Abstract

Distributed Java-based software systems are increasingly deployed onto heterogeneous mobile platforms with limited battery resources. In this domain, it is a crucial issue to preserve the energy resource and prolong the system's lifetime. To address this problem, we first suggest a framework that allows the system engineer to estimate the energy consumption of a distributed Java-based software system both during system construction-time and during runtime. Based on our energy estimation framework, we then present two efficient approaches for reducing the software system's energy consumption and consequently increasing the system's lifetime. Finally, we discuss our strategy for evaluating our estimation framework and energy saving approaches.

1. Research Problem

Modern software systems are predominantly distributed, dynamic, and mobile. They increasingly execute on heterogeneous platforms, many of which are characterized by limited resources, and are also implemented in Java more and more because of its platform independence and intended use in network-based applications. One of the key resources, especially in long-lived systems, is battery power. Unlike the traditional desktop platforms, which have uninterrupted, reliable power sources, the newly emerging computing platforms (e.g., PDAs, cellular telephones, “smart dust” motes) have finite battery lives. These are either “single use” devices that are not rechargeable (e.g., motes), or have to be collocated with a charging unit and rendered stationary for a period of time (e.g., cell phones). In such settings, minimizing the system's power consumption, and thus increasing its lifetime, becomes as important as the more traditional quality-of-service concerns, such as reliability, security, fault-tolerance, availability, communication latency and so forth. While many previous approaches have suggested ways of reducing the energy consumption in mobile devices, they only focus on reducing the energy consumption incurred by individual mobile applications in a single mobile device, and do not consider a distributed software system running on heterogeneous hosts. Hence, these approaches are not adequate for the distributed domain.

To address the energy awareness problem in distributed environments, we first suggest a framework that estimates the energy consumption of a distributed software system implemented in Java both during system construction-time and during runtime. Our primary objective in devising this estimation framework is to enable an engineer to make informed decisions when constructing a distributed software system, such that the system’s overall energy consumption is reduced, and its duration increases. Based on our energy estimation framework, we present two feasible approaches, which can be effected during system construction-time for reducing the distributed system’s overall energy consumption and thus extending its lifetime. Especially, we focus on devising the application-level energy saving approaches rather than the hardware-level energy optimization techniques such as adjusting a CPU clock speed dynamically.

2. Related Work

There have been a number of energy saving approaches that can be realized at the hardware level. Several CPU dynamic voltage scaling techniques [2,6] have been suggested for adjusting the CPU clock speed based on the characteristics of tasks and thus saving the energy resource due to the reduced clock speed. Hard disk spin-down policies [13] have been studied for reducing the energy consumption of hard disk I/O operations by adapting to the user’s access patterns and priorities. Since these previous approaches are supported and managed by a host’s OS, they can be easily integrated with our application-level approach for further saving the energy resource in the distributed software
Several researchers have also presented application-level energy saving techniques on mobile devices. Remote execution techniques [3,8] supported by middleware frameworks have been suggested to reduce the power consumption incurred by mobile applications. Remote execution techniques split the functionality of an application into several components and enable some of the components to be executed in resource-abundant servers connected with wall power. In addition, several studies have proposed the way of adjusting the quality levels of output produced by mobile applications for meeting the user-specified battery duration [4,7]. Most of these previous approaches provide the frameworks for estimating the energy consumption of individual applications. However, their frameworks are restricted to multimedia applications (e.g., video player, map viewer, speech recognition, and so forth). As mentioned earlier, they also focus only on individual mobile applications in a single host rather than a distributed software system running on multiple hosts.

Previous research has suggested energy consumption models for embedded software at the level of CPU instructions [10,11]. Although these approaches produce accurate energy consumption estimates for embedded software, a system engineer would face at least two problems in using them to estimate the energy consumption of a distributed software system: (1) they only consider the energy consumption incurred by the CPU and do not take into account other system elements such as main memory; and (2) since these previous approaches have not modeled the communication energy cost caused by the remote interactions among software components, the engineer cannot use them to accurately estimate the energy consumption of a software system distributed across multiple hosts.

