JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS

BEFTIGRE: Behaviour-driven Full-tier Green Evaluation of Mobile Cloud Applications

Samuel J. Chinenyeze*, Xiaodong Liu and Ahmed Al-Dubai
School of Computing, Edinburgh Napier University, Edinburgh, United Kingdom

ABSTRACT

With the resource constrained nature of mobile devices and the resource-abundant offerings of the cloud, several promising optimisation techniques have been proposed by the green computing research community. Prominent techniques and unique methods have been developed to offload resource intensive tasks from mobile devices to the cloud. Although these schemes address similar questions within the same domain of mobile cloud application (MCA) optimisation, evaluation is tailored to the scheme and also solely mobile focused, thus making it difficult to clearly compare with other existing counterparts. In this work we first analyse the existing/commonly adopted evaluation technique, then with the aim to fill the above gap we propose the BEFTIGRE approach which adopts the behaviour-driven concept for evaluating MCA performance and energy usage – i.e. green metrics. In order to automate the evaluation process, we also present and evaluate the effectiveness of a resultant API and tool driven by the Beftigre approach. The API is based on Android and has been validated with EC2 instance. Experiments show that Beftigre is capable of providing a more distinctive, comparable and reliable green test results for MCAs.

KEYWORDS: Green Mobile Cloud, Green Metrics Evaluation, Mobile Cloud Evaluation, Mobile Offloading Comparison, Behaviour-driven Evaluation, BDD for MCA.

1. INTRODUCTION

The research on green computing investigates ways to optimise software to improve its resource or particularly energy usage, thus the concept of software energy efficiency. The novelty concept behind mobile cloud computing (in terms of green software) is to offer the use of resource abundant cloud as a surrogate to alleviate resource-intensive tasks from mobile devices, thus achieving greener (more energy efficient) software processes on mobile. As the most mobile battery consuming applications are those with computation-intensive features [1], [2], the energy or resource gain (or improvement) for mobile devices is often realised in the computation offloading techniques of mobile cloud applications (MCA). These techniques extend a local mobile architecture to a distributed (mobile-cloud) architecture as shown in Figure 1.

The MCA architecture generally comprises three sets of components [3]–[6]: mobile offloadable components (MCs) are first identified[1]. These components are subsequently replicated on the server (as server components, SCs) to improve mobile resource usage. The third is the offloading scheme components – which handle decision making based on monitoring data of current environmental state to predict when offload process is beneficial. The scheme can be launched in both tiers (Figure 1). To evaluate the efficiency of such systems (MCA with offloading schemes) two abstract phases are taken into account, they are mobile device test and server test; however, most research focus is on the mobile end [3]–[6]. For example, various works [3]–[5] achieve rigorous test by randomization of environment conditions such as network (bandwidth and latency) and server/cloud resource (CPU and memory), using throttling and load generation respectively, in order to evaluate mobile power usage and performance. Consequently, the results of the evaluation do not highlight the scheme's implications on cloud tier. Although the implementation of the MCA offloading schemes takes into full account the architectural change, the evaluation of the schemes does not clearly account for the impact of the scheme on the full system tiers.

Furthermore, customary with the justification of the relevance of an approach is a comparison against existing counterparts. Comparison (and evaluation) of current MCA architectures is achieved using architecture scenario based approach – that is, comparing the offloading scheme with an architecture scenario (such as local, server or optimal as shown in Figure 1). Although conclusions can be reached with this approach, the evaluation is constrained to the scope of the literature as different sources define varying architecture scenarios; thus, presenting difficulty in evaluating offloading schemes for adoption in software development process. These and more challenges are later explained. Furthermore, Johann et al [7] demonstrated that the granularity of test either reveals or hides the actual behaviour (energy-efficiency) of software; and therefore can impact any conclusions made. Similarly, due to varying architecture scenarios adopted by different existing offloading techniques it is challenging to compare between techniques, as any conclusions made would be difficult to verify.

To address the identified gap of coarseness and inconsistencies in the evaluation of MCAs we proposed a behaviour-driven full-tier approach for green evaluation of MCA (Beftigre). Beftigre is focused on green metrics, such as software energy usage, resource usage and performance evaluation [7], [8].

Behaviour-driven: Behaviour-driven development (BDD) is a design approach to aid collaboration between non-technical contributors (such as business analysts, or users) and software engineers. Consequently, BDD gears towards more verifiable and collaborative test process by being able to compare expected behaviours with actual results, following standard simplified scenarios – constructed by simple language clauses: GIVEN, WHEN and THEN [9]. Beftigre adopts BDD concept and simple clause approach to simplify the comparison and evaluation of offloading schemes; and thus, simplifying software design decisions. *Full-tier*: the approach adopts the concept of fine-grained software testing to present the implications of an offloading scheme on the mobile tier as well as on the cloud tier. By evaluating the system as a whole, the approach can detect whether an offloading scheme is aware of both mobile and cloud resource consumption. The full-tier objective of the approach is also assisted by the BDD concept. To facilitate the approach, an API and analysis tool have been implemented for Android platform. The API and tool have been evaluated using existing offloading schemes on a computation-intensive application.

The rest of this paper is organised as follows: Section 2 motivates the research and outlines the challenges to the existing technique adopted in MCA evaluation. Section 3 presents the proposed solution

[1] The analysis for offload-able classes is necessary due to the inadequacy of cloud to perform some mobile specific functions such as those tied to sensors, cameras, GPS etc.; and also to identify the application features that are resource-intensive.

(called Beftigre). Section 3 presents the design of an API and tool which implements the Beftigre approach for Android mobile and cloud evaluation of MCA. The design is presented for individual components of Beftigre. This is followed by experimental evaluations in Section 5. The paper concludes with a review of related work in section 6 and opportunities for future research; in section 7.

2. MOTIVATION AND CHALLENGES

In this section, we use examples from the research to highlight the challenges/difficulties of architecture scenario approach in the evaluation and comparison of offloading schemes. Thus, deriving the goals and novelty of the proposed solution.

2.1 Motivating Example

Let us consider a situation in the development of mobile cloud application. The choice of the offloading scheme would be a critical decision, as it is the core functionality which transforms mobile application to MCA [3]–[6]. Assuming a development team chooses to use an existing scheme, they will need to evaluate and compare between existing offloading schemes. Two offloading schemes have been selected, one based on single thresholding [4] – ST for brevity, and another based on multi-layer perceptron (MLP) [3] – known as POMAC. Further details on how the schemes work, including their decision making/learning process, is presented later in the evaluation section. Also selected is an optical character recognition (OCR) Android application called Mezzofanti, used in the literature [3], [4], to validate the schemes. From the literature [3], [4], the computation-intensive offloadable component is the OCR functionality. The Data presented in Table I is obtained from the literature using WebPlotDigitizer[2].

To achieve the evaluation of individual schemes and comparison between the schemes (ST vs. POMAC), mobile-centric architecture scenarios provided by the literature are used. Mobile-centric means that the approach provides green metrics results for the mobile tier only (i.e. performance and energy usage). Although mobile-centric architecture scenario approach is prevalent in the research [3]–[6], it possesses challenges which make it difficult to come to a satisfactory conclusion for evaluation and comparison of schemes.

2.2 Research Challenges

The research challenges identified for mobile-centric architecture scenario are presented below;

i. Inconsistency in evaluation results of scenarios for an offloading scheme

To evaluate POMAC, Hassan et al. [3] defines four[3] scenarios (illustrated by Figure 1). The efficiency of POMAC is evaluated by comparing the POMAC scheme against other defined architecture scenarios using percentage difference. Deducing from Table I, we can conclude POMAC energy usage as approximately 5% inefficient compared to both local and optimal scenarios and 118% efficient compared to server scenario. Although the local and optimal percentage differences seem to arrive at the same conclusions, there is no clear relationship between these scenarios. This is shown by ST which has approximately 71%, 43% and 8% energy improvement based on comparison to local, server and optimal scenarios respectively. This challenge makes it difficult to arrive at easily verifiable conclusions for schemes.

ii. Variability of architecture scenarios (making it difficult to compare between offloading schemes)

Different sources use varying scenarios to evaluate proposed schemes, e.g. four[3] scenarios are used to evaluate POMAC [3], while five[4] scenarios are used to evaluate ST [4]. Consequently, scenarios will have to be matched (as shown in Table II) in order to establish a basis for comparison. This process introduces complexity in comparing schemes especially since scenarios which may be congruent (by inference) may have slightly different definitions from each other (based on the actual literature implementation). For example, although five scenarios are defined in [4], only four from [4] can be used to match with [3] as shown in Table II. Furthermore, as well as being a challenge to the comparison, the viability of scenarios introduces difficulty in communicating evaluation results between the development teams.

