Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms

Xinwen Zhang, Sangoh Jeong, Anugeetha Kunjithapatham, and Simon Gibbs

Computer Science Lab., Samsung Information Systems America, San Jose, CA, USA
{xinwen.z,sangoh.j,anugeetha.k,s.gibbs}@samsung.com

Abstract. We propose a new elastic application model that enables the seamless and transparent use of cloud resources to augment the capability of resource-constrained mobile devices. The salient features of this model include the partition of a single application into multiple components called weblets, and a dynamic adaptation of weblet execution configuration. While a weblet can be platform independent (e.g., Java or .Net bytecode or Python script) or platform dependent (native code), its execution location is transparent – it can be run on a mobile device or migrated to the cloud, i.e., run on one or more nodes offered by an IaaS provider. Thus, an elastic application can augment the capabilities of a mobile device including computation power, storage, and network bandwidth, with the light of dynamic execution configuration according to device's status including CPU load, memory, battery level, network connection quality, and user preferences. This paper presents the motivations, concepts, typical elasticity patterns, and cost consideration of elastic applications. We validate the augmentation capabilities with an implemented reference architecture and example applications.

1 Introduction

Applications on smartphones traditionally are constrained by limited resources such as low CPU frequency, small memory, and a battery-powered computing environment. For example, the iPhone 3G is equipped with 412MHz CPU, 512MB RAM, and a battery allowing about 5 hours of talking time. The new Samsung Galaxy Android phone has 528MHz CPU, 128MB RAM, and battery offering about 6.5 hours of talk time. Both devices have up to 7.2 Mbps 3G data network connection. Compared to today's PC and server platforms, these devices still cannot run compute-intensive applications such as complex media processing, search, and large-scale data management and mining.

Cloud computing delivers new computing models for both service providers and individual consumers including infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS), which enable novel IT business models such as resource-on-demand, pay-as-you-go, and utility-computing [7]. From the perspective of service providers, cloud computing is often viewed as a vast and scalable platform for service delivery. We suggest a new perspective, one tuned to the needs of mobile devices. We consider cloud computing as a means to extend or augment the capabilities of resource constrained devices.

There are several approaches to realize this perspective. One approach is to duplicate the runtime environment of the device in the cloud and then run the application either on the device or in the cloud. The off-device runtime environment is sometimes called a “surrogate” [14], a “clone” [10], or a cloudlet [17]. Virtual machine technology is often used to host and isolate the off-device runtime so making this approach fit well with emerging IaaS platforms such as Amazon EC2 [1]. Running a device clone in the cloud has some attractive properties such as enhanced CPU and memory resources which lead to better performance. Furthermore, applications do not need any modification – the clone and the physical device can run identical binaries. However, this approach has disadvantages too. First, the application on the clone may need to access the physical hardware on the device. For example, consider a GPS application or simply the question of how an application running in the clone interacts with the user. It is certainly possible to transfer device I/O between the device and clone environment over the network, but this may impact responsiveness and battery use. Secondly, simply replacing one processor with another fails to take full advantage of cloud compute resources. Ideally, a cloud application should be able to run in a highly parallel fashion distributed over many cloud nodes. Thirdly, completely duplicating a device and running it on the cloud increases the complexity of device management. For example, the cloud system needs similar security protection and data privacy control as those on the device since it runs all possible applications with data resources from the original device.

The above considerations lead us to focus on application level augmentation instead of cloning a complete device environment. Often these applications are data-parallel with high compute-to-communication ratio. Examples include media processing, search, and data mining. Our goal is to design an architecture and related middleware to enable *elastic applications* which consist of multiple components called *weblets*, each of which can be launched on a mobile device or in the cloud. The decision of where to launch a weblet is based on application configuration and/or the status of the device such as its CPU load and battery level. Ideally the application model could also support migration of weblets between the device and cloud platform during runtime. While offloading and delegating computing have been proposed by many researchers [11, 9, 14, 13], the novelty of our approach lies in enabling flexible and optimized elasticity by considering multiple factors including device status, cloud status, application performance measures, and user preferences (e.g., different running modes of an application including power-saving mode, high speed mode, low cost mode, offline mode, or in terms of expected application throughput).

To enable this new application model, many challenges exist in different areas, including management of heterogeneous computing environments, data management and communication dependencies between weblets, state synchronization between weblets, and cost-effective dynamic execution configuration. The middleware should provide infrastructure for seamless and transparent execution of elastic applications and offer convenient development support. This paper first gives the concepts and typical elasticity patterns (Section 2). We then focus on the optimization of cost-effective execution configuration by considering multiple factors (Section 3), which we believe is one of the most critical and unique components of the application model. We then present a high-level description of an implemented reference framework including deployment and runtime architecture and software development kit (SDK), and some example applications built with this framework (Section 4). We then show some experimental results which confirm the augmentation capabilities of our approach (Section 5). We present some related work and oversee further research themes along this novel application model at the end of this paper (Section 6 and 7).

