2010 IEEE 3rd International Conference on Cloud Computing

Provisioning Web Services From Resource Constrained Mobile Devices

Mahbub Hassan, Weiliang Zhao and Jian Yang
Department of Computing
Macquarie University
Sydney, Australia
mhassan,wzhao,jian@science.mq.edu.au

Abstract—The increasing processing power, storage and support of multiple network interfaces are promising the mobile devices to host services and participate in service discovery network. A few efforts have been taken to facilitate provisioning mobile Web services. However they have not addressed the issue about how to host heavy-duty services on mobile devices with limited computing resources in terms of processing power and memory. In this paper, we propose a framework which partitions the workload of complex services in a distributed environment and keeps the Web service interfaces on mobile devices. The mobile device is the integration point with the support of backend nodes and other Web services. The functions which require the resources of the mobile device and interaction with the mobile user are executed locally. The framework provides support for hosting mobile Web services involving complex business processes by partitioning the tasks and delegating the heavy-duty tasks to remote servers. We have analyzed the proposed framework using a sample prototype. The experimental results have shown a significant performance improvement by deploying the proposed framework in hosting mobile Web services.

Keywords-Web service; mobile device; cloud computing; performance; resources management

I. INTRODUCTION

Cloud computing is the way of simplifying access to the complex Web based applications, services and platforms, instead of having them hosted in local computer or mobile device. For example, instead of having a laptop to do the actual word processing, Google Docs service on the cloud can be used to do this to conserve resources for other functions. Complex applications, which take months or years to implement on a company's servers (e.g. Customer Relationship Management applications), can now fully reside in the cloud and can be accessed from anywhere. Besides, applications for mobile devices are getting more and more popular since general consumers have started to utilize advanced capabilities of their mobile devices. Many applications require significant processing power and memory in the devices. The mobile cloud computing can help to reduce the workload of mobile devices by exploiting remote resources on the cloud. Mobile cloud computing promises the adoption in the businesses in future, by enabling even the resource demanding services on smart phones or laptop computers. For example, a mobile phone can use extra storage from the cloud for multimedia files like pictures or music. Using such architecture, business users will benefit from collaboration and data sharing, and personal users will benefit from social networking mashups that let them share multimedia files or incorporate their address books and calendars.

Hosting Web services on mobile devices is a new concept. This will bring great convenience for the owners of mobile devices to administrate and manage their business services anytime and anywhere. Hosting Web services on mobile devices can create a lucrative market. For example, a mobile device with a GPS receiver can host a service to track its exact location at any time. This type of service can be used by shipping companies to track the delivery and estimate the arrival time directly. In a supply chain management system [1], the updated services offered by a person can be available through Web services on his mobile device. In emergency or disaster situations, skilled people like doctors and nurses can be located using mobile Web services for help.

Most of the available mobile devices have low resource capabilities. Compared to stationary nodes, mobile devices have less memory, slower processing speed, and unreliable communication links. Running complex and complicated services on a mobile device will consume most of its resources and may obstruct the device to perform its core functions (e.g. voice calls). It is not feasible and necessary for a single mobile device to execute complex processes independently. Services should be designed in a way to put less workload on the mobile device and delegate heavy-duty tasks to backend servers on the cloud. Web service (WS) hosting on mobile devices demands a flexible, light-weight and scalable execution environment, which can exploit resources from the cloud as necessary.

This paper proposes a framework for hosting Web services involving complex business processes on mobile devices. The mobile device acts as the integration point with the support of backend servers and remote Web services. The framework provides support for executing functions locally which require the resources of the mobile device, and delegating the heavy-duty tasks to backend servers. The proposed partitioning framework provides a high level design about how to assign different tasks to be executed on a mobile device and backend nodes on the cloud. Different partitioning schemes are analyzed based on the order of the execution process. A comparison between the partitioned and not-partitioned execution engine has shown a significant improvement in the performance.

978-0-7695-4130-3/10 $26.00 © 2010 IEEE
DOI 10.1109/CLOUD.2010.30

The rest of the paper is organized as follows. Section II highlights some related work in the area of mobile Web services. Section III presents the architectural overview of the proposed framework. Section IV explains the details of the partitioned execution engine. Section V discusses the different partitioning schemes for mobile WS execution environment. Section VI presents the proposed partitioned mobile Web service framework. The estimation of overheads to determine the performance tradeoffs is illustrated in Section VII. In Section VIII, experimental result shows the performance improvement by using the partitioning framework. Finally, conclusions are summarized in Section IX.

