HiLiCLoud :High Performance and Lightweight Mobile Cloud Infrastructure for Monitor and Benchmark Services

Dai Hoang Tran, Chuan Pham, Cuong T.D, Dung T.N, Nguyen H.T, Eui-Nam Huh, Choong Seon Hong
Department of Computer Engineering, Kyung Hee University
Email: {dai.tran, pchuan, dtcuong, ntiendung, nguyenth, johnhuh, cshong}@khu.ac.kr

***Abstract*—In the area of cloud infrastructure environment, the management tool to monitor and control the cloud resources is the important factor that can drive the cost benefit of the cloud vendors. But most these tools are bundled within the high cost commercial platforms and are optimized to run on desktop computers. With the vision that Mobile Cloud Computing will be the future technology paradigm that dominates the IT industry, we want to create a cloud management tool that is open source, fast, lightweight and mobile friendly. We take the initial steps by implementing our framework using several popular technologies such as RESTful, Java Message Service, JSON, and we call it "High performance and Lightweight Mobile Cloud Infrastructure Monitor and Benchmark Service" or HiLiCloud. The initial testings show competitive evaluation results.**

I. Introduction

Mobile devices such as smartphone and tablet are gradually becoming an essential part of our life. They are effective and powerful communication devices that we can use at any moment, and are not bounded by place and time. Their rich environment of applications and services developed by worldwide developers make them become a powerful new trend in the IT technology development (e.g., Android Google application, iPhone iOS applications). However, mobile devices still face many challenges because of their limited resource capacity such as battery life, storage, network bandwidth, etc [1]. This leads to the prevention of service quality improvement.

On the other hands, Cloud computing (CC) nowadays is considered to be the next generation's computing infrastructure, that provides low cost hardware (e.g., storages, CPU, networks) and software (e.g., application programs) utilities to users by leveraging its powerful and modular infrastructure. Additionally, CC enables elasticity for on-demand resources usage of users. As a result, the combination of mobile devices and cloud computing complements each others, forming what is called "Mobile Cloud Computing" (MCC). MCC brings brand new type of services and facilities to mobile users with the power of cloud computing [2].

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2015-(H8501-15-1015)) supervised by the IITP(Institute for Information & communications Technology Promotion).
*Dr. CS Hong is the corresponding author

In the space of cloud infrastructure environment, there are always a need for cloud management tools to monitor and assess the cloud resources. Usually these applications are bundled within the high cost commercial such as VMWare Horizon Suite. There are also free options such as Xen Citrix[1], but they are proprietary in source code, and are not mobile friendly since most of them are desktop applications, or web applications and are not optimized for mobile usage. As mobile devices are now ubiquitous and MCC now becomes the strong back-end for mobile devices, we want also the ability to dynamically access, control and monitor our cloud resources on the these devices. As for such requirements, we are building a framework for cloud resources management that could be used on mobile devices and meet its characteristics such as battery life and network bandwidth limitation. We call it "High Performance and Lightweight Mobile Cloud Infrastructure Monitor and Benchmark Service" or HiLiCloud. The main principles we are following during the design and implementation of HiLiCloud are that the response time has to be minimum, response data has to be lightweight, the system memory footprint has to be minimal. Thus we give more focus to the design and implementation of communication between components, that is applying RESTful model, JSON data format and JMS technology.

The remaining of the paper is organized as follows: section II provides background information about the communication techniques we used. Section III details the design and implementation of our framework HiLiCloud, section IV shows how the HiLiCloud performs in our initial testing stages, and finally we draw our conclusion in section V.

II. Background

This section elaborates the concepts of mobile cloud computing. Further, it explains the cloud resources management platforms, the web services and communication technologies that are widely used in modern software services and applications.

A. Cloud Resource management

Virtualization is the key concept behinds the success of Cloud Computing. A cloud infrastructure is usually powered

[1] http://www.xenserver.org

by powerful physical machines, and its quartet physical resources of processing power, storage, input/output (IO) and memory get abstracted as shared resources, and can be provided as different layers of virtualized services. With these physical components get virtualized, they can be combined to form a Virtual Machine (VM), which can function like a real physical computing machine, and has the flexibility in resources utilization. As the Cloud Computing getting popular, the Cloud vendors want to have credible resources management platform that fully utilizes their cloud resources, maximize their cost benefit, quickly scale workload and resources without obstructing the provided customer services, thus guarantee performance, Quality of Service (QoS), scalability and adaptability.