2 Concepts & Elasticity Patterns

2.1 Concepts and Benefits

We define elastic applications as having two properties. First, following the client/server split of traditional web applications, an elastic application is split or partitioned so that execution occurs partially on the device and partially on the cloud. Previous work has proposed many mechanisms for splitting an application into modular components for remote execution or *cyber foraging* purposes, such as [8, 9, 11, 13, 16, 18]. For elastic devices we assume application developers can determine how to organize weblets based on their functionalities and runtime behaviors such as computation demand, data dependency, and communication need, which we believe should be part of high-level design consideration of an application. Elastic middleware should provide necessary SDK and tools allowing developers to implement and test their designs. One principle for partitioning applications is that each weblet should have minimum dependency on others. This is not only for robustness but decreases communication overhead between weblets during runtime.

Second, the *execution configuration* of an elastic application is not static, instead it is determined when the application is launched and potentially modified during runtime. By execution configuration, we mean the assignment of application partitions to execution units (e.g., cores or virtual machines), either on the device or in the cloud. The left hand side of Figure 1 shows some possible execution configurations for an application using three weblets.

There are several benefits that the elastic application concept offers to mobile users and application developers deriving from coarse-grained application partitioning and dynamic configuration. First, elastic applications are not constrained by the compute capabilities of today's mobile platforms and can be configured to take advantage of multiple processing cores when available. If more compute (or storage) is needed then this can be obtained from the cloud. As devices become more powerful, compute and storage can shift back to the device. On the other hand, mobile device compute and storage need not be designed to satisfy the most demanding applications. Device resources can be modest (and less power consuming) since the more demanding applications can acquire resources from the cloud. From a performance perspective, the ability to allocate resources in the cloud and migrate functionality gives the device great flexibility. For example, performance can be increased or optimized to fit various goals (such as responsiveness, monetary cost, or power consumption). Furthermore, application components that are partitioned for migration can also be replicated. The failure then of one instance of a replicated component need not compromise the application. Also, the elastic application model offers a testbed for future technologies of mobile devices. Applications that run on the cloud today can move to the device in future products. This greatly extends the lifetime of applications and reduces development costs.

Weblets	Web Services
HTTP (REST interface)	HTTP (REST or SOAP interface)
single client	many clients
client is application root or other weblet	clients are generally browsers or other web services
short-lived & long-lived requests	generally short-lived requests
dynamic endpoints (may migrate)	fixed endpoints
lifetime is client dependent	lifetime is client independent
runs on servers or client (cloud or device)	runs on servers
push to client possible	not available or non-standard

Table 1. Weblets vs. Web Services

2.2 Elasticity Patterns

We now consider elastic applications and weblets in more detail. Our motivation for using weblets is that developers are familiar with the web application model and so can easily transition from the client/server partitioning of web applications to the more general form of partitioning found in elastic applications. Furthermore, programming methods used for web applications, for example AJAX and REST, are adapted by weblets. To see the similarities and differences of web applications and elastic applications, it is interesting to compare weblets with traditional web services. We highlight some areas for comparison in Table 1.

In designing a web application, a key issue is determining what logic will run on the server and what on the client. For early web sites, the client was mainly used for rendering and input, but now with JavaScript, AJAX, and plug-ins such as Flash and Silverlight, many tasks can be performed by the client. With elastic applications there is a similar issue, but because several weblets can be created by a single application, the topology of elastic applications is more varied. It appears these topologies fall into some common patterns, what we call *elasticity patterns,* several of these are shown on the right hand side of Figure 1 and briefly summarized as follows.

Replication Patterns: Pools and Shadowing Weblet replication refers to running multiple weblets with the same interface, i.e., accepting the same types of request. There are two forms of replication: *pools* and *shadowing*. Weblet pools allow an application to leverage cloud CPU cycles and augment its throughput. With this pattern, the application issues requests that are routed to weblets as they become available. Weblet pools are well suited for applications that are easily divided into similar tasks, for example processing sets of images or scanning sets of files. Closely related to pools is shadowing in which the same request is sent to a set of replicated weblets in parallel. Shadowing can be used for fault tolerance and latency control. For example, shadowing a weblet on the device with a copy on the cloud can help the application recover from loss of network connectivity or loss of battery power. Shadowing can also enable more flexible latency control for an application, e.g., the device can use the earliest response from multiple shadowed weblets on the cloud.

Splitter Pattern With the splitter pattern, a set of worker weblets perform variant implementations of a shared interface. For example, the workers may encapsulate adapters to access different social networks, or codecs to process different media formats. The application is decoupled from the various implementations by a splitter weblet that

