PERFORMANCE ANALYSIS OF WEB SERVICE TECHNOLOGY FOR BUILDING A MULTICLOUD MIDDLEWARE FOR MOBILE DEVICES

Nithya RAVI
Research Scholar, Anna University, Chennai, India
nithyaravi01@gmail.com

Mala THANGARATHINAM
Associate Professor, Anna University, Chennai, India
malanehru@annauniv.edu

Abstract: *The presence of numerous cloud service offerings has lead to working with services and APIs of different cloud vendors. These APIs are not interoperable and the data stored into one cloud is non-transferrable to other clouds. In addition, today's cloud users are mobile devices and consuming a cloud service onto mobile device poses another set of risks. One way to handle this problem is to devise a generic middleware with a unified cloud API set to handle the API heterogeneity at the cloud end and make it suitable for use with mobile devices. Thus, this paper attempts to identify the best suitable API technology for the middleware by doing an extensive comparative study of the existing protocols.*

Key words: *mobile devices, vendor lock-in, cloud services, API, SOAP, REST.*

1. Introduction

The buzz around cloud computing has reached a fever pitch. It has emerged as a mainstream phenomenon where any type of IT resources are delivered as a service over the network to novice users who need not have knowledge of, expertise in, or control over the cloud infrastructure that supports them. With the enormous offering that the cloud provides, that market today has an equal enormous offering of different cloud providers. Too often, people end up in a lock-in situation because important decisions were made casually or without any upfront thought. This can occur easily with the cloud because services are so easy to sign up for and use. At some point later, companies are at the mercy of cloud providers whenever rates increase or terms and conditions change. Every cloud vendor exposes his business functionality by means of his vendor specific APIs. This creates vendor lock-in, and a user is not free to use the services of multiple providers simultaneously without investing in upfront cost, time and energy. This limitation of cloud interoperability hampers cloud adoption and does not drive integration of accessing hybrid cloud services.

Fortunately, some open source communities and interested forums have come up with some approaches to handle this problem. On the same lines, few open source libraries have also come into existence to manage the interoperability and portability issue. But, however, efforts have not been introduced to service mobile clients. Mobile devices are limited in terms of computational power when compared to their PC counterparts. Being locked-in by a cloud provider is not what end customers plan to experience. The paper [1] also mentions that developing native applications individually to suit heterogeneous providers is an arduous and expensive task. Hence, there arises a need to write a single code that works with more than one cloud provider simultaneously, regardless of the differences in the API set.

A majority of cloud service APIs are based on SOAP and REST. Therefore, it is essential to develop a single point of contact for all providers. Hence, a generic middleware with a single stack of API set is proposed, and using this single API call, the client can access or leverage the benefits of global portability among cloud services. This saves considerable amount of time, and reduces the complexity, as there is just one version of the client application to be developed for invoking multiple services. To address the low computational resources needs of mobile devices, a performance analysis of the existing cloud service APIs (SOAP and REST) is carried out to find out the best suitable type for access from mobile devices. The results have recommended REST style of APIs as more efficient and consume fewer resources when compared to that of SOAP. With this basis, it is identified that the generic Multicloud middleware can be developed with RESTful service APIs for servicing mobile clients.

The rest of the paper is organized as follows: Section 2 discusses about the challenges of cloud interoperability with some insight into the existing approaches. In Section 3, the emergence of Mobile Cloud computing is discussed and Section 4 presents the vision of Multicloud middleware for Mobile devices. Section 5 gives a brief overview of the Web service technologies with current state-of-the-art. Performance analysis of SOAP and REST are presented in section 6. Section 7 discusses about the API heterogeneity and finally section 8 concludes the paper and points out future research directions.

2. Cloud Interoperability Overview

2.1 Need for Cloud Interoperability

Cloud computing has evolved as a disruptive technology and picked up speed in 2008 and 2009 with the presence of many vendors in the cloud computing space. As the cloud market continues to grow, the number of commercially available cloud-based service offerings is also increasing at a higher rate. Cloud services are offered in three styles: SaaS, PaaS, and IaaS. Each cloud service provider exposes such services through a specific set of APIs and a user interacts with the cloud using that API leading to something called the Cloud API propagation [2]. Cloud providers belonging to a common category, (say IaaS providers), may provide the same functionality but differ in their APIs list. The APIs either use different names / URIs, or use different protocols (such as SOAP or REST) to invoke them. Applications that are developed for one cloud may not be compatible with one another. Therefore, applications must suffer some changes when it is necessary to move from one cloud to another. From an end user's perspective, this becomes a serious and tiring issue when trying to access hybrid cloud services to achieve some functionality. This will not reflect good programming practice of developing several versions of a single application for specific vendors. This eventually kills the purpose of using cloud by limiting cloud choice because of vendor lock-in, portability, and the ability to use the cloud services provided by multiple vendors. Therefore, the situation demands for a *"Write once, Run anywhere"* paradigm to handle this cloud interoperability issue. Fortunately, some interoperability approaches have been initiated and the same has been discussed below.

2.2 Cloud Interoperability approaches

According to the paper [3], there are two groups called the SDO and SSO. Standards developing organization (SDO) are technically involved in developing and publishing standards for cloud computing, while Scientific Consortia and Standards-setting Organization (SSO) is involved in promoting the adoption of emerging technologies, without the intention of developing their own standards. Table 1 begins with a gist of some of the prominent standard initiatives[4-14] and open working groups[15-19].