III. HiLiCloud Framework

A. Architecture

HiLiCloud framework has its focus on performance and mobile friendly, so the designed architecture is composed of several modules that provide high throughput and fast, lightweight responses. Figure 1 shows the HiLiCloud framework. At the heart of HiLiCloud is the high performance Asynchronous JMS Handler (AJH), that responsible for generating and digesting JMS messages between modules. The AJH queues its messages in ActiveMQ broker, a popular open-source Message Broker[2]. When a HTTP request is initiated by an end-user or an administrator, the Balancer receives the incoming request and authenticates the user. Depend on the user's type, it will expose full (for administrator) or partial (for normal user) of its RESTful API. User now can invoke subsequent requests for the framework services. The Balancer then delivers the requests in form of JMS message to the Controller module, and as mentioned above, all of the delivery is handled by AJH. The Controller modules consists of smaller components that make the decision for different types of service.

One particular important component of the Controller is the Status Repository (SR). The SR will receive and store all the information of the whole system. It will be the central hub for other components to make intelligent analysis, derive behaviors and instrument other modules' activities. Its source of information comes from the Hypervisor Monitor, which tracks hypervisor activities using third party library called LibVirt, and the labor Service Virtual Machines (Service-VMs).

The Service VMs provides framework services to users. It maintains few sub-modules such as Resources Monitor, Extensible Service API (ESA) and the JSON Processor. The Resources-Monitor keeps track of Virtual Machine status, and regularly updates is status back to the SR in the Controller by a defined interval. We call these update actions with the name "heartbeats". The ESA is the set of services provided by HiLiCloud that users can query for. ESA is designed to follow a strict and general application interface, thus gives us the flexibility in adding new services when needed. Currently,

[2]http://activemq.apache.org

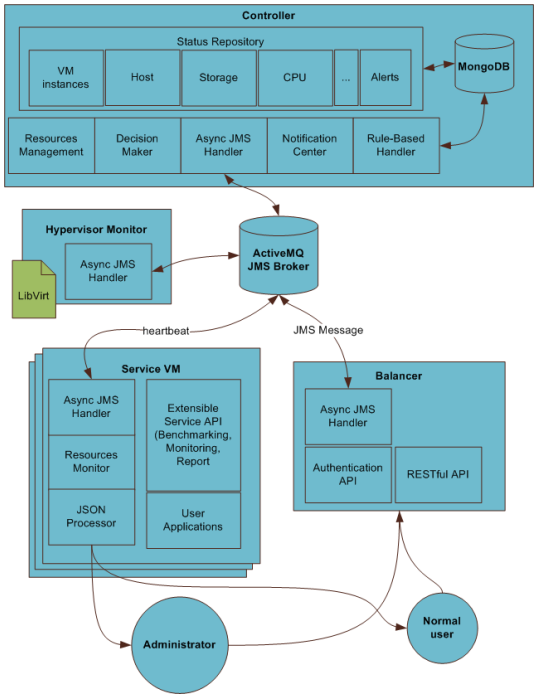

Fig. 1: HiLiCloud Architecture

we provide three main services, they are Benchmarking, Monitoring and Report services. When the Service-VM finishes its invoked service task, the result is passed to JSON-Processor sub-module. JSON Processor deals with the serialization of response data into JSON format, and gives HTTP responses to users in JSON format.

1) Asynchronous JMS Handler: At the heart of HiLiCloud is the AJH component. AJH will create and regulate the flow of all JMS messages when HiLiCloud modules talk to each others. Each HiLiCloud module has its own AJH. The AJH run in a separate and self-control thread, that is constantly watching the ActiveMQ broker status by subscribing to a specific topic. Upon receiving messages, it will hand over to other components inside the module to further process the payload data.

Traditionally, JMS messages usually get handled in synchronous manner by the AJH when it receives messages. As a results, JMS messages get queued up in a long queue inside the database of the message broker. To prevent this situation since it dramatically slows down the processing speed of the whole framework, we implement the **asynchronous** interface to let the AJH receives multiple messages at one, then it can spawn multiple worker threads to handle the content of each JMS message.

Initially, all the JMS messages will be fetch directly into the Controller which is located in a physical machine to make decisions on user requests. But we find that this behavior causes drawbacks in some cases, such as when users just want to query simple system status information of a particular VM. This kind of message can be passed directly to user's

```
<?xml version="1.0"?>
<rule>
    <id>1012</id>
    <source name="service-vm-001"
            ip="192.168.1.101" />
    <destination name="service-vm-003"
                 ip="192.168.1.103" />
    <service type="report">
        <route type="direct">
    </service>
</rule>
```

Fig. 2: An example rule of RBH module in XML format

specific service-VM without going through the Controller. To simplify this behavior when the Controller doesn't need to involve in every request actions, we add a sub component called "Rule based Handler" (RBH) in the Controller. By updating the behavior rules in RBH, the AJH can intelligently send JMS messages to intended service-VM faster, thus gives better response rate and throughput to users. The content of a rule of RBH module can be serialized to XML file format and deserialized to keep in cache memory for access efficiency. One example rule of the RBH in XML format is presented in Figure 2. The reason why we use XML in this case is because this rule object is meant for internal processing, XML gives us more flexible in syntax to defining our rules than JSON format.