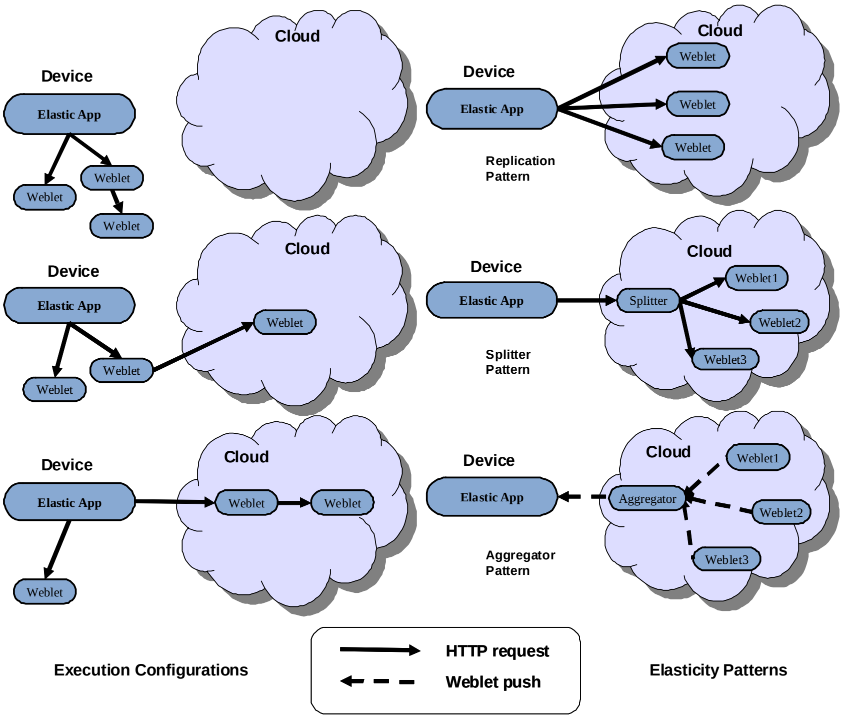

Fig. 1. Execution configurations and elasticity patterns.

routes requests to appropriate workers. This pattern increases application extensibility since new worker weblets are added without changing the application structure. Splitting can also enhance the user experience by converging multiple services on a single device. For instance, in the case where the worker weblets access different social networks, the splitter weblet's interface provides a unified or converged interface to a range of social networking services.

Aggregator Pattern An elastic application can also aggregate computations from multiple worker weblets. In this pattern, an aggregator weblet collects information from multiple worker weblets and uses *weblet push* to relay this information to the device. For example, an application can run multiple weblets in the cloud as background threads that monitor the user's web accounts (e.g., emails or instant messages), the aggregator weblet pushes events (such as account activity) to the device. In some cases the splitter and aggregator patterns are combined or overlaid, the splitter pushes requests to the workers while the aggregator pushes events back to the device.

3 Cost Optimization for Elastic Applications

3.1 Cost Model

The augmented computation of an elastic application is not free but introduces costs to the mobile device and user, which depends on when and where a weblet is running and communications within weblets or between weblets and Internet. Furthermore, elastic applications can exhibit variant runtime behaviors with dynamic execution configurations, such as power consumption, monetary consummation, application performance, and even security and privacy properties. Therefore, the dynamic execution configuration of an elastic application is decided based on some cost saving objectives, which form a cost model in our framework. As Figure 2 shows, the cost model takes inputs of sensor data from both device and cloud sides, and runs optimizing algorithms to decide execution configuration of applications. Device and cloud related data such as battery level, network conditions, device loads, cloud loads and other performance data including current latency of the application, are obtained from appropriate sensing modules. The output of the cost model is possible actions that lead to the optimal execution configuration for the application, such as allocating resources on the cloud, launching/migrating weblets on/to device and/or cloud, selecting/switching between different network interfaces, replicating and shadowing weblets on cloud, etc.

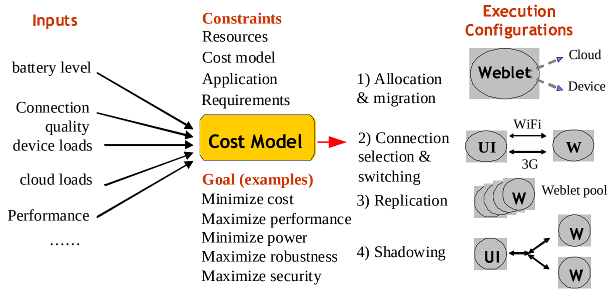

Fig. 2. Cost model of elastic applications.

An important part of the cost model is choosing the attributes or objectives that should be optimized. We consider the following four attributes in our current elastic application framework, while new cost objectives can be integrated easily.

Power Consumption Each application/weblet running on a mobile device consumes battery power by using CPU cycles, memory and radio module for communication with peer weblets on the cloud and/or external web services. The power consumption of a weblet on the device heavily depends on the I/O operations it performs [19, 5, 4]. In addition, different communication channels, such as W-CDMA, WiFi (802.11/a/b/g/n) etc., consume different power [2, 6, 3]. Considering the above, it is evident that although launching/migrating weblets to clouds should ideally save power consumption of computation on the device, the power consumption of network interfaces may override the benefits of the migration.

