Elastic HTML5: Workload Offloading Using Cloud-Based Web Workers and Storages for Mobile Devices

Xinwen Zhang, Won Jeon, Simon Gibbs, and Anugeetha Kunjithapatham

Computer Science Laboratory, Samsung Information Systems America
75 W. Plumeria Drive, San Jose, CA 95134

Abstract. In this position paper, we propose the concept of *Elastic HTML5*, which enables web applications to offload workload using cloud-based web workers and cloud-based storage for mobile devices. Elastic HTML5 is a collection of software components and functions in for a web runtime agent (e.g., web browser); this includes components and methods to create and manage web workers in the cloud so as to augment the computation functionality of a browser-based application running on a device. Elastic HTML5 also includes the components and methods to create and manage elastic storage between the main thread of a web application and its web workers. With these functions, a web application can run in elastic manner such that, whenever necessary, the mobile device can obtain resources from the cloud, including computation and storage, and when necessary, it can run offline at the device side completely.

1 Introduction

HTML5 is being proposed by W3C as the next generation of HTML. Although the specification has been under development since 2004, many features of HTML5 are supported by the latest builds of browsers including Firefox, Chrome, Safari and Opera. Microsoft is also starting to support HTML5 within Internet Explorer. Web workers and local storage are important concepts and features in HTML5. A web worker is a script that runs in the background within the browser. For example, in the Chrome browser, a web worker is an independent background renderer process. A web application (e.g., an HTML page) can communicate with a web worker with `onMessage()` and `postMessage()` methods. A web page usually can only talk to web workers from the same origin. Each web application or web worker can have persistent local storage (such as SQL database). Compared with traditional cookies, local storage has the benefits of larger data size, supporting variant data types, and first class object names to access data with JavaScript.

Mobile devices such as cellular phones, in general, are constrained platforms with limited computational power and storage, support for a fixed set of codecs and data formats, and limited capability to access and process web services. To

M. Griss and G. Yang (Eds.): MOBICASE 2010, LNICST 76, pp. 373–381, 2012.
© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2012

address these restrictions for computing-, networking- and storage-intensive web applications, we propose the concept of *Elastic HTML5*, which enables an elastic version of the web worker model in HTML5 and allows workers to run on remote cores. Elastic storage between a device and cloud dynamically extends the capability of a web application and synchronizes data between the web application's main thread and its web workers.

Our elastic scheme enables transparent distribution of the computation and storage of a single web application into multiple locations including client browser environments and remote cloud nodes. The main function to support this feature is a pair of proxy web workers on client and cloud sides to relay and route messages between other web workers. Rather than a simple and rigid solution where nearly all processing and storage is done either on the cloud or on the device, we want to be able to migrate functionality between the device and cloud, as and when it is required. This ability allows the device to adapt to different workloads, performance goals, and network latencies. For example, an application could run locally when the device workload is light, but as the workload increases, more and more of its computation can be shifted off the device to the cloud.

Some of the advantages of elastic HTML5 include:

- It can transparently augment mobile browser capability with cloud computing and storage resources.
- It needs no change to the existing web programming model (i.e., HTML, CSS, and JavaScript) and facilitates cross-platform development.
- It introduces no new programming language to application developers.
- It uses a simple extension for integration with web applications, via web worker and storage APIs.
- It requires a simple extension and integration with mobile platforms which support HTML5 web worker and storage specifications.

The rest of this paper is organized as follows. Section 2 covers some background and related work about HTML5 and computation offloading. We highlight some features and functions of elastic HTML5 in Section 3, including launching web workers in the cloud and runtime message flow. We discuss some future research directions towards practical elastic HTML5 at the end of this paper.

2 Background

HTML5. The W3C HTML5 working draft specifies the APIs of HTML5 applications. Web workers and local storage are important features of HTML5. A web worker can be forked from the main browsing context in the browser runtime and can communicate via `postMessage()` and `onMessage()`. An HTML5 application can use local storage of a client, such as SQL database, to store application-specific data. So far, there has been no prior consideration of running web workers and storage in the cloud for the client-side browsing sessions.

Remote Execution Systems. Cyber foraging [7,8,16] 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,

called *surrogates*, to offload the execution of an application or parts of an application running on a mobile device. Adaptive Offloading [10], Coign [11], and R-OSGi [14] leverage programming language and application runtime middleware to transform applications into distributed systems. However these remote execution mechanisms and architectures are not for the browser environment and many of them are tightly coupled with specific programming languages such as Java. In addition, most of them require a new application model, which target splitting legacy applications to run in distributed manner.