II. RELATED WORK

In our partitioning framework, one part of the WS execution engine runs on a mobile device and the other part runs on a backend node. Partitioning execution engine is conceptually similar to separating an application into different parts that can be executed on different nodes. Static application partitioning is separating application executions at design time. Typical examples of this type of partitioning are client server applications [2]. Dynamic application partitioning is offloading some executions to a nearby surrogate node at run time. This type of partitioning is more challenging than static partitioning. More research in this area can be found in [3], [4], [5] and [6].

Not much work has been published in the area of Web service hosting on mobile devices. One of the earliest works in this area was proposed by IBM. They developed a prototype for a shopper-kiosk application where a shopper can use his wallet services to pay bills [7]. But this effort was specific to a particular scenario and cannot be generalized. Srirama et al. have investigated Web service hosting on mobile devices in detail [8]. It is extended in [9] and [10] to provide a secure communication and an access control for mobile Web service provisioning. The authors presented a distributed semantics-based authorization mechanism for accessing mobile Web services. Pham et al. [11] proposed a light-weight Simple Object Access Protocol (SOAP) server architecture for mobile devices and provided an implementation with Java micro edition (J2ME). The proposed SOAP server is useful in providing access to Web services via Hypertext Transfer Protocol (HTTP). Riva et al. [12] proposed a mobile service framework, which can reflect dynamic context changes in an ad-hoc network. This method monitors the context of a service requester and forwards the requester to a relevant host. However, the proposed framework is designed for an ad-hoc environment and requires the client to monitor the context. A light-weight infrastructure referred to as Micro-Services has been proposed in [13], which is capable of hosting Web services from mobile devices. It is limited to performing simple and short operations and does not consider the intermittent bandwidth characteristic of the wireless medium.

All the efforts discussed earlier are useful only for hosting simple WS applications. To accomplish hosting complex WS applications on mobile devices, we propose a WS application partitioning, which allows some partitions to be executed on a powerful backend node.

III. MOBILE WEB SERVICE PROVIDER ARCHITECTURE

This paper presents a framework for hosting Web services from mobile devices. The proposed WS provider architecture is based on three main modules: transport handler, execution engine, and deployed Web services (Figure 1). Accommodating all parts of the modules in a single node is not feasible for resource constrained devices. Hence, the execution engine can be partitioned to execute the resource consuming partitions on a larger backend node. A backend node is a computing node that executes tasks on request and sends back the results to the provider. The functional details of the individual modules are presented below.

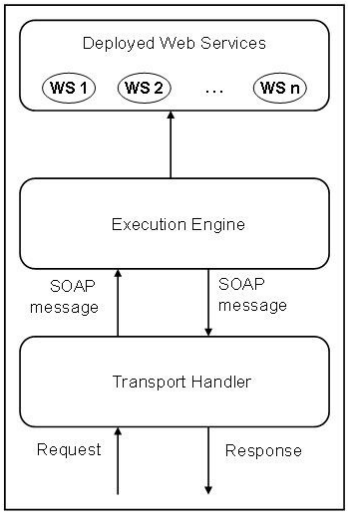

Figure 1. Architecture of the mobile WS provider.

A. Transport Handler

One of the major advantages of SOAP is its independence of the underlying transport mechanism. This allows Web service applications to select the appropriate transport mechanism according to its availability and the requirement of the quality of service. HTTP is used as the transport mechanism in our framework, because it is firewall friendly and most commonly used protocol for exchanging SOAP messages. The transport handler processes the HTTP requests and extracts the SOAP messages to forward them to the execution engine.

B. Execution Engine

This is the core module of the WS provider architecture. This engine is responsible for parsing the incoming request message (SOAP message) and invokes a particular method of a Web service class based on the requested Web service. A mapping between the URIs of the deployed Web services and their class instances is also maintained by the execution engine. This mapping is used to invoke the requested operation of a Web service.

