Model Reference Adaptive Control - Compute Server Fan Control

Paul Strefling

April 1, 2012

Contents

0.1 Introduction

It has been found that hard drive failure rates are a function of many parameters. However their operating temperature plays a significant role in their lifetime. A study by Google incorporated included a disk drive population exceeding 100,000 disks over the period of 9 months. Their results found that hard drives operating between 30 and 40 degrees had an annual failure rate less than half that of temperatures above 45 degrees Celsius or below 30 degrees Celsius.

The motivation behind this project is to develop a fan controller to regulate hard drive temperatures regardless of manufacturer, utilization, installation, ambient temperature or fan specifications. Specifically this controller has the ability to regulate the temperatures of a Redundant Array of Independent Disks (RAID) regardless of the number of disks installed.

0.2 Obtaining the Code

The code for this project is hosted using the services provided by github.com. To obtain the source code, execute the following command on your machine

```
git clone git://github.com/strefli3/Fan-Controller.git
```

0.2.1 What is contained in the Repository

There are several directories in this repository that contain code and documentation for this particular project. Here is a brief overview of the structure:

1. **AVR_Code**

   Code that is executed on the AVR Atmega128A microcontroller. The control logic or Model Reference Adaptive Control strategy is captured in this code.

2. **Bootloader**

   A bootloader is provided with this project such that the firmware on the microcontroller can be updated with out the need of an ICSP and thus can be performed remotely.

3. **Python_Daemon**

   This is the daemon which runs on the host system. It extracts hard drive temperatures, formats them into a readable string, and handles all the communications with the microcontroller.

4. **Documentation**

   You would find this document, and documents like it in this directory.

0.3 Tailoring the Daemon to your Configuration

As previously mentioned the daemon extracts hard drive temperatures, formats them into a readable string, and handles all the communications with the microcontroller.

After obtaining the code from the git repository copy the daemon to an acceptable location on the host system, for example:

```
mkdir /usr/local/MRAC
cp -r ./Python_Daemon/* /usr/local/MRAC/
```

There is only a single file that needs to be edited to tailor this daemon to your system. This file, hdd_temp.py obtains the hard drive temperatures, formats them into an array and returns said array.

0.3.1 Hard Drive Locations

The following discussion will follow the example of a typical Rackables Storage sever which contains 16 hard drives as shown in Table 1. Hard drive Temperatures are stored in an array of the following format

Table 1: Typical Hard Drive Configuration

Fan Bank 1	Fan Bank 2	Fan Bank 3	Fan Bank 4
HDD 01	HDD 05	HDD 09	HDD 13
HDD 02	HDD 06	HDD 10	HDD 14
HDD 03	HDD 07	HDD 11	HDD 15
HDD 04	HDD 08	HDD 12	HDD 16

```
Temps[Fan_Bank][HDD]
```

For example, if */dev/sda* was in the position of *HDD* 09 then the following assignment would reflect this, taking into account that arrays start at 0 not 1:

```
self.Temps[2][0]=int(TTable['/dev/sda'])
```

0.3.2 Obtaining Hard Drive Temperatures

Referring to hdd_temp.py hard drive temperatures are obtained using the *GNU* command *hddtemp*. Implemented inside Python, this command is executed as follows:

```
def hddget(self):

cmd='hddtemp /dev/sd[abcdefg]'
        hddCMD=Popen(cmd, shell=True, stdout=PIPE, stderr=PIPE)
      stdout, stderr = hddCMD.communicate()
```

```
        hdbuf=stdout+stderr
    return hdbuf
```

For this particular example, the command was called with the argument of 7 different hard drives, a, b, c, d, e, f, g. Change this command to reflect the amount of hard drives that you have configured.

0.3.3 Assembling the Temperatures into an Array

Again assuming that 7 hard drives are installed, the following example will place them accordingly to their physical location. Notice that you need to adjust the argument in the $if statement$ to reflect the number of physical drives.

```
if len(TTable) == 7:

        self.Temps[1][0]=int(TTable['/dev/sdd'])
        self.Temps[1][1]=int(TTable['/dev/sde'])
        self.Temps[1][2]=int(TTable['/dev/sdf'])
        self.Temps[1][3]=int(TTable['/dev/sdg'])

        self.Temps[2][0]=int(TTable['/dev/sda'])
        self.Temps[2][1]=int(TTable['/dev/sdb'])
        self.Temps[2][2]=int(TTable['/dev/sdc'])
```

