Architecture and Measured Characteristics of a Cloud Based Internet of Things API

Geoffrey C. Fox, Supun Kamburugamuve
School of Informatics and Computing and Community Grids Laboratory
Indiana University, Bloomington IN 47408 USA
{gcf, skamburu}@indiana.edu

Ryan Hartman
Community Grids Laboratory
Indiana University, Bloomington IN 47408 USA
rdhartma@indiana.edu

ABSTRACT

The Internet of Things (IoT) many be thought of as the availability of physical objects, or devices, on the Internet [1]. *Given such an arrangement it is possible to access sensor data and control actuators remotely. Furthermore such data may be combined with data from other sources, e.g., with data that is contained in the Web and/or operated on by scalable cloud hosted system(s) to create services far richer than can be provided by isolated embedded systems* [2,3]. *This is the vision of the Internet of Things.*

We present a cloud-compatible open source messaging system and extendable API named 'IoTCloud' that enables developers to write scalable high performance IoT and sensor-centric applications. The IoTCloud software is written in Java and built on popular open source packages such as Apache Active MQ [4] *and JBoss Netty* [5]. *We will present an overview of the IoT Cloud architecture and describe its developer API.*

Next we introduce the FutureGrid – a geographically distributed and heterogeneous cloud test bed [6,7] – used in our experiments. *Our preliminary results indicate that a distributed cloud infrastructure like the FutureGrid coupled with our flexible IoT framework is an environment suitable for the study and development of new, scalable, collaborative/IoT/sensor-centric applications.*

Finally we report on our initial study of measured characteristics of this middleware system running on the FutureGrid.

KEYWORDS: Internet of Things, IoT, distributed cloud, collaboration, sensor-centric applications, smart objects, FutureGrid

1. INTRODUCTION

Since the standardization of the TCP/IP protocol suite over thirty years ago, the Internet has grown steadily from a collection of small research networks to a ubiquitous worldwide network of interconnected networks that serviced nearly two billion unique users in 2009 [8]. Now with the widespread availability of wireless Internet connectivity combined with the low cost of miniature electronic devices it possible to imagine the Internet expanded to include objects, embedded with sensors, communicating over the Internet in in vast numbers [9]. These *smart objects* are ordinary physical things that are augmented by a small computer which includes a sensor or actuator and a communication device [3]. A *smart object* is thus an *embedded system*, consisting of a thing (the physical entity) and a component (the computer) that processes the sensor data and supports a, usually, wireless communication link to the Internet [2].

> Smart objects can be battery-operated, but not always, and typically have three components: a CPU (8-, 16- or 32-bit micro-controller), memory (a few tens of kilobytes) and a low-power wireless communication device (from a few kilobits/s to a few hundreds of kilobits/s). The size is small and the price is low: a few square mm and few dollars [10].

The technical issues involved with the IoT vision are not related to the functional capabilities of *smart objects* – there many such embedded systems are connected to the Internet today – but in the potentially massive numbers of *smart objects* involved. Some examples of these issues are: identification of smart objects, management and organization of networks of smart objects, data privacy and trustworthiness. Therefore, there is a need for scalable systems that will explore these and the other related technical issues involved in creating IoT applications. As a step towards this our preliminary effort is on focused developing a sensor-centric middleware capable of supporting applications that incorporate a wide variety of

sensor types and a large number of geographically distributed sensors. We also discuss the suitability of distributed clouds for hosting our middleware and discuss an initial measurement of the performance of our system in the context of scalable real-time collaborative sensor-centric applications.

The rest of this paper is organized as follows. Firstly, we present the architecture of our IoTCloud framework and describe the API available to end-users. Secondly, we give an overview of the FutureGrid [6,11], the underlying heterogeneous distributed cloud infrastructure that we conduct the experiments on. Then, we discuss the experimental setup and report performance measurements in several scenarios. Lastly, we present our conclusions and future work.

2. IoTCloud Architecture

We have adopted an open architecture for the IOTCloud middleware system. From a very high view the architecture is based on four main components.

1. IOTCloud Controller
2. Message Broker
3. Sensors
4. Applications

IOTCloud Controller component is responsible for managing the Message Broker, Sensors and Application. It is at the center of the IOTCloud architecture and coordinates the communication between sensors and applications. Also it maintains the status information about the system and publishes it to the interested applications. Sensors produce a series of data over the time and applications can register for acquiring this data. Also applications can control the sensors as well as receive the status information about the system. Figure 1. Shows the architecture of the IOTCloud in a very high view.

We will look at each of these parts in more details separately.

IOTCloud Middleware

IOTCloud Contrller is a coordinator and a configuration manager between the communication parties. It creates the paths to establish the connections among the Applications and Sensors. These paths are created in the message brokers. Also it maintains the status information about the various sensors and applications. For example if an application wishes to know, the names of the sensors that belong to a particular group, that information is available in through the middle-ware system.

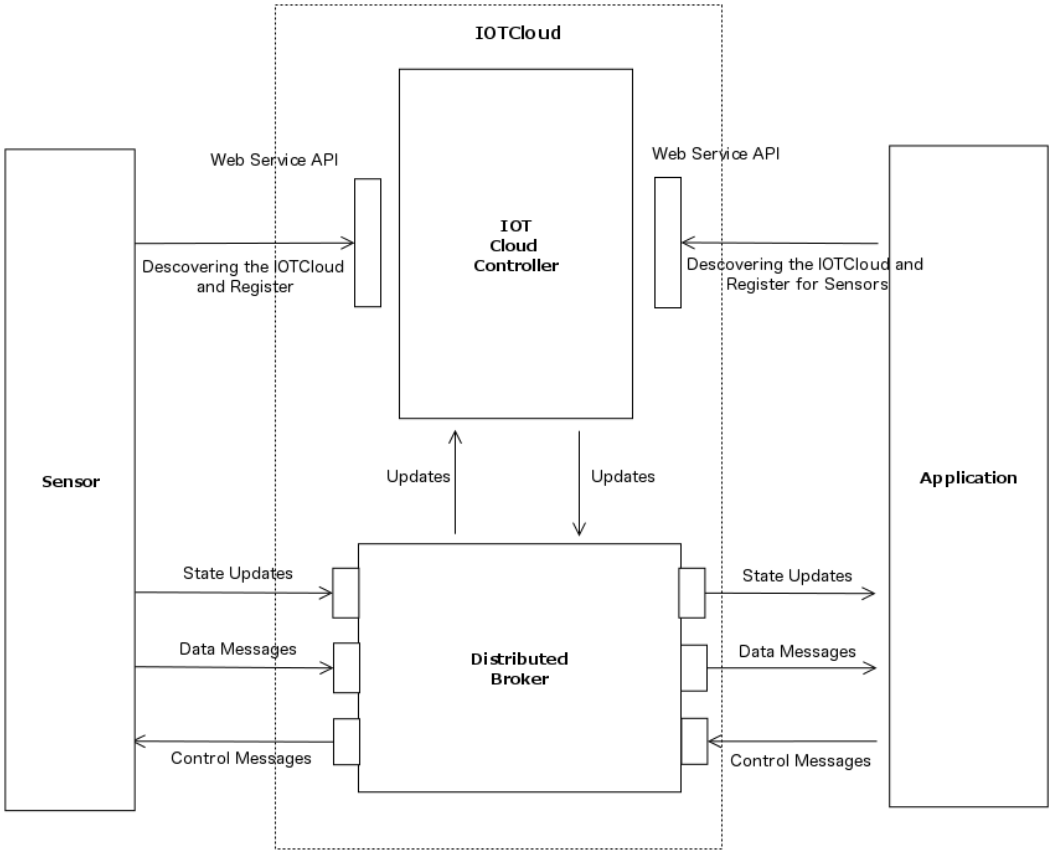

Figure 1.

We believe that the interoperability is a key, to a technology that is designed to connect various devices together. To gain the interoperability, IOTCloud exposes a Web Services [4] API for applications and sensors. Sensors use this API for registering to the IOTCloud and getting the required information. Application also uses the web services API for same purposes. The web services are written using Apache Axis2 and hosted in a Jetty server.

Apart from the status information about the system, IOTCloud stores various configuration information needed by the applications and sensors to communicate with each other. The information about the brokers and how to connect to them are stored in the controller, so that sensors and applications can use this information without the knowledge of explicit configurations of underlying message transfer mechanisms.

Message Broker

In general sense, we use the term "message broker" to refer a system that accept messages and distribute it to set of targets. We use a JMS [5] style message broker to handle block type data and streaming message broker to handle streaming data. All the routing rules and configuration of the broker are managed by the controller and the specific details of the broker is hidden from the application and sensor developers.

JMS Message Broker

This broker is used for distributing data messages, all the updates and control messages making it a key part in the IOTCLoud architecture. IOTCloud supports a JMS 1.1 specification compatible message broker. JMS style broker runs as a separate entity from the controller. We use Apache ActiveMQ [6] as the message broker for this implementation.

Streaming Message Broker

Streaming message broker is a simple low footprint HTTP server written using the Netty [7] technology. It is a streaming server with a constant memory usage for streaming the messages to multiple targets.

Sensor

A Sensor is a component that can publish time dependent data. The sensor module facilitates the development of sensors by providing mechanisms for exchanging data between the actual sensor and the IOTCloud. A typical example is a GPS sensor. A user trying to connect a GPS to the IOTCloud should use the Sensor module to register itself to the middle-ware systems as well as to send and receive information.

The IOTCloud supports two types of sensors and the distinction is made upon the nature of the data the sensor is producing. There are sensors that produce data in discrete time intervals. The best example is a GPS sensor. It only produces data in a fixed rate and the data is independent of each other for most of the practical purposes. Due to that, we though it is better to fit this type of sensors in to a category called block sensors. They support sending data as independent messages. The other type of sensors produces a continuous data stream. The best example is, a video camera producing a stream in very high data rate. IOTCloud has the streaming sensor type to handle sensors producing a data stream.

IOTCloud by design supports both types of sensors using different message transfer mechanisms. For block type of data, it uses a JMS style messaging. So the data packets are discrete and there is no correlation between the messages. For streaming data, IOTCloud uses a HTTP based streaming message server. We have chosen HTTP because of its wide adoption and support for streaming data via the chunking.

Apart from sending data, every sensor should listen for the control messages. A control message carries information about an operation that the sensor should perform. One of the best examples is a sensor that has start and stop functions like a video camera. An application that is registered to the IOTCloud should be able to turn on or off the video camera. This information is sent as a control message to the sensor.

Applications

Data produces by the sensor should go through the IOTCloud to the applications. Applications listen on the sensor data and control the sensors by sending control data. Any application can listen on one or more sensors. After the registration to the IOTCLoud, application can choose the sensor from which it receives data, using filters. If the application directly knows the name of the sensor it can readily bind to it. But most of the time applications are looking for sensors with some criteria like for example a GPS sensor that belongs to the group "Indiana Lab".

There is a notification mechanism to send updates about the sensor status to the applications. Some of these update messages are produced by the IOTCloud itself and some are produced by the sensors. Applications can choose to accept all or set of updates about the system.

Update Distribution

Since the sensors and applications are ad-hoc it is required to send the states changes to the applications. When a client joins it only knows about the available sensors. A sensor of interest to the Application may come online after the Application starts its operation on the available sensors. So

the application should know about the new sensor immediately upon its arrival. These kinds of updates are delivered to the interested sensors whenever they are available. Also IOTCloud maintains the state of its sensors internally. It periodically pushes this information to the interested applications as well. All the updates are handled through the message brokers and the middle-ware system so that the system knows about the status of the connected components.

Application Programing Interface (API)

From the users perspective the important APIs are the Sensor API and Application API. The sensor API is designed to facilitate easy development of Sensors and publish data to the IOT cloud. Apart from the data publication these APIs are used to control the behavior of the sensors depending on the control messages coming from the Applications as well as IOT Cloud.

The Application API is designed to facilitate the easy consumption of the data published by the sensors. Also the API has mechanisms for applications to listen on the changes to the sensors and IOT cloud. Also it has methods for sending control information to the sensors as well.

Both these API's are available as Java APIs for the initial version. We believe these API's should be available in languages like C/C++ as well. This is possible because we are using standards based technologies like Web Services and HTTP for communicating between the sensors and the applications.

Sensor API

Sensor API provides routines to register the sensor to the IOTCloud and publishing the data to it. Also sensors can listen for the controls messages sent both by the middleware system as well as applications.

First a sensor has to initialize the IOTCloud libraries by pointing the bootstrap configuration files. These configuration files include the information needed by the underlying transports that does the actual web service messaging.

Then it can register itself to the sensor cloud by providing its name, type of the sensor and group. Type of the sensor denotes weather it is a block sensor or a streaming sensor. Once it is registered it can start sending the data. Data should be prepared by the user and should be wrapped inside the message API provided by the IOTCloud.

Also after starting its operation, sensors receive control information from the middleware. It is up to the sensor implementer to act upon this control data. After completing its operation a sensor should notify the middleware system that it is going to terminate. Middleware system also periodically checks for the status of the sensors. If a sensor failed to respond within a given time sensor cloud assumes that the sensor no longer available and closes the communication paths.

Application API

The application API consists of several classes for registering an application with the IOTCloud and getting sensor data. Here is the typical sequence of actions an Application will take for registering itself and getting some data.

First it has to initialize the IOTCloud libraries by pointing to the configuration files. This is similar to the process by the sensors.

Then it can register itself with the IOTCloud by pointing to the IOTClouds URL. Here the underlying APIs will talk to the controller to get information about the protocols it supports and the mechanism for connecting to it. For example the application receives information about the transports that are supported by the controller. Here the ports of the HTTP transport or the JNDI properties of the JMS transport are sent through this call. So it is a discovery process for the Application.

Once the Application is registered to the controller it can request information about the sensors that are connected to the system. Various filters can be used to get only the required list of sensors. For example it can get only the list of GPS sensors by specifying the filter that filters the sensors by the group of the sensors.

Once the Application has the names or the ID's of the sensors it can register itself for that sensors data stream. When there is data from the system the registered application are notified and they can receive the data. Data is processes according to the application and the sensor requirements. Once the application is no longer interested in receiving data, it can un-register itself from the IOTCloud.

Message API

IOTCloud has two broader categories of message types and these represent control messages and data messages. Both Application API and Sensor API use these message types for message access.

Control Message is a message with a specific command and some key value pairs associated with it. The meaning of control messages is a contract between the sensor and

the client. There are some reserved control messages that IOTCloud explicitly uses for controlling the behavior of sensors in pre-defined situations like when shutting down the IOTCLoud.

Since different sensors can send different messages with different semantics IOTCloud doesn't try to infer or act upon the message itself. For the middleware a message is just a data block or a stream. Message schema is strictly a contract between the sensors and its consumers. Since there are two types of sensors namely block type sensor and streaming sensors, IOTCloud provide mechanisms for supporting block data of messages and streaming data messages.

It is up to the sensor and application programmer to figure out the meaning of the data messages with different mime types. Some sensors like still cameras may send pictures in JPEG format and vedio cameras may send it in a vedio-encoding format. It is up to the sensor application to figure out the type of the messages and decode it according to its requirements.

3. FUTUREGRID

FutureGrid [8] is a part of the Extreme Science and Engineering Discovery Environment (XSEDE) [14]. The FutureGrid provides a capability that makes it possible for researchers to tackle complex research challenges in computer science related to the use and security of grids and clouds. These include topics ranging from authentication, authorization, scheduling, virtualization, middleware design, interface design and cybersecurity, to the optimization of grid-enabled and cloud-enabled computational schemes for researchers in astronomy, chemistry, biology, engineering, atmospheric science and epidemiology [11]. The project has several computing clusters at different locations with a sophisticated virtual machine and workflow-based simulation environment to support research on cloud computing, multicore computing, new algorithms and software paradigms.

Unlike production cloud systems like the Amazon EC2, Microsoft Azure or Google App Engine for commercial applications, or XSEDE for scientific computing, FutureGrid, by contrast, is oriented towards developing tools and technologies rather than providing production computational capacity [15].

FutureGrid is an infrastructure comprising currently approximately 4,000 cores at six sites - Indiana University (11 Teraflop IBM 1024 cores, 7 Teraflop Cray 684 cores, 5 Teraflop Disk Rich 512 cores), University of Chicago (7 Teraflop IBM 672 cores), University of California San Diego Supercomputing Center (7 Teraflop IBM 672 cores), University of Florida (3 Teraflop IBM 256 cores), Purdue University (4 Teraflop Dell 384 cores) and Texas Advanced Computing Center (8 Teraflop Dell 768 cores) - connected by a high-speed, network which is dedicated except for public link to Texas Advanced Computing Center. It is an experimental test-bed that could support large-scale research on distributed and parallel systems, algorithms, middleware and applications. Figure 2. shows the connectivity of the six sites.