Table 1
Standards and Open Working Groups

Standards	
ETSI (2018)	Interoperable solutions for IaaS
Messina (2014)	Two working groups P2301 and P2302 working on cloud interoperability. P2301 focuses on cloud portability, while P2302 focuses on cloud-to-cloud interoperability and federation.
ITU-T (2018)	Works on next generation networks in conjunction with cloud computing
NIST (2018)	Covers cloud architectures, security and deployment strategies
Karmakar & Pilz, 2012	Focuses on security challenges in the cloud, consumer-provider collaboration in maximizing quality of service, and specification for enhancing the portability of cloud applications and enabling the interoperable description of application and infrastructure cloud services
CDMI (2018)	Focus on data management in the cloud
TMForum (2018)	Encourages and stimulates the growth of an open marketplace for cloud services.
Aradhana et al., 2011	Study on the risks & benefits involved with cloud portability
OCCI (2018)	Management API for deployment, autonomic scaling, and monitoring of IaaS resources
OVF (2018)	Standard portable platform for representing virtual machines
DMTF (2012)	Focus on multi vendor interoperability for enterprise in system, tools and applications.
Open Working Groups	
Open Group (2018)	Collaborates on standard models & frameworks to eliminate vendor lock-in for enterprises
CSCC (2018)	Guidance on critical elements to consider when negotiating an SLA
OCC (2018)	Benchmarks for dealing with large data clouds
Motohashi (2011)	Standardization of network protocols and interfaces namely, the Intercloud protocol and the Cloud Resource Data model.
Cohen (2018)	Unifies various cloud APIs and abstracts it under a single, open and standardized interface. Holds a specification (details for integration with other management models) and schema (model descriptions). Uses Semantic Web & OWL. Uses the Resource Description

	Framework (RDF) to describe a cloud data model.

2.3 Multicloud Libraries

Few open source communities have come up with some of the open source libraries that leverage the cloud interoperability and portability problem. Some of those libraries [20-26] are discussed in Table 2 below.

Table 2
Multi-Cloud Libraries

Library	Description
Jets3t (2018)	Java toolkit for accessing S3 and Google Storage services for storage.
Jclouds (2018)	Compatible with OpenStack for the compute and Blobstore feature. Not compatible with any mobile platform.
Darryl (2011)	Multi-cloud API that supports Eucalyptus and Rackspace with XML/JSON instructions. No support for mobile platform.
ZCloud API (2018)	Supports storage from S3, Sun Cloud service, Eucalyptus Walrus, Mezeo, as well as private storage clouds. No support for mobile platforms.
Joe (2018)	Rest API for compute and storage purposes. A top-level Apache project that works from EC2 to RedHat Enterprise.
Libcloud (2018)	Python library for multi-cloud management. Supports LoadBalancers as a service, DNS as a service, Rackspace and AWS for compute cloud, S3 for storage and others.
Typica API (2018)	Supports compute service of EC2 and Eucalyptus. No support for mobile platforms

Most of these libraries are at its infancy stage, and are evolving with better features everyday. Cloud computing has started reaching its acceptance in the mobile domain and hence, the real benefits of cloud attains completeness in satisfying the needs of a mobile device access to hybrid cloud services offered by multiple cloud providers.

3. Transforming from Cloud to Mobile Cloud Computing

Mobile devices are invading our lives, and its use is growing at an unprecedented rate. Due to the convenience it offers, people enjoy being associated with mobile for activities such as mobile banking, BYOD (Bring Your Own Device), social networking or even online shopping [27]. With this shift in consumers' expectations, newer mobile applications are being developing rapidly to satisfy the needs of a wider range of audience. However, despite the rapid advancements and developments of smartphones, they are intrinsically limited by several factors such as computational speed, storage capacity, processing power, etc. when compared to PCs. On an average, the processing power of a mobile device is 3 times lesser than that of a personal computer, RAM memory is 4 times lesser, storage is 30 times lesser, and the display size is 5 times smaller.

Smartphones lack the luxury of performing very high compute intensive tasks due to the unavailability of required computing power and limited battery lifetime**.** And as such, applications that run on mobile devices are not business class applications. To help smartphones overcome these challenges, smartphones have been backboned with the power of Cloud computing, bringing about a new research domain called the Mobile Cloud Computing (MCC). In MCC, mobile devices delegate all data processing, storage and other intensive operations to the clouds. According to Shah [28], mobile applications have already started leveraging the cloud and have become a necessity to solve complex problems in science and engineering fields**.** Some of them are the Apple iCloud and the Amazon Silk Browser. Apple's iCloud stores customers' photos, videos, apps, calendars, etc. on Amazon EC2 and Windows Azure, and synchronizes them with all the iOS devices. Amazon Silk Browser is a cloud-accelerated "split browser". On a web page request, the browser dynamically decides which sub-components of the browser run on the mobile and which has to be delegated to Amazon EC2 depending on the page complexity and network conditions. Thus, today's market highly utilizes the immense capacity of cloud to bridge the limitations of mobile devices.