C. Deployed Web Services

This module contains a stack of the deployed Web services and the list of the Web services is maintained in an XML file. A new Web service can be deployed by simply putting its implementation package on the class path in the XML file. A sample of the deployment configuration of a Web service 'SearchArticle' is shown in below.

```
<webservice>
    <uri>http://mobilews.com/searchArticle</uri>
    <class>webservices.SearchArticle</class>
    <operation>searchArticleByDate</operation>
</webservice>
```

IV. Partitioning Mobile Web Service Execution Engine

The key objective of our framework is to minimize the execution load on the service provider mobile device. A WS application can be executed on the mobile device and a few of its tasks can be offloaded to a powerful backend node. The control of the application remains with the service provider and only the tasks requiring more computing resources are offloaded to the backend node. The service designer can decide which set of tasks to execute on the mobile device and which on the backend node. The tasks that do not depend on resources of the mobile device can be executed on the backend node. Such execution engine partitioning also makes the system scalable in terms of the number of concurrent clients.

Our partitioning technique is based on the processing of SOAP messages in such a way that the execution on the resource constrained mobile device is minimized. At this stage, we focus on splitting the execution engine into two partitions only: one for the mobile device that hosts the mobile execution engine, and the other for the backend node that hosts the static execution engine. A WS execution engine performs a series of tasks while invoking a Web service. These tasks may include conforming to WS specifications for address resolution, WS-policy and transaction management, verification of security, and so on. Conforming to each WS specification can be considered as one processing task for the execution engine. For example, verification of XML signature is one task and decrypting a SOAP message can be another task for the execution engine. Likewise, invoking the actual Web service application can be considered as another separate task.

To explain the concept of a distributed execution engine, let us consider a WS invocation that requires clients to send encrypted request with identity of the requestor. The tasks that are performed by the WS execution engine are: verifying client identity (T1), decrypting incoming message (T2), verifying message integrity (T3), invoking other Web services (T4), incorporating value-added services (T5), and signing response message with service provider's certificate (T6). Figure 2(a) shows the sequence of all these tasks performed by a single WS execution engine. Figure 2(b) shows the execution of these tasks by a distributed WS execution engine.

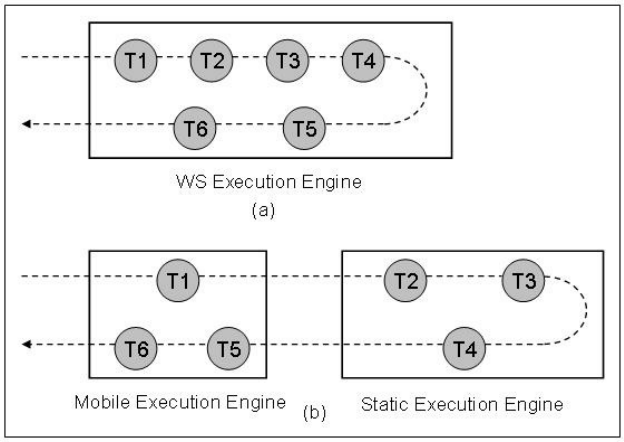

Figure 2. Performing WS execution tasks (a) as a single execution engine, (b) as a distributed execution engine.

As mentioned earlier, the execution engine is divided into two parts: the *mobile execution engine* and the *static execution engine*. The mobile execution engine is designed to be deployed on the mobile device and is responsible to process the tasks that require local resources or need actions of the service provider. The static execution engine is to be deployed on a backend node and is responsible for handling tasks that demand more computing resources. For example, if signing a response message requires a security certificate of the mobile service owner, then such tasks are better not to be placed on a backend node. Considering these facts, division of tasks between the mobile and the static execution engine is presented in Figure 2(b). In this partitioning scheme, the mobile execution engine verifies client identity (T1), incorporates value-added services (T5), and signs the response message (T6). The static execution engine decrypts incoming message (T2), verifies message integrity (T3), and invokes the required Web services (T4).

V. Mobile Web Service Partitioning Schemes

Three different partitioning schemes can be devised for mobile Web services. All schemes are assumed to have an asynchronous mode of communication between WS clients and WS providers. In brief these schemes are presented below.

