobile devices. In: Proceedings of the 9th International Conference on Mobile Systems, Applications and Services. ACM, pp. 43–56.
Satyanarayanan, M., 2001. Pervasive computing: vision and challenges. IEEE Pers. Commun. 8 (4), 10–17.
Sharifi, M., Kafaie, S., Kashefi, O., 2012. A survey and taxonomy of cyber foraging of mobile devices. IEEE Commun. Surv. Tut. 14 (4), 1232–1243.
Shiraz, M., Gani, A., Khokhar, R.H., Buyya, R., 2013. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. IEEE Commun. Surv. Tut. 15 (3), 1294–1313.
Simanta, S., Ha, K., Lewis, G., Morris, E., Satyanarayanan, M., 2012. A reference architecture for mobile code offload in hostile environments. In: International Conference on Mobile Computing, Applications, and Services. Springer, pp. 274–293.
Wu, W., Kelly, T., 2004. Safety tactics for software architecture design, computer software and applications conference, 2004. COMPSAC 2004. In: Proceedings of the 28th Annual International. IEEE, pp. 368–375.
Yang, K., Ou, S., Chen, H.-H., 2008. On effective offloading services for resource–constrained mobile devices running heavier mobile internet applications. IEEE Commun. Mag. 46 (1).
Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R.P., Mao, Z.M., Yang, L., 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. CODES/ISSS '10, ACM, New York, NY, USA, pp. 105–114.
Zhang, X., Kunjithapatham, A., Jeong, S., Gibbs, S., 2011. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. Mobile Netw. Appl. 16 (3), 270–284.