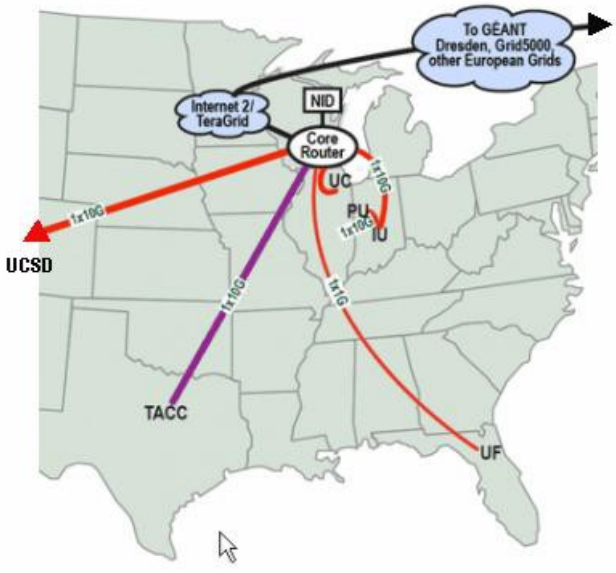

Figure 2. FutureGrid connectivity

FutureGrid includes services accessible to users to run HPC (High Performance Computing) jobs such as MPI, or Hadoop. It also supports several popular Grid and Cloud environments including the Eucalyptus, Nimbus and OpenStack Clouds.

Eucalyptus [16, 17] is an open source software platform that implements an Infrastructure-as-a-Service (IaaS)-style cloud computing. Eucalyptus provides an Amazon Web Services (AWS)-compliant, EC2-based web service interface for interacting with the cloud service. Additionally, Eucalyptus provides Walrus, an AWS storage-compliant service, and a user interface for managing users and images.

Nimbus is an open source toolkit that allows one to turn a cluster into an IaaS cloud [18]. Nimbus on FutureGrid allows users to run virtual machines on FutureGrid hardware. A Nimbus account user can easily upload custom-built virtual machine (VM) image or customize an image provided by FutureGrid. When a VM is booted, it is assigned a public IP address (and/or an optional private address). The VM is accessible by logging in as root via SSH. A user can then run services, performs computations, and configure the system as desired. After using and configuring the VM, the modified VM image can be saved to the Nimbus image repository.

OpenStack is an open source, IaaS cloud computing platform

founded by Rackspace Hosting and NASA and widely used in industry [18]. It includes three components: Compute (Nova), Object Storage (Swift) and Image Service (Glance). OpenStack is also fully Amazon EC2 complaint and supports an (AWS)-complaint web interface. OpenStack images are manipulated with the familiar euca2ools [19]. Our FutureGrid experiments were performed using OpenStack for virtual machine deployment/management.

4. Performance Characteristics

Previous experiments have investigated our messaging system performance at the network, message, and application level [20-22]. We find that a pub/sub based middleware is an appropriate model for scalable sensor-centric, collaborative, and IoT applications. In this section we present the performance characteristics of a single message broker as a function of the number of subscriber instances.

For this experiment we created a virtual sensor to simulate a typical data stream from an IP Camera. We selected the popular TRENDnet TV-IP422WN ip camera as our baseline [21]. The TV-IP422WN camera publishes audio/video data over an RTSP stream at a rate of approximately 1800kbps when using the following encoding:

> Video: codec MPEG4; width: 640; height: 480; format: YUV420P; frame-rate: 30 frames/sec; Audio: codec PCM_MULAW; sample rate: 8000; channels: 1; format: FMT_S16

In order to simulate video sensors of this type we publish randomized data in 7680 bytes packets at a rate of 30 packets per second. It is worth noting that this frame rate and packet size is also reasonable simulation of Microsoft Kinect sensor [24].

Our software was deployed on the FutureGrid using a single OpenStack m1.large instance [25] to host the middleware and the simulated sensor. Subscribers were then deployed across multiple m1.large instances as necessary. Figure 3. shows the average message latency versus the number of clients. Latencies of less than 300 milliseconds are required for real-time video conferencing applications [20].