Monetary Cost Execution of a weblet on a cloud platform may involve a monetary cost for the application user, based on the exact resources consumed on the platform. Usually, a commercial cloud service provider measures the cost of a computing task based on the amount of CPU cycles, storage, and communication traffic (in and out) of a cloud platform [1]. The monetary cost of a weblet running on the cloud platform is determined by the size of the input data consumed by the weblet (including those from peer weblets on the device for the same application and external web services), total execution time of the weblet on the cloud platform, data size/rate for intra-cloud communication between this weblet and others within the same cloud service provider (if applicable), and any other attributes that affect these parameters, such as network status affecting data transmission rate.

Performance Attributes As an elastic application potentially runs across different platforms, latency is an important design consideration. There are different aspects of latency, such as impact on the user experience when using the application's UI and network latency with different network connections and traffic status, and the application latency to finish a particular computing task. Throughput also can be an important objective for some applications. For example, an application that does image analysis to find similar pictures from a large database needs maximum throughput. To achieve this, the heavy computing tasks are be launched or migrated to the cloud, although there is a tradeoff between doing this and the data communication overhead: too much communication may slow down the overall application throughput. Given this, building a good performance model is more challenging than power and monetary aspects. In general, to optimize latency, throughput and some application-specific options, CPU cycles and memory used by the weblets, along with the available network bandwidth for communication between the device and the cloud should be carefully evaluated.

Security and Privacy Security is increasingly concerned in web-based computing systems. A mobile device potentially contains many user secrets and privacy-sensitive data, such as: contacts, SIM information, credit card details and many other credentials that may be needed to consume web services. Naturally, a mobile user may trust her device more than the cloud platform which is controlled by a third-party service provider. As launching or migrating a weblet to the cloud may also require offloading user data to the cloud, the user security and privacy concerns are even higher with an elastic mobile device. A weblet on the device or the cloud may need to access external web services on behalf of the user. For cost modeling purposes, we need to evaluate if a weblet requires any user data and if the user has strong concerns about offloading such data to the cloud. If the user has concerns over doing this, the weblet that requires this data should be launched on the device only and never migrated. Furthermore, during runtime, if a weblet needs to acquire external user data from other web services, which usually requires user credentials (username/password, public key certificate, or any other security credentials), the weblet may have to be migrated back to the device.

3.2 Optimizing Execution Configuration with Cost Objectives

Once a cost model is developed for a particular application, a mechanism is needed for efficient and intelligent dynamic execution configuration, e.g., via some lightweight machine learning algorithms at the device side. In our implementation of one elastic application, we use Naïve Bayesian Learning techniques to find the optimal weblet configuration (# of weblets on device and cloud), given device status (in terms of CPU, memory and network consumptions), user preference (in terms of expected # of images that should be concurrently processed), and history data of the application.

As Figure 3 shows, a vector '$\mathbf{x}$' consists of values representing device status components such as the upload bandwidth, throughput, power level, memory usage and file cache. A vector '$\mathbf{z}$' consists of values representing user's preferred setting for cost objectives including monetary cost, power consumption, and processing speed. The configuration variable 'y' has values from 1 to N (max number of possible configurations), where each value maps to a specific configuration pair. Given all these data, the following expression can be applied to determine the most optimal configuration.

$$y^* = \operatorname*{argmax}_y p(y) \prod_{i=1}^{L} p(x_i|y) \prod_{j=1}^{M} p(z_j|y) \tag{1}$$

In the above expression, x_i is the i-th status component value that can have different number of states for each component and z_j is a j-th preference component, where $i \in \{1, 2, \cdots, L\}$ and $j \in \{1, 2, \cdots, M\}$, with L and M representing the number of components in the status vector and the number of components in the preference vector, respectively.

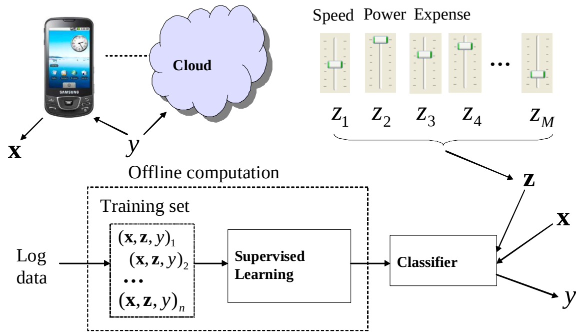

Fig. 3. Weblet scheduling through Machine Learning techniques.

Note that it is relatively easy to determine dynamic configurations in this application since it has only one type of weblet. For a general application with multiple types of weblets, each having different runtime behaviors, the optimization can be very complex and the computation itself may override the cost savings. Considering that an elastic application can be installed and executed by many users on similar devices, a service-oriented cost optimization implementation can save computation cost for the device.

4 Reference Implementation and Application Development

4.1 Reference Architecture

To experiment with this new application model, we have developed a reference framework including application bundle, architecture, and some example elastic applications. Our framework works with Amazon EC2 and S3. Figure 4 shows the main functional components.

In our current framework design, a typical elastic application consists of a UI component, one or more weblets, and a manifest. Weblets are autonomous software entities that run either on the device or cloud and expose RESTful web service interfaces via HTTP. The manifest is a static XML file that contains metadata for the application. It could be used to specify any requirements and constraints for the application and the individual weblets, such as: the digital signature needed to download/migrate the weblets, requirements for compute power, network and storage, time limits for weblet execution, maximum instances of the weblet that can be launched on the device and the cloud, if a weblet can be launched/migrated to the cloud and specifics about handling data required/generated by the application/weblets etc.

On the device side, the key component is the device elasticity manager (DEM) which is responsible for configuring applications at launch time and making configuration changes during run time. The configuration of an application includes: where the application's components (weblets) are located, whether or not components are replicated or shadowed (e.g., for reliability purposes), and the selection of paths used for communication with weblets (e.g., WiFi or 3G if such a choice exists). Each device also provides sensing data on the device such as processor type, utilization, and battery state. This data is made available to the elasticity manager and is used to determine when and where a new weblet instance should be launched.

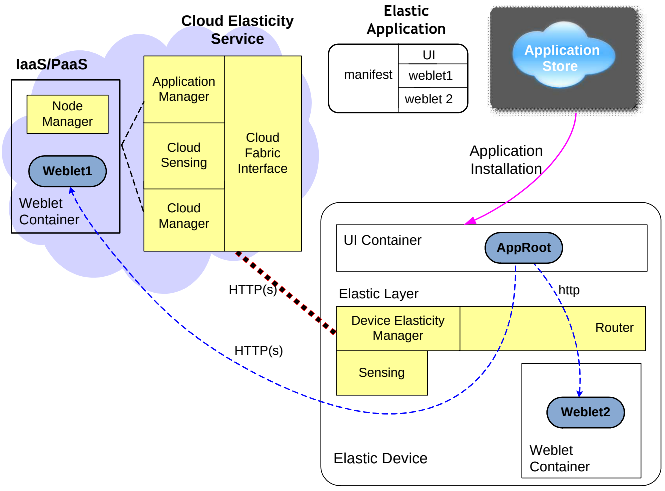

Fig. 4. Reference architecture for elastic application.

The cloud elasticity service (CES) consists of the cloud manager, application manager, and sensing information collection. The cloud manager is responsible for allocating resources from, and releasing to, underlying cloud nodes. It maintains usage information, including compute, bandwidth and storage, for the various weblets running on the cloud. The application manager provides functions to install and maintain applications on behalf of elastic devices, and helps launch weblets on different cloud nodes. Sensing information refers to the collection of operational data on the cloud platform. These data are made available to the cloud manager to assist it in tracking usage. As a service provider, the CES exports a web service, referred to as the cloud fabric interface (CFI) to elastic devices and applications. A node manager on each cloud node oversees resources associated with a particular node (server) within the cloud. It communicates directly with the cloud manager and application manager. Each node runs one or more weblet containers which are the weblet runtime environments hosted on an Amazon EC2 instance.

4.2 SDK Development

We have implemented a preliminary SDK for according to the reference architecture, which is used to develop the basic interfaces of weblets in our example applications. Using this SDK as a base, developers can build elastic applications in high-level languages such as JavaScript, Java, and C#. Currently the SDK has C# bindings; however we plan to extend it to other languages.

A typical elastic application includes a `AppRoot` component and one or more weblets. The `AppRoot` is the part of the application that provides the user interface and issues requests to weblets. All of these are packaged into one bundle, which includes the binaries of weblets and a manifest describing the application, and most importantly, the developer-signed hash values of the individual weblets. Figure 5 shows a state diagram illustrating the lifecycle of a weblet, including the various states that a weblet can be in and the actions that cause the state transitions. A weblet is an independent functional unit of an application that performs computing, storing, and networking tasks. It resembles an embedded or dedicated web server and presents a web service interface (i.e., it is accessed via HTTP). In our SDK, an abstract class called `AbstractWeblet` is defined to represent the core behavior of weblets. Other specific types of weblets can be implemented as subclasses of `AbstractWeblet` and extend its methods as required. Each weblet is associated with a weblet type and identified through a unique id. Once an application has defined one or more weblet types, it

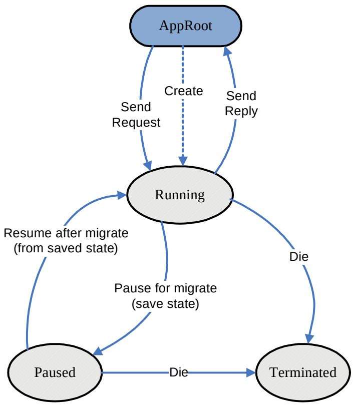
AppRoot
Create
Send Request
Send Reply
Running
Resume after migrate (from saved state)
Die
Pause for migrate (save state)
Paused
Die
Terminated

Fig. 5. Lifecycle of a weblet. A weblet is always created by the `AppRoot`, and can be in state of `Running`, `Paused`, or `Terminated`.

can use the DEM to create instances (i.e., to create specific weblets) and issues requests to these weblets.