CloneCloud [9] takes the approach of cloning the entire user's mobile device environment on a remote server(s). Applications can then be quickly restarted on or migrated to the remote machine when the user's machine is running low on resources. Whereas CloudCloud needs to change existing applications to make them partial on device and partial on cloud, our scheme does not need to change the application model used by web applications. It uses existing interfaces for communication between JavaScript contexts and works transparently with HTML5 applications. More specifically, existing HTML5 applications can use our techniques without any modification.

Virtual Machine Migration. Virtual machine migration [12,17] and VM-based cloudlet [15] are complementary approaches to enable users to seamlessly access their applications and data across multiple and heterogeneous devices in general. It also enables users to instantly continue/restore an application on a different device, when their current machine is running low on resources. The fundamental design objective of our elastic web application model is to remove the constraints of specific mobile platforms by providing a distributed framework that extends the device into the cloud. The salient feature of the elastic model is that it can offer a range of elasticity patterns between resource-constrained devices and Internet-based clouds [19]. Each pattern in turn can be realized by several execution configurations. A comprehensive cost model can be used to dynamically adjust execution configurations thus optimizing application performance in terms of power savings, monetary savings, or throughput.

Browsing Through Remote Rendering. Flashproxy [13] supports active web content on mobile devices using a proxy to splice active content out of web pages and replace it with an Ajax-based remote display component. The spliced active content executes within a remote sandbox on the proxy. However, Flashproxy is specific for the Flash plugin, and it has to change Flash bytecode to access a proxy. Similarly, Opera Mini is a JavaME application that displays web pages that are reformatted and compressed by Opera servers [4]. Deepfish is Microsoft's experiment in remote rendering for mobile browsing [2].

Compared with our scheme, Flashproxy has only one proxy server and does not offer an elastic storage mechanism. Also, Opera Mini and Deepfish work for web page format/rendering only, so they do not distribute computation or storage of a single web application into many locations.

3 Elastic HTML5

3.1 HTML5 Web Worker

The W3C WHATWG specifies HTML5 web worker [5]. Web workers allow JavaScript to run in parallel on a web page, without blocking the user interfaces. Typically, a web worker is an independent browser execution context (i.e., thread or process) from any web page. It runs in background and talks with the web page that creates it via dedicated message channels. It can talk with web servers via `XMLHttpRequest` (for HTML4) and `WebSocket` (for HTML5) methods. Furthermore, it can store data persistently in local database. A web worker can be either dedicated, in that it works for one web page, or shared by multiple pages, e.g., to transfer data or messages between them. Also, it can create subworkers, delegate tasks, aggregate results, and can be a shared library for other web workers.

We have tested the support of web worker functionalities in various mainstream browsers. As a web worker runs in an independent browser execution context, it cannot access global variables and DOM (Data Object Model) from the main thread of a web page. Also, as accessing to session and local storage are through global variables in the current HTML5 specification, a web worker cannot access a web page's session and local storages. However, according to the specification, a web worker can access local database storage, which can be independent to each web worker, or can be shared with the main thread of the application, although so far based on our experiment, none of the current mainstream browsers support this function. Therefore, this position paper focuses on the elasticity of web worker and database-based storage. However, if other forms of local storage are supported for web worker in the future, it can be applicable to them.

3.2 Elastic Web Worker

The basic idea for elastic web application with HTML5 is to run web worker in a cloud platform, e.g., private cloud such as home and enterprise, or public cloud such as Amazon EC2 [1] and Google AppEngine [3]. Figure 1 shows the overview of our elastic web worker. To transparently support this elasticity, the following requirements need to be supported for web worker:

- A web worker should communicate with a web page's main thread with existing APIs (i.e., message-based communication channels), no matter it is running on local browser environment or in cloud environment.
- Web workers can communicate with each other with existing APIs no matter they are located together in one location or in separate locations.
- A web worker should be able to access database storage with existing APIs, no matter the database is in device side or in cloud. That is, the database should appear "locally" no matter where the web worker is running.
- When running in cloud, a web worker should be able to access external web servers and comply with the same security policies as it runs in the client-side browser environment.