[2] WebPlotDigitizer: data extraction tool; http://arohatgi.info/WebPlotDigitizer [01-Jul-2016].

[3] Four scenarios are defined by [3] for evaluating POMAC, they are OnDevice, OnServer, Optimal and POMAC.

[4] Five scenarios are defined by [4] for evaluating ST, they are Smartphone only, Offloading w/All objects, Offloading w/Necessary objects, Offloading w/Necessary objects (delta), and Offloading w/Threshold check.

The summary column of Table II is used to match the scenarios from the literature ([4] and [3]). *Local* is the execution of the application without any offloading. *Server* is a scenario where all offloadable objects are always executed on the server. *Optimal* is a scenario where only offloadable objects assessed as having better remote execution time compared to that of local are offloaded. These tasks are mainly computation or data-intensive [3], [4]. Also, the optimal scenario is only optimal from the perspective of mobile tier efficiency (i.e. mobile performance or energy savings). *The Scheme* is based on extending the previous optimal scenario with the decision-making[5] mechanisms for offload. It refers to the proposed offloading schemes in the literature. While the optimal scenario is focused on mobile-tier efficiency, the Scheme (which extends the optimal scenario with a decision maker) ought to achieve some level of overall efficiency (by a decision maker which is also cloud-aware). An objective of this study is to present an approach which can assess the overall efficiency of schemes without requiring multiple scenarios, for evaluation efficiency and accuracy.

iii. Coarse-granularity of evaluation

Different sources use different levels of experimental rigour. For example; [3] performed a more rigorous experiment for POMAC evaluation (as the scheme is based on MLP), compared to [4]'s experiment for ST which is not as rigorous. Comparing ST energy with POMAC (using optimal scenario as reference) gives approximately 8% gain/savings in ST and 5% loss in POMAC. The case may be that in adverse environmental conditions ST scheme fails to save mobile energy (which is true from later experiment – section 5). Also since the analysis is mobile-centric, it fails to provide the overall implications of a scheme's decision to or not to offload. This challenge poses difficulty when deciding schemes with an overall efficiency (i.e. mobile as well as cloud resource awareness). We later show that with full-tier evaluation one can better understand if a scheme just keeps offloading to the server, or if it checks server availability (i.e. robustness).

3. BEFTIGRE APPROACH

The challenges and difficulties of the mobile-centric architecture scenario approach fall into the category of (green[6]) software testing/evaluation [7], [10]. In response to the identified challenges, we propose an evaluation approach called Beftigre, which adopts the use of the behaviour-driven technique to address challenge 1 and 2. Furthermore, we treat challenge 3 as a software testing granularity problem, consequently resolved by fine-grained testing of the mobile tier and cloud tier (i.e. full-tier). In particular, this paper makes the following novel contributions:

3.1 Behaviour-driven Comparison – as single scenario

Beftigre adopts BDD concept. BDD simplifies software testing by using clauses which can easily be communicated across development team to construct a scenario for testing [11]. Similarly, Beftigre's *Comparator* (Figure 2) uses these clauses in form of annotations to specify the expected (THEN) MCA system behaviour, given a set of conditions (GIVEN and WHEN). The annotations can be applied to a test function (TFn; which is a method to test the mobile component – as shown in Figure 2), and the information supplied through the annotations is then used (by Full-tier Analyser) to evaluate or compare against the actual test results. Thus, the behaviour-driven comparison process acts as a single scenario replacing varying architecture scenarios, and alleviating inconsistency of architecture scenarios; therefore a solution to challenge 1 and 2. Furthermore, the annotations are used to implement full-tier comparison (to address challenge 3) by using @Given and @When annotations to specify mobile and server *expected* preconditions respectively, for comparison on Then clauses.

Markers are objects for counter-based instrumentation which has attributes that can be written to file for processing. The purpose of Markers is to integrate instrumentation counters with annotation information (of comparator component) for efficient single scenario comparison. The start and finish counters in Markers component are vital to calculating *actual* values of THEN clause (mobile energy and time, server CPU and memory used) which are used to compare against *expected* values of @Then annotation. Thus,

[5] Decision making is the check on the environment conditions of the communications which influence the offloading. Decision making mechanisms can be based on single (static) thresholds [4] or predictive learning [3].

[6] Green is the term used for software optimisation and testing based on the energy- and/or resource- efficiency and performance metrics [8], [35]. Since offloading schemes target energy-efficiency of mobile devices, similarly scenario based comparison is within the green software testing category.

Marker interface contributes to the behaviour-driven solution to challenge 2 and contributes a foundation for full-tier analysis (i.e. based on THEN clauses).

3.2 Full-tier Analysis – results from mobile and cloud perspective

Beftigre evaluates a MCA from mobile (power usage and performance) and cloud (resource usage) perspective to produce full-tier actual test results. The analysis is accomplished by *Full-tier Analyser* (details presented later in Algorithm 1 and 2). The aim is that by full-tier analysis (as opposed to mobile-centric testing) an offloading technique can be finely evaluated in terms of its impact on the system as a whole. [7], shows that a fine-grained approach to energy measurement (using counters) aid reveal specific energy usage points. Similarly, to identify specific energy points, Beftigre adopts fine-grained measuring distributed across the mobile tier (using *Markers*) and cloud tier (using *Metrics Collector*). Consequently, by revealing the mobile to cloud resource consumption implications of a MCA application, the full-tier analysis objective addresses challenge 3.

3.3 Unified Monitoring – which facilitates full-tier results

The required data for full-tier analysis is collected by *Power Monitor* and *Markers* for mobile energy and performance respectively; by *Metrics Collector* for cloud resource usage; and by *Resource Simulator* for determining cloud resource availability and network. By unified monitoring of mobile and cloud tier, the aforementioned components contribute to the full-tier evaluation. Specifically, these components (Figure 2) contributes to the research objectives as follows;

Power Monitor is achieved by PowerTutor model via an API for seamless integration with the BDD objective of Beftigre. The power monitor generates PowerLog, which (in conjunction with counters from Marker component) is used to compute *actual* mobile energy usage which is used to compare against the expected used energy attribute of @Then annotation. This is a step which contributes to the mobile tier of the full-tier evaluation, using behaviour-driven concept.

Server Monitor provides the logic for monitoring and serving the cloud tier metrics useful for the behaviour driven full-tier evaluation. It uses *PerfMon Server Agent* to compute metrics for *actual* cloud-tier values of Then clause (i.e. percentage CPU and memory usage), and socket server programs to compute metrics for *actual* values of Where clause (i.e. available percentage CPU and memory).

Resource Simulator makes it possible to evaluate the MCA scheme based on different set environmental conditions. Furthermore, it provides the capability for introducing controlled rigour (i.e. experimental replication) to the evaluation process (as shown later in section 5). In the implementation section, the simulator is provisioned with the server monitor.

Metrics Collector provides the logic (client programs) for saving the cloud tier metrics received from the server monitor component. The metrics collector persists the *actual* values for When clause and cloud-tier values of Then clause. By generating evaluation data for cloud-tier, the metrics collector contributes to the full-tier solution to challenge 3.

The aforementioned components focus on monitoring: power and performance for the mobile tier, and resource usage for cloud tier, because they are the popularly investigated green metrics for MCA domain [1]–[4], [12]. Mobile energy is increasingly gaining research interest due to the resource-constrained nature of mobile devices and the increasing demand for rich mobile applications. Resource usage (specifically CPU and memory) is a commonly investigated metrics when monitoring workload impact on the cloud [12], [13]. In the current Beftigre approach, the scope of the energy measurement is at the application level, i.e. the overall energy consumption of the mobile execution. The overall mobile energy is measured using power tutor model and applies to all network types (including WLAN and 3G).