A. Backend node based scheme

A backend node executes a part of an application of the WS provider and sends the results back to it. In this scheme, a client request is directly received by a resource constrained mobile device. The WS application is executed on the mobile device and few of its components are offloaded to a powerful backend node for improving performance. The interactions of different components in this scheme are shown in Figure 3(a). On receiving a WS request, the mobile device runs some parts of the requested WS application locally and offloads rest of the execution to a backend node. As shown in Figure 3(a), it is the responsibility of the mobile WS provider to collect results from the remote partitions,

aggregate the results and send the final response back to the client.

In this scheme, the mobile WS provider itself has the control of coordinating the different partitions. It can either use a design time or a run time application partitioning strategy. With the run time strategy, a number of run time parameters are used to decide when to offload to a backend node. An advantage of this scheme is that the mobile device controls the different partitions. This makes the services provisioned from mobile WS providers to the clients. From business point of view, this scheme opens up completely new opportunities for small businesses by being mobile Web service providers. Our work proposes a new framework based on this scheme.

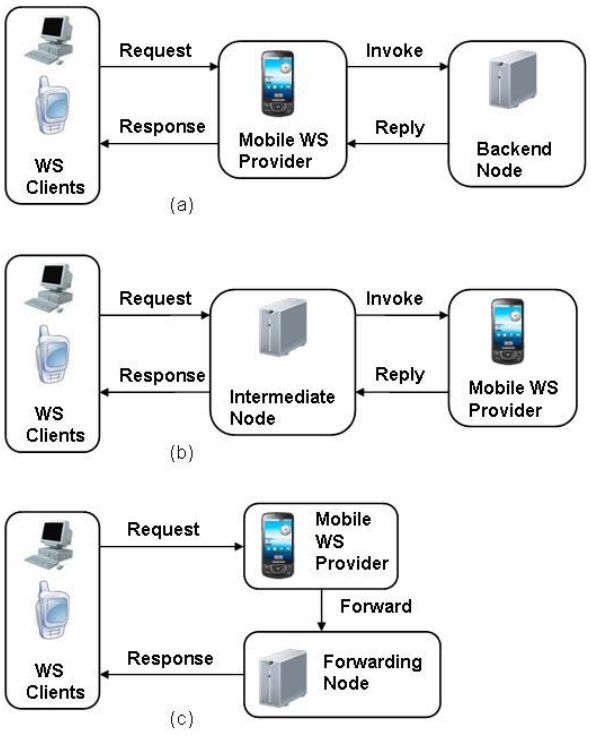

Figure 3. (a) Backend node based scheme, (b) Intermediate node based scheme, (c) Forwarding node based scheme.

B. *Intermediate node based scheme*

An intermediate node intercepts the WS client requests and processes partly before forwarding them to the mobile WS provider. In this scheme, an intermediate node works as a proxy or a surrogate node. To the external world, a service is assumed to be hosted on the intermediate node. A high level overview of this scheme is shown in Figure 3(b). The intermediate node receives the WS requests, executes some tasks locally and assigns other tasks to the mobile WS provider. It is also the responsibility of the intermediate node to create a final response and send it to the client.

The use of an intermediate node as a service proxy ensures availability of the service at all time, comparing to hosting the service directly on a mobile device. This scheme is most suitable for design time application partitioning strategies. In this scheme, the mobile WS provider has less control on the WS application. The WS applications that require more access to the resources of mobile devices or more involvement of device owners are not suitable for this scheme.

C. *Forwarding node based scheme*

The objective of this scheme is to alleviate the drawback of collecting results from backend node and aggregating locally on the mobile WS provider. Some tasks can be executed on the mobile device first and then rest of the execution can be moved to a forwarding node. In this scheme, responsibility of sending the final response to the client is delegated to the forwarding node. A high level overview of the interactions of this scheme is presented in Figure 3(c).

This scheme is suitable for WS applications which require the partition on the mobile device to be executed first. This scheme requires the forwarding node to communicate with clients directly, which may not be feasible in many scenarios (e.g. peer to peer services). Besides, this scheme allows less control to the mobile WS provider and introduces additional binding issues between the forwarding node and the client.