4. MultiCloud Middleware for Mobile Devices

Every cloud vendor who comes up with a new set of proprietary APIs everyday is generally observed to be slow in providing support for mobile devices. A cloud service is basically a Web service. Web services are software functions that are exposed over the Web to perform some task. Consuming Web services from mobile devices is certainly different from that of PCs. The APIs available for direct deployment on a PC are not suitable for mobile deployment, due to the integration issues with the compiler and reference to other external libraries required by the API. Some run time issues emerge due to the platform restrictions and compiler inabilities. Also, a mobile device is developed

in a variety of OS such as Android, Apple iOS, Symbian, etc. Each platforms component differs from one another in its architecture and implementation, and this can also greatly reduce the opportunity of accessing cloud from mobile phones. However, some cloud providers have initiated to provide its cloud service support by offering a separate SDK for different mobile devices such as Android and Apple. Some of those providers are Amazon, Apple iCloud, Google Drive, Dropbox, Microsoft SkyDrive, SugarSync, Box, etc. But here again, there is vendor lock-in with each service, and a user is forced to develop different mobile applications for different cloud providers. Hence, in an effort to ease the development at the client side, and with the intention to invoke hybrid cloud services, it is desirable to develop a generic middleware to handle the platform independence feature of the mobile device and solve the interoperability problems of different clouds. Cloud integration and mobile performance optimization will be the key drivers of the middleware. The Multi-Cloud Middleware for Mobile devices should include features like:

- Unified Cloud API
- Protocol transformation
- Result Optimization
- Data Integration
- Advanced SLA Controls
- Billing and Monetization
- Mobile performance optimization

5. Cloud Service Invocation from Mobile Devices

Any service exposed over the cloud is a Cloud service. Cloud services expose their behavior through APIs. An API is a set of instructions for interacting with an application via a programming language. According to Programmable Web [29], currently there are 20,613 APIs recorded in its registry. Many service providers use this public registry to advertise their APIs. Cloud APIs are of two Web service technology types, namely the SOAP (Simple Object Access Protocol) and the REST (REpresentational State Transfer). SOAP and REST have been brought forward to implement RPC calls over the Web. Both define standards for application integration, but differ in their methodology and effectiveness. For instance, SOAP follows a Service Oriented Architecture style with the request and response formats in XML, however REST follows the Resource Oriented Architecture style. The request format is in XML or JSON, and the response format could be in XML, JSON, HTML, RAW, PDF, JPEG, CSS, RSS, Serialized PHP and CSV. The CRUD (Create, Read, Update, Delete) operations performed on SOAP are GET and POST, whereas the CRUD operations on REST protocol are GET, PUT, POST, and DELETE.

To handle the vendor-lock in limitation, a middleware is planned to provide a unified API abstracting all other cloud APIs. The end user communicates with only the middleware API and the API in turn communicates with the respective cloud API. Also, the APIs of the middleware need to be offered in a manner that is optimized for access from mobile devices. Hence, it necessitates developing a common API set based on a specific technology, either being the SOAP or REST. Thus, the objective of our work is to find out the technology basis of developing the middleware. A thorough examination of SOAP and REST is carried out to explore the best suitable Web service technology to use for developing the cloud APIs and extending its support to mobile devices. With this knowledge in hand, let us proceed to understand the characteristics of the two technologies SOAP and REST in detail.

5.1 Simple Object Access Protocol

Simple Object Access Protocol was introduced by the World Wide Web Consortium (W3C) to integrate applications of diverse platform and language. It is a protocol specification for Web service communication. A client communicates with a Web service by exchanging SOAP messages. For the purpose of interoperability, SOAP messages use XML as the data format along with standard XML types. SOAP works mainly on top of HTTP, and as well works with other transport protocols such as SMTP, TCP, etc. SOAP communications revolve around GET and POST operations over HTTP. Since SOAP messages are text based and self describing, they can easily convey information between services of heterogeneous computing environments. SOAP also handles security and reliability of the messages involved. Figure 1(a) and Figure 1(b) depicts the skeleton of SOAP request and response messages.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header/>
   <soapenv:Body/>
</soapenv:Envelope>
```

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
   <ns:sayHelloResponse xmlns:ns="http://hello">
      <ns:return>Hello World!</ns:return>
   </ns:sayHelloResponse>
</soapenv:Body>
   </soapenv:Envelope>
```

Fig. 1(a), 1(b). SOAP Request & Response Messages

5.2 Representational State Transfer

Representational State Transfer (REST) is an architectural style introduced by Roy Fielding for

building large-scale distributed hypermedia systems. REST mainly focuses on the data of an element and its corresponding state. It is perceived to be simple because it leverages the existing Web standards. Any data or information such as a document, image, or temporal service, is termed as resources and they are the key aspect of REST. Each resource is identified by a distinguished URI (Uniform Resource Identifier). The resources are acted upon using the four HTTP operations: GET, POST, PUT, and DELETE. While SOAP based Web services expose service APIs through WSDL document, RESTful style of Web services expose services through a Web browser using the four HTTP methods. GET is used to retrieve resource, POST to create new resource, PUT to update or modify resource and DELETE to delete the resource. REST classifies any kind of Web service operations into these main four "verbs". Since, everything revolves around resources; REST is a resource-oriented technology while SOAP is a service-oriented technology. REST is a stateless communication protocol in the sense that the client should contain all the necessary information for a server to understand the request and should not leave any information by assuming some stored context at the server. This feature can be helpful when there is a failure in the communication and also improves scalability at the server end as it need not maintain state of all clients and keep track of them. But yet to improve the efficiency in responding to requests, cache constraints are introduced explicitly by labeling as cacheable. Figure 2(a) and Figure 2(b) depicts the skeleton of REST request and response messages.

```
http://localhost:8000/restful/resources/helloworld
```

```
<data contentType="text/plain" contentLength="25">

<![CDATA[This is a Hello from REST]]>

</data>
```