4. DESIGN DETAILS

To automate the proposed approach we have implemented an *API* (consisting of power monitor, marker and comparator interface) and a *desktop tool* (consisting of server monitor, metrics collector, and full-tier analyser interface).

Beftigre API has been built for the Android platform and can be applied to the test module (Figure 3a) or application module (Figure 3b) of an android project for measuring power and performance of the application. *Beftigre desktop tool* (called *the orchestrator* for brevity) is used to launch server monitoring and testing, and also provides data analysis feature for the logs collected from mobile and cloud domain.

Server monitoring and testing in the orchestrator has been tested for Linux based servers (specifically on Amazon EC2 VMs).

4.1 The Power Monitor Interface

Power monitoring in Beftigre is achieved using PowerTutor [14] model. Although PowerTutor monitor is widely adopted in the research for mobile power monitoring, it is worth noting that PowerTutor seems to produce accurate energy measurement for specific brands and models of mobile devices and rough estimates for others [14]. However, as exact measurements may not be necessary for relative comparisons, we adopt the widely used PowerTutor model, leveraging the core logic of the monitor for Beftigre API. Furthermore, two methods are exposed to start and stop the monitor. In Beftigre API, the PowerTutor[7] based monitor is not a UI-based application but a service which runs in the background to monitor the application power usage. startPowerMonitoring() launches PowerTutor, while stopPowerMonitoring() stops and saves power data to PowerLog.

The purpose of adapting the monitor as an API is for seamless integration with mobile test package; consequently, allowing for ease of control of the monitoring process from the code. Furthermore, as a background process, the monitor does not interfere with the application being tested. As shown in lines 14 and 28 of Figure 3a, the monitor is started and stopped right before and after the call to register and save marker objects within setup and teardown methods of android test framework, respectively. This is to ensure that the monitor captures all test execution process.

4.2 The Marker Interface

Markers are objects for counter-based instrumentation (line 6) which has attributes that can be written to file for processing. A marker object takes a *label* attribute as a parameter which is used to associate counters with test functions. After markers have been created, registerMarkers() is used to validate and assign unique identifier to all marker objects to be used in the test, while saveMarkers() is used to save and write the attributes of registered Marker objects to MarkerLog. A marker object (Mn, n is an integer typed unique identifier) consists of the following attributes: Mn_start, Mn_finish, Mn_label, and Mn_anno. Mn_start and Mn_finish are references to the timestamps in ms used to identify the execution block for the test section. Mn_label is the reference to the label assigned during marker creation. And Mn_anno is a reference to the annotations assigned to the test function (i.e. lines 17 to 19).

Beftigre carries out four types of exception handling associated with Marker objects: DuplicateLabelException is thrown if two or more markers were registered with the same label. Labels are used to create a unique identifier for markers, no two markers can have the same label. DuplicateStartMarkerException and DuplicateFinishMarkerException are thrown from the Marker class. The former is thrown when the API identifies that start() was called more than once on a registered marker while the latter is thrown when finish() is called more than once on a registered marker. UnevenMarkerException is caused if a marker set was incomplete. For example when a marker is started by calling start() method, and not finished. All registered markers must have a complete set. A marker is said to have a complete set if it calls a start() and a finish() method.

With the Beftigre API, one test method can be created within a test project to test the overall application – this includes only a marker object and annotations (see Figure 3a). Multiple markers can be applied within the application project; however, annotations are not applicable in this scope (see Figure 3b) because the annotations of the main test method (in Figure 3a) covers any markers used within application project (in Figure 3b). Furthermore, service-based API features (such as start and stop power monitoring, and base status service) are only declared once from the test projects, as the call from test project captures the application module execution.

4.3 The Comparator Interface

The comparator is implemented as a method named getBaseStatus() which calculates *actual* mobile CPU and memory availability, consequently; contributing to the single scenario and behaviour-driven objective, by providing *actual* values for Given clause. It is made as the last API call (line 29), to ensure the process does not add any overhead to the power and performance readings; furthermore, it is designed as an android

[7] PowerTutor; https://github.com/msg555/PowerTutor [01-Jul-2016].

alarm monitor service, to ensure the process completes even after the test is terminated (i.e. tearDown() method on line 30).

The annotations: @Given, @When, and @Then are the basis on which offloading schemes are evaluated or compared. @Given annotation specifies the *expected* percentage CPU and memory availability of the mobile device, both of which are integer typed. Recall, the *actual* value for Given clause is obtained from getBaseStatus(). @When annotation specifies the *expected* bandwidth (bps), latency (ms), and server percentage CPU and memory availability, all of which are integer typed. The *actual* value for When clause is obtained from the server monitors. @Then annotation specifies the *expected* mobile elapsed time (ms) and used energy (mJ), and cloud percentage used CPU and memory, during the period of the test. All parameters are integer typed except used energy which is double typed.

@Given is a precondition for the mobile end, while @When is the precondition for the server end. @Then annotation is the postcondition with elements specifying values to be asserted or compared against. Both the pre and post conditions are based on the full-tier concept (i.e. mobile and cloud involved).

4.4 The Server Monitor Interface (with Simulator)

The server monitor interface is used to orchestrate three monitoring processes and two resource simulation processes on the server. For server monitoring: *PerfMon Server Agent* monitors the percentage CPU and memory usage. *CPUMemoryAvail* computes percentage CPU and memory availability using SIGAR API[8]. And *BandwidthLatency* is used for obtaining bandwidth and latency. The last two monitors are java socket server programs. For resource simulation: *Traffic Control Utility* is used to simulate different network conditions (using the parameters: bandwidth and latency)[9], *Stress Utility* is used to simulate CPU and memory load (using the parameters: CPU load and memory load). TC and Stress are both Linux utilities. To start the monitoring the aforementioned simulation parameters are passed to the Server Monitor Interface (as shown in Figure 4).

4.5 The Metrics Collector Interface

Following the start of server monitors and simulators, is the launch of metrics collectors. The Metrics Collector Interface is composed of three metrics collector processes within the orchestrator. *CPUMemoryAvailClient* records the percentage CPU and memory availability obtained from *CPUMemoryAvail* monitor. *BandwidthLatencyClient* records bandwidth and latency, which is measured by sending packets to and from the *BandwidthLatency* monitor. *CPUMemoryAvailClient* and *BandwidthLatencyClient* collectors execute only once (i.e. when metrics collectors are launched) and then terminates. *PerfMon Metrics Collector* runs continually for a scheduled time, recording server CPU and memory usage at intervals using Apache JMeter binaries. Furthermore, *PerfMon Metrics Collector* is launched as a listener, based on HTTP request sampling, within JMeter test plan. The test plan implements a loop controller to ensure that metrics recording process is continuous until a scheduled duration is elapsed. The duration is any speculated time, in seconds, which covers the evaluation process.

Network and resource availability metrics from Socket clients are first logged into MetricsLog, followed by the Resource usage metrics from *PerfMon Metrics Collector*. The purpose of using *PerfMon Metrics Collector* for continuous recording of resource usage at intervals is so that the average percentage resource usage (of the MCA scheme being evaluated) can be computed after the test is done. See Algorithm 1 for further details.

4.6 The Full-tier Analyser Interface

Completing the test on the mobile and server tier generates three logs: MarkerLog and PowerLog from the mobile; and MetricsLog from the server. The full-tier analyser interface (analyser for short) analyses data stored in the logs both from mobile and cloud tier to produce full-tier results. The analyser also references the behaviour-driven annotation from the test for comparison. The analyser contributes the final solution to the challenges in section 2 by consolidating the contributions of all the Beftigre components.

Algorithm 1 and 2 presents the *Evaluate* and *Assert* functions of full-tier analyser respectively, used to compute the results of MCA test. *Evaluate* presents the algorithm useful for evaluating an offloading scheme (self-evaluation) which gives a full-tier result (mobile performance and energy, with cloud CPU

[8] SIGAR API; https://support.hyperic.com/display/SIGAR [01-Jul-2016].