Several studies have profiled the energy consumption of Java Virtual Machine (JVM) implementations. Lafond et al. [5] have measured the energy consumption of Sun Microsystems’ KVM [15], and showed that the energy required for memory accesses usually accounts for 70% of the total energy consumed for KVM. Vijay et al. [12] have also discussed the characteristics of the energy consumption by cache and main memory for executing a JVM. However, none of these studies suggest a model that an engineer can use for estimating the energy consumption of a distributed Java-based software system either prior to or during runtime.

### 3. Research Hypotheses

Our research aims to develop the framework that can be used by the engineer as part of a software system’s design process for reducing the distributed system’s overall energy consumption and thus extending its life-time, and is based on the following research hypotheses:

**Hypothesis 1.** In a distributed software system, each component provides its services for other components. An interface is a common mechanism for exporting a component’s service. Thus, a component has its own implementation behind each interface, and executes the corresponding implementation whenever one of its interfaces is invoked. We hypothesize that the energy behavior of each component can be characterized by, and is directly proportional to the energy costs of its constituent interfaces.

**Hypothesis 2.** In some domains (e.g., distributed real-time embedded systems), the system’s expected runtime environment (e.g., expected inputs to an interface, possible system states, available network bandwidths, etc.) is known or can be predicted with a high accuracy during system construction-time. We hypothesize that an estimation framework can be devised such that energy estimates made at system construction-time fall on the average within 5% of the estimates at runtime for these domains.

**Hypothesis 3.** In the domains like those mentioned in Hypothesis 2, the overall energy consumption of a distributed software system can be reduced by selecting a more energy-efficient deployment architecture at system construction-time. However, finding the most energy-efficient deployment architecture is an exponential problem as in the most general case its complexity is \(k^n\), where \(k\) and \(n\) are the numbers of the target hosts and software components, respectively. We hypothesize that an approximation algorithm with the polynomial-time complexity in terms of the number of hosts and components can be devised such that the estimated energy consumption of the distributed system with the deployment architecture suggested by the approximation algorithm either will exceed the system’s energy consumption for the most energy-efficient deployment architecture by at most 10% on average (if the optimal deployment is known), or will be less than the statistical average energy consumption (i.e., calculated by averaging the system’s energy consumption for a set of randomly selected deployment architectures) by 40% on average, in the case where the optimal deployment architecture is not known.

**Hypothesis 4.** There usually exist multiple candidate architectural styles for the target distributed software system, which can meet the desired requirements of traditional QoSs (e.g., availability, performance, communication latency, etc.). We hypothesize that the framework can be devised with the ability to find the most energy-efficient style among these candidate architectural styles during system construction-time. We also hypothesize that the amount of energy saving estimated at system construction-time by selecting the most
energy-efficient style will be on the average within 5% of the actual energy saving measured at runtime.

4. Proposed Solution

In a distributed software system, each component usually provides its services for other components via its constituent interfaces. For example, Figure 1 shows a component $c_1$ on host $H_1$, $c_1$’s provided interfaces, and the invocation of those interfaces by remote components. In this distributed domain, each host may have a different battery capacity while each component usually requires a different amount of energy for providing its services. To address the energy awareness problem in the distributed software system, we propose the following framework for estimating the software system’s energy consumption either prior to or during runtime.

**Energy Estimation Framework.** The energy consumption of a software component consists of its computational and communication energy costs. The computational cost is mainly due to CPU processing, memory access, I/O operations, and so forth, while the communication cost is mainly due to the data exchanged over the network for the components’ interactions. In order to accurately determine the computational cost of a software component, we focus on the component’s constituent interfaces.

In Java, the effect of invoking an interface can be expressed in terms of the execution of the 256 Java bytecode types, and of the native methods provided by the JVM. Java bytecodes are platform-independent codes interpreted by JVM’s interpreter, while native methods are library functions (e.g., java.io.FileInputStream’s read() method) provided by JVM. Native methods are usually implemented in C and compiled into dynamic link libraries, which are automatically installed when JVM is installed. JVM also provides a mechanism for synchronizing multiple threads via an internal implementation of a monitor. In a multi-threaded environment, monitor enter/exit operations are executed as a result of invoking a synchronized method. Based on this effect, the computational energy consumption of invoking an interface can be calculated by summing up the energy costs of all of the bytecodes, native methods, and monitor operations executed during the invocation. Furthermore, the overall computational energy cost of each component can be calculated by aggregating the computational energy costs of its constituent interfaces.