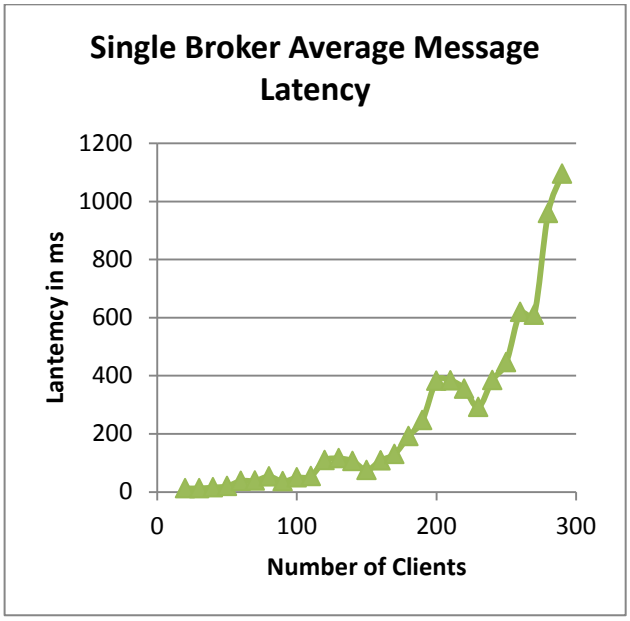

Figure 3.

Therefore in terms of message delivery times alone a single broker is capable of supporting approximately 200 clients participating in a simulated real-time video conferencing application. However, in real-time of collaborative video applications message latency is not the only factor, uniformity of the message latency must also be considered. In order to achieve a satisfactory user experience the video packets must also be delivered in a uniform (i.e. non-jittery) manner [19]. Figure 4. shows the average jitter versus number of clients.

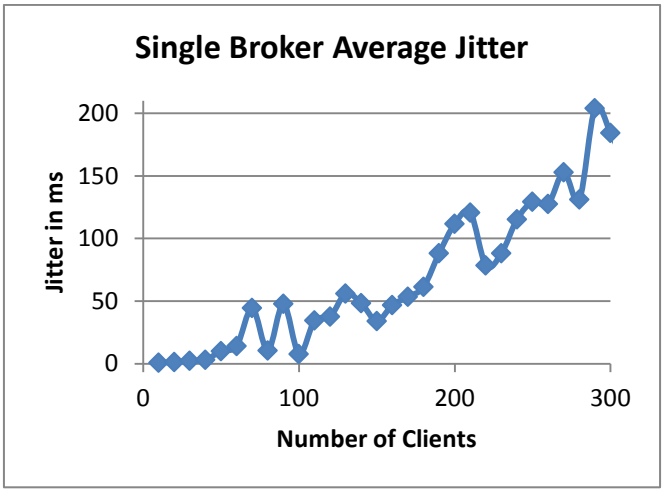

Figure 4.

Here we see acceptable jitter until we reach approximately 150 clients. This number is a better estimate of the true number of clients a single broker can support and maintain good performance. To verify this conclusion we also examine how the jitter various over time. These results are show in Figure 5.

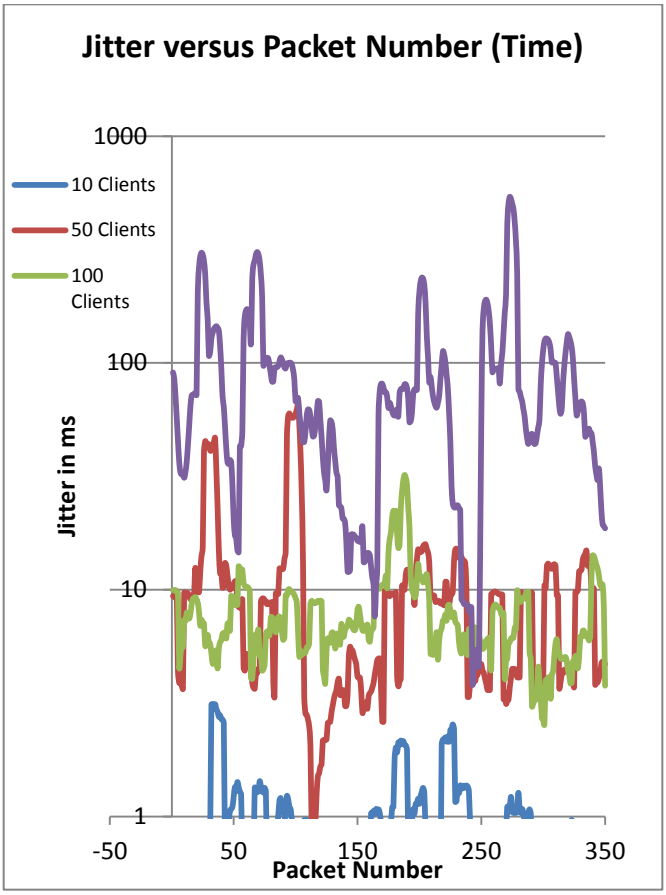

Figure 5.

The here the jitter (as a function of time) is acceptably low until the number of clients reaches approximately 150 clients. We have demonstrated therefore that a single broker is capable of supporting 150 clients participating in a real-time video conferencing application where 640x480 video is streamed at 30 frames per second. Scaling is achieved by deploying additional brokers in the cloud to support an arbitrary client load.

In previous work [18-20] we have shown a single broker is capable of supporting larger number of clients in the case where the sensor data packet size was lower. Many IoT applications will consist of sensors with significantly smaller packet sizes and transmission frequencies e.g. gps, rfid, ZigBee etc. In these cases we expect our system to support potentially thousands of clients with a single broker. Further work is planned to examine this scenario.

5. Conclusion

We presented our open sourced IoTCloud framework and gave an overview of its architecture and the extensible API we provide to develop scalable IoT and other sensor-centric applications. Next we described the FutureGrid, the experimental testbed for cloud development, we use for our development and testing. Finally we conducted a preliminary study to analyze the performance characteristics of our middleware in the context of high end real-time video sensors.

KEYWORDS: Internet of Things, IoT, distributed cloud, collaboration, sensor-centric applications, smart objects, FutureGrid

ACKNOWLEGMENTS

We thank Alex Ho of Anabas Inc., Sankarbala Manoharan and Vignesh Ravindran of Indiana University for their important contributions to this work. This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High- Performance Grid Test-bed." Other partners in the FutureGrid project include U. Chicago, U. Florida, San Diego Supercomputer Center - UC San Diego, U. Southern California, U. Texas at Austin, U. Tennessee at Knoxville, U. of Virginia.

BIOGRAPHY

GEOFFREY C. FOX received a Ph.D. in Theoretical Physics from Cambridge University and is now the Associate Dean for Research and Graduate Studies at the School of Informatics and Computing Indiana University Bloomington and professor of Computer Science, Informatics, and Physics at Indiana University where he is director of the Community Grids Laboratory. He previously held positions at Caltech, Syracuse University and Florida State University.

SUPUN KAMBURUGAMUVA is a PHD Student in the Computer Science Department of Indiana University Bloomington. He is an Apache Software Foundation member and is a contributor on many open source projects including Apache Web Services and Apache Synapse. He has been actively contributing in developing middleware systems in to cloud environment, which is being his major area of interest.

RYAN HARTMAN is software developer with experience in a wide range of topics, technologies and industries. He has led many successful projects from numerical modeling of physical systems, to Doppler Weather Radar control systems, to web-based software distribution systems and currently onto cloud computing.

REFERENCES

1. Postscapes. *Internet of Things Definition*. [accessed 2012 February 6]; Available from: http://postscapes.com/internet-of-things-definition.

2. Hermann Kopetz, *Real-Time Systems Series*. 2011, Spring US, ISBN 978-1-4419-8236-0. pages. 307-323. DOI: 10.1007/978-1-4419-8237-7_13

3. Luigi Atzori, Antonio Lera, *The Internet of*

Things: A survey. Computer Networks Volume 54, Issue 15, 28 October 2010. pages. 2787–2805. DOI: 10.1016/j.comnet.2010.05.010

4. RAO, A. 2002. Web Services Unleashed, garage insigh http://www.garage.com/newsletter/index.shtml

5. Hapner, M., Burridge, R., Sharma, R., Fialli, J., and Stout, K. 2002. Java Message Service. Sun Microsystems Inc., Santa Clara, CA.

6. Apache. *ActiveMQ open source messaging system.* [accessed 2012 February 6]; Available from: http://activemq.apache.org/.

7. JBoss. *Netty - the Java NIO Client Server Socket Framework*. [accessed 2012 February 6]; Available from: http://www.jboss.org/netty.

8. Geoffrey Fox. *FutureGrid Platform FGPlatform: Rationale and Possible Directions* (White Paper). 2010 [2012 February 6]; Available from: http://grids.ucs.indiana.edu/ptliupages/publications/FGPlatform.docx.

9. Javier Diaz, Gregor von Laszewski, Fugang Wang, Andrew J. Younge and Geoffrey *Fox FutureGrid Image Repository: A Generic Catalog and Storage System for Heterogeneous Virtual Machine Images* 3rd IEEE International Conference CloudCom on Cloud Computing Technology and Science, Athens Greece, November 29 - December 1 2011 http://grids.ucs.indiana.edu/ptliupages/publications/jdiazCloudCom2011.pdf

10. Google. *Public Data*. [accessed 2012 February 7]; Available from: http://www.google.com/publicdata/explore?ds=d5bncppjof8f9_&ctype=l&met_y=it_net_user&hl=en&dl=en

11. The Internet of Things Council Homepage, [accessed 2012 February 8]; Available from: http://www.theinternetofthings.eu/what-is-the-internet-of-things

12. Adam Dunkels, JP Vasseur. *IP for Smart Objects* Internet Protocol for Smart Objects (IPSO) Alliance White Paper #1 September 2008 Available From: http://www.sics.se/~adam/dunkels08ipso.pdf

13. XSEDE Homepage, [accessed 2012 February 7]; Available from https://www.xsede.org/home

14. Geoffrey Fox. Interview on FutureGrid. 2009 September 29 [accessed 2012 February 7]; by Sander Olson Available from: http://nextbigfuture.com/2009/09/interview-of-geoffrey-fox-director-of.html.

15. Nurmi D., Wolski R., Grzegorczyk C., Obertelli G., Soman S., Youseff L., and Zagorodnov D., The Eucalyptus Open-Source Cloud-Computing System, in 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. CCGRID '09. 18-21 May, 2009. Shanghai. pages. 124-131. DOI: 10.1109/CCGRID.2009.93.

16. Eucalyptus Open Source Cloud Software. [accessed 2012 February 7] Available from: http://open.eucalyptus.com/.

17. Nimbus Cloud Computing for Science. [accessed 2012 February 7]; Available from: http://www.nimbusproject.org/.

18. OpenStack. *Open source software for building private and public clouds.* [accessed 2012 February 7] Available from: http://openstack.org

19. OpenStack. *Euca2ools Commands*. [accessed 2012 February 7] Available from: http://docs.openstack.org/diablo/openstack-compute/starter/content/Euca2ools_Commands-d1e2791.html

20. Geoffry Fox, Alex Ho, and Eddy Chan, *Measured Charatericstics of FutureGrid Clouds for Scalable Collaborative Sensor-Centric Grid Applications*. IEEE International Symposium on Collaborative Technologies and Systems CTS 2011, Waleed Smari, Editor. May 23-27, Philadelphia. http://grids.ucs.indiana.edu/ptliupages/publications/cts_2011_paper_mod_6%5B1%5D.pdf

21. Shrideep Pallickara, Hasan Bulut, Pete Burnap, Geoffrey Fox, Ahmet Uyar, and David Walker. *Support for High Performance Real-time Collaboration within the NaradaBrokering Substrate*. 2005 May [accessed 2011 March 11]; Available from: http://grids.ucs.indiana.edu/ptliupages/publications/NB-Collaboration_update.pdf.

22. Ahmet Uyar and Geoffrey Fox, *Investigating the Performance of Audio/Video Service Architecture I: Single Broker*, in IEEE International Symposium on Collaborative Technologies and Systems CTS05. May, 2005, IEEE. St. Louis Missouri, USA. pages. 120-127. http://grids.ucs.indiana.edu/ptliupages/publications/SingleBroker-cts05-submitted.PDF. DOI: http://doi.ieeecomputersociety.org/10.1109/ISCST.2005.1553303.

23. Trendnet. *Secure View Wireless Day/Night Pan Tilt Zoom Internet Camera*. [accessed 2012 February 8] Available from: http://www.trendnet.com/products/proddetail.asp?prod=155_TV-IP422W

24. Microsoft. *Kinect for Windows.* [accessed 2012 February 8] Available from: http://www.microsoft.com/en-us/kinectforwindows/

25. Open Stack. *OpenStack Manuals*. [accessed 2012 February 8] Available from: http://docs.openstack.org/diablo/openstack-compute/starter/content/Instance_Type_Management-d1e2734.html