[9] Slow: simulates low bandwidth, high-latency; https://gist.github.com/obscurerichard/3740206 [01-Jul-2016].

and memory used) that is, the *actual* values of Then clause. *Assert* presents the algorithm useful for comparing between offloading schemes. *Assert* extends the *Evaluate* function in order to determine the relationship between the schemes using the behaviour driven annotations, by asserting which scheme is more efficient based on the given and where conditions or whether the expected and actual values are from the same/similar system/offloading scheme. The functions which make up the analyser are presented below in the order of execution;

- *ExtractExpected* extracts the expected value of all attributes of annotations from MarkerLog.
- *ExtractActual* extracts the actual (or measured) values of Given and When clause, i.e. mobile CPU and memory availability from PowerLog; and cloud CPU and memory availability, bandwidth and latency from MetricsLog (used in line 2 of Algorithm 2).
- *Map* obtains the start timestamp (TS) and finish timestamp (TF) from MarkerLog, and matches them to that of the PowerLog and MetricsLog in order to obtain the exact mobile power (PS to PF), cloud CPU (CS to CF) and cloud memory (MS to MF) used by the MCA during the evaluation.
- *Evaluate* (Algorithm 1) firstly computes the elapsed time (ms), used energy (mJ), average CPU usage (%) and average memory usage (%) using the data from *Map* function.
- *Assert* (Algorithm 2) compares Then clause *actual* values from *Evaluate* function with the @Then annotation *expected* values from *ExtractExpected* function to assert a result following some condition. *Assert* is achieved in two stages below;

  Stage 1 Assertion:
  Result – The assertion result (lines 4-8, Algorithm 2) gives the percentage increase or decrease[10] between expected scheme and actual scheme at a full-tier scale (i.e. mobile elapsed time, mobile used energy, cloud CPU usage, and cloud memory usage).
  Condition – The assertion is performed if the Given and When values (from *Evaluate* and annotation) are within ± 5% range[11] (meaning schemes are comparable).

  Final Assertion:
  Result – asserts that the expected and actual schemes are the same or similar system (lines 9-11, Algorithm 2).
  Condition – the assertion is true if any three values of Then clauses of *Stage 1 Assertion* are within ± 1%. In other words if after comparing *actual* and *expected* of Then clause, and any three of its attributes (say; mobile time, energy and cloud CPU) has percentage increase or decrease within the range of ± 1%, then the *actual* and *expected* schemes are similar or the same.

After the above analysis process is completed, a CSV file containing the computed values/summary of analysis is generated for reporting purpose. The following section presents an evaluation of the Beftigre approach using real-life application.

5. EXPERIMENTS AND EVALUATION

5.1 Experimental Setup

i. Experimental Settings

We have evaluated the performance of Beftigre and its effectiveness in MCA evaluation and comparison by applying it to three real-world open-source Android mobile applications used in existing research [3], [5]. The purpose of using case study applications from the literature is because of their identified taxonomy (for offloadable tasks). Table III presents details of the mobile applications used. Table III shows that computation- and data- intensive taxonomy applications were used for a more rigorous evaluation. The mobile device configuration used for the experiment is a Samsung Galaxy S3 running Android 4.3 on Quad-core 1.4 GHz, with 1.5GB memory. While the cloud configuration is an Amazon EC2 m3 instance running Linux Ubuntu 14.04 64 bit, with Intel Ivy Bridge 2.5GHz CPU and 3.75GB memory. We implemented offloader and launcher (Figure 1) using Java socket API which sends requests and receives responses for offloaded components.

[10] Percentage increase (in actual value compared to expected value) means increased or more resource demand, while percentage decrease means decreased or less resource demand.

[11] Existing experimental research e.g. [18], uses 5% range as acceptable range of comparison between expected and actual power readings (for power meters vs power models, respectively). We therefore apply ±5% range to our green metrics.

Furthermore, the *research questions* investigated by the experiment are:

- How does Beftigre compare to non-BDD in terms of effectiveness in MCA evaluation? This question is addressed in point 1 to 3 of section 5.2.
- To what extent can replication be achieved in Beftigre? This point is addressed by point 4 of section 5.2.

The *dependent variables* in the experiment are, i) the evaluated mobile cloud metrics, which is the result of evaluating a scheme, and they are: mobile performance, mobile energy, cloud CPU and memory usage; and ii) assertions, which is the result of comparing between two schemes.

The *independent variable* in the experiment is the type of evaluation approach used for the MCA:

- *Beftigre*: the behaviour-driven full-tier approach. This is the proposed approach in this paper – which is focused on constructing scenarios from behaviour-driven parameters of clauses (such as Given, When and Then). Thus, this does not require multiple architecture scenarios and also integrates well with the development process.
- *Non-BDD*: the mobile-centric architecture scenario approach presented in motivation section. This is the current type of evaluation used in current research – which is focused on capturing accuracy in test result by using as many architecture scenarios as possible (such as Local, Server and Scheme scenarios).

In the evaluation of Schemes, *statistical analysis methods* have been applied to ascertain the effectiveness of Beftigre in achieving rigour. The methods: mean and standard deviation were used for the evaluation of Beftigre approach as shown in Table VI. The non-BDD approach maintains the use of averaging on varying architecture scenarios (a method used in existing literature [3], [4]), and thus evaluated in this research, as presented later in Table IV. The software packages and technical documentation for the experiment have been made publicly available at http://beftigre-mca.appspot.com.

ii. Simulation Parameters

We use stress and TC utility (also utilised by Beftigre's Resource Simulator) to provide parameters which simulate different environmental conditions to test the schemes. The above parameters are used to maintain the same level of rigour for both Beftigre and Non-BDD approaches. The simulation parameters[12] (also used in Tables IV and V) are as follows:

- *WLAN*: consists of 30mbps bandwidth, 20ms latency, 2 CPU worker loads and 2 memory worker loads. With these parameters, local execution time is greater than remote execution time, which is appropriate for offload.
- *Outlier*: consists of 20mbps bandwidth, 200ms latency, 2 CPU worker loads and 2 memory worker loads. These parameters are used to verify the offloading schemes.
- *3G*: consists of 500kbps bandwidth, 200ms latency, 5 CPU worker loads and 5 memory worker loads. With these parameters, local execution time is less than remote execution time.

iii. Sample Evaluated Offloading Schemes

For each approach (Beftigre and Non-BDD) we compare two offloading schemes:

- *Scheme1* is based on threshold-based policy, similar to [4].

In threshold-based offloading scheme [4], a method is offloaded only when its parameter data size is greater than a predefined threshold value. The scheme in [4] is implemented based on runtime checkpointing which incurs transmission overhead due to varying offload data size, hence the use of data size for thresholding. In the experiment we omit runtime checkpointing, thus making the offload data size fixed for all the applications used in the experiment. Consequently, we use a predefined threshold based on the network rather than data size. Therefore, the *criteria for offloading* in Scheme1 is when bandwidth is greater than 500bps, and latency is greater than 150ms. The bandwidth and latency values are obtained by sending packets to and from the server. The premise behind the effectiveness of static thresholds – such as Scheme1 – is that the local execution time of the application for the threshold used is always greater than the remote execution time; therefore, using the threshold would amount for time or energy savings [4]. Within the *Simulation Parameters* section, the WLAN offload favourable condition simulates environment favourable for the static threshold of Scheme1.

- *Scheme2* is based on perceptron algorithm, similar to [3].

[12] The settings use in WLAN, Outlier and 3G are found to be commonly used in experiments [3], [15].

Scheme2 uses multiple criteria for offloading – based on learned data (as opposed to predefined static threshold). *Offload criteria* are bandwidth, latency, server and mobile CPU and memory availabilities. The adapted perceptron algorithm used in the experiment is open sourced[13]. To extract learning data for Scheme2, we implement a special version of the application having the offloadable component execute remotely (LearnRemote) and locally (LearnLocal). LearnRemote and LearnLocal are instrumented with random simulation parameters (including WLAN, Outlier, and 3G) to generate the following metric; mobile CPU and memory available, server CPU and memory available, bandwidth, latency, and elapsed time. The generated data is then used to build the training dataset classified for *remote* or *local* execution. A data subset is classified as a *remote* data if the remote execution time is greater than local execution time, otherwise, it is classified as *local* data.

5.2 Results and Discussion

The results of the experiment have been presented in Tables IV and V, and Figures 5-8. Following the full-tier and behaviour-driven objective of Beftigre, the approach has been evaluated against the earlier described mobile-centric architecture scenario approach (Non-BDD).

1) Inconsistency Challenge to Non-BDD

The Non-BDD approach (Table IV) presents the elapsed time and used energy of the mobile device during the period of the experiment. Since the same level of rigour was applied on all experiment, sample population of 9 (3 samples from each simulation parameters – 3G, WLAN, and Outlier) was used. Table IV further specifies three architecture scenarios: Local, Server and the scheme being evaluated. The results of the Local and Server scenarios are the same for both schemes and are consequently used as a basis for evaluating the schemes (using percentage difference), and subsequently used for comparison in the Non-BDD approach.

The samples for non-BDD evaluation (Table IV) was broken down, using Linpack as an example, into respective resource metrics as shown in Figures 5-7. These results show inconsistencies and complex correlation between scenarios. For example, the *Local* scenario (which is affected by less environmental inconsistencies – only mobile CPU and memory availability – Figure 7), begins with its first sample as 90% and 14%, whereas Scheme1 and Scheme2 are 98% and 18%, 85% and 17% for CPU and memory availability respectively. This inconsistency follows through the samples, and also occurring in the *Server* scenario. Even more, the Server scenario inconsistencies are more profound as it also involves network and server resources – Figures 5 and 6.

Conclusively, Non-BDD is not very effective for the comparison of schemes as each sample that make up the mean of the experiments are affected by different conditions which are unrelated to the scheme. With Non-BDD, we can, however, make a generalised conclusions, such as scheme1 is more energy efficient compared to scheme2 on the basis of local and server scenarios – this would be based on the assumption that a more rigorous experiment would span different environmental conditions, as 35.8mJ to 2.1J ratio is a significant saving on Scheme1. The significant saving is due to the simulation parameters used which seemed to favour Scheme1, as the robustness of Scheme2 in adverse conditions seemed to be compromised by its energy intensive decision making – especially if the adverse conditions are not extreme – further explained in next point. Having extracted a conclusive result using Non-BDD approach, it is difficult to tell the behaviour of the scheme, for instance, given a specific (kind of) environmental condition.

Also notice that for MatCalc application which is data-intensive, Non-BDD (Table IV) shows Scheme1 to be 31.61% more energy-efficient compared to Server whereas Scheme2 is only 19.79% more efficient. Beftigre approach (Table V) shows the reverse to be the case. This is understandable because Scheme2 is based on trained data and is aware that the application is data-intensive, and therefore stops subsequent offload. The misconception of results associated with the Non-BDD approach is due to the randomisation of samples – in which case there is more presence of favourable conditions (such as *WLAN*) than unfavourable (such as *3G*). Also with NQueen application, the same inconsistency issue reoccurs in Non-BDD (Table IV) where Scheme1 and Scheme2 are presented with very close results for elapsed time, i.e. 8.63% and 8.74% respectively, which is however clarified by Beftigre approach (Table V). Also, there is no consistency across applications with the Non-BDD approach (Table IV), for example; Scheme 1 is more

[13] Perceptron Algorithm in Java; https://github.com/nsadawi/perceptron [01-Jul-2016].

energy-efficient in Linpack application (using Local scenario) compared to Scheme 2, but in NQueen the reverse is the case. Beftigre, however, maintains consistency across applications.

2) Beftigre Full-tier Effectiveness

Table V presents the results of sample evaluation and comparison using the Beftigre approach. First, the schemes are self-evaluated[14] using the outlier simulation parameter (20mbps bandwidth, 200ms latency, 2 CPU loads and 2 memory loads), these are provided through the Orchestrator's Server Monitor Interface. For Linpack: the evaluation of *Scheme1* gives 22208ms and 3008mJ, and Scheme2; 21887ms and 3181mJ for mobile-tier green metrics (elapsed time and used energy). Furthermore, the cloud CPU and memory usage during the process are obtained – as 58% and 28% for Scheme1, 60% and 32% for *Scheme2*. From Table V evaluation, we can predict; from the CPU and memory usage of both schemes; that Scheme1 was executed locally and did not offload whereas Scheme2 did.

Recall that in Scheme1, the criteria for offload is set to be as bandwidth >= 500bps and the latency <= 150ms, as a result, using the outlier, Scheme1 was not offloaded – thus consuming more time and energy. This means that Scheme1 is not robust enough to be aware of conditions beyond the parameters of its criteria constraints. Conversely, Scheme2 which learns from trained data is not only aware of adverse, environmental conditions but also incurs training overhead – which in this case is more energy consequential than time. The overhead is caused by the decision-making process; communication to and from the server to calculate cloud CPU and memory availability, as well as mobile resource availability and network states, prior to deciding to offload. Inspecting from the full-tier analysis, we can deduce that, based the outlier parameters Scheme2 is both inefficient in mobile and cloud tier. However, since it is aware of the environment, its benefits would be more appreciated in adverse conditions (which may include mobile devices with very low computing capacities). Furthermore, as Scheme2 was found to be inefficient for mobile and cloud tier due to an excessive network transmission, improving Scheme2 would demand an insight into overall transmission energy during offloading and decision making. In the current experiment, transmission energy was not investigated, but future work on Beftigre would look into the investigation/adoption of successful methodologies [16], [17] for assessing such metrics.

3) Robustness of Beftigre Assertion

As well as providing the full-tier capacity to deduce the behaviour of a scheme, by adopting BDD concepts Beftigre also makes it easy to communicate the expected goal of schemes within software teams – from business analyst to software engineers. For example, Table V shows the comparison of the schemes based on the earlier evaluated simulation parameters. By re-executing Scheme1 we see changes in the actual experimental values, however, these changes are within ± 1% of the percentage increase or decrease between the expected of Scheme1 and actual of Scheme1 (to satisfy line 9 of Algorithm 2). The reverse is the case when Scheme1 is compared against Scheme2, the percentage increase/decrease are beyond ± 1%.

Consequently, by developing a test plan (using simulation parameters) earlier, and a sample application, schemes can be better evaluated, compared and communicated between development team – thus adopting the wider software engineering objective of BDD. Furthermore, by adopting Beftigre, one can avoid inconsistencies and the difficulty of the variability of architecture scenarios through the use of guided annotations, while providing a finer granularity of results through full-tier analysis/evaluation.

4) Reproducibility Effectiveness of Beftigre

[14] For self-evaluation of an application (which does not require annotations), @Given values can be set to 0. However to compare against results of a previous test or between offloading schemes, all values for all annotations are required.

Table VI uses the statistical method (of standard deviation) to verify the effectiveness of Beftigre for repeatability or reproducibility of its test results. The verification is achieved using: mean (on five samples for each Scheme run with *Outlier* parameters), standard deviation and 5% range criterion. Recall that in Beftigre approach, 5% range criterion is the basis on which two schemes are compared. In other words: two schemes are comparable if the preconditions (@Given and @When) of the expected scheme are within ± 5% range of the actual preconditions – as shown by *inRange(a, b)* method of Algorithm 2. The 5% range criterion is popularly explored in research [18] and in a similar context as Beftigre (i.e. for comparison). The purpose for which the range criteria was applied in Beftigre approach is in consideration of the unpredictable and varying nature of the MCA environment. The difficulty in predicting MCA environments is a challenge which in practice contributes to the inconsistencies of existing MCA evaluation approach (as discussed in Section 2). To achieve a solution with a better consistency of results, Beftigre applies environmental control and scoping. Control is orchestrated by the Resource Simulator while the range criteria (of Full tier analyser) provides scope for comparison. Together the aforementioned features achieve reproducibility as shown in Table VI.

For example, from Table VI varying samples have been executed using the *Outlier* parameter settings, and the deviation gives the varying range of the samples – demonstrating natural inconsistencies in MCA. However, despite the inconsistencies, there are mostly overlaps in the values of the environmental metrics (i.e. CPU, memory, etc.) for the schemes (Scheme1 and Scheme2). Furthermore, the deviations are also within the ± 5% range criteria (not more or less), thus validating the scoping effectiveness of the Full-tier analyser (achieved by *inRange*) for comparison of schemes. Therefore the extent to which replication can be achieved in our evaluation approach is by using range criteria and applying an element of control to the environment.

5.3 API Performance

The API performance evaluation was performed on Windows 10 x64 PC, with Intel i7 2.20GHz CPU and 8GB memory. Furthermore, the mean value of 30 test samples (on local execution) was used to investigate the performance (overhead) of Beftigre API on mobile testing by comparing the build, setup and execution time of *Default* test setup against *Beftigre* setup (Figure 8). The *Default* setup, is the conventional android test setup with Robotium API for UI test, while *Beftigre* setup adds the Beftigre API to the default setup.

Build time; measured using Android Studio's build functionality; is the time to (re)build the modules and libraries of the project. Setup and execution time are measured using Java timestamp utility. Setup time is the time it takes to initialise all test library objects used (i.e. Lines 11-16 of Figure 3a). Execution time is the time from setup to test completion.

From Figure 8, less than 0.3% increase is observed in the execution time, this is due to the persistence of Marker objects as logs. Also, as the power monitor runs as a different android service (and as a different process) to the test process, its execution has no effect on the test; this is similar to executing PowerTutor application external to the application under test. Consequently, the results of the evaluation show that Beftigre API has no significant overhead to the android test process.

5.4 Dependencies

As our approach requires monitoring resources, writing logs, and analysing results, some configurations are required on the Android manifest (these are mostly dependencies inherited by the use of PowerTutor monitor). Permissions include; internet, access fine location, access coarse location, write external storage, access Wi-Fi state, read phone state, access network state and receive boot completed. Furthermore, two services are required which is the UMLLoggerService for power monitoring and BaseService for the getting the base CPU and memory usage of the device – utilised in the @Given clause.

Furthermore, key dependency for the effectiveness of the Beftigre evaluation is the BaseService, which is required after the android test. We have observed that although the test runs as a service, in order to log (synchronise) to the same file as the MarkerLog, the device is required to still be connected to the PC. This is a precautionary measure (to avoid interrupting the BaseService) as interrupted singleton services in android will be terminated automatically. Android services can alternatively be programmed to restart when interrupted – however, this is not efficient for the evaluation process, as the evaluation time will be increased. We use android shared preferences API to manage synchronisation of logs.

6. RELATED WORK

Mobile cloud application testing often involves evaluation of mobile power usage and performance, and cloud resource usage. This section splits related work into tool/API components used in MCA evaluation;

- Mobile power monitors – which focus on evaluating a mobile application's power usage,
- Cloud resource monitoring – which focuses on evaluating the resource usage of a set load in a given time and
- Test frameworks – which focus on evaluating an application's requirements; to assert that a given requirement works as expected.

Furthermore, the section highlights how Beftigre API and Tool unifies the functions provided by the three aforementioned components.

6.1 Mobile Power Profilers

Software power profiling are achieved in two ways [7], [19]: either by hardware approach (i.e. involving power meters) [7] or by modelling [14], [20]–[22]. The use of energy models is imperative to power measurements, particularly for mobile applications, especially because the hardware approach can be a complex and expensive process [23]. PowerTutor [14]; is a power model (and android application) for measuring resource and power usage of android mobile applications. Hao et al. also presents a power model called eLens [21] (an extension of their previous work, eCalc in [20]) which aims at fine-grained mobile energy profiling at the bytecode level. Some other models [24]–[26] have been proposed, however, PowerTutor is the most adopted in the research [3], [4]. The monitor, however, is provisioned as a mobile application, with Beftigre we transform the monitor into an API suitable for use within a mobile test project. Furthermore, monitoring in current models are only single-tiered – i.e. either mobile or cloud (as presented in 6.2), however, Beftigre performs a unified monitoring of mobile and cloud tier for full-tier results.

Intel Energy Checker (IEC) is a software development kit (SDK) designed to measure the performance and energy consumption of a system [27]. The IEC power model requires the use of external power monitors [7]. This form of setup, however, is neither applicable to mobile nor cloud systems, as cloud environment is based on a setup too complex for power meters [23]. Trepn profiler [28], is useful for mobile instrumentation. A good work has been done by the Qualcomm team on the profiler; which includes the display of markers from the code for fine-grained analysis. Trepn however, is tied to a processor type (Snapdragon) for the accuracy of readings. Consequently, MCA evaluation is out of scope for Trepn. In response to these gaps, Beftigre API extends the conventional code markers by a behaviour-driven test approach which uses annotations to capture mobile-cloud evaluation concerns. Furthermore, the Beftigre orchestrator is used to incorporate libraries which also monitors the cloud and integrate logs from both domains for a unified analysis.

6.2 Cloud Resource Monitoring

System Information Gatherer And Reporter (SIGAR) is an API which provides an interface for gathering system information associated with a variety of system resources such as the processor, network, disk and memory. There also exist vendor specific monitors such as Amazon CloudWatch [13] which provide APIs tied to the vendor's cloud platform. PerfMon Server Agent[15] is an open source server monitoring library built on SIGAR API. It is popularly used for gathering resource usage information of cloud VMs [29], [30]. However, Server Agent is not self-sufficient for MCA, therefore, we integrate it into our proposed Beftigre Orchestrator alongside server network throttling and resource load generation. The integration into the orchestrator makes it possible to capture the resource usage for specific environment conditions.

Furthermore, pertaining to cloud monitoring, current research investigate from a resource metrics perspective or power metrics perspective. Fang et al. [12] proposed an approach known as TARGO which is built on Amazon CloudWatch [13]. The objective of TARGO is VM optimisation based on monitor data, and the core metrics investigated is CPU utilisation. In Beftigre, we evaluate both CPU and memory, and by using open-sourced PerfMon Server Agent, the approach is vendor independent. Current work on power metrics assessment for cloud often adopt power models, as hardware meters are complex to implement for cloud infrastructures [23], [31]. Network transmission energy is also popularly investigated [16], [17] in the context of assessing cloud energy. In Beftigre, we have scoped our investigation for mobile power and cloud resource as they are mostly investigated in MCA domain, however, future work would include cloud and network energy assessment.

[15] PerfMon Server Agent; https://jmeter-plugins.org/wiki/PerfMonAgent [01-Jul-2016].

6.3 Test Frameworks

Several test frameworks have been proposed for mobile application test. These can be generally classified into test case design libraries (e.g. JUnit [32]), test automation libraries (e.g. Robotium[16] and Espresso [33]) and behaviour-driven libraries (e.g. Jbehave[17]), with JUnit as the de facto standard for Android. These frameworks can also be used together for mobile testing as shown in [34]. JUnit is a well-known library for implementing test cases, Robotium or Expresso does not entirely replace JUnit, but rather has its strength in UI test automation. Jbehave is also a unique supplement, as per behaviour-driven development (BDD). It specifies system behaviour from which test cases and (later) reports, are derived. To the best of our knowledge, none of these test frameworks has been explored for power measuring, or mobile-cloud based assertions/testing. As a response to the gap, we adopt the BDD concept, similar to Jbehave, to specify behaviour (based on any evaluation test knowledge), then we derive a test case from the behaviour and adequately provide a report based on the behaviour. Furthermore, in Beftigre, we integrate the BDD concept into cloud-based mobile systems for full-tier results.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have demonstrated an approach for MCA evaluation called Beftigre, which is based on behaviour-driven evaluation (facilitated by the API) and full-tier analysis (facilitated by the tool). We have shown that; by presenting a basis for comparison in a programmable syntax (i.e. using behaviour-driven annotations), different offload schemes can be compared between each other. Furthermore, an API and a tool were developed to realise and scale up the concept. An evaluation of the proposed API shows that there is no significant performance overhead to the test process. Furthermore, the API also integrates with existing test automation frameworks; for example, Robotium has been employed for testing (line 22 of Figure 3a) with Beftigre API.

The Beftigre tool also automates the analysis process using the generated logs (from the API and the tool's server metrics collector). An evaluation of the full-tier analysis proves accuracy and effectiveness in the specification of the behaviour of a MCA. By adopting BDD concepts, Beftigre makes it easy to communicate expected goal of offload schemes within software teams; from business analyst to software engineers using annotation clauses, and to seamlessly test for actual results (comparing evaluated outcomes with annotation values). Furthermore, we have demonstrated that Beftigre is capable of achieving reproducibility in test evaluation process through the use of a range criterion (5%) and tracked control of the environment (Resource Simulator).