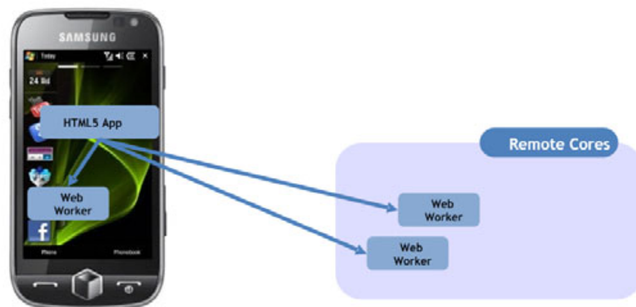

Fig. 1. Concept of Elastic Web Worker

3.3 Launching Web Worker

Figure 2 shows the infrastructure to launch a web worker in elastic HTML5. The high-level work flow is following:

1. Whenever the main thread of an HTML5 application (e.g., a loaded web page) invokes a web worker, it sends the request to a device-side elastic web worker manager (EWWM), which decides where to launch the web worker - in the same browser environment of the device, or on cloud. EWWM provides the same web worker message APIs to JavaScript such that there is no need to change any existing HTML5 application.
2. EWWM checks security policies on launching the web worker, such as the same-origin policy (SOP), according to the HTML5 specifications. A web worker is invoked by EWWM only when it passes the security checks.
3. EWWM maintains a table with each loaded web page, which includes the information of all the names of active web workers (accessed by the main thread in the web page) and others including URLs if they are running in the cloud, and the message ports if they are running locally.
4. EWWM decides where to launch the target web worker. When the worker is launched locally, EWWM invokes the web worker with parameters from the first step, and updates the table with the related information of the launched web worker. The decision can be made by considering some cost objectives [19].
5. If EWWM decides to launch the target web worker in the cloud, it files the request to cloud web worker service (CWWS), along with the parameters and code of the web worker. CWWS in turn arranges necessary resources such as a specific CPU core, storage, etc. and launches the web worker with the transferred parameters.
6. CWWS returns the end point (URL) of the web worker to EWWM, which in turn updates its active web worker table with the related information.
7. After these steps, the main thread and invoked web workers can talk to each other, using existing message-based web worker APIs.

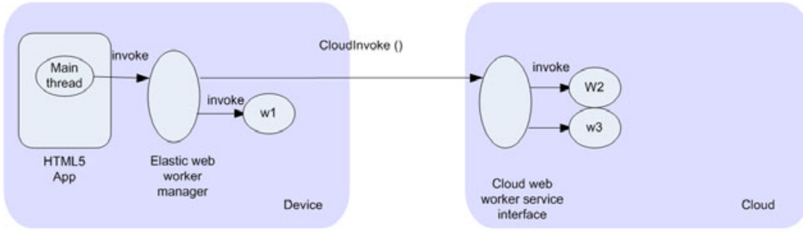

Fig. 2. Launching Web Worker in Elastic HTML5

In order to save the cost of transferring the code from device to cloud, EWWM can send the web worker invocation request with parameter of the web worker's absolute URL. Then, CWWS fetches the web worker code and invokes it in cloud. Again, the security policies are enforced by EWWM and CWWS such that web workers will be executed with the same origin as the device-side web application, although its code can be retrieved from a different origin.

In real implementation, EWWM can be a browser extension, a plugin, or a usual web worker. Also, CWWS can be web service interface, or simply a cloud-based web worker.

3.4 Runtime Message Flow

Figure 3 shows the runtime interaction among a web application and elastic web workers. Two proxy web workers, residing on device and cloud side and representing the role of EWWM and CWWS, respectively, route and relay messages between original web workers and web application main thread. All communication between web workers are through `postMessage()` and `onMessage()` interfaces specified by HTML5. The communication between two proxy web workers are bidirectional, e.g., via `WebSocket` APIs [6], a dedicated TCP connection, or a long lived HTTP connection, which is not visible to the main thread and other web workers. The proxy web workers act as *distributed elasticity layer* for web applications.

A cloud web worker can access external web servers (e.g., via Ajax) directly, according to the same security policies in the client-side browser environment. Optionally, a cloud web worker can only access external web servers via the device-side proxy web worker, which can enforce security policies in more efficient and consistent way. To support this, the Ajax APIs should be wrapped or rewritten to embody this indirect calling. With this design, each web worker and the main thread have exactly the same interfaces as specified by HTML5 or its future versions. All communications and location-awareness are handled by the proxy web workers transparently.