2) Controller: The brain of the HiLiCloud framework is the Controller module. It controls, monitors and instruments the whole framework to keep the whole system stable and speed up the response rate of framework services. There are several components inside the Controller, We will elaborate the details of the important ones: The Status repository (SR) keeps all records of cloud system status using an embedded MongoDB, the "Notification center" (NC) that constantly tracks the abnormally of the system status within defined thresholds. Upon detection of system abnormally changes, the NC will quickly notify the related modules to handle the situations. And the Decision maker (DM) acts as a Java daemon that make final decisions of the framework, and give commands to other components.

3) Heartbeat: Initially, we put the Controller under quite heavy pressure workload as aside from controlling the whole system, it also has to pull other modules' health status to see if everything is working fine. To relax this pressure, we use a technique that commonly applied in many popular open source distributed middle-wares called heartbeat [11]. Every module has a built-in Heartbeat class that sends out updated JMS messages about its health status to the Controller. Not only this reduces the strain on the Controller, but also now we can manage the system more effectively.

4) Service Virtual Machines: The last module in our HiLiCloud framework is the Service Virtual Machine (service-VM). Like any public cloud infrastructure, the service-VM is the unit that provides cloud services in different model types, such as Platform-as-a-Service type to host user developed applications, or Software-as-a-Service (SaaS) to provide commercial grade applications to be used by end-users. Our HiLiCloud framework can be fitted into these cloud service types as Paas or SaaS [10]. End-users can use HiLiCloud as a standalone SaaS to monitor and benchmark their cloud infrastructure, or developers can use HiLiCloud along with their developed applications to check for the system health status and adjust system resources in critical moments. Within these service-VMs, their components include the Resources monitor, the JSON processor and the Extensible service API. The Resources monitor (ReM) records the system status of the Service-VM and pushes that information to the RM. JSON processor performs serialization of responses to JSON format and give those lightweight data back to users. And the Extensible service API (ESA) is the logical implementation of RESTful service API, that each service-VM can provide. This includes Benchmarking service (BS), Monitoring service (MS) and Reporting service (ReS).

IV. Testing and performance tuning

A. Testing methods

With the focus of having a monitoring and benchmarking framework that provides high throughput and lightweight responses as cloud services, our testing approach is to simulate a busy business scenario in which the users send lot of requests to the system. We implement a RESTful endpoint that when get invoked from the browser, will constantly generate different types of request in short period to exercise the framework, and the framework will send JSON responses to display on the web browser of user's devices. During that process, we keep track of the amount of requests, the system resources fluctuation, the throughput of the system to have the evaluation in both sides of the server and the client.

B. Evaluation

We are still actively working on implementing the HiLiCloud framework. Since it is not possible to show all the framework evaluation aspects at this moment, we focus on the first few evaluation metrics, the mean response time (MRT), the memory usage (MU) and CPU load (CL) of the HiLiCloud framework. We wrote a benchmark service that a normal user can access at the RESTful URL patternhttp://⟨ip⟩/HiLiCloud/vm/all/benchmark/response_time/⟨params⟩, whereas *params* value specify the amount of requests. The response of this benchmark is a random string of JSON text in about 400-500KB size. We run the test multiple times, with *params* value ranges from 50, 100 to 1000, 2000 and run in multiple devices in different network conditions. Figure 3 shows the benchmark result. To find relative contexts for comparison, we run the benchmark on normal desktop with wire Internet connection, mobile with wireless and 3G connection.