Fig. 2(a), 2(b). REST Request & Response Messages

5.3 State-of-the-art

Several research activities have been carried out for estimating the performance and efficiency of Web services. With Web services extending its support to lower end devices such as mobile phones, care is taken to offer mobile friendly Web services such that it can meet the hardware limitations of mobile devices. Hamad et al. [30] have evaluated the performance of SOAP and RESTful Web services for mobile devices. The message size and processing time involved in handling an array of string concatenation and floating number addition are found and the results recommended REST style of Web services as the apt type for mobile devices. Mobile devices can be Web service consumers, at the same time; they can provision Web services themselves. In the latter case, a mobile device should have the necessary capacity to accommodate Web servers to deploy services on them. Recently, certain servers have been developed specifically for mobile devices such as RhoSync, iFMW, FineWS and I-Jetty. Aijaz et al. [31] have compared the REST based Mobile Web server provisioning against the SOAP architecture in terms of HTTP payload. In addition, the synchronous and asynchronous type of interaction for a RESTful Mobile Web service is discussed. The results indicated that a synchronous and asynchronous Mobile Web service communication consumes ≈96% and ≈75% reduced payload when compared to SOAP. The paper [32] is a more elaborate extension of [31] by discussing the architecture of a REST-interfaced Mobile Web server with emphasis on synchronous interaction strategy for short-lived Mobile Web service. The results showed promising signs of optimized processing performance, lesser latency and reduced payload when compared to SOAP. In the same range of idea, the authors Mizouni et al. [33] have evaluated the QoS of REST and SOAP Web services on the basis of response time, availability, throughput and scalability. Results indicate RESTful Web services to be superior when compared to SOAP for mobile devices.

A Web service needs to be made available in a mobile environment by handling un-interrupted connectivity when the mobile device moves from one location to another. Mobility becomes a key feature as far as a mobile device is considered. AlShahwan & Moessner, [34] investigate into mechanisms of providing un-interrupted Web services from resource constrained mobile devices such as J2ME. Preliminary work is carried on to find the best type of Web services (SOAP or REST) offered from mobile devices and some intermediate components are introduced to retain continuous network connectivity. The results revealed that RESTful based Mobile Web Services are more scalable and reliable when compared to SOAP based Mobile Web Services. A mobile phone has limited luxury in terms of computation, storage and battery lifetime. Hence, it may not be able to efficiently execute a large service on its own. Ideally, a better idea would be to partially offload its execution to other mobile phones, to let the mobile device execute its huge compute-intensive service. AlShahwan et al. [35] extends the idea of paper [34] by facilitating the offloading of services and service fragments to other mobile devices. The REST implementation outperformed SOAP by processing cycles, reduced delay and lesser message size for the distributed service execution scenario.

The objective of our work is to further explore the feasibility of providing RESTful Web services from mobile devices and to compare its performance against SOAP in terms of message size, processing time, data formats and throughput to suggest the base for developing the middleware.

6. Comparison Levels

We have measured the performance of SOAP and REST on various parameters such as the size of the SOAP and REST request / response messages, time taken to process a REST / SOAP service, and the scalability or the load a service can handle. Three types of case study have been considered for the evaluation of the above parameters. They are:

- ✓ **Hello Service** – No input, but the response is a simple Hello World message.
- ✓ **Arithmetic Service** – Given two input parameters, the service responds with the addition of the two numbers.
- ✓ **Video Service** – The user demands for a video, and the video is transferred as a binary attachment.

The SOAP and REST Web services have been implemented using the AXIS2 (Apache eXtensible Interaction System) framework and Jersey (JSR-311 reference implementation) library respectively. The analysis has been carried out on a Web server with a PC client. The next section compares the message size of the Web services for three scenarios.

6.1 Message Size

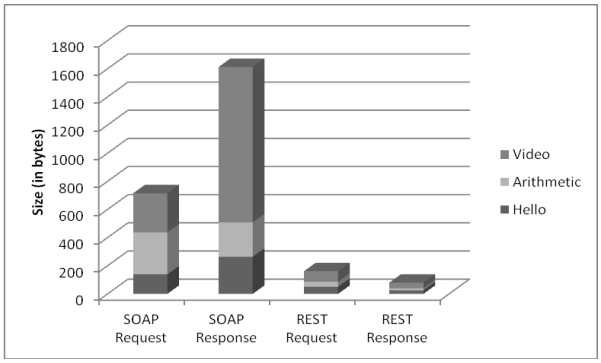


Fig. 3. Comparison of message size of SOAP and REST request / response messages

Figure 3 depicts a huge marginal difference in the message sizes of the two protocols. The size of the messages have been evaluated from only the body of the HTTP packet for both the cases. The HTTP header details are not considered here. A SOAP envelope contains extra parameters in addition to the actual response data. This creates an unnecessary increase of payload size, whereas REST derives only the actual payload. Also, SOAP uses XML which is heavy weight when compared to REST which uses JSON as the data format. In addition to JSON, REST also uses XML as the data format.