The DEM can decide to migrate a running weblet from the device to the cloud or vice-versa; weblet migration is transparent to the application. When a weblet is running on device and the DEM decides to migrate it to a cloud node, the DEM issues a `Pause` request to the weblet, this causes the weblet to close its request interface, release resources and save state. The DEM then sends the saved state to the cloud via the CFI. After the state has been transferred to the cloud, the weblet is resumed and restores itself from the saved state. The CFI returns the new connection information for the weblet (e.g., IP address, port, and session tokens) to the DEM so that the DEM may continue to route requests to the weblet on cloud.

4.3 Example Applications

To demonstrate the elastic application model, we have developed several test applications with our SDK and deployed on the reference architecture with Amazon EC2. The simplest is an image processing application in which various filtering operations are applied to set of images. Following the replication pattern, a weblet pool is created on the cloud; images are then processed in parallel by pool members. The application can adjust the size of the pool, so it is possible to compare throughputs for different execution configurations. For example, the application running on a mobile device can be configured to offer the same throughput or greater as the application running on a PC.

A second example is a form of augmented reality in which real-world objects are detected and enhanced. This application runs tracking and rendering on the device and uses the splitter pattern with a set of matcher weblets on the cloud. Each matcher searches for different objects within video frames. The splitter collects information on identified objects and relays this to the device for rendering. By running the matchers in the cloud, many more objects can be detected (per unit time) than when the application runs fully on the device.

5 Experimental Validation of Elasticity

We validate the elasticity of our framework by using the aforementioned image processing application as benchmark. This application consists of only one type of weblet called `ImageWeblet`. Its functionality is to perform image filtering with an algorithm specified by the user. The weblet is replicated on the device and the cloud, as and when required. The total number of weblet instances spawned depends on application load and the number of weblets in the cloud, both specified by the user. The application UI enables the user to do the following configurations during runtime: online (can launch weblet at cloud) or offline (all weblets are running on the device) mode of the application, number of weblets to run on the cloud (if in online mode), the filtering algorithm to be used, and the number of images (workload) to process at the same time. The images used in by the experiment are 24-bit color with size 240 x 360. Figure 7 shows a snapshot when it is running on a Samsung Galaxy smart phone with Android 1.6.

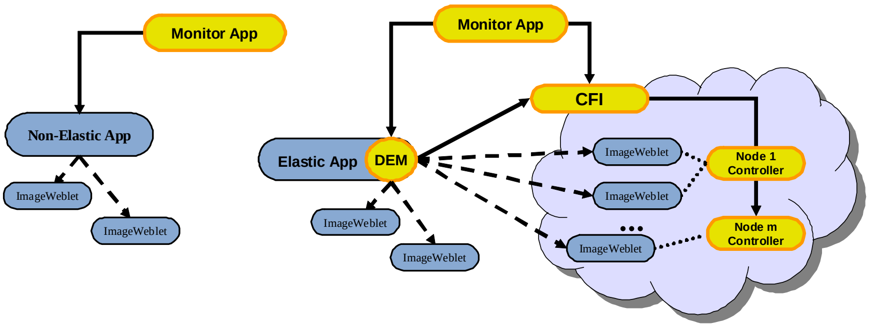

Fig. 6. Experiment configuration for elastic image processing application.

The goal of our validation is to compare the performance of an elastic device (ED) and a non-elastic device (NED) running the same image processing application. Figure 6 shows an overview of the demo configuration and system setup. For the elastic device, the application uses an in-house cloud comprising of 8 Linux boxes. A non-elastic version of the application is also running independently in order to compare it with the elastic version. Essentially, the non-elastic version uses only the device to run weblets, whereas the elastic version uses both the device and the cloud. The setup also includes PCs to host the CFI and a performance monitor application. The CFI is implemented with PHP scripts on a Linux server with Apache and MySQL.

Fig. 7. Snapshot of elastic image processing application on Samsung Galaxy.

The performance monitor collects several measurements, including the available upload/download bandwidth (KB/sec), application workload (number of images to be processed) and throughput (the number of image tiles processed/sec), average CPU usage (%), and available memory (MB), from the test device and from the cloud. In addition, it also maintains information about the total number of weblets started for the application and the individual number of weblets running on the device and the cloud.

Each configuration has a unique composition of device weblets and cloud weblets. We set the maximum number of weblets as 16 and consequently, more than 100 different configurations are possible. The configuration specifying 1 device weblet & 0 cloud weblets is considered the default configuration for the non-elastic device. Among all possible configurations, we chose the 74 configurations where the number of device weblets is less than or equal to 4 (due to limitations with CPU utilization) for the experimental analysis. For each configuration, the data was collected 20 times and the average values were considered for final comparisons.