Future work will involve investigation of specific energy consumption of access networks (i.e. transmission energy), and studying its impact as a behaviour attribute (annotation) on full-tier results. Also, future work can investigate possible options to specify custom annotation attributes – especially for the WHERE clause within the API. This could assist to define a test scope for a given scheme, at the same time not breaking the comparability objective of Beftigre, since the scope will be specified as annotations. Also, the API and tool have been demonstrated for Android and EC2 mobile-cloud platforms, future work could look into extending the approach for multi-platform support – Azure cloud, IOS, Blackberry. We are currently working on an easy install virtual environment for a simulated mobile cloud experimentation for Beftigre. This will be provided on the publicly available documentation site at http://beftigre-mca.appspot.com.

REFERENCES

[1] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: Application Energy Metering Framework for Android Smartphones Using Kernel Activity Monitoring," in *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, 2012, p. 36.

[2] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh, "Who killed my battery?: analyzing mobile browser energy consumption," in *Proceedings of the 21st international conference on World Wide Web*, 2012, pp. 41–50.

[3] M. A. Hassan, K. Bhattarai, Q. Wei, and S. Chen, "POMAC : Properly Offloading Mobile Applications to Clouds," in *Proceedings of the 6th USENIX conference on Hot Topics in Cloud*

[16] Robotium; https://github.com/robotiumtech/robotium [01-Jul-2016].

[17] JBehave; http://jbehave.org [01-Jul-2016].

Computing, 2014, pp. 1–6.

[4] Y.-W. Kwon and E. Tilevich, "Energy-Efficient and Fault-Tolerant Distributed Mobile Execution," in *2012 IEEE 32nd International Conference on Distributed Computing Systems*, 2012, pp. 586–595.

[5] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring android Java code for on-demand computation offloading," in *Proceedings of the ACM international conference on Object oriented programming systems languages and applications - OOPSLA '12*, 2012, vol. 47, no. 10, p. 233.

[6] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services - MobiSys '10*, 2010, vol. 17, pp. 49–62.

[7] T. Johann, M. Dick, S. Naumann, and E. Kern, "How to measure energy-efficiency of software: Metrics and measurement results," in *2012 First International Workshop on Green and Sustainable Software (GREENS)*, 2012, pp. 51–54.

[8] P. Bozzelli, Q. Gu, and P. Lago, "A systematic literature review on green software metrics," VU University Amsterdam, The Netherlands, 2013.

[9] C. Solís and X. Wang, "A study of the characteristics of behaviour driven development," in *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011*, 2011, pp. 383–387.

[10] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston: Addison-Wesley Professional, 2003.

[11] M. Hüttermann, "Specification by Example," in *DevOps for Developers*, Berkeley, CA: Apress, 2012, pp. 157–170.

[12] D. Fang, X. Liu, L. Liu, and H. Yang, "TARGO: Transition and reallocation based green optimization for cloud VMs," *Proc. - 2013 IEEE Int. Conf. Green Comput. Commun. IEEE Internet Things IEEE Cyber, Phys. Soc. Comput. GreenCom-iThings-CPSCom 2013*, pp. 215–223, 2013.

[13] A. Shackelford, "Monitoring the Application," in *Beginning Amazon Web Services with Node.js*, Berkeley, CA: Apress, 2015, pp. 171–208.

[14] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "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*, 2010, pp. 105–114.

[15] M. A. Hassan, Q. Qi Wei, and S. Songqing Chen, "Elicit : Efficiently Identify Computation-intensive Tasks in Mobile Applications for Offloading," in *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2015, pp. 12–22.

[16] V. C. Coroama and L. M. Hilty, "Assessing Internet energy intensity: A review of methods and results," *Environ. Impact Assess. Rev.*, vol. 45, pp. 63–68, 2014.

[17] K. Hinton, J. Baliga, M. Feng, R. Ayre, and R. Tucker, "Power consumption and energy efficiency in the internet," *IEEE Netw.*, vol. 25, no. 2, pp. 6–12, 2011.

[18] C. Seo, S. Malek, and N. Medvidovic, "Estimating the energy consumption in pervasive java-based systems," in *6th Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2008*, 2008, pp. 243–247.

[19] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *ACM SIGOPS Oper. Syst. Rev.*, vol. 47, no. 3, pp. 42–49, Nov. 2013.

[20] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating Android applications' CPU energy usage via bytecode profiling," in *Proceedings of the 1st International Workshop on Green and Sustainable Software, GREENS 2012*, 2012, pp. 1–7.

[21] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *Proceedings of the 35th International Conference on Software Engineering, ICSE 2013*, 2013, pp. 92–101.

[22] D. Li, S. Hao, J. Gui, and W. G. J. Halfond, "An Empirical Study of the Energy Consumption of Android Applications," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 121–130.

[23] A. Kansal, F. Zhao, and A. A. Bhattacharya, "Virtual Machine Power Metering and Provisioning," in *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010, pp. 39–50.

[24] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-Grained Power Modeling for Smartphones Using System Call Tracing," in *Proceedings of the sixth conference on Computer systems - EuroSys '11*, 2011, p. 153.

[25] A. Pathak, Y. C. Hu, and M. Zhang, "Fine Grained Energy Accounting on Smartphones with Eprof," in *Proceedings of the 7th ACM european conference on Computer Systems - EuroSys '12*,

2012, pp. 29–42.
[26] R. Mittal, A. Kansal, and R. Chandra, “Empowering developers to estimate app energy consumption,” in *Proceedings of the 18th annual international conference on Mobile computing and networking - Mobicom ’12*, 2012, p. 317.
[27] J. Tayeb, K. Bross, C. S. Bae, C. Li, and S. Rogers, “Intel Energy Checker Software Development Kit User Guide,” 2010. [Online]. Available: https://goo.gl/Yrtbn9. [Accessed: 01-Jul-2016].
[28] L. Ben-Zur, “Developer Tool Spotlight-Using Trepn Profiler for Power-Efficient Apps,” 2011. [Online]. Available: https://goo.gl/6jJNQA. [Accessed: 01-Jul-2016].
[29] C. E. Gómez, C. O. Díaz, C. A. Forero, E. Rosales, and H. Castro, “Determining the Real Capacity of a Desktop Cloud,” in *High Performance Computing*, vol. 565, Springer International Publishing, 2015, pp. 62–72.
[30] S. Shekhar, H. Abdel-Aziz, M. Walker, F. Caglar, A. Gokhale, and X. Koutsoukos, “A simulation as a service cloud middleware,” *Ann. Telecommun.*, vol. 71, no. 3–4, pp. 93–108, 2016.
[31] Y. Guo and X. Zhou, “Coordinated VM resizing and server tuning: Throughput, power efficiency and scalability,” in *Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2012*, 2012, pp. 289–297.
[32] G. Nolan, “Android Unit testing,” in *Agile Android*, Berkeley, CA: Apress, 2015, pp. 15–24.
[33] G. Nolan, “Espresso,” in *Agile Android*, Berkeley, CA: Apress, 2015, pp. 59–68.
[34] N. Saleem, “Testing Android Apps with Robotium and JBehave,” 2013. [Online]. Available: http://goo.gl/q6WMMU. [Accessed: 01-Jul-2016].
[35] G. Lami, L. Buglione, and F. Fabbrini, “Derivation of Green Metrics for Software,” in *Software Process Improvement and Capability Determination*, T. Woronowicz, T. Rout, R. V. O’Connor, and A. Dorling, Eds. Springer Berlin Heidelberg, 2013, pp. 13–24.

Table I. MCA offloading schemes evaluation and comparison by architecture scenarios (showing the mobile time and energy).

Architecture Scenarios	**ST** [4]		**POMAC** [3]	
	Elapsed Time (ms)	**Used Energy (J)**	**Elapsed Time (ms)**	**Used Energy (mJ)**
Local	49331.55	86.59	3930.33	4854.24
Server	27673.79	63.86	34873.15	19839.42
Optimal	17486.63	44.73	3986.21	4845.10
The Scheme	10347.59	41.33	4242.32	5085.80
Local % diff.	130.65	70.76	-7.64	-4.66
Server % diff.	91.14	42.84	156.62	118.38
Optimal % diff.	51.30	7.90	-6.23	-4.85
Note: *Local % diff.*, *Server % diff.* and *Optimal % diff.* is the percentage difference of the scheme in comparison to Local, Server, and Optimal scenarios respectively. A negative value signifies a loss in energy savings or performance.				

Table II. Variability of Architecture Scenarios.

Summary of Architecture Scenarios	**ST** [4] **Specific Architecture Scenarios**	**POMAC** [3] **Specific Architecture Scenarios**
Local	Smartphone only	OnDevice
Server	Offloading w/All objects	OnServer
Optimal	Offloading w/Necessary objects (delta)	Optimal
The Scheme	Offloading w/Threshold check	POMAC

Table III. Characteristics of mobile applications used (also explored in [3], [5], [15]).

Applications	**Offloadable task**	**Taxonomy**	**Description**
Linpack[18]	*run()* method of *Linpack* class	Computation-intensive	Benchmarking application

[18] Linpack; https://github.com/pedja1/Linpack [01-Jul-2016].

MatCalc[19]	*times(Matrix B)* method of *Matrix* class	Data-intensive	Matrix calculator application
NQueen[20]	*findSolution(int board[], int n, int pos)* method of *NQueen* class	Computation-intensive	NQueen computation application

Table IV. Non-BDD Evaluation and Comparison (based on mean values of samples [3], [4]).

Architecture Scenarios	**Scheme1** (3G, WLAN, Outlier)		**Scheme2** (3G and WLAN, Outlier)	
	Elapsed Time (ms)	**Used Energy (mJ)**	**Elapsed Time (ms)**	**Used Energy (mJ)**
Linpack				
Local	21952.50	3168.23	21952.50	3168.23
Server	21948.67	2539.87	21948.67	2539.87
Schemes:	21953.33	2206.08	21945.33	3102.39
Local % diff.	-0.0038	35.8055	0.0327	2.1
Server % diff.	-0.0212	14.0663	0.0152	-19.9395
MatCalc				
Local	6313.22	492.48	6313.22	492.48
Server	8340.03	711.17	8340.03	711.17
Schemes:	6403.33	517.07	7009.98	583.10
Local % diff.	-1.42	-4.87	-10.46	-16.85
Server % diff.	26.27	31.61	17.33	19.79
NQueen				
Local	17123.02	2702.13	17123.02	2702.13
Server	16980.88	2009.89	16980.88	2009.89
Schemes:	15707.00	2609.05	15689.52	1898.99
Local % diff.	8.63	3.51	8.74	34.91
Server % diff.	7.79	25.94	7.91	5.67
Note: *Local % diff.* and *Server % diff.* is the percentage difference of the scheme in comparison to Local and Server scenarios respectively. A negative value is used to signify a loss in energy savings or performance.				

Table V. Beftigre Evaluation and Comparison/Assertion (between Scheme1 vs. Scheme2) showing Full-tier Results (i.e. Then clauses) – *Outlier* simulation parameters used.

Label	**@Given: mobile**		**@When: cloud/network**				**@Then: mobile**		**@Then: cloud**		**Final Assert**
	CPU	**Mem.**	**CPU**	**Mem.**	**Bandw.**	**Lat.**	**Time**	**Energy**	**CPU**	**Mem.**	
Linpack Evaluation											
Scheme1	93	18	41	67	560	243	22208	3008	58	28	-
Scheme2	81	41	41	62	557	247	21887	3181.08	60	32	-
Linpack Compare: Scheme2 expected on Scheme1 actual											
Expected	81	41	41	62	557	247	21887	3181.08	60	32	-
Actual	92	17	41	68	559	241	22304	3029.44	58	29	-
Assert	-	-	-	-	-	-	1.91% more	4.77% less	3.33% less	9.38% less	Different
Linpack Compare: Scheme1 expected on Scheme1 actual											
Expected	93	18	41	67	560	243	22208	3008	58	28	-
Actual	92	16	41	68	563	240	22233	3031	58	29	-
Assert	-	-	-	-	-	-	0.11% more	0.76% more	0%	3.57% more	Similar system
MatCalc Evaluation											
Scheme1	90	23	38	56	505	211	7033	613.41	55	26	-
Scheme2	92	22	38	56	512	224	6442	537.07	55	26	-
MatCalc Compare: Scheme2 expected on Scheme1 actual											
Expected	92	22	38	56	512	224	6442	537.07	55	26	-
Actual	94	21	38	58	507	209	7075	625.12	55	26	-
Assert	-	-	-	-	-	-	9.37% more	15.15% more	0%	0%	Different
MatCalc Compare: Scheme1 expected on Scheme1 actual											
Expected	90	23	38	56	505	211	7033	613.41	55	24	-
Actual	91	22	38	56	501	217	7085	609.91	55	24	-

[19] MatCalc; https://github.com/kc1212/matcalc [01-Jul-2016].

[20] NQueen; https://github.com/acelan/NQueen [01-Jul-2016].

Assert	-	-	-	-	-	-	0.75 more	0.57% less	0%	0%	Similar system
NQueen Evaluation											
Scheme1	84	25	43	67	552	246	16344	2810.11	59	29	-
Scheme2	89	25	42	65	540	244	15702	1908.09	59	30	-
NQueen Compare: Scheme2 expected on Scheme1 actual											
Expected	89	25	42	65	540	244	15702	1908.09	59	30	-
Actual	85	25	42	68	553	241	16289	2797.42	59	30	-
Assert	-	-	-	-	-	-	3.67% more	37.80% more	0%	0%	Different
NQueen Compare: Scheme1 expected on Scheme1 actual											
Expected	84	25	43	67	552	246	16344	2810.11	59	29	-
Actual	85	25	43	68	553	243	16302	2785.90	59	30	-
Assert	-	-	-	-	-	-	0.26% less	0.87% less	0%	3.39% more	Similar system

Key: *Given*: mobile CPU and memory availability (%), *When*: cloud CPU and memory availability (%), bandwidth (bps) and latency (ms), Then: mobile elapsed time (ms), mobile used energy (mJ), cloud used CPU (%) and cloud used memory (%).

Table VI. Replication capability of Beftigre Evaluation (verified using *Outlier* parameters).

	@Given: mobile		**@When: cloud/network**			
	CPU	**Mem.**	**CPU**	**Mem.**	**Bandw.**	**Lat.**
Linpack						
Scheme1 Mean	92.54	18.50	41.30	67.30	560.12	243.03
Scheme1 Deviation	1.13	0.84	0.65	1.30	11.25	5.43
Scheme1 5% range	4.63	0.93	2.03	3.37	28.01	12.15
Scheme2 Mean	84.25	42.01	41.50	63.40	558.31	247.80
Scheme2 Deviation	1.25	1.03	0.55	1.44	9.96	4.04
Scheme2 5% range	4.21	2.10	2.08	3.17	27.92	12.39
MatCalc						
Scheme1 Mean	92.91	23.20	38.06	56.85	506.45	217.00
Scheme1 Deviation	1.21	0.77	0.71	0.82	5.00	2.01
Scheme1 5% range	4.65	1.16	1.90	2.84	25.32	10.85
Scheme2 Mean	91.44	22.31	38.42	56.10	517.66	220.60
Scheme2 Deviation	2.22	0.90	0.74	1.31	4.02	6.03
Scheme2 5% range	4.57	1.12	1.92	2.81	25.88	11.03
NQueen						
Scheme1 Mean	84.60	24.84	43.44	68.70	550.98	245.31
Scheme1 Deviation	1.01	1.09	0.58	2.77	9.03	3.93
Scheme1 5% range	4.23	1.24	2.17	3.44	27.55	12.27
Scheme2 Mean	86.08	25.06	42.02	65.22	548.40	249.16
Scheme2 Deviation	2.30	1.02	0.99	2.40	7.72	3.20
Scheme2 5% range	4.30	1.25	2.10	3.26	27.42	12.91