A Design of Flexible Data Channels for Sensor-Cloud Integration*

Jose Melchor
Computing & Software Systems
University of Washington Bothell
Bothell, WA 98052, U.S.A.
Email: pepemel@u.washington.edu

Munehiro Fukuda†
Computing & Software Systems
University of Washington Bothell
Bothell, WA 98052, U.S.A.
Email: mfukuda@u.washington.edu

***Abstract*—The recent popularity of sensor networks and cloud computing has brought new opportunities of sensor-cloud integration that facilitates users to immediately forecast the future status of what they are observing. The main challenge is how to establish a series of elastic data channels from sensors to mobile users through data-analyzing jobs running in the cloud. The variation of network protocols, the dynamism of cloud-computing resources, and even user mobility result in the following three design problems. Data-analyzing programs must: (1) deal with the major sensor and cloud communication protocols; (2) include hard-coded sensor manipulation and data retrieval operations; and (3) handle job/user migration in dynamic cloud environments, which needs the recovery of broken data connections. To address these problems, we are developing the *Connector* software tool kit that provides cloud jobs with uniform data channels, allows them to exchange standard input/output as well as graphics with remote users, and re-establishes their data channels upon a job or user migration. This paper presents *Connector*'s system overview, implementation, and performance consideration.**

***Keywords*-data channels; cloud computing; sensor-data analysis; fault tolerance; middleware tools;**

I. Introduction

The emergent dissemination of sensor networks and cloud-computing services has brought new opportunities of sensor-cloud integration [1] that will facilitate users not only to monitor their objects of interest through sensors but also to analyze future status of these objects on the fly by using cloud services. For instance in agriculture, crop growers need to monitor real-time orchard temperature for frost protection purposes as well as to forecast the overnight temperature transition by applying sensor data to temperature-prediction models. This type of on-the-fly analysis demands a substantial amount of computing power and storage only in frost-critical seasons, particularly at night, which is therefore fitted to cloud services.

One of the biggest challenges in such sensor-cloud integration is how to establish a series of data channels from sensors to mobile users through data-analyzing jobs running in the cloud. The variation of network protocols, the dynamism of cloud-computing resources, and even user mobility result in the following three design problems. Data-analyzing programs must: (1) deal with the major sensor and cloud communication protocols (such as UDP, SNMP, FTP, and HTTP); (2) include hard-coded sensor manipulation and data retrieval operations (such as packet filtering and web-portal checking); and (3) handle job migration for performance and resource-availability reasons, which needs the recovery of broken data connections.

In general, data-analyzing model designers and users are not well skilled in various network protocols and distributed job coordination. Therefore, to address these problems, we are developing the *Connector* data-channel software tools: *Connector-API*, *Connector-GUI*, *Web Server*, and *Sensor Server*. Connector-API gives cloud applications Java FileInputStream/OutputStream as uniform data channels to be connected to sensors, cloud resources and mobile users, by hiding all underlying network protocols and defining hard-coded parameters into an independent configuration file. Connector-GUI runs on a mobile user side to serve as a portal to data-analyzing jobs in the cloud, which exchanges standard input, output, and graphics between the user and the jobs. Web Server facilitates an alternative GUI to a user if her/his mobile device is not capable of working as a TCP or an X server. Sensor Server runs on a wireless network's master node to (de-)activate each sensor device, to filter sensor data, and to deliver them to cloud jobs. These tools work independently in accordance with each user's needs but also handle job migration and user mobility collaboratively by resuming broken data channels.

This paper presents the Connector software tool kit's contribution to sensor-cloud integration from the viewpoints of functionality and execution performance. The rest of the paper is organized as follows: Section I reviews related work and points out features necessary for sensor-cloud integration; Section III gives an overview of the system model and specification; Section IV explains the implementation techniques; Section V considers the performance; and Section VI concludes our discussions.

*This research is being conducted with partial support from UW Provost International Grants, Faculty-led Program

†Corresponding author. Email: mfukuda@u.washington.edu, Phone: 1-425-352-3459, Fax: 1-425-352-5216

II. Related Work

This section discusses the current technologies available for sensor-data analysis with cloud computing in terms of (1) data retrieval, (2) data storage, and (3) channel recovery and redirection.