Figure 8 shows the performance of the elastic device over 74 configurations. In comparison with the throughput of about 6 tiles/sec for the default/non-elastic device configuration (1 device weblet, 0 cloud weblets), the throughputs of all other configura-

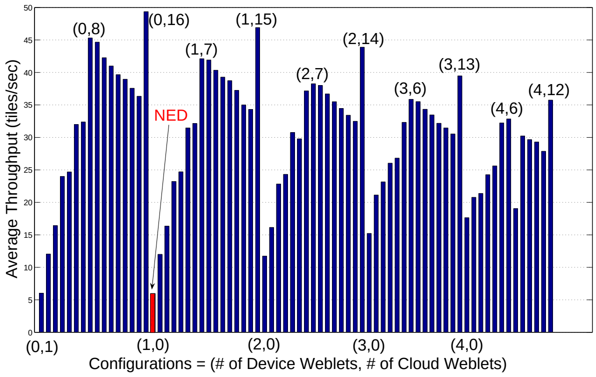

Fig. 8. Throughputs vs. configurations

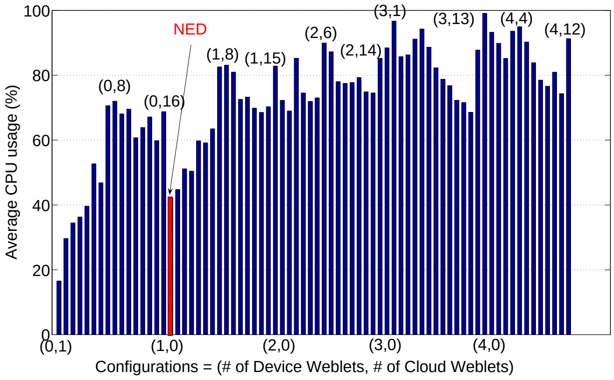

Fig. 9. CPU usage vs. configurations

tions are better. We can observe that the throughput for the configuration with 0 device weblet and 16 cloud weblets has the highest throughput among the 74 configurations tested. The configuration with 16 device weblets has the best performance, as there are a total of 16 images in load 3. A surprising observation is that the configuration with 8 weblets performed better than configurations with 9-15 weblets (a result of internal application logic). This indicates that an intelligent weblet scheduling is essential to identify the most efficient weblet configuration.

CPU usage is more predictable overall, in that more device weblets lead to more CPU usage. However, the trend is interesting when comparing the number of device weblets. For configurations with up to 2 device weblets, running more cloud weblets leads to more CPU usage. For configurations with 3 and 4 device weblets, a general trend is that running more cloud weblets reduces the CPU usage. By combining CPU usage data in Figure 9 with the throughput data in Figure 8, we are able to identify the configurations that lead to low CPU usage and high throughput: for instance, configurations (0,2), (0,3) and (0,4) have lower CPU usage (than that of a non-elastic device)

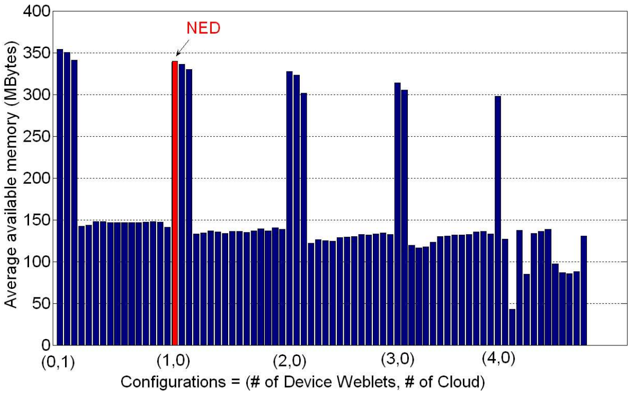

Fig. 10. Available memory vs. configurations

and higher throughput. This results in more available CPU cycles for other applications and improves multi-tasking capabilities.

Figure 10 shows interesting but not easily comprehensible results regarding the available memory versus configurations. Certain configuration such as (0,1), (0,2), (0,3), (1,0), (1,1), (1,2), (2,0), (2,1), (2,2), (3,0), (3,1), and (4,0) have much available memory. Most of the configurations have only up to 4 total weblets and using only the device weblets consumed only a little memory up to 4 device weblets. Meanwhile, other configurations up to (3, 13) have similar available memory, but there are much variation between (4,1) and (4,12). It is not clear why the system behaves that way, but it could be related to cache operations and memory paginations. This will need further investigations. Combining this result with Figure 8 can lead to a memory-constrained optimal configuration. Of course, it would also be conceivable to find a good configuration constrained on both memory and CPU.

6 Related Work

The elastic model builds on previous work in the areas of remote execution and application offloading. Cyber foraging [18, 9, 8, 14] is a common approach explored by many to augment the capability of resource-constrained mobile devices. The basic idea is to dynamically discover and make use of nearby resources, aka surrogates, to offload the execution of an application or parts of an application running on a mobile device. Compared to these approaches, elastic application model has more flexible deployment patterns to parallelize tasks on multiple remote cores.

Narayanan et al. [15] use historical application logging data to predicate the fidelity of an application, which decides its resource consumption. However, in this work, only aspects of device hardware and application inputs are considered. In our cost model, we consider more comprehensive factors not only on device side, but also on cloud side. Uniquely, we incorporates user preferences in terms of cost objectives. Gurun et al. [12] extend the network weather service (NWS) toolkit in grid computing to predict offloading, which can be leveraged as an implementation mechanism for our cost model.

