# Topics in Software Development
# - Aspect Oriented Programming -
# Final Project Report

Vitus Lorenz-Meyer

July 21, 2006

# Chapter 1

# Introduction

This Report describes my efforts in the name of a final project in the above named class in the fall of 2005 at UTEP. The class was an introduction to the Aspect Oriented Programming paradigma that has recently emerged. Thus, the final project is meant for you to demonstrate your level of understanding of that concept.
I originally proposed to implement a logging aspect for Coral [1, 3]. The motivation was the fact that Coral's current logging facility had failed and thus been discontinued. I also proposed to try to generate logs that are more general, i.e. network-centric rather than node-centric, which is a shortcoming the old system suffered.
Aspects are seperate source files that get woven into the original source code by an aspect oriented compiler or weaver. This compiler has therefore to parse the original source code. Since I could not get the Aspect C++ compiler to like Coral I had to discard that approach. Instead I developed a seperate system completely outside of Coral which I will explain in this Report.
I think that the class taught me to not always try to fix a program from within. If there is something wrong with the logging of that particular program, maybe there is a way to write an aspect outside the original source/program to fix it. In the case of Coral I was able to fix some of the things I criticized by using that approach. I I had to name a way in which this class helped me to achieve this goal, this is the only I can come up with.

# Chapter 2

# Concept overview

Coral's former logging system implemented an approach that is know as *central warehousing*. This entails sending the logs of all participating sources towards a central place for storage (possibly in a DBMS), and then deleting them on the distinct nodes.

This is bad in two ways:
First, it generates a high amount of traffic directed at a single node. Second, it demands storage for all that traffic to be available at that node. If the data is to be stored in a DBMS it might end up grinding the DBMS to a halt just by the sheer amount of data (rows).
That is why I have taken the approach of leaving all the data at the edges of the network (the nodes) and sending just summaries to certain points in the net on demand. This cuts down traffic significantly if demand is low enough. Even if the information being sent around in this alternative equals the amount of the central warehousing approach it is not going to generate the same amount of network traffic, because only summaries are sent over the wire.
To be able to achieve this requests for certain data need to be delivered to the edges of the network, and digests need to be sent back to the requesters. While theses summaries are *en-route* they are aggregated at crossroads to further summarize the data. The natural structure to easily achieve this is a tree.
Consequently, the infrastructure concept that we created facilitates and at the same time relies on a tree. The model is the following:
The user supplies three or less executables (could be scripts) to the system, along with a list of IPs (and optionally a username for ssh). The first one of those is executed on each node on the IP list to produce the data of interest. If it needs to be summarized rather than just concatenated, the second script is executed at the crossroads of the data (internal nodes in the tree, because the data moves up the tree) and is responsible for producing a digest of the inputs of all children (two in most cases). Should the data need post-processing the third script is supplied to do that, for example in the case of averaging a value, which is not a prefix operation.

This model works with all operators that are associative, because the operator can be applied to any subset without changing the outcome. Associativity means the following (for addition and real numbers): $x + (y + z) = (x + y) + z$. These operations are commonly known as *prefix operations*. Because of their nature they can be parrallelized and, ultimately, distributed.
One particular implementation of a distributed prefix infrastructure, done by the Google labs in 2004, has been called MapReduce [2]. It allows the user to supply an arbitrary executable program to reduce the input domain to a (possibly smaller) tuple space - the 'Map' phase, as well as another program to reduce exactly two of those those tuples to form a new one - the 'Reduce' phase.
Because of the similarity and inspiration we have drawn from MapReduce and our intention to work on PlanetLab [4, 5], we have nicknamed our system *PlanetLab MapReduce* (PLMR).

# Chapter 3

# Program

The first part I created was a script that takes the three executables as parameters and essentially runs a centralized version of the system. For that it uses rsync to sync a given folder or the init-executable with a list of IPs and then runs that initializer remotely. If aggregator and evaluator were given they are run locally, that is centralized. The result is the same as if it were run utilizing a tree, but it generates more network traffic and a higher load on the central node. The output is printed to stdout. The output of a normal run, executing a script that just emits the three load averages from the uptime command at each server it is run on, would look like this:

```
derDoc@~/Desktop/UTEP/classes[520]0$ ./pwhn.py -a ips.txt -D scripts/ -i init.php -u utep_1
---- vxargs rsync ----
/tmp/pwhn-out exists. Continue will destroy everything in it. Are you sure? (y/n) y
exit code 0: 3 job(s)
total number of jobs: 3
---- vxargs ssh ----
exit code 0: 3 job(s)
total number of jobs: 3
---- Results ----
(1,2.64, 2.92, 2.91)
(1,2.11, 2.19, 2.10)
(1,1.65, 2.11, 2.34)
derDoc@~/Desktop/UTEP/classes[521]0$
```