It is not required to strictly follow the physical placement. For example, if $Fan0$ and $Fan1$ provided cooling for both $Bank0$ and $Bank1$, and the same for $Fan2$ and $Fan3$ then the following would be appropriate as well:

```
if len(TTable) == 7:

        self.Temps[0][0]=int(TTable['/dev/sdd'])
        self.Temps[0][1]=int(TTable['/dev/sde'])
        self.Temps[0][2]=int(TTable['/dev/sdf'])
        self.Temps[0][3]=int(TTable['/dev/sdg'])

        self.Temps[1][0]=int(TTable['/dev/sdd'])
        self.Temps[1][1]=int(TTable['/dev/sde'])
        self.Temps[1][2]=int(TTable['/dev/sdf'])
        self.Temps[1][3]=int(TTable['/dev/sdg'])

        self.Temps[2][0]=int(TTable['/dev/sda'])
        self.Temps[2][1]=int(TTable['/dev/sdb'])
        self.Temps[2][2]=int(TTable['/dev/sdc'])

        self.Temps[3][0]=int(TTable['/dev/sda'])
        self.Temps[3][1]=int(TTable['/dev/sdb'])
        self.Temps[3][2]=int(TTable['/dev/sdc'])
```

0.3.4 Setting Up Email

Setup your email address in HDD_Temp_Push.py and be sure your system can send email to external addresses. Notifications of controller failure and communication problems will be sent directly to your inbox.

0.4 Initial Testing

0.4.1 Testing with micro USB

Using a micro USB cable, attach the Fan Controller to your host machine. Ensure that the jumper just under the USB port is set to USB rather than HDD. It is shipped in this configuration.

Once plugged in, execute *dmesg* in a terminal, you should see something similar to this:

```
USB Serial support registered for FTDI USB Serial Device
ftdi_sio 6-1:1.0: FTDI USB Serial Device converter detected
usb 6-1: Detected FT232RL
usb 6-1: Number of endpoints 2
usb 6-1: Endpoint 1 MaxPacketSize 64
usb 6-1: Endpoint 2 MaxPacketSize 64
usb 6-1: Setting MaxPacketSize 64
usb 6-1: FTDI USB Serial Device converter now attached to ttyUSB0
```

0.4.1.1 Testing your Configuration

To test your configuration, open a terminal and execute the following:

```
python HDD_Temp_Push.py
```

If hdd_temp.py has been configured correctly, you can expect output similar to the following:

```
python HDD_Temp_Push.py
Opening Serial Port...
Try
00,00,00,00,37,41,41,41,40,41,40,00,00,00,00,00,36,36,36,36
Fan_Bank 1: 0, Ts: 0, u: -128.00000, Tm: 30.00000

Fan_Bank 2: 133, Ts: 41, u: 5.38787, Tm: 30.16549

Fan_Bank 3: 133, Ts: 41, u: 5.38787, Tm: 30.16549

Fan_Bank 4: 0, Ts: 0, u: -128.00000, Tm: 30.00000

F1:0.00 F2:52.16 F3:52.16 F4:0.00
```

The line $00, 00, 00, 00, 37, 41, 41, 41, 40, 41, 40, 00, 00, 00, 00, 00, 36, 36, 36, 36$ can be interpreted as follows:

$$T_{00}, T_{01}, T_{02}, T_{03}, T_{10}, T_{11}, T_{12}, T_{13}, T_{20}, T_{21}, T_{22}, T_{23}, T_{30}, T_{31}, T_{32}, T_{03}, Td_0, Td_1, Td_2, Td_3 \quad (1)$$

where T_{ij} is the temperature of the j^{th} hard drive in the i^{th} fan bank and Td_i is the desired temperature of the i^{th} fan bank. Lines of the format

```
Fan_Bank 2: 133, Ts: 41, u: 5.38787, Tm: 30.16549
```

are formatted as follows:

```
Fan_Bank 2: PWM_Value, Ts: max(T), u: Control_Input, Tm: Model_Temperature
```

where PWM_Value is between $0 - 255$, $max(T)$ is the maximum temperature of the drives in the i^{th} fan bank, $Control\_Input$ is between -128 and 127 and $Model\_Temperature$ is the current desired temperature of the array. The last line of the output,

```
F1:0.00 F2:52.16 F3:52.16 F4:0.00
```

is the duty cycle for each fan bank, $0 - 100$.

0.5 Hardware Installation

It is recommended that the following procedure be completed without the system running

Install the fan controller with the provided hardware in a suitable location. Using the provided USB cable, connect the fan controller to an internal USB header. Be certain that you are attaching it to the correct header, and that all of the pins correspond to the pins on the fan controller PCB. Consult your system board's manual. Note, pin assignment for the USB cable are printed on the PCB. Attach the power and fans to the fan controller.

0.6 Testing that hardware configuration

When the system is powered on, the fans should begin spinning, if they do not, then there is a power issue, check your connections.

Once installed and booted, open a terminal and execute the following command

```
python HDD_Temp_Push.py
```

If all is working correctly, you should expect to see output similar to that in the **Testing your Configuration** section. If you get an error, such as

```
USB1 Failed... Disabaling Serial Output
```

your USB cable is not connected correctly, please check your connections.

0.7 Final Installation

0.7.1 Installing the software as a cron job

Add the following line to your crontab by executing

```
crontab -e
```

```
 */5 * * * * /usr/local/MRAC/Fan_Watchdog.sh > /dev/null 2>&1
```

It should be noted that this software should not be ran as root, but simply as a normal user. If a normal user can successfully run the python daemon, then install the cronjob to that user's crontab.