A. Sensor-Data Retrieval

Bare wireless networks use the *push*-based data-retrieval model, where each data item is placed into a UDP or an SNMP packet, and thereafter forwarded through the network master node to designated user sites. Users are responsible to filter all incoming packets to detect only those of their interest. On the other hand, the *pull*-based model is available in more sophisticated wireless networks such as sensor databases [2] and commercialized sensor portals, where users initiate a data retrieval query or an HTTP request to a sensor network or a web portal. The problem is that these models require user programs to be hard-coded for filtering packets and sending database queries or HTTP requests.

B. Sensor-Data Storage

For future reuse, users need to maintain past sensor data in provider-specific cloud storage such as Tripod FTP, Amazon S3, and Hadoop. There are many Java packages available to take advantage of these storage: Commons Net 2.2 API [3], Amazon S3 JetS3t [4], and Hadoop API [5]. Although these packages are all designed to return very common Java *DataInput/OutputStream*s to user programs, the users must still understand the respective storage concept and connection set-up procedures. For instance in Amazon S3, a user is supposed to first locate the bucket including a given file, then retrieve the *S3Object* file wrapper, and finally obtain its *DataInputStream*.

Open Data Kit [6] is another promising software that eases the data storage design, mobile device GUI definition, and automated file uploading/downloading operations. However this tool focuses on the easiness of GUI-level data manipulation rather than that of code-level data transfer.

C. Data-Channel Recovery and Redirection

Job migration and user mobility should be dealt with in the dynamic computing environment in the cloud. These features need support for recovering and redirecting existing data channels to new locations where the job and the user reside. Condor (which is supported in Amazon EC2) runs a user-side shadow process that keeps track of a remote job and delivers all I/O data from it to the local user [7]. Rocks/Racks adds snapshot-maintenance features to the standard UDP/TCP protocols, so that it re-delivers lost packets and messages to a migrating job as well as re-establishes broken TCP connections [8].

However, on-the-fly sensor-data analyses do not always require all sensor data to be re-delivered to them. In fact, it is even common that some sensor devices are out of battery or that packets are dropped during wireless communication. Therefore, of importance is to keep applications and users unaware of their migration by simply recovering and redirecting data channels to their new location.

In summary, we need the following three features when integrating sensor networks and cloud services: (1) hiding underlying different network/storage protocols from applications, (2) separating connection set-up and data-sampling operations from applications, and (3) supporting location unawareness with automatic data-channel recovery and redirection.

III. System Model and Specification

This section defines our system model used in sensor-cloud integration, clarifies the design principles of the **Connector** data-channel software tool, and introduces its specification.

A. Design Principles

Figure 1 illustrates the sensor-cloud integrated system that we assume. Launched somewhere in the cloud, each sensor-data analyzing program runs as a client of various network protocols, retrieves data from sensor networks, stores results to remote storages, and interacts with its mobile user. To cover these features, we are designing the **Connector** data-channel software tool kit: (1) **Connector-API:** a program-side I/O and graphics package, (2) **Connector-GUI:** a user-side GUI, (3) **Web Server:** a web-based GUI, and (4) **Sensor Server:** a sensor-side data publisher. We are based on the following three design principles: (1) facilitating Java *FileInput/OutputStream*-based uniform channels by hiding all underlying network protocol; (2) separating connection set-up and data-sampling work from user code into an independent configuration file; and (3) automating channel recovery and redirection upon a job/user migration.

B. Specification

Connector-API allows a cloud job to behave as various protocol clients (including FTP, HTTP, and X windows) to access remote data through the major Java classes such as FileInput/OutputStream, Frame, and Graphics. File-to-URL mapping is no longer hard-coded in a user program but is rather specified in a file map, using a quadratic notation of: $(name, URL[, interval, extract])$, where each parameter represents (1) a file *name* used in a user program, (2) the corresponding *URL* including a user account and password, (3) a repetitive access to a given Web site every *interval* seconds, and (4) only text data *extract*ed from the Web. Figure 2 shows an example of file-to-URL mapping. The three files: sensor1, file2.txt, and file3.txt, all named in a user program, are respectively retrieved from the corresponding sensor, FTP, and HTTP servers. In particular, file3.txt provides the user program with data items that are repeatedly read from the same website every 5 seconds. Since an application needs