6.2 Processing Time

The processing time taken to obtain the request and provide the desired response is shown in Figure 4 below.

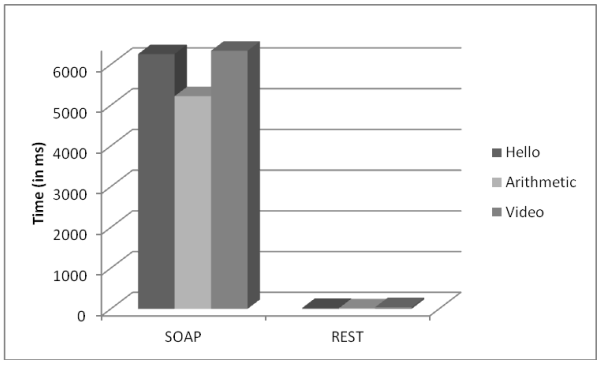


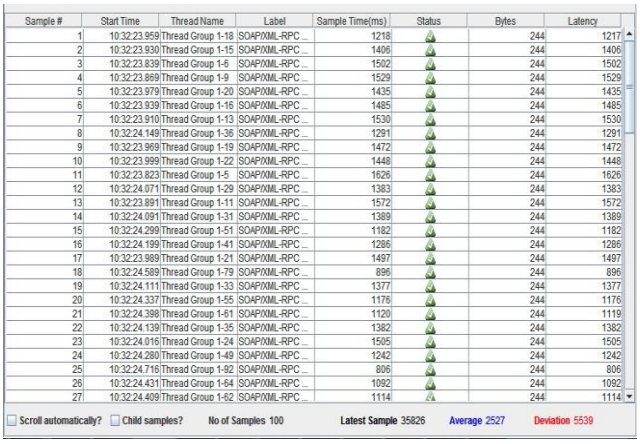
Fig. 4. Time taken to process REST and SOAP service

REST responds far quickly when compared to SOAP style of services. The average processing time involved between the request and response is inclusive of connection setup, network latency, jitter, service processing time. Due to the overhead nature of verbose XML, SOAP incurs huge processing overhead when compared to REST. JSON comparatively occupies lesser time to respond than XML.

6.3 Throughput

Apache Jmeter [36] is used to test the performance and scalability by configuring test plans and perform load testing for Arithmetic SOAP and REST services. The scalability feature is tested on three parameters. They are:

- ✓ Number of users sending the requests
- ✓ Ramp-Up period determining how often the user request arrives. By default, it is considered that all requests arrive at the same time.
- ✓ Loop count deciding the number of requests, a user sends. In our case, 100 users are considered to send the request at the same time with each user sending only one request.



Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Latency
1	10:32:23.959	Thread Group 1-18	SOAP/XML-RPC ...	1218		244	1217
2	10:32:23.930	Thread Group 1-15	SOAP/XML-RPC ...	1406		244	1406
3	10:32:23.839	Thread Group 1-6	SOAP/XML-RPC ...	1502		244	1502
4	10:32:23.869	Thread Group 1-9	SOAP/XML-RPC ...	1529		244	1529
5	10:32:23.979	Thread Group 1-20	SOAP/XML-RPC ...	1435		244	1435
6	10:32:23.939	Thread Group 1-16	SOAP/XML-RPC ...	1485		244	1485
7	10:32:23.910	Thread Group 1-13	SOAP/XML-RPC ...	1530		244	1530
8	10:32:24.149	Thread Group 1-36	SOAP/XML-RPC ...	1291		244	1291
9	10:32:23.969	Thread Group 1-19	SOAP/XML-RPC ...	1472		244	1472
10	10:32:23.999	Thread Group 1-22	SOAP/XML-RPC ...	1448		244	1448
11	10:32:23.823	Thread Group 1-5	SOAP/XML-RPC ...	1626		244	1626
12	10:32:24.071	Thread Group 1-29	SOAP/XML-RPC ...	1383		244	1383
13	10:32:23.891	Thread Group 1-11	SOAP/XML-RPC ...	1572		244	1572
14	10:32:24.091	Thread Group 1-31	SOAP/XML-RPC ...	1389		244	1389
15	10:32:24.299	Thread Group 1-51	SOAP/XML-RPC ...	1182		244	1182
16	10:32:24.199	Thread Group 1-41	SOAP/XML-RPC ...	1286		244	1286
17	10:32:23.989	Thread Group 1-21	SOAP/XML-RPC ...	1497		244	1497
18	10:32:24.589	Thread Group 1-79	SOAP/XML-RPC ...	896		244	896
19	10:32:24.111	Thread Group 1-33	SOAP/XML-RPC ...	1377		244	1377
20	10:32:24.337	Thread Group 1-55	SOAP/XML-RPC ...	1176		244	1176
21	10:32:24.398	Thread Group 1-61	SOAP/XML-RPC ...	1120		244	1119
22	10:32:24.139	Thread Group 1-35	SOAP/XML-RPC ...	1382		244	1382
23	10:32:24.016	Thread Group 1-24	SOAP/XML-RPC ...	1505		244	1505
24	10:32:24.280	Thread Group 1-49	SOAP/XML-RPC ...	1242		244	1242
25	10:32:24.716	Thread Group 1-92	SOAP/XML-RPC ...	806		244	806
26	10:32:24.431	Thread Group 1-64	SOAP/XML-RPC ...	1092		244	1092
27	10:32:24.409	Thread Group 1-62	SOAP/XML-RPC ...	1114		244	1114