For the communication energy cost, we focus on modeling the energy consumption due to the remote communication based on UDP. In comparison to TCP, UDP is a much more lightweight protocol (e.g., it provides no congestion control and error recovery mechanisms). Therefore, it is the dominant networking protocol in embedded and resource-constrained domains. Previous research [1,14] has showed that the actual energy consumption of wireless network communication is directly proportional to the amount of transmitted and received data, and inversely proportional to the actual bandwidth between two hosts. Therefore, the communication energy cost due to invoking an interface can be calculated by using the size of messages exchanged and the transmission and receipt bandwidths on a target host during the invocation. Moreover, the overall communication energy cost of a component can be estimated by aggregating the communication energy costs of its interfaces.

However, in addition to the computational and communication energy costs, there are two additional energy costs for executing a Java component incurred by a JVM process: (1) Within a JVM process, a separate thread is usually created and manages Garbage Collection; (2) Since JVM runs as a separate user-level process in an OS, implicit OS routines such as context switches, process scheduling, and paging, are internally executed by an OS for managing the execution of a JVM process. We call the energy consumption due to the GC thread and implicit OS routines *infrastructure energy overhead*. Once we have estimated the energy costs of all the components, as well as the infrastructure energy overhead, we can simply calculate the distributed software system’s overall energy consumption by adding all of these energy costs. In our preliminary work [9], we have explained how our framework can be used by a software engineer for estimating the system’s overall energy consumption both at system construction-time and at runtime, in more detail.

**Deployment Architecture.** An initial deployment architecture may have a significant impact on the overall energy consumption of the distributed system. For example, we can reduce the communication energy consumption by collocating frequently communicating components on the same host or deploying them onto the hosts with the high bandwidths. At system construction time, we can estimate the system’s overall energy consumption for each possible deployment architecture by using our estimation framework. However, determining the most energy-efficient deployment architecture is an exponentially complex problem in terms of
the numbers of the target hosts and software components. Hence, an efficient approximate algorithm leveraging the energy estimates from our framework is needed for finding a more energy-efficient initial deployment architecture. This approach would be particularly useful in domains (e.g., distributed real-time systems) where the system’s expected runtime context is known or can be predicted easily at system construction time.

Architectural Style. The distributed system’s architectural style can also impact the system’s overall energy consumption as an architectural style has its own unique pattern of interactions among components. Several candidate architectural styles may satisfy the desired requirements of traditional QoSs for the target distributed system. Our estimation framework should be extended for estimating the amount of energy saving achieved via choosing each of these candidate architectural styles, and thus finding the most energy-saving style.

5. Evaluation

We have evaluated our energy estimation framework for a large number of distributed application scenarios by using several mobile devices (Compaq iPAQs, laptops) and an HP 3458-a digital multimeter [9]. Hypothesis 1 has been evaluated by selecting various Java-based components, invoking their interfaces on a mobile device, and comparing each component’s energy cost estimated from our framework with its actual energy consumption measured from a multimeter. Hypothesis 2 has been evaluated by setting the almost same system’s runtime context both at system construction-time and at runtime and comparing energy estimates at construction-time with those at runtime.

For evaluating Hypothesis 3, we will run our approximate algorithm for particular application domains (i.e., those mentioned in Hypothesis 2) in both simulated distributed environments and actual mobile devices. To evaluate Hypothesis 4, we will use the same application domains as Hypothesis 3, run our extended estimation framework for finding the most energy-saving architectural style with its estimated amount of energy saving, and finally compare this estimated amount with the actual amount of energy saving measured at runtime.

6. Future Work and Contributions

So far, we have finished implementing our estimation framework that estimates the energy consumption of distributed Java-based software systems both at system construction-time and at runtime. As part of our future work, we plan to devise an approximate algorithm that suggests an energy-efficient deployment architecture by leveraging our estimation framework. We will also extend our current estimation framework for finding the most energy-saving architectural style for the target distributed software system.

As discussed earlier, all of the previous approaches considered only reducing the energy consumption of individual mobile application in a single host, and thus are not appropriate for the distributed environments. However, our work addresses a more complex problem of saving the energy resources in the distributed Java-based software system. As our solution to this problem, we suggest the energy estimation framework with a high accuracy [9], and present two efficient energy-saving approaches leveraging our framework.

7. References