CloneCloud takes the approach of cloning the entire user's mobile device environment on a remote server. Applications can then be quickly restarted on or migrated to the remote machine when the user's machine is running low on resources [10]. Similar virtual machine-based approach is used by cloudlet [17]. As mentioned in Section 1, our elastic application model offloads computing tasks in more fine-grained level such that it leverages the parallel computing advantage of cloud resources.

Some research work extend existing programming language and application runtime middleware to transform applications into distributed systems [11, 13, 16]. Adaptive Offloading [11] leverages Java's object oriented design to identify possible partitions for a Java application and modified the JVM to support such partitioning. Coign [13] makes use of the location transparency supported by COM and converts an application built from COM components into a distributable application. R-OSGi [16] extends the centralized module management functionality supported by the OSGi specification to enable an OSGi application to be transparently distributed across multiple machines. The main limitation with these approaches is that they are tied to one particular language or specification and hence not suitable for a wide range of applications. Compared to these approaches, our proposed elastic application model is programming language independent, and can be extended to many existing application middleware.

7 Conclusions and Future Research Themes

We propose an elastic application programming model aiming to remove the constraints of specific mobile platforms by providing a distributed framework that extends the device into the cloud. The salient feature of this model is that it offers a range of elasticity patterns between resource-constrained devices and Internet-based clouds. Each pattern in turn can be realized by several execution configurations. A comprehensive cost model is used to dynamically adjust execution configurations thus optimizing application performance in terms of a set of objectives. We present the high level design of elasticity framework and primitive experimental results with an example application.

There are a set of directions that need further research efforts. First of all, we use a simple weblet launching scheduling mechanism in our example application, while a general cost optimization engine is very desired for elastic applications with comprehensive considerations based on our cost model. Further, as aforementioned in the elasticity patterns, weblets of a single application may share application data and state. Since weblets run in different locations, it is desirable to replicate data to increase performance, but then data integrity and synchronization become issues. As another issue, code and computation migration is a traditional problem in many systems [9, 20]. How to support runtime weblet migration thus enhance mobile user experience but at the same time achieve the transparency and seamlessness is challenging. Furthermore, integrity and data security of weblets running on cloud are essential problems for many applications. We have designed a lightweight protocol to distribute shared secrets and session keys between weblets for mutual authentication purposes [21]. However, how to build strong trust between weblet runtime environments on cloud and device is an open problem.

References

1. Amazon ec2, http://aws.amazon.com/ec2/.
2. Rfmd data sheet, http://www.rfmd.com/databooks.
3. Wifi power consumption analysis,
http://nesl.ee.ucla.edu/fw/documents/reports/2007/poweranalysis.pdf.
4. Samsung corp., flash/smartmedia/filesystem memory databook, 2000.
5. Samsung semiconductor dram products,
http://www.usa.samsungsemi.com/products/family/browse/dram.htm, 2001.
6. Analog devices data sheet, analog device inc.,
http://www.analog.com/productselection/pdf, 2003.
7. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.
8. R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H. Yang. The case for cyber foraging. In *Proc. of the 10th ACM SIGOPS European Workshop*, 2002.
9. R. K. Balan, M. Satyanarayanan, S. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In *Proc. of The 1st International Conference on Mobile Systems, Applications, and Services*, pages 273–286, 2003.
10. B.-G. Chun and P. Maniatis. Augmented smartphone applications through clone cloud execution. In *USENIX HotOS XII*, 2009.
11. X. Gu, A. Messer, I. Greenberg, D. Milojicic, and K. Nahrstedt. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing*, page 66.
12. S. Gurun, C. Krintz, and R. Wolski. Nwslite: A light-weight prediction utility for mobile devices. In *Proc. of International Conference on Mobile Systems, Applications, and Services*, 2004.
13. G. C. Hunt, M. L. Scott, G. C. Hunt, and M. L. Scott. The coign automatic distributed partitioning system. In *Proc. of the 3rd Symposium on Operating Systems Design and Implementation*, pages 187–200, 1999.
14. O. R. J. Porras and M. D. Kristensen. *Dynamic Resource Management and Cyber Foraging*, chapter Middleware for Network Eccentric and Mobile Applications. Springer Press, 2008.
15. D. Narayanan, J. Flinn, and M. Satyanarayanan. Using history to improve mobile application adaptation. In *Proc. of the 3rd IEEE Workshop on Mobile Computing Systems and Applications*, 2000.
16. J. S. Rellermeyer, G. Alonso, and T. Roscoe. R-osgi: distributed applications through software modularization. In *Proc. of the ACM/IFIP/USENIX International Conference on Middleware*, 2007.
17. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, (4), 2009.
18. J. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proc. of the 3rd Working IEEE/IFIP Conference on Software Architecture*, 2002.
19. N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. In *Proc. of the Int. Symposium on Computer Architecture*, 2000.
20. C. Xian, Y. H. Lu, , and Z. Li. Adaptive computation offloading for energy conservation on battery-powered systems. In *ICPADS*, 2007.
21. X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong. Securing elastic applications on mobile devices for cloud computing. In *Proc. of ACM Cloud Computing Security Workshop*, 2009.