Scroll automatically? Child samples? No of Samples 100 Latest Sample 35826 Average 2527 Deviation 5539

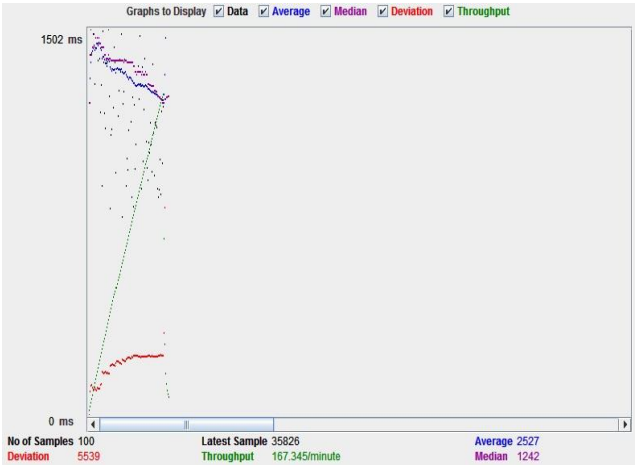
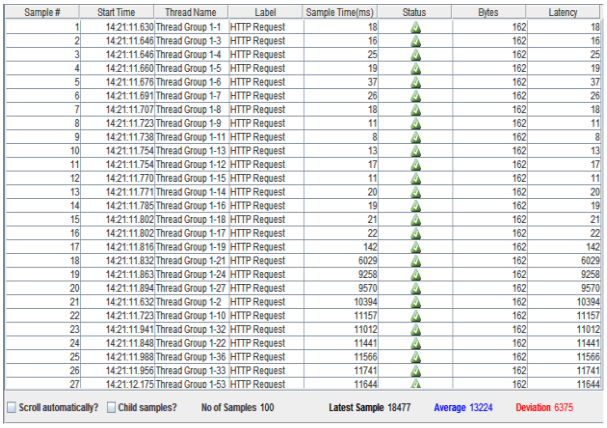


Fig. 5(a), 5(b). Scalability test for Arithmetic SOAP service

The above figures Figures 5(a) and Figure 5 (b) shows the average parsing time and their standard deviations for the SOAP messages over 100 independent samples. The throughput i.e., the number of requests that the service can handle is identified to be approximately 167 requests per minute. Now, let us compare it against the REST service in Figure 6 (a) and 6 (b).



Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Latency
1	14:21:11.630	Thread Group 1-1	HTTP Request	18		162	18
2	14:21:11.646	Thread Group 1-3	HTTP Request	16		162	16
3	14:21:11.646	Thread Group 1-4	HTTP Request	25		162	25
4	14:21:11.660	Thread Group 1-5	HTTP Request	19		162	19
5	14:21:11.676	Thread Group 1-6	HTTP Request	37		162	37
6	14:21:11.691	Thread Group 1-7	HTTP Request	26		162	26
7	14:21:11.707	Thread Group 1-8	HTTP Request	18		162	18
8	14:21:11.723	Thread Group 1-9	HTTP Request	11		162	11
9	14:21:11.738	Thread Group 1-11	HTTP Request	8		162	8
10	14:21:11.754	Thread Group 1-13	HTTP Request	13		162	13
11	14:21:11.754	Thread Group 1-12	HTTP Request	17		162	17
12	14:21:11.770	Thread Group 1-15	HTTP Request	11		162	11
13	14:21:11.771	Thread Group 1-14	HTTP Request	20		162	20
14	14:21:11.785	Thread Group 1-16	HTTP Request	19		162	19
15	14:21:11.802	Thread Group 1-18	HTTP Request	21		162	21
16	14:21:11.802	Thread Group 1-17	HTTP Request	22		162	22
17	14:21:11.816	Thread Group 1-19	HTTP Request	142		162	142
18	14:21:11.832	Thread Group 1-21	HTTP Request	6029		162	6029
19	14:21:11.863	Thread Group 1-24	HTTP Request	9258		162	9258
20	14:21:11.894	Thread Group 1-27	HTTP Request	9570		162	9570
21	14:21:11.632	Thread Group 1-2	HTTP Request	10394		162	10394
22	14:21:11.723	Thread Group 1-10	HTTP Request	11157		162	11157
23	14:21:11.941	Thread Group 1-32	HTTP Request	11012		162	11012
24	14:21:11.848	Thread Group 1-22	HTTP Request	11441		162	11441
25	14:21:11.988	Thread Group 1-36	HTTP Request	11566		162	11566
26	14:21:11.956	Thread Group 1-33	HTTP Request	11741		162	11741
27	14:21:12.175	Thread Group 1-53	HTTP Request	11644		162	11644