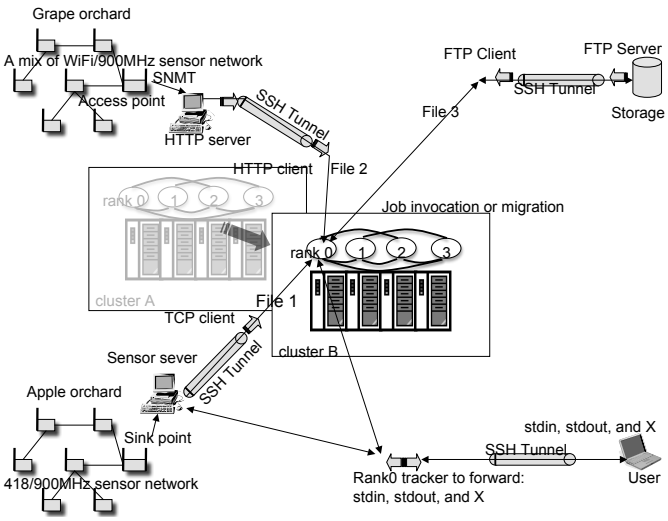

Figure 1. Redirectable Stream-Oriented Channels

to communicate with a remote user or a web server, this file-to-URL mapping allows a user to specify her/his mobile device or a given web server with *-i ip_address* and *-p port* options.

# -i	ip_address		
# -p	port		
sensor1	sftp://account:password@hercules.uwb.edu/sensor1		
file2.txt	ftp://account:password@ftp.tripod.com/temperature.txt		
file3.txt	http://www.weather.com/today/Bothell+WA+98011	5	extract

Figure 2. File-to-URL mapping in Connector.jar

Figure 3 is an example user program that takes the file map shown in Figure 2. It first starts a Connector daemon thread (line 6). While the user program writes to the standard output, reads *sensor1*, and writes *file2* locally, in background the daemon forwards "Recording..." to a remote user's console (line 8), connects to *hercules.uwb.edu/sensor1* (line 10), contacts with *ftp.tripod.com* as an FTP client (line 13), and transfers data from the sensor to the ftp server (lines 15-17). The Connector daemon is capable of behaving as a (secure) FTP, HTTP, and X client.

Connector-GUI runs on a user-local machine to facilitate GUI by forwarding keyboard/mouse inputs to and by receiving standard outputs from a remote user program. It is also capable of transferring files between user-local disks and each user program. To provide these features, the GUI runs as a TCP server to keep track of a nomadic user program by accepting its connection request upon a migration; scrutinizes each message for its data delivery; and also works as an X proxy client to display graphics.

Web Server facilitates an alternative GUI to a mobile user if her/his mobile device is not capable of working as an TCP

```
1   import Connector.*;          // Import Connector package
2   import java.util.Scanner;
3   public class TemperatureRecording {
4   public static void main( String[] args ) {
5     // Initialize a Connector daemon thread
6     Connector System = new Connector("file.map");
7     // forward the message to a remote user
8     System.out("Recording...");
9     // Connect to an orchard sensor
10    FileInputStream in=new FileInputStream("sensor1");
11    Scanner input=new Scanner( in );
12    // Connect to ftp.tripod.com
13    FileOutputStream out=new FileOutputStream("file2");
14    DataOutputStream output=new DataOutputStream(out);
15    while( input.hasNextLine( ) ) {
16      // Transfer data from the orchard sensor to ftp.tripod.com
17      output.writeUTF( input.nextLine( ) );
18    }
19    System.close( );             // Terminate the Connector daemon
20  } }
```

Figure 3. A user program importing Connector-API

server or an X server. This server acts as Connector-GUI by presenting the same GUI menu to a user's web browser, allowing a remote cloud job to establish TCP connections with the web server, forwarding the user's menu inputs to the remote job as the standard input, and relaying this job's standard and graphical outputs back to the web browser as an HTTP response.