VI. PARTITIONED MOBILE WEB SERVICE FRAMEWORK

In the proposed partitioned mobile Web service framework, the division of tasks to process a SOAP message between the two execution engines (mobile execution engine and static execution engine) is defined at the time of deploying a Web service through a partitioning strategy. The strategy uses an XML schema that defines the XML elements for each execution task. A sample partitioning strategy is as below.

```
<partition>
    <mobile><IDENTITY required=”true”/></mobile>
    <static><WEBSERVICE class-name=” ”/></static>
    <mobile><SIGNATURE required=”true”/></mobile>
</partition>
```

In this partition, verification of a WS client identity and signing a response message are assigned to the mobile execution engine. The task of invoking a Web service is assigned to the static execution engine. The sequence of tasks performed by the execution engines is the same as specified in the partitioning strategy. In this example, there are two ‘<mobile></mobile>’ XML blocks. The ‘<mobile></mobile>’ block that comes after the ‘<static></static>’ block, contains a task to be done on the mobile device after the completion of the static execution engine. Light-weight and open source packages KSOAP [14] and KXML [15] will be used in the implementation of the execution engine partitions. KXML is an XML parser based on pull parsing and is an implementation of XMLPULL parser API [16]. KSOAP is used for processing SOAP messages.

The overall framework is depicted in Figure 4 and the details of the components are discussed below.

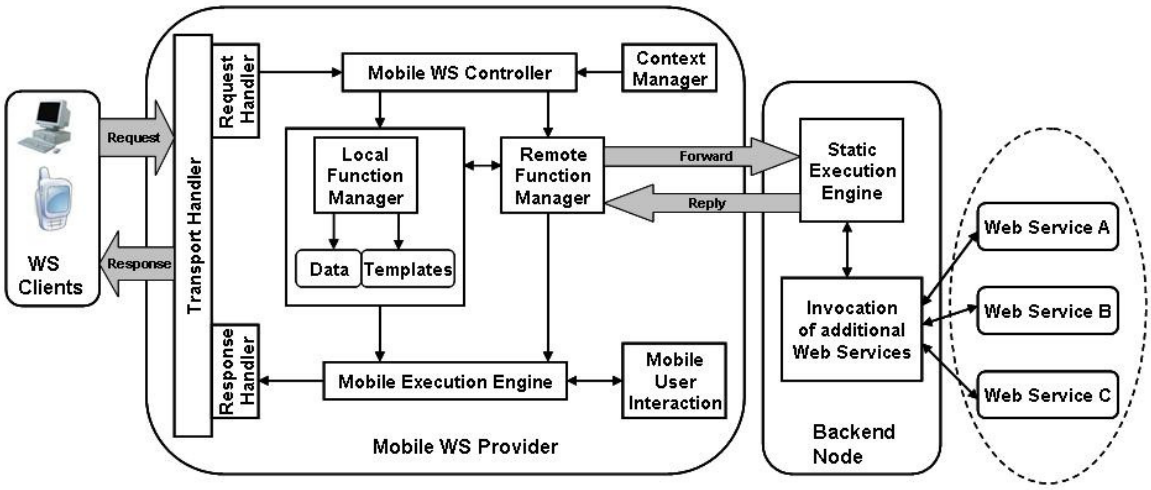

Figure 4. A partitioned mobile Web service framework.

A. *Mobile WS Controller*

A WS client request (SOAP message) is first received by the Transport Handler in any compatible transport mechanism and the Request Handler forwards the message to the Mobile WS Controller. The controller decides whether or not to partition the execution, and which tasks are to be offloaded. The partitioning strategy is configurable because the partitioning depends both on the nature of a Web service and the resource capabilities of the mobile device and the backend node. The Context Manager monitors the availability of resources within the WS provider and periodically provides feedback to the mobile WS controller. Depending on the context of available resources, the controller decides to allocate more tasks to be offloaded to the backend node, or to execute all tasks locally.

B. *Function Managers*

After splitting the execution engine tasks, the controller passes the SOAP message blocks to Local Function Manager and Remote Function Manager accordingly. The local function manager further separates the data and templates from the message blocks to reduce the processing load significantly on the resource constrained mobile device. The data and templates are passed to the Mobile Execution Engine for local WS execution. The remote function manager forwards message blocks to the Static Execution Engine on a backend node. The static execution engine processes the message and performs the tasks assigned to it. The static engine invokes additional Web services as necessary to process the forwarded messages and replies back the results to the remote function manager. These results are then passed to the mobile execution engine to be aggregated with the local execution results. The two execution engines use similar implementation, but they differ in terms of tasks they perform at run time.

C. *Mobile Execution Engine*

This component is responsible for executing the local WS execution tasks within the mobile WS provider. The user or administrator of the mobile device can administrate the mobile execution engine if necessary. For some request processing, it might be necessary to get the administrator input to proceed or to deliver responses. After getting the data and templates separated from the local function manager, the mobile execution engine performs the tasks costing less computing power. Then it aggregates the results with the results obtained from the backend node. At the last step, the transport handler sends the response message (SOAP message) to the original WS client from the mobile host.

D. *Context Manager*

The context manager monitors the available resources on the mobile host and helps the mobile WS controller to allocate tasks accordingly to the execution engines. For example, when the available memory or processing power is low, the controller can allocate more tasks to the backend node, based on the configured partitioning strategy. At the run time, if available resources become low due to any principal operations (e.g. voice calls) of the mobile device, the controller can offload more tasks.

The context manager proposes to partition the WS execution tasks by calculating a context suitability using the following equation,

$$Context\ suitability = \sum \frac{CR_i - RR_i}{CR_i} \times w_i \qquad (1)$$

where, CR_i is the current context value of the i th resource, RR_i is the required context value of the i th resource, and w_i is the weight of the i th resource.

If the calculated context suitability value becomes negative, the mobile WS controller partitions the tasks of the execution between the local function manager and the

remote function manager. As shown in Equation (1), the context manager has the least context information required to execute a Web service, and proposes to partition based on this. For instance, let us consider the case of a mobile host with available 500M memory and 400MHz CPU. For simplicity, we suppose that the weight values w_1 and w_2 are 0.5 and 0.5 respectively. When a Web service executes locally and the required resources are 100M memory and 300MHz CPU, the context suitability will be calculated as,

$$Context\ suitability = \frac{500-100}{500} \times 0.5 + \frac{400-300}{400} \times 0.5$$

$$= 0.4 + 0.125 = 0.525$$

In this case, the context manager will propose to execute the Web service locally within the mobile host. Now let us suppose that the mobile host has 300M memory and 100MHz CPU available. With the same required resources for executing the service, the context suitability in this case is calculated as,

$$Context\ suitability = \frac{300-100}{300} \times 0.5 + \frac{100-300}{100} \times 0.5$$

$$= 0.333 + (-1) = -0.667$$

In this case, as the context suitability is negative, the context manager will propose the mobile WS controller to partition the execution and offload the resource intensive tasks to a remote backend node.

VII. Performance Model

Partitioning mobile WS execution across multiple nodes adds overheads of coordination and communication among the different partitions. Therefore, the expected performance from partitioning a mobile Web service should be improved with the consideration of these overheads. Web service hosting requires a WS execution environment on the mobile device. This execution environment handles receiving a WS request, de-serializing the request message, invoking the requested WS, serializing the results into a response message, and sending the response to the client. If we denote the CPU time required by the execution environment by T_{en} , and the CPU time required by the WS application itself by T_{app} , then the overall response time T_{mws} is the sum of T_{en} and T_{app} (network delays are not considered).

$$T_{mws} \cong T_{en} + T_{app} \quad (2)$$

T_{en} includes the CPU time spent on sending and receiving SOAP messages and executing WS protocols. If the execution engine performs total n number of tasks, then total CPU time required by the WS application is:

$$T_{app} = \sum_{i=1}^{n} T_{ti} \quad (3)$$

Equation (3) only considers CPU time while running all the tasks sequentially on a single mobile device. Let us assume that we group the tasks into two sets to be executed in two partitions. One set of tasks is executed on the mobile device and the other set of tasks is offloaded to a static remote node. Now equation (3) can be written as,

$$T_{app} = \sum_{i=1}^{m} T_{ti} + \sum_{j=m+1}^{n} T'_{tj} \quad (4)$$

where, $i = 1, 2, \dots, m$ are the tasks executed locally on the mobile device and $j = m+1,\ m+2, \dots n$ are the tasks executed remotely.

There are two additional overheads that will occur if the WS application is partitioned across multiple nodes. One overhead arises from the coordination of different partitions, and the other overhead arises from the transfer of data between the partitions. If we denote these two overheads by Δ_{cd} and Δ_{tn} respectively, then equation (4) becomes,

$$T_{app} = \sum_{i=1}^{m} T_{ti} + \sum_{j=m+1}^{n} T'_{tj} + \Delta_{cd} + \Delta_{tn} \quad (5)$$

Therefore, the overall response time of a mobile WS invocation can be obtained by combining equations (2) and (5),

$$T_{mws} \cong T_{en} + \sum_{i=1}^{m} T_{ti} + \sum_{j=m+1}^{n} T'_{tj} + \Delta_{cd} + \Delta_{tn} \quad (6)$$

Equation (6) gives an estimate of the time required to invoke a mobile WS that is partitioned across multiple nodes. In this estimate, the executions on the mobile device and on the remote node are assumed to be not concurrent.

VIII. Performance Results

To investigate performance of the partitioned execution engine for mobile hosts, a test prototype has been developed to compare the end to end response time for different number of concurrent clients. The sample Web service 'SearchArticle' performs the operation 'SearchArticleByDate' and provides a list of articles written on a particular date by a journalist. The Web service extracts the input parameter 'date' from the incoming SOAP message and returns the list of articles as a response. The average response time for each input value is measured by sending 10 requests to the service provider and then taking the average.

The WS clients are run on a Toshiba Satellite A100 laptop equipped with a RAM of 1 GB and 1.83 GHz Intel processor. The backend static node is a desktop computer equipped with a CPU speed of 3.0 GHz and a RAM of 1 GB. Both machines operate with Windows XP professional operating system. The mobile WS provider is deployed on a Nokia E63 smart phone equipped with 369 MHz ARM11 processor and a RAM of 71 MB, running on Symbian OS v9.2 operating system. The Java ME technology available on the device is, JSR 172 J2ME Web Services Specification. The light weight WS execution engine is based on J2ME and deployed on the smart phone. The client machine, the smart phone and the backend node are equipped with wireless interfaces using IEEE 802.11g standard, and they communicate using a wireless local area network.

In this preliminary test, we partitioned the WS execution engine at design time. The task of client's authentication checking is partitioned to be offloaded at the backend node when a client sends a request SOAP message. After verifying the signature, the backend node sends the message to the WS provider. The mobile WS provider does all the tasks of Web service invocation for the authorized clients and sends back the results to them. In the second test, the same client requests were executed without partitioning the execution engine. The client's authentication checking is done within the mobile execution engine. Each client operates cyclically and sends one request at a time. The system is stressed by increasing the number of concurrent WS clients. Table 1 shows the average response times obtained from these observations. Plotting the values from Table 1, we obtain Figure 5.

TABLE I. COMPARISON OF RESPONSE TIMES WITH AND WITHOUT PARTITIONING

Concurrent clients	Average response time when partitioned (ms)	Average response time when not partitioned (ms)
1	1186.82	1766.53
3	2466.35	4232.21
6	3742.41	7542.37
9	5013.26	13708.82
12	8651.44	18857.34

As we can see for both of the cases, demand for resources increases with an increase of concurrent clients and the mean response time increases. Figure 5 shows that the mean response time for the partitioned execution engine is significantly lower than the mean response time with no partitioning. The result of distributing the execution engine from the mobile host is very promising. For a single client the improvement in the response time is 32% and for 12 concurrent clients the improvement is 54%.

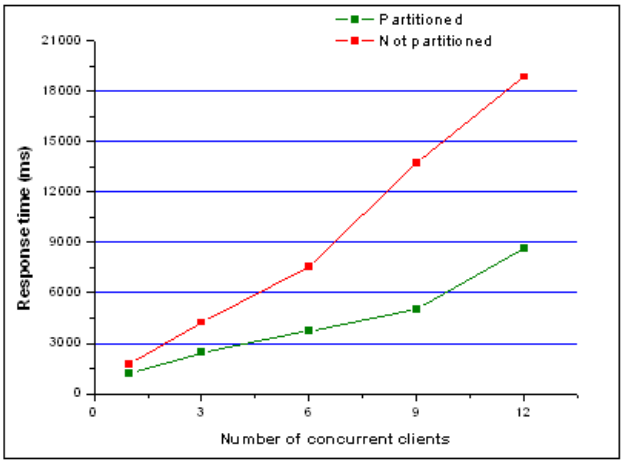

Figure 5. Impact of execution engine partitioning on performance.

IX. CONCLUSIONS

This paper presents a framework for provisioning Web services from resource constrained mobile devices promising to create mobile cloud in future. The WS execution engine is devised as the kernel of the proposed framework. The architecture of the WS execution engine has been provided with the details of each component. A prototype has been tested for analyzing the performance improvement by the implementation of the proposed framework. As we can imagine from the observation, partitioning of different computing tasks can significantly improve the performance of the services hosted on mobile devices.

The proposed framework is based on the backend node scheme. In the future, efforts will be given to extend the framework by including the intermediate node and the forwarding node schemes in different business scenarios. In this paper, we considered only two partitions for the WS execution engine. Further work is required to investigate the possibility and feasibility of more than two partitions. Meanwhile, we only considered partitions that are executed sequentially, and we plan to extend the framework to be able to support concurrent execution of the partitions.

REFERENCES

[1] J. Wortmann, H. J. Pels and H. Jagdev, "Collaborative Systems for Production Management", 2003, Springer.

[2] A. Tanenbaum and M. Steen, "Distributed Systems: Principles and Paradigms", 1st edition, Jan 2002, Prentice Hall.

[3] G. Hunt and M. Scott, "The Coign Automatic Distributed Partitioning System", *In the Proceedings of the 3rd Symposium on Operating System Design and Implementation (OSDI'99)*, New Orleans, LA, U.S.A., Feb 1999, pp. 187-200.

[4] V. Jamwal and S. Iyer, "Automated Refactoring of Objects for Application Partitioning", *In the Proceedings of the 12th Asia-Pacific Software Engineering Conference* (APSEC'05), Taipei, Taiwan, Dec 2005, pp 671-678.

[5] D. Chandra, C. Fensch, W. Hong, L. Wang, E. Yardımci and M. Franz, "Code Generation At The Proxy: An Infrastructure-Based Approach To Ubiquitous Mobile Code", *In the Proceedings of the 5th*

ECOOP Workshop on Object-Orientation and Operating Systems (ECOOPOOOSWS02), Malaga, Spain, Jun 2002.

[6] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. Giuli and X. Gu, "Towards a Distributed Platform for Resource-Constrained Devices", *In the Proceedings of International Conference on Distributed Computing Systems (ICDCS02)*, Vienna, Austria, Jul 2002, pp. 43-56.

[7] S. McFaddin, C. Narayanaswami and M. Raghunath, "Web Services on Mobile Devices – Implementation and Experience", *In the Proceedings of the 5th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA'03)*, Monterey, CA, U.S.A., Oct 2003, pp. 100-109.

[8] S. Srirama, M. Jarke and W.Prinz, "Mobile Web Service Provisioning", *In the Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, Guadeloupe, French Caribbean, Feb 2006, pp. 120-128.

[9] S. Srirama and A. Naumenko, "Secure Communication and Access Control for Mobile Web Service Provisioning", *In the Proceedings of International Conference on Security of Information and Networks (SIN 2007)*, May 2007.

[10] S. Srirama, M. Jarke and W. Prinz, "A performance evaluation of mobile web services security", *In the Proceedings of 3rd International Conference on Web Information Systems and Technologies (WEBIST 2007),* Barcelona, Spain, Mar 2007.

[11] L. Pham and G. Gehlen, "Realization and Performance Analysis of a SOAP Server for Mobile Devices," *In the Proceedings of 11th European Wireless Conference, 2005*, vol. 2, Nicosia, Cyprus, Apr 2005, pp. 20-27.

[12] O. Riva, T. Nadeem, C. Borcea, and L. Iftode, "Mobile Services: Context-Aware Service Migration in Ad Hoc Networks", *IEEE Transaction on Mobile Computing,* IEEE Educational Activities Department, (to appear).

[13] Pratistha, D., Nicoloudis, N., Cuce, S., "A Micro-Services Framework on Mobile Devices", *In the Proceedings of International Conference on Web Services*, 2003, Nevada, USA.

[14] KSOAP2 project, *available at* http://ksoap2.*sourceforge.net/*, [Accessed Oct 20, 2009].

[15] KXML2 project, *available at* http://kxml.sourceforge.net/kxml2/, [Accessed Oct 20, 2009].

[16] XML PULL Parser API, *available at http://www.xmlpull.org/.* [Accessed: August 12, 2009].