Scroll automatically? Child samples? No of Samples 100 Latest Sample 18477 Average 13224 Deviation 6375

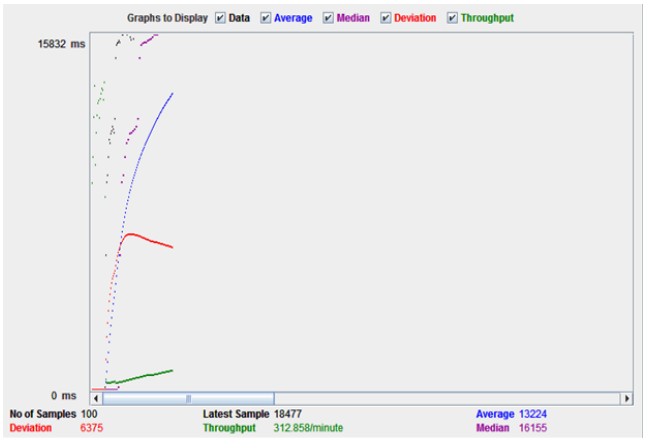


Fig. 6(a), 6(b). Scalability test for Arithmetic REST service

From the figures Figure 6(a) and 6(b), it is known that the throughput of an Arithmetic REST service is observed to be 313 requests per minute which is higher than that of a SOAP service. This clarifies that a REST service can handle more loads at the same time, and is specifically beneficial in a cloud environment, where all users has to be satisfied on an all time basis.

On a PC environment, it is understood from the above results that RESTful Web services are far more efficient when compared to SOAP based Web services. However, it is likely to test for Web service access from mobile devices and test the competence of SOAP and REST.

An Arithmetic SOAP and REST service have been deployed on a Web server and an Android application is developed to access the Web service. The test bed environment consists of an Android phone of ICS (Ice Cream Sandwich) version with a RAM of 1GB and internal phone storage of 16 GB. It holds a 1GHZ U8500 Dual Core processor. The Internet speed at which the Web service and mobile application communicates is 2 Mbps. Figure 7 (a), 7 (b), 8(a) and 8 (b) shows the screenshot of the computational resource occupied by the application to access the Web service.

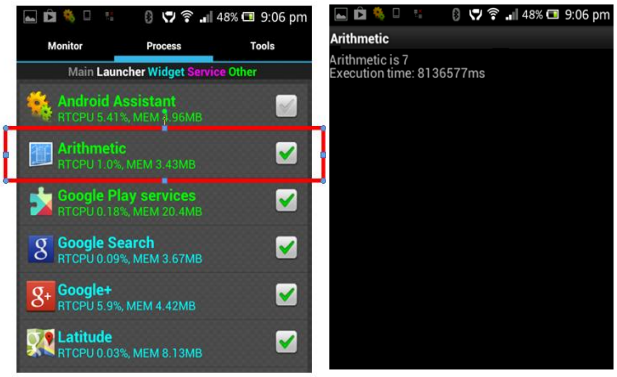


Fig. 7(a), 7(b). Screenshot of mobile arithmeticSOAP application usage and its execution time.

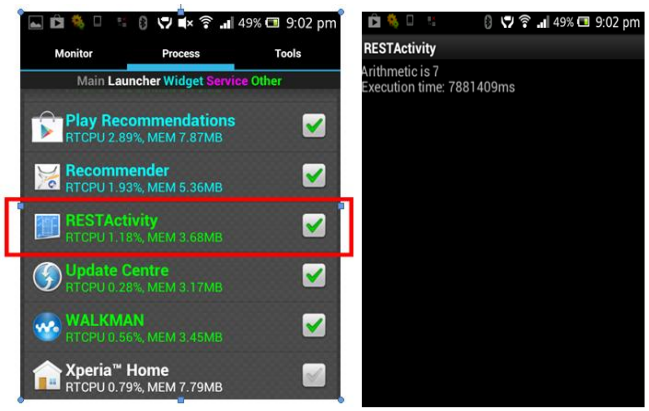


Fig. 8(a), 8(b). Screenshot of mobile arithmeticREST application usage and its execution time.

The mobile applications have been developed for consuming the SOAP and REST types of Web services. It is observed that REST application consumes a RTCPU (Real-Time CPU) of 1.18% when compared to SOAP which consumes a RTCPU of 1.0%. This difference is due to the stateless nature of REST. The REST style of Web service records no

preconceived information about the client, and therefore, it provokes to consume little more time when compared to SOAP. This might be due to transmission delays during request or response time. Overall, REST is the preferred protocol due to its stateless nature.

7. Conclusions and Future Directions

The need for a multicloud middleware for mobile devices has been explored and the performance analysis of the cloud service APIs have been carried out and the results recommend RESTful APIs as more efficient in terms of performance and processing overhead, and occupies less storage on the phone since the data format is JSON. The study on the entire stack of middleware has to be carried out to bring a full-featured generic middleware. The need for "open APIs" has to be propagated far and wide.