A web worker can be migrated from device to the cloud or vice versa. As each web worker communicates with others and external web servers with the

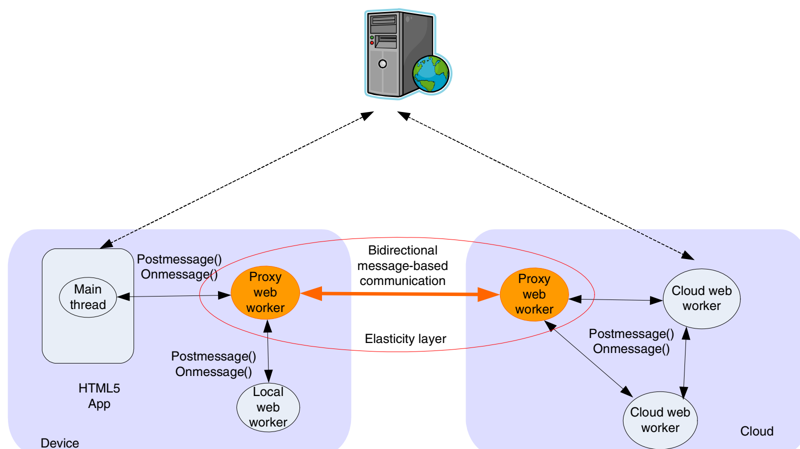

Fig. 3. Message Flow of Elastic Web Worker

same interfaces as specified by HTML5, a web worker can seamlessly run after migration. The decision for the migration can also be based on some objectives.

3.5 Elastic Storage

Each web worker can have its local database storage. To maintain transparency, a web worker can access database no matter where it is running, thus management and synchronization of local storage between device and the cloud has to be supported. Figure 4 shows the method to enable this idea. By this description, we assume that there is only one active copy of a data item in a database - either the data is in cloud-side or in device-side.

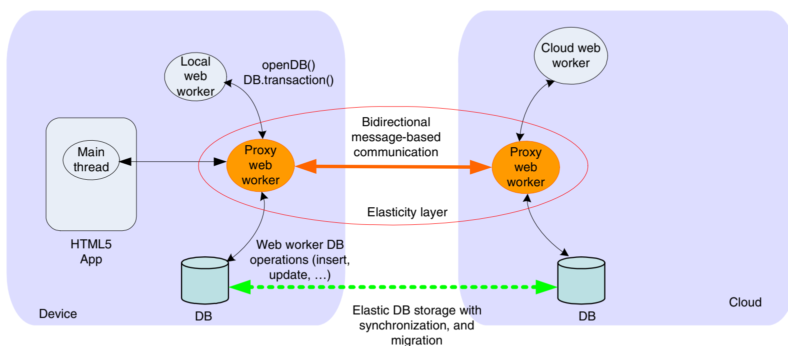

Fig. 4. Database Storage and Web Worker in Elastic HTML5

Similar to web worker APIs, the elasticity layer provides the same interfaces specified by HTML5 to local database storage. when a web worker invokes APIs to access database such as `openDB()`, the proxy web worker looks up at both device and cloud sides to check if the target database exists. If not, the database is created locally (local to the invoking web worker) and database transactions are performed as usual, e.g., with `DBtransaction()` method. Otherwise, the elastic layer routes the message to the corresponding database either in device or cloud side, and the corresponding proxy web worker finishes the operations and returns results. Thus, although a database is located in the cloud, it works like a local storage to device web workers.

To support offline operation, the device-side and cloud-side database should be synchronized or even migrated. This can be done either online or offline. Also, a database synchronization or migration can be done along with the invocation or migration of a web worker. As alternatives, the database synchronization and migration can be done by elasticity layer or other mechanisms.

According to different operations from different web workers, the databases at device and at cloud can be split from a single database, e.g., each has different tables, or each has different attributes of the same table. Database splitting and union operations are possible along with invocation and migration of web workers.

4 Concluding Remarks and Future Work

While we believe elastic HTML5 bridges the web-based programming model for mobile devices and powerful computing resources from cloud in a friendly manner, there are several challenges to be resolved before it can be widely useful for general web applications. First, the network latency is a major issue for many highly user interactive web applications. To minimize latency, web workers having tight dependency on updating UI parts of an application should run at the device side. In general, there can be a break-even point by considering the benefit of offloading web workers to cloud to have fast computation and the cost of network latency, which needs comprehensive study in our future work.

With distributed web workers and storage, running an elastic HTML5 application should decide what logic will run on the cloud and what on the client. As the topology of elastic applications is more varied, we can identify several common patterns, such as web worker pools and shadowing on cloud, task splitting web workers, aggregating web workers, and many others [19]. Also, to decide where a web worker should be launched, and when an running web worker should be migrated between a mobile device and cloud, some cost factors should be considered, e.g., for minimizing power consumption of the device. Overall, we believe a cost model is necessary for elastic HTML5 applications.

Since web workers run in different locations, it is desirable to replicate database storage to increase performance, but then data integrity and synchronization become issues. Further, code and application state computation migration is a traditional problem in many systems [8,18]. It is a challenging task to support runtime web worker migration thus enhance mobile user experience, but at the

same time achieve the transparency and seamlessness. Furthermore, integrity and data security of web workers running on cloud are potential problems. We have designed a lightweight protocol to distribute shared secrets and session keys between distributed application components running on mobile devices and cloud nodes for mutual authentication purposes [20]. However, how to build strong trust between web worker runtime environments in the cloud is an open problem that we will explore.

References

1. Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/
2. Deepfish, http://en.wikipedia.org/wiki/Microsoft_Live_Labs_Deepfish
3. Google AppEngine, http://code.google.com/appengine/
4. Opera Mini, http://www.opera.com/mobile/
5. Web Workers, Draft Recommendation (August 23, 2010), http://www.whatwg.org/specs/web-workers/current-work/
6. WebSocket API, Editor's Draft (August 10, 2010), http://dev.w3.org/html5/websockets/
7. Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., Yang, H.: The case for cyber foraging. In: Proc. of ACM SIGOPS European Workshop (2002)
8. Balan, R., Satyanarayanan, M., Park, S., Okoshi, T.: Tactics-based remote execution for mobile computing. In: Proc. of MobiSys (2003)
9. Chun, B.-G., Maniatis, P.: Augmented smartphone applications through clone cloud execution. In: Proc. of USENIX HotOS XII (2009)
10. Gu, X., Messer, A., Greenberg, I., Milojicic, D., Nahrstedt, K.: Adaptive offloading for pervasive computing. IEEE Pervasive Computing 3(3) (2004)
11. Hunt, G.C., Scott, M.L.: The Coign automatic distributed partitioning system. In: Proc. of OSDI (1999)
12. Kozuch, M., Satyanarayanan, M.: Internet suspend/resume. In: Proc. of IEEE WMCSA (2002)
13. Moshchuk, A., Gribble, S.D., Levy, H.M.: Flashproxy: transparently enabling rich web content via remote execution. In: Proc. of MobiSys (2008)
14. Rellermeyer, J.S., Alonso, G., Roscoe, T.: R-OSGi: Distributed Applications Through Software Modularization. In: Cerqueira, R., Pasquale, F. (eds.) Middleware 2007. LNCS, vol. 4834, pp. 1–20. Springer, Heidelberg (2007)
15. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based Cloudlets in mobile computing. IEEE Pervasive Computing 8(4) (2009)
16. Sousa, J., Garlan, D.: An architectural framework for user mobility in ubiquitous computing environments. In: Proc. of IEEE/IFIP Working Conference on Software Architecture (2002)
17. Travostino, F.: Seamless live migration of virtual machines over the man/lan. In: Proc. of SC (2006)
18. Xian, C., Lu, Y.H., Li, Z.: Adaptive computation offloading for energy conservation on battery-powered systems. In: Proc. of ICPADS (2007)
19. Zhang, X., Jeong, S., Kunjithapatham, A., Gibbs, S.: Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms. In: Cai, Y., Magedanz, T., Li, M., Xia, J., Giannelli, C. (eds.) Mobilware 2010. LNICST, vol. 48, pp. 161–174. Springer, Heidelberg (2010)
20. Zhang, X., Schiffman, J., Gibbs, S., Kunjithapatham, A., Jeong, S.: Securing elastic applications on mobile devices for cloud computing. In: Proc. of ACM Cloud Computing Security Workshop (2009)