Pervasive Computing Coursework

Pervasive Home

1 Introduction

For this coursework you will be required to develop a pipeline for capturing and managing sensor data. The scenario is the development of a system to support a home instrumented with sensors that are able to capture light and temperature information.

Important information:

> **Group coursework:** This is a group coursework. It will be undertaken in the groups that you have already been working in for the sensor tutorials.
>
> **Assessment:** This coursework will be assessed by a group presentation and by assessment of code submitted electronically.
>
> Project presentations will take place on **Tuesday 13th March 2012 from 3-5pm**
>
> **Code SUBMISSION DEADLINE: Midday, Tuesday 13th March 2012**
>
> Details of endpoint URLs for supporting services will be provided on the Coursework page of the course website: https://www.doc.ic.ac.uk/~globus/coursework.

2 Equipment

Every group will be provided with access to the following:

- Sensor hardware
  - 2 x Sensor gateway boards with USB connection
  - 4 x Mote sensor devices (1 for use with a gateway board, 3 for temperature/light sensing) – during development and testing you may wish to use a second sensor device in combination with the second gateway board to aid debugging.
- 2 x CouchDB instances running on the LeSC Cloud
- Access to a web-based visualisation interface that will display data you send to it.

3 Coursework Exercise

Your group will develop the infrastructure for one home in a high-tech street where all the houses have central monitoring of temperature and light to help the inhabitants keep track of their energy usage. The energy company collects information from all homes allowing them to compare energy use between homes in the street and provide advice on efficient energy consumption and a community incentive to regulate energy use.

You will build a full pipeline that provides sensing, communication, processing and submission of your data to the energy supplier (see Figure 1).

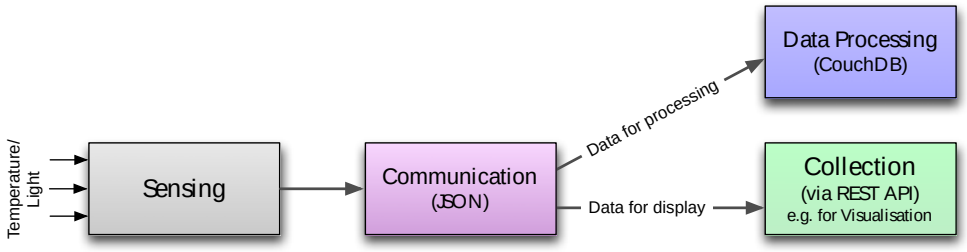

Figure : Pervasive Home Pipeline

You will need to carry out the following tasks:

A) Sensors (50%):

1. Set up **three** of your sensor devices so that they are capable of capturing light and temperature. Write code that will sample *light* and *temperature* every second and report the values to the base-station. Each device must flash its YELLOW LED when it takes a sample. From the base-station, the values will be sent to the Data Processing and Collection services. (see part B). (10 marks)

2. Write code to allow the three sensing nodes to inter-communicate their *light* values. When a node receives a 'Dark' light reading from at least one neighbour, then it indicates this by flashing its own GREEN LED for 20ms. This must happen while continuing to send all sensor readings to the base-station as in part A1.

   A reading is considered to be 'Dark' if the value is less than 100. 'Bright' is considered to be readings above 100. (10 marks)

3. Add code that generates an event message when a node suspects a fire. A given node suspects a fire when it detects a *recent* increase in temperature of 5 ^{o}C and it and all its neighbours sense that it is currently 'Dark'. To identify a recent increase in temperature of 5 ^{o}C each sensor should maintain a log of its last 30 temperature readings and look for a 5 ^{o}C temperature increase (or an increase in the ADC value of 20 over a 30 sample period). When a node detects a fire, it will indicate this with a RED LED.

   If one or more nodes detect a fire, the base-station should warn the users by sending a 'FIRE' message to the data collection service specifying the sensors that have detected a fire. (15 marks)

4. Modify the code that you use to sample light and temperature so that the samples are taken at the same time. This synchronisation will be observable by the fact that all of the devices will flash their YELLOW LEDs at the same time. You can either compose your own scheme to do this or integrate an established scheme into your code. (15 marks)

B1) Communications (25%):

1. Write a Java program on the host (base-station) PC to:
    a. Retrieve the raw data produced by your sensors from the gateway board.
    b. Convert the hexadecimal data from the gateway node into human-readable values and encode the data received from the gateway node in JSON format using the org.json API (http://www.json.org) and sensor data format shown in section 5.

       You should assign each of your three sensors a logical identifier – 0, 1 or 2. This number is used for the `sensorId` parameter described in the API.

2. Write a REST client to send this JSON data to:
    a. the data collection service using the energy display REST API detailed in section 5.
    b. your CouchDB instance for storage and processing (see section B2).

   Once started, your program should sit in a loop, receiving data from your sensor nodes and forwarding this data to the energy display.

   When the previous data reading received from one or more of your sensors reports a suspected fire, send an event message using the event call defined in the REST API.

   In the event call, you must include a list of the IDs of all sensors that reported a fire condition in their last message they sent to the base station (see parameter `sensorIdList`).

B2) Data processing (25%):

1. You will need to use MapReduce to process the data stored to CouchDB from the sensor nodes on your host PC. Write JavaScript Map and Reduce functions in CouchDB's web interface to carry out the following processing:
    a. Create time-series data providing 10-second averages for light and temperature data *for each sensor*. Implement this using a single design document (i.e. one map and one reduce method).

   You will need to upload your data to CouchDB using CouchDB's REST API. This will have been done as part of the code written for part B1.1. To access your CouchDB instances via the REST API, you need to provide your group's CouchDB username and password. The authentication method used is HTTP Basic authentication. If you are using the Restlet REST client library, see Restlet's "Security package" documentation for details of how to add client

authentication to your Restlet calls: http://wiki.restlet.org/docs_2.0/13-restlet/27-restlet/46-restlet.html

Please use the provided database named "*sensor_data*" on your CouchDB instances. This has been pre-configured with security to prevent access to your design documents and data from outside your group. The account username is "*admin*" and the password has been provided to you by e-mail.

You should write your CouchDB Map and Reduce functions as a "temporary view" for testing purposes. Once you are happy with your code, save it as a view. This view will then be available through a CouchDB URL that can be used to extract processed data from your view for assessment purposes. *NOTE: Ensure you also keep an offline copy of your map and reduce functions!*

> For the REST API to CouchDB's views, see:
> http://wiki.apache.org/couchdb/HTTP_view_API

Note: It is important to specify a *group_level* of at least 1 to get a full output from CouchDB's REST service for visualisation.

4 Assessment

The coursework will be assessed through a practical demonstration of your work and through analysis of your code.

Code submission:

You will need to submit **3 archive files (ZIP or GZIPPED TAR format)** via CATE:

1. The sensor code that you have written for your sensor devices – the file should be named

   sensor-group[group number].zip/.tar.gz

2. Java code for the communications application that you have written to run on your host PC – the file should be named comms-group[group number].zip/tar.gz
3. Text files containing your JavaScript map and reduce functions for your CouchDB view – file should be named couchdb-group[group number].zip/.tar.gz

Presentation:

You will be required to give a group presentation covering the following:

1. Power up your sensors and demonstrate the requirements described in section A.
2. Run your Java application, developed for part B1, on the host PC and show that:
   a. Data is being sent to the data collection API.
   b. Data is being sent to CouchDB.
   c. You are able to differentiate between a FIRE event on one or two sensors and a full fire event signalled by all three sensors.

3. Your CouchDB map/reduce code will be tested by using your CouchDB view to access the processed data.

5 Pervasive Home Data Collection – REST API

We will provide an endpoint for a "data collection" service that you can use to send your data to. The service will provide feedback as to whether your call has been successful.

The URL of the data collection service can be found on the Pervasive Computing Coursework page at https://www.doc.ic.ac.uk/~globus/coursework.

The REST API for the Pervasive Home data collection service is described below.

Data will need to be sent in JSON format. Responses will be received as JSON data. Ensure that you set the "`Content-Type:`" header of your requests to "`application/json`" to show that you're sending JSON data and the "`Accept:`" header to "`application/json`" to show that your code can receive JSON as a response.

`POST` `/energyInfo/dataSample` – Send one or more samples of data from your sensor(s)

Accept response content: `application/json`

The JSON parameters for this API operation are:

`groupId: string` Your group number….

`key: string` Your API "key" as provided with your group's coursework sheet

`groupName: string (optional)` An optional name for your group (groupId is still a required parameter)

`sensorData: array` An array of one or more sensor data sample objects. The structure of a sensor data sample is shown below:

Sensor data object structure:

`sensorId: number` Either 0, 1 or 2 depending on which of your three sensors has sampled the packet

`nodeId: number (optional)` The number of the node that has generated this reading

`timestamp: number` Timestamp for the reading in milliseconds since the epoch

`temp: number (double)` Temperature value from sensor OR `null` if a lux value is provided (a single sensor cannot provide both temp and light in a single data sample).

`lux: number` Light intensity value from sensor OR `null` if a temp value is provided (a single sensor cannot provide both temp and light in a single data sample).

Responses:

`OK: Boolean` `true` if the call has completed successfully, `false` if not.

If `OK` is `false`, some error information is provided:

`errorCode: string` A code identifying the error. One of `AUTH_ERROR`, `ERROR_MISSING_VALUE`, `ERROR_INVALID_DATA`, `ERROR_READING_DATA`

`errorMessage: string` A message describing the error

`POST` `/energyInfo/event` – Send an event identified by your sensors.

Accept response content: `application/json`

`groupId: string` Your group number….

`key: string` Your API "key" as provided with your group's coursework sheet

`groupName: string (optional)` An optional name for your group (group number is still a required parameter)

`eventType: string` Value: "`FIRE`"

`eventMessage: string` A string describing the event.

`sensorIdList: array [number]` An array of one or more `number`s (0-2) specifying the sensors that have detected a fire condition. NOTE: A full FIRE event will only be triggered when all three sensor IDs appear in the sensorIdList parameter in a single request.

Responses:

`OK: Boolean` `true` if the call has completed successfully, `false` if not.

If `OK` is `false`, some error information is provided:

`errorCode: string` A code identifying the error. One of `AUTH_ERROR`, `ERROR_MISSING_VALUE`, `ERROR_INVALID_VALUE`, `ERROR_UNKNOWN`

`errorMessage: string` A message describing the error