References

1. Lachgar, M., Abdali, A: *Modeling and generating native code for cross-platform mobile applications using DSL*. In: Intelligent Automation & Soft Computing, 23 (2016), No.3, p. 445–458.
2. Cohen, R: Cloud API Propagation and the Race to Zero (Cloud Interoperability). Jan 20, 2009. Retrieved from http://www.elasticvapor.com/2009/01/cloud-api-propagation-and-race-to-zero.html
3. Hogan, M., Liu, F., Sokol, A., Tong, J: NIST Cloud Computing Standards Roadmap. NIST CCSRWG-092, First Edition, Gaithersburg, 2011.
4. ETSI (http://www.etsi.org).
5. Messina, J: IEEE Project 2301 – Guide for Cloud Portability and Interoperability Profiles (CPIP). IEEE Computer Society, 2014. Retrieved from http://standards.ieee.org/develop/project/2301.html
6. ITU Telecommunication Standardization Sector. ITU (https://www.itu.int/en/ITU-T/Pages/default.aspx).
7. National Institute of Standards and Technology. NIST (https://www.nist.gov/itl).
8. Karmakar, A., Pilz, G: OASIS Cloud Application Management for Platforms (CAMP) TC. In: OASIS – Advancing open standards for the information society, 2012. Retrieved from https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=camp.
9. Cloud Data Management Interface (CDMI). SNIA. (https://www.snia.org/cdmi).
10. TMForum (https://www.tmforum.org).
11. Aradhana, C., Balaji, R., Jim, P., Joe, W., Michele, D., Tushar, B: Interoperability and Portability, Cloud Security Alliance Group 4.
12. Open Cloud Computing Interface (http://occi-wg.org/).
13. Open Virtualization Format. DMTF. (https://www.dmtf.org/standards/ovf).
14. DMTF Standard: Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol – An Interface for Managing Cloud Infrastructure. Distributed Management Task Force Standard, DSP0263, 2012. Retrieved from https://www.dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.1.pdf
15. The Open Group (http://www.opengroup.org).
16. Cloud Standards Customer Council. CSCC. (http://www.cloud-council.org).
17. Open Commons Consortium (http://occ-data.org/index.html)
18. Motohashi, K. (2011). Global Inter-Cloud Technology Forum. NTT DATA Agilenet L.L.C. Retrieved from https://www.dmtf.org/sites/default/files/20110518_ISO_JTC1_SG38_SGCC_GICTF_2.pdf
19. Cohen, R: Unified Cloud Interface. 2018.Retrieved from https://code.google.com/archive/p/unifiedcloud
20. JetS3t (http://www.jets3t.org)
21. Jclouds (http://jclouds.incubator.apache.org/documentation/reference/supported-providers/)
22. Darryl, E: New Multi-Cloud API, New Add Server Assistant, and Community Translations. Aug 25, 2011. Retrieved from http://www.rightscale.com/blog/rightscale-news/new-multicloud-api-new-add-server-assistant-and-community-translations
23. ZCloud API (http://www.zmanda.com/zcloud.html).
24. Joe, B: DeltaCloud. Feb 17, 2012. Retrieved from http://readwrite.com/2012/02/17/a-look-at-deltacloud-the-multi/
25. Libcloud (http://clean-clouds.com/2013/).
26. Typica API (http://code.google.com/p/typica/).
27. Park, S., Seo, C., Yi. J: Cyber Threats to Mobile messenger apps from identity cloning. In: Intelligent automation & Soft computing, 22(2016), No.3, p.379-387.
28. Shah, S: Recent advances in Mobile Grid and Cloud Computing. In: Intelligent automation & Soft computing, 24(2018), No.2, p.285-298.
29. Programmable Web (http://www.programmableweb.com/).
30. Hamad, H., Saad, M., Abed, R: Performance Evaluation of RESTful Web Services for Mobile Devices. In: Arab Journal of e-Technology, 1 (2010), No. 3, p.72-78.
31. Aijaz, F., Ali, S.Z.Chaudhary, M.A., Walke, B: Enabling High Performance Mobile Web Service Provisioning. In: Proceedings of the 70th Vehicular Technology Conference VTC 2009, 2009, Anchorage, AK, USA, p.1-6.
32. Aijaz, F., Ali, S.Z., Chaudhary, M.A., Walke, B: Enabling Resource-oriented Mobile Web Server for Short-Lived services. In: Proceedings of the IEEE 9th Malaysia International Conference on Communications MICC 2009, 2009, Kuala Lumpur, Malaysia, p. 392-396.
33. Mizouni, R., Serhani, M.A., Dssouli, R., Benharref, A., Taleb, I: Performance Evaluation of Mobile Web Services. In Proceedings of the IEEE 9th European Conference on Web Services ECOWS 2011, Sept 2011, Lugano, Switzerland, p. 184-191.
34. AlShahwan, F., Moessner, K: Providing SOAP Web Services and RESTful Web Services from Mobile Hosts. In: Proceedings of the IEEE 5th International Conference on Internet and Web applications and Services ICIW 2010, May, 2010, Barcelona, Spain, p.174-179.
35. AlShahwan, F., Moessner, K., Carrez, F: Distributing Resource Intensive Mobile Web Services. In: Proceedings of the IEEE International Conference on Innovations in Information Technology IIT 2011, April, 2011, p. 41-46.
36. Jmeter(http://jakarta.apache.org/jmeter/)