Sensor Server runs on a sensor-network master node in charge of (1) managing all its sensor devices, (2) behaving as an FTP server to make all sensor devices accessible as files from applications, (3) detecting changes in sampled data, and (4) handing off active connections to nomadic jobs. For these purposes, it reads from Connector-GUI, Web Server, or a given configuration file several commands to add, delete, and change a sensor's IP address, MAC24 address, and name as well as to detect data-sampling conditions, as shown in Figure 4.

add	192.168.15.21	sensor1
add	0x082be4	sensor2
detect	sensor2 $<=$ 33.6	absolute

Figure 4. Sensor-to-File mapping in Connector Server

IV. Component Implementation

A. Connector-API

Connector-API creates a daemon thread in the background when a user program instantiates a Connector object. As illustrated in Figure 5, the daemon connects to a remote Connector-GUI that is waiting at the IP address and port defined in its file map. (See the first two lines in Figure 2.) It then allows the user program to exchange standard input/output data and graphics with the GUI. If no GUI exists on the specific site, the daemon thread simply disables any functions that rely on the GUI.

The daemon thread possesses multi-threaded capabilities. Whenever it detects a new instance of Connector-API supported classes, (i.e., FileInput/OutputStreams, Frame, and

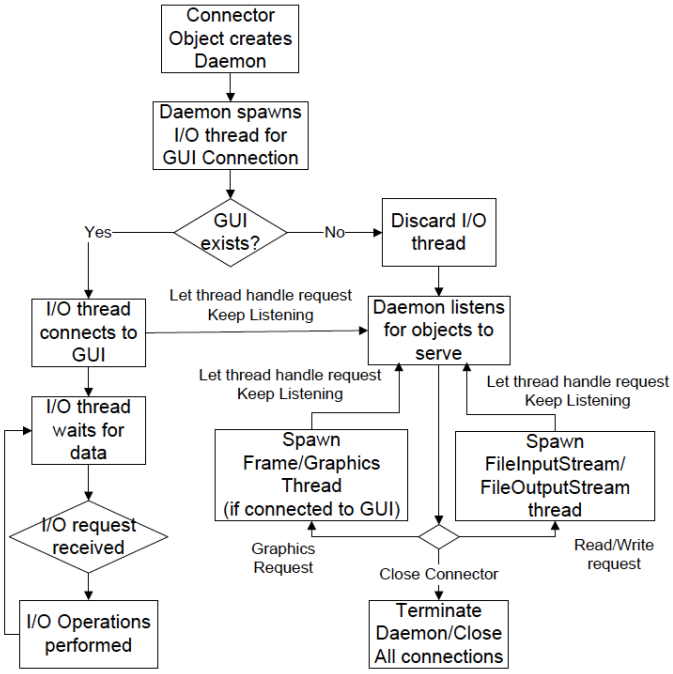

Figure 5. Daemon life cycle

Graphics), the daemon spawns a new child thread that behaves as an FTP, HTTP, or X client to exchange data with a remote file, Web or X server.

B. GUI

As shown in Figure 6, Connector-GUI uses two channels of communication to deal with data coming from a remote user job: one used for handling standard input/output and the other for graphics, both using the same single port. Upon an initialization, Connector-GUI prompts the user to choose a port for accepting connections from a remote job, and thereafter enters a listening state. Interaction begins once a remote job successfully connects to the GUI. When an application finishes, the GUI goes into a listening state once again.

User jobs interacting with the GUI may migrate to a different computing node at any time. Therefore, Connector-GUI has the ability to sense a job migration by sending back heartbeat messages to the job. Once a remote job settles on a new computing node and then requests connections to the GUI, it resumes the communication and continues performing tasks.

Connector-GUI has two techniques to retrieve graphics from a remote application. One uses the GUI as an X proxy client and the other involves retrieving JPEG images sent from the application. As an X proxy client, the GUI reads RMI-like messages (of an object reference, a method name, and arguments) from applications, and calls the corresponding methods locally on actual Java Frame and Graphics

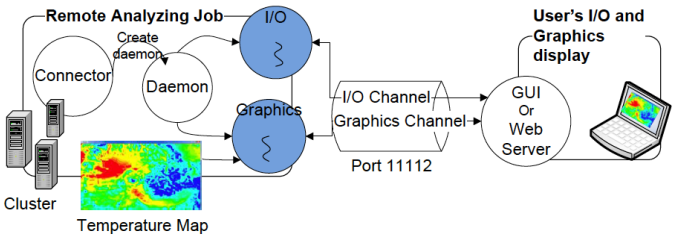

Figure 6. GUI and Web Server

objects to display graphics. On the other hand, every time the GUI receives a new JPEG image, it repackages and displays the image locally on a Java Frame.

C. Web Server

Web Server allows mobile users to access cloud-based data-analyzing jobs from web browsers (in case their mobile devices are not capable of running Connector-GUI, in other words, unable to act as a TCP and X server). It supports all the Connector-GUI's features: monitoring and interacting with remote sensors, exchanging standard input/output data between a user and cloud jobs, and delivering graphics from jobs to her/him. In order to start a session, the very first web menu requires a user to specify a port number that Web Server uses for accepting a TCP connection from a job. Once a connection is established, the first servlet stores a pair of this user's session id and the corresponding socket in the server-internal hash table, and returns to her/him a new HTML page that displays standard input/output fields as well as buttons to disconnect or retrieve remote data. As far as incoming HTTP requests have the same session id. the subsequent servlets use the same socket to relay data between this user and her/his job.

D. Sensor Server

Sensor Server provides two types of channels to allow interactions between users and sensors. One is a data channel used for a remote application to retrieve sensor data through Connector-API as if it read data from local files. The other is a control channel for a remote user to manage sensors through Connector-GUI.

Figure 7 illustrates a sensor server's interaction with an application and two GUIs. Invoked on a wireless network master node, a sensor server itself uses Connector-API to download a sensor-to-file map, exampled in Figure 4, which may be located somewhere else such as in a remote FTP server. It then sets up the accessibility of sensor devices with that map.

Once a sensor server completes reading a sensor-to-file map, it starts behaving as a multi-threaded FTP server to accept all FTP-GET requests from remote applications and to provide them with sensor data of their interest, using an independent child thread. (Note that such a child

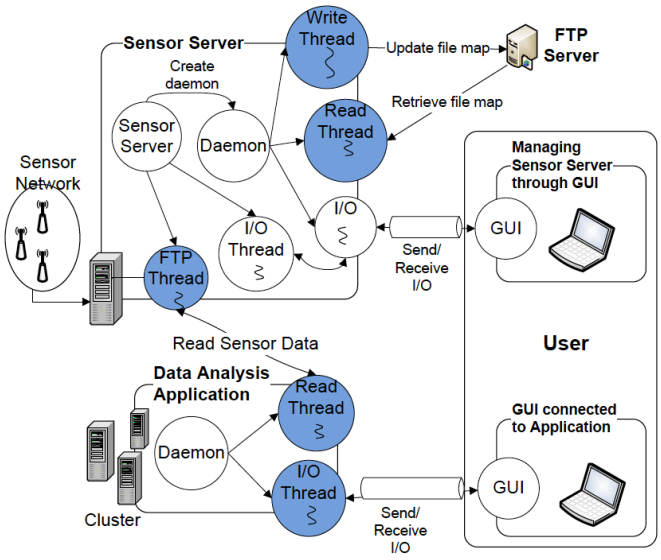

Figure 7. Visualization of Sensor Server

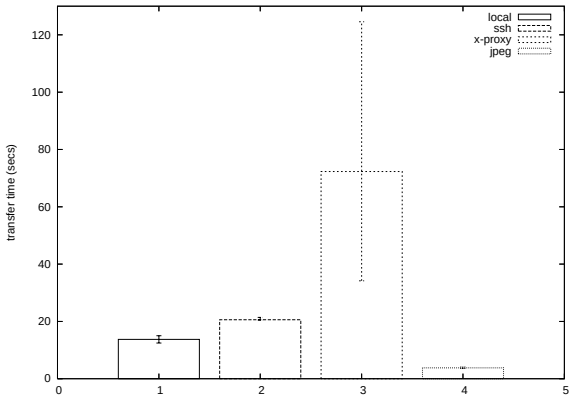

Figure 8. Performance of graphics forwarding

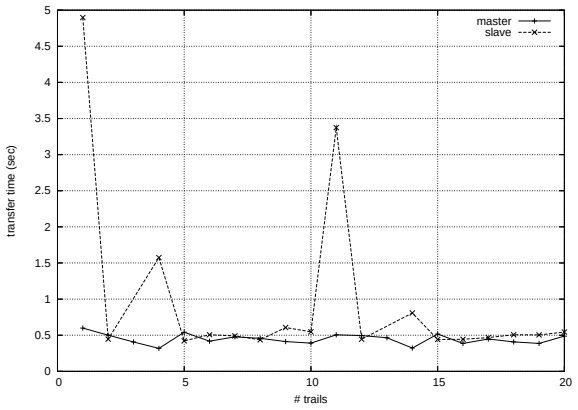

Figure 9. Performance of sensor-data forwarding

thread is a non-permanent instance, as shown shaded in Figure 7.) Applications simply read sensor data through Connector-API's FileInputStream that conveniently resumes them whenever new data items are available, thus keeping the jobs unaware of the underlying FTP transactions