From the result, it is obvious that the response time from the wired PC connection is very small and consistent. The MRT of mobile device show some interesting aspects. For a small amount of requests, the performance is somewhat consistent, but for large request quantity, we see that the 3G connection's MRT approaching the wireless connection's MRT. This indicates that our framework performance get fluctuated when the amount of request is so dense in a short

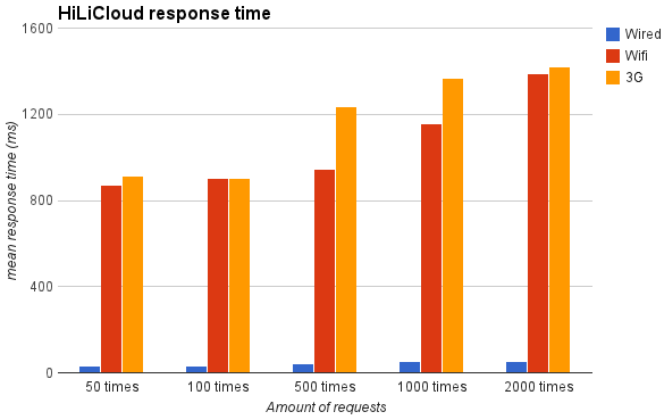

Fig. 3: Response time benchmark

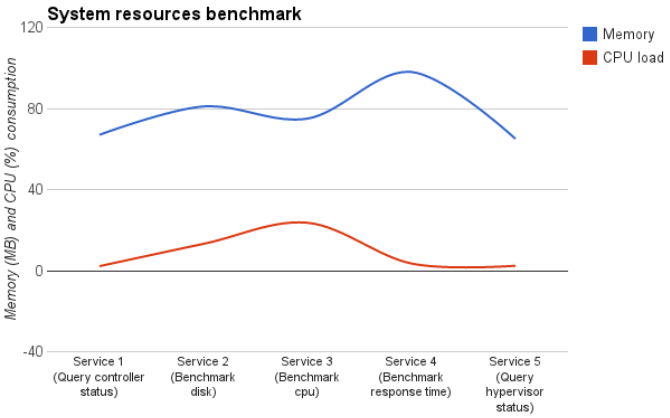

Fig. 4: System performance benchmark

period of time, we need to do deep investigation in the future work. For the MU and CL evaluation from Figure 4, it shows that our framework does not consume that much memory when different services are invoked and the CL shows also low CPU peak load for different kind of benchmarks. Thus it shows its lightweight characteristic.

V. Conclusions and Future Work

In conclusion, even though our framework achieve fairly competitive performance, we believe that we can give it more tuning and optimization to speed up the whole framework. The SPDY protocol [12] is the next option we consider for the improvement of HiLiCloud. On the scalability aspect, HiLiCloud still has it weakness at the message broker as now it is the single point of failure. We have considered to use a high performing model of the broker that multiple instances of the it can run in isolated VM environment, thus provides better tolerance and achieves highly scalable message broker module. We also plan to integrate another service layer that takes into account the user's Service Level Agreement, that can monitor and adjust the service quality based on user's agreements. And last but not least, we will expand the API of the monitoring and benchmark services, that in turns makes this framework become our base MCC service for more sophisticated researches in mobile cloud computing area.

References

[1] M. Satyanarayanan, Fundamental challenges in mobile computing, in Proceedings of the 5th annual ACM symposium on Principles of distributed computing, pp. 1-7, May 1996.

[2] Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu. 2013. Mobile cloud computing: A survey. Future Gener. Comput. Syst. 29, 1 (January 2013), 84-106. DOI=10.1016/j.future.2012.05.023 http://dx.doi.org/10.1016/j.future.2012.05.023

[3] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing, Mobile Networks and Applica- tions, vol. 16, no. 3, pp. 270284, 2011.

[4] White Paper, Mobile Cloud Computing Solution Brief, AEPONA, November 2010.

[5] Papazoglou, M.P.; Traverso, P.; Dustdar, S.; Leymann, F., "Service-Oriented Computing: State of the Art and Research Challenges," Computer , vol.40, no.11, pp.38,45, Nov. 2007

[6] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, SOAP Version 1.2 Part 1: Messaging Framework, Published on the internet, Jun. 2003, w3C Recommendation. [Online]. Available: http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[7] Davis, D.; Parashar, M., "Latency Performance of SOAP Implementations," Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on , vol., no., pp.407,407, 21-24 May 2002

[8] Maeda, K., "Performance evaluation of object serialization libraries in XML, JSON and binary formats," Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on , vol., no., pp.177,182, 16-18 May 2012

[9] R. T. Fielding, Architectural styles and the design of network-based software architectures, Ph.D. dissertation, University of California, Irvine, 2000.

[10] Rimal, B.P.; Eunmi Choi; Lumb, I, "A Taxonomy and Survey of Cloud Computing Systems," INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on , vol., no., pp.44,51, 25-27 Aug. 2009

[11] "Heartbeat (computing)." Wikipedia. Wikimedia Foundation, 22 Aug. 2014. Web. 04 Sept. 2014. http://en.wikipedia.org/wiki/Heartbeat_(computing)⟩

[12] "SPDY Protocol." SPDY Protocol. N.p., n.d. Web. 04 Sept. 2014. http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft2