Raspberry Pi Cloud: Educational Web Application

James Burke - jamesjburke91@gmail.com

September 21, 2013

1 Introduction

This document contains details upon the Raspberry Pi Educational Tool and what was completed over the course of the summer internship. It details information on design, how to execute, and a number of known problems in which there wasn't enough time to fix. The wiki section on the bitbucket repository might contain some useful information or links as well.

Please don't hesitate to contact me if you have any questions regarding any aspect of it's implementation.

2 Design

2.1 Web App

The Web Application has been implemented using Node JS alongside the Node JS packages Express (`http://expressjs.com/`) and faye-websocket (`https://github.com/faye/faye-websocket-node`).

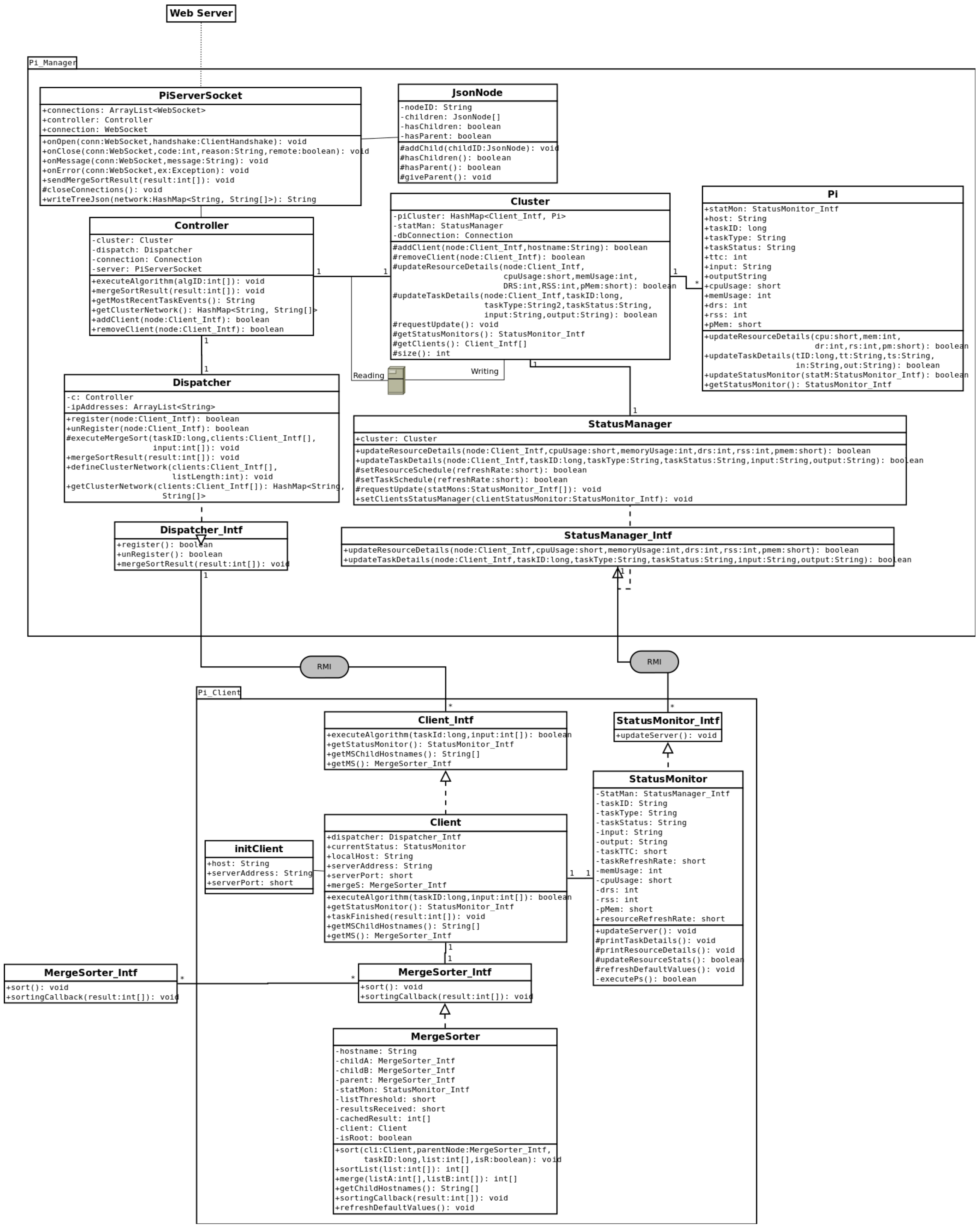
Express was used as it removes the necessity for writing a large amount of boiler-plate code, and handles hosting of the web server incredibly easily. Faye-Wesocket is an easy-to-use package to handle communication across a websocket. "Socket.io" seems to be the most popular package to use for sockets in Node js, however I have a lot of trouble co-ordinating it with a Java socket package.

A package called Jade (`http://jade-lang.com/`) is also being used, which is a templating language. It isn't vital to the project, however it makes coding html easier. Another template engine being used is Stylus (`http://learnboost.github.io/stylus/`) which makes writing CSS code easier as well.

2.2 Pi Manager & Client

The back-end which manages all Pi's is handled through two main components, the Pi Manager and Pi Clients. Each Raspberry Pi will run one instance of a Pi Client, of which will connect and register them to an instance of a Pi Manager. The Pi Manager coordinates the execution of algorithms and handles communication between itself and the Web App (via web sockets).

The following is a detailed class diagram as to how the Pi Manager and Pi Client work, comments are also left inside the code to aid in understanding.


Web Server
Pi_Manager
PiServerSocket
+connections: ArrayList<WebSocket>
+controller: Controller
+connection: WebSocket
+onOpen(conn:WebSocket,handshake:ClientHandshake): void
+onClose(conn:WebSocket,code:int,reason:String,remote:boolean): void
+onMessage(conn:WebSocket,message:String): void
+onError(conn:WebSocket,ex:Exception): void
+sendMergeSortResult(result:int[]): void
#closeConnections(): void
+writeTreeJson(network:HashMap<String, String[]>): String
JsonNode
-nodeID: String
-children: JsonNode[]
-hasChildren: boolean
-hasParent: boolean
#addChild(childID:JsonNode): void
#hasChildren(): boolean
#hasParent(): boolean
#giveParent(): void
Controller
-cluster: Cluster
-dispatch: Dispatcher
-connection: Connection
-server: PiServerSocket
+executeAlgorithm(algID:int[]): void
+mergeSortResult(result:int[]): void
+getMostRecentTaskEvents(): String
+getClusterNetwork(): HashMap<String, String[]>
+addClient(node:Client_Intf): boolean
+removeClient(node:Client_Intf): boolean
Cluster
-piCluster: HashMap<Client_Intf, Pi>
-statMan: StatusManager
-dbConnection: Connection
#addClient(node:Client_Intf,hostname:String): boolean
#removeClient(node:Client_Intf): boolean
#updateResourceDetails(node:Client_Intf, cpuUsage:short,memUsage:int, DRS:int,RSS:int,pMem:short): boolean
#updateTaskDetails(node:Client_Intf,taskID:long, taskType:String2,taskStatus:String, input:String,output:String): boolean
#requestUpdate(): void
#getStatusMonitors(): StatusMonitor_Intf
#getClients(): Client_Intf[]
#size(): int
Pi
+statMon: StatusMonitor_Intf
+host: String
+taskID: long
+taskType: String
+taskStatus: String
+ttc: int
+input: String
+outputString
+cpuUsage: short
+memUsage: int
+drs: int
+rss: int
+pMem: short
+updateResourceDetails(cpu:short,mem:int, dr:int,rs:int,pm:short): boolean
+updateTaskDetails(tID:long,tt:String,ts:String, in:String,out:String): boolean
+updateStatusMonitor(statM:StatusMonitor_Intf): boolean
+getStatusMonitor(): StatusMonitor_Intf
1
1
1
*
1
1
Reading
Writing
1
1
Dispatcher
-c: Controller
-ipAddresses: ArrayList<String>
+register(node:Client_Intf): boolean
+unRegister(node:Client_Intf): boolean
#executeMergeSort(taskID:long,clients:Client_Intf[], input:int[]): void
+mergeSortResult(result:int[]): void
+defineClusterNetwork(clients:Client_Intf[], listLength:int): void
+getClusterNetwork(clients:Client_Intf[]): HashMap<String, String[]>
StatusManager
+cluster: Cluster
+updateResourceDetails(node:Client_Intf,cpuUsage:short,memoryUsage:int,drs:int,rss:int,pmem:short): boolean
+updateTaskDetails(node:Client_Intf,taskID:long,taskType:String,taskStatus:String,input:String,output:String): boolean
#setResourceSchedule(refreshRate:short): boolean
#setTaskSchedule(refreshRate:short): boolean
#requestUpdate(statMons:StatusMonitor_Intf[]): void
+setClientsStatusManager(clientStatusMonitor:StatusMonitor_Intf): void
Dispatcher_Intf
+register(): boolean
+unRegister(): boolean
+mergeSortResult(result:int[]): void
1
StatusManager_Intf
+updateResourceDetails(node:Client_Intf,cpuUsage:short,memoryUsage:int,drs:int,rss:int,pmem:short): boolean
+updateTaskDetails(node:Client_Intf,taskID:long,taskType:String,taskStatus:String,input:String,output:String): boolean
1
RMI
RMI
Pi_Client
*
*
Client_Intf
+executeAlgorithm(taskId:long,input:int[]): boolean
+getStatusMonitor(): StatusMonitor_Intf
+getMSChildHostnames(): String[]
+getMS(): MergeSorter_Intf
StatusMonitor_Intf
+updateServer(): void
StatusMonitor
-StatMan: StatusManager_Intf
-taskID: String
-taskType: String
-taskStatus: String
-input: String
-output: String
-taskTTC: short
-taskRefreshRate: short
-memUsage: int
-cpuUsage: short
-drs: int
-rss: int
-pMem: short
+resourceRefreshRate: short
+updateServer(): void
#printTaskDetails(): void
#printResourceDetails(): void
#updateResourceStats(): boolean
#refreshDefaultValues(): void
-executePs(): boolean
initClient
+host: String
+serverAddress: String
+serverPort: short
Client
+dispatcher: Dispatcher_Intf
+currentStatus: StatusMonitor
+localHost: String
+serverAddress: String
+serverPort: short
+mergeS: MergeSorter_Intf
+executeAlgorithm(taskID:long,input:int[]): boolean
+getStatusMonitor(): StatusMonitor_Intf
+taskFinished(result:int[]): void
+getMSChildHostnames(): String[]
+getMS(): MergeSorter_Intf
1
1
1
1
MergeSorter_Intf
+sort(): void
+sortingCallback(result:int[]): void
*
*
MergeSorter_Intf
+sort(): void
+sortingCallback(result:int[]): void
MergeSorter
-hostname: String
-childA: MergeSorter_Intf
-childB: MergeSorter_Intf
-parent: MergeSorter_Intf
-statMon: StatusMonitor_Intf
-listThreshold: short
-resultsReceived: short
-cachedResult: int[]
-client: Client
-isRoot: boolean
+sort(cli:Client,parentNode:MergeSorter_Intf, taskID:long,list:int[],isR:boolean): void
+sortList(list:int[]): int[]
+merge(listA:int[],listB:int[]): int[]
+getChildHostnames(): String[]
+sortingCallback(result:int[]): void
+refreshDefaultValues(): void

2.3 Database

A small database is required for the Pi Manager and Pi Client to operate and is implanted with mysql. Cluster.java writes to the database, creating a history of 'events' that have occurred during an execution of an algorithm (a 'task'), where Controller.java reads from it and send it on to the Web App.

The schema for the database is as follows:

```
table Task:
 +---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| task_id | bigint(20)  | NO   | PRI | NULL    |       |
| type    | varchar(25) | NO   |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+


table Event:
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| task_id          | bigint(20)  | NO   | MUL | NULL    |       |
| status           | varchar(50) | YES  |     | NULL    |       |
| input            | text        | YES  |     | NULL    |       |
| output           | text        | YES  |     | NULL    |       |
| timestamp        | bigint(20)  | NO   | PRI | NULL    |       |
| ip               | varchar(16) | NO   | PRI |         |       |
| percentageMemory | tinyint(4)  | YES  |     | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
```

3 Execution

3.1 Pi Manager

The Pi Manager is executed through the following commands (these are all contained in the "runManager" script':

```
export CLASSPATH=/pathto/project/raspberryPiCloud/resources/java_websocket.jar:$CLASSPATH
export CLASSPATH=/pathto/project/raspberryPiCloud/resources/mysql-connector-java-5.1.26-bin.jar:$CLASSPATH
export CLASSPATH=/pathto/project/raspberryPiCloud/resources/gson-2.2.4.jar:$CLASSPATH
java -Djava.security.policy==tempPolicy -Djava.rmi.server.hostname=192.168.100.100 pi_cloud.piManager.Controller
```

Each of the export commands exports a .jar file that is required by the Pi Manager. These are all contained inside the "resources" folder, and details on their purpose can be found in comments within the code.

"-Djava.security.policy" specifies the rmi policy used, and "java.rmi.server.hostname" specifies the IP address the server will use. Ensure that this IP address matches the IP that each client tries to connect to.

With these commands executed the clients can now register to the Pi Manager and the web server can now establish a connection to it.

3.2 Client

A client can be executed by simply entering the following command:

```
java pi_cloud.piClient.initClient
```

3.3 Web Server

The web server can be executed by entering the following command. Ensure that the Pi Manager is active before doing so.

```
node app.js
```

4 Known Issues

There are a number of issues which are currently unsolved, however most are minor:

4.1 Web Server

1. **Global list containing the user-entered list is not thread-safe.** Because of this, the web app is only really fit for one user to be using it at a time, which is obviously impractical. There is a variable contained in app.js called "globalList", once this is synchronised *and* each list being input from the web-app is appended to it rather than over-writing it, multiple users should be able to use the site.

2. **When executing a merge-sort task, it will need to be executed twice so as to retrieve the correct network hierarchy.** This is caused by an issue regarding sequence of events. When a merge-sort task is asked to be executed, the web-server first asks the back-end for the network's hierarchy and only afterwards will it ask to execute a merge sort. The problem is that the back-end only defines it's network hierarchy when it is asked to execute a merge-sort, and so therefore a merge-sort needs to be executed at least once for the correct network hierarchy to be displayed.

3. **When viewing the visualisation, the page needs to be refreshed.**These problems are caused by the fact that the visualisation page is being loaded *before* the merge sort task as been completed. Refreshing the page is required so that the information the merge-sort task returns is displayed. The issue regarding the merge-sort

4. **The 'Previous' button on the visualisation page will not work.** This is caused by the fact that the default values for each node isn't stored and therefore when backtracking, original/default constant isn't being loaded.

4.2 Pi Manager / Pi Client

Two major issues were found in the final days of the internship, and are as follows:

One is that the distributed merge-sort is actually being executed synchronously, just about defeating the whole purpose of it all! Callback methods are being used however not correctly. Examining MergeSorter.java and the sort method will solve the problem. It's a minor fix, but a major problem.

The final one is one which I'm really unsure of the issue. The application will consistently work correctly with 3 or less clients, however when 4 or more are used it starts to behave very unpredictably. Events written to the database are in an odd order, and sometimes a node in the visualisation isn't the same as one written to the database. I suspect this might be to do with the order callback methods are being made, however I'm really unsure. Apologies to anybody who has to work this one out.

5 Future Work

The following is a list of smaller requirements I hoped to implement but never found the time.

1. **Use only the minimum number of clients required to execute the merge sort.** Currently all clients available are included in the merge-sort algorithm, even if only 1 client is required to perform the sort;

2. **Display time-to-complete values**, comparing TTC values between a sort on a single node and a sort across multiple.

3. **Collapse / expand tree where appropriate.** When viewing the visualisation, it may be useful to only show the 3rd tier of the heirarchy when a value actually reaches it, meaning that the visualusation will expand and collapse as data as passed through it.

4. **Include system statistics in visualisation.** The amount of memory the process is using is currently being calculated, however not included in the visualisation. Details on this can be found in StatusMonitor.java and in the method executePs(). Most of the work is done for this to be included in the visualisation.