When managing sensors through Connector-GUI, a sensor server spawns a new thread that establishes Connector-API FileInput/OutputStream as a control channel to receive instructions from the GUI. When changes are made to the sensor configuration, the sensor server updates its sensor-to-file map.

V. Performance Consideration

We measured the performance of forwarding graphics to GUI and sensor data to a remote user job.

A. Graphics Forwarding

Graphics forwarding was evaluated with Wave2D (a two-dimensional wave simulation program) in terms of its 20-repetitive execution time measured from a graphics generation in Wave2D to its pop-out on a GUI. Figure 8 shows the average forwarding time over 1Gbps network of four different test cases: case 1 displays a graphics locally where Wave2D runs; case 2 forwards a graphics to a remote X server through SSH; case 3 uses Connector-GUI as an X client proxy; and case 4 forwards a JPEG to the GUI. The results demonstrated the efficiency of JPEG transfers.

B. Sensor-Data Forwarding

Sensor-data forwarding was evaluated with a simple test program that repeat reading sensor data 18 - 20 times from a remote sensor server through Connector-API's Java FileInputStream. We measured the time elapsed for FileInpuStream.read( ) to send an FTP-GET to the sensor server and thereafter to receive new sensor data. Figure 9 compares the performance of data retrieval from the master sensor (where a sensor server runs) and that from a slave sensor located 45 feet away from the master node. The average time of data retrieval from the master and the slave is 0.473 and 0.970 seconds respectively, which means that a sensor server is capable of handling at least 61.85 sensors per minute.

VI. Conclusions

We have designed the *Connector* data-channel tool kit intended for use in sensor-cloud integration. It provides file-based elastic data-transfer channels from sensors to cloud jobs as well as from cloud jobs to mobile users and/or remote file storages. The prototype demonstrated its competitive performance of forwarding graphics over network and capability of handling 60+ sensors per minute. Our next plan is to complete the initial version by adding

Connector-Web Server to the tool set and to use it in sensor-data analyzing applications such as on-the-fly orchard temperature prediction.

ACKNOWLEDGMENT

The authors would like to thank Mr. Stephen Dame (AgComm), Mr. Todd Elliod (Valhalla Wireless), and Mr. Josh Larios (UW Bothell) for their continuous advice on the installation and management of our temperature sensor network in our laboratory.

REFERENCES

[1] M. M. Hassan, B. Song, and E.-N. Huh, "A framework of sensor-cloud integration opportunities and challenges," in *Proc. of the 3rd International Conference on Ubiquitous Information Management and Communication*. Suwon, Korea: ACM, January 2009, pp. 618–626.

[2] TinyDB: A Declarative Database for Sensor Networks, "http://telegraph.cs.berkeley.edu/tinydb/."

[3] Commons Net 2.2 API, "http://commons.apache.org/net/api/index.html."

[4] Amazon JetS3t Toolkit, "http://jets3t.s3.amazonaws.com/toolkit/toolkit.html."

[5] Apache Hadoop, "http://hadoop.apache.org/."

[6] C. Hartung, Y. Anokwa, W. Bruentte, A. Lerer, C. Tseng, and G. Borriello, "Open Data Kit: Tools to build information services for developing regions," in *Proc. of Int'l Conf. on Information and Communication Technologies and Development - ICTD 2010*, London, U.K., December 2010.

[7] Condor Project, "http://www.cs.wisc.edu/condor/."

[8] V. C. Zandy and B. P. Miller, "Reliable network connections," in *Proc. of the 8th Annual International Conference on Mobile Computing and Networking – MOBICOM'02*. Atlanta, GA: ACM Press, September 2002, pp. 95–106.