Figure 3.1: Sample run of PLMR script

## 3.1 GUI

Then I created a GUI in Java around that script to be able to plot the results nicely. It basically lets the user select and/or edit the list of IPs and the three scripts/executables. The current version is able to run either the script or a native java implementation of it, executing all neccessary commands. It lets the user input a username for ssh, a name for the run in the results table, the

number of threads desired as well as the method of choice. The main screen after startup looks like this:

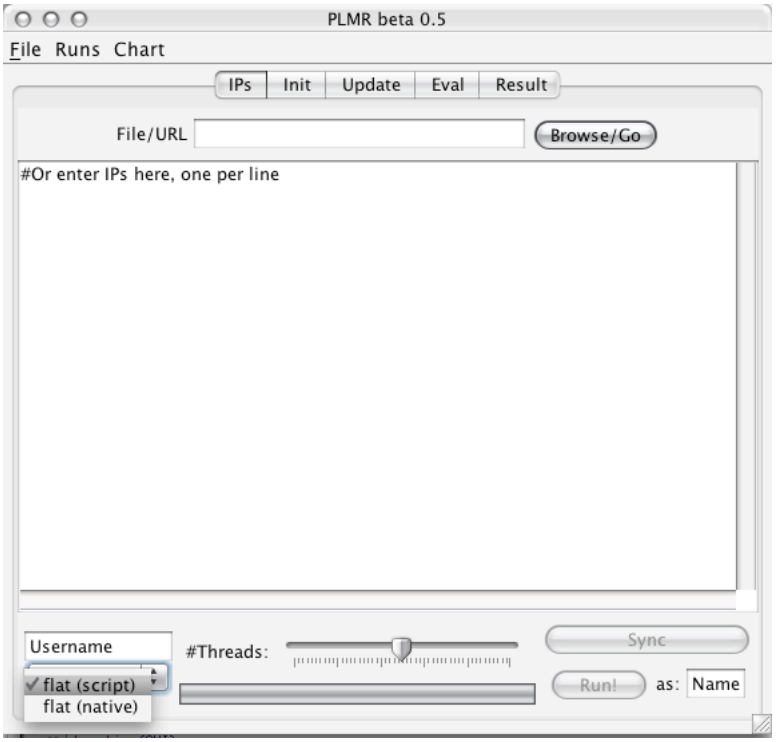


Figure 3.2: Main screen of PLMR after startup

If the output is in the right format, i.e. it is ASCII and consists of tuples enclosed in parentheses, and has either 2 or 3 elements, the first of which can be a string, PLMR is able to plot them nicely. The following picture is the plot of the results of a run that simply emits a (<hostname>,<load avg.>) tuple for each server (it was run on 3 PlanetLab servers).

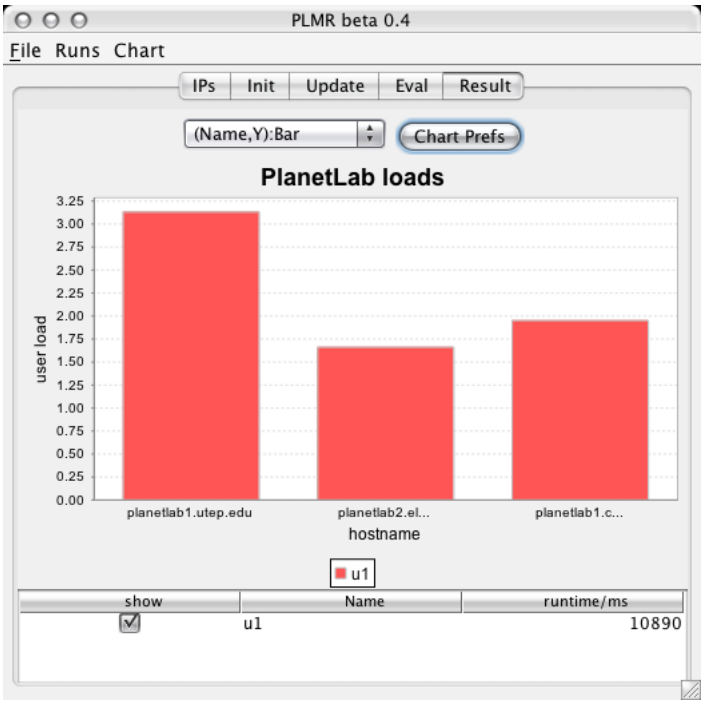


Figure 3.3: Bar-Chart Plot of Results of simple (hostname,load avg.) run

# Chapter 4

# Application to Coral

To run Coral on our share of PlanetLab servers I used PLMR to rsync a folder with the executables over and then kicked it off using a simple shell script that basically just starts the three neccessary processes. This, in my opinion, shows nicely how versatile PLMR already is, because it cannot only be used for statistics collection and plotting but also for simple things like starting a program on a set of servers.

## 4.1 Example 1

Next, I wrote scripts to get some interesting stats from coral. The first one is a script that just extracts the total webserver hits, i.e. the number of inbound requests seen to the local coral-webserver, as logged by the coral-monitor. The perl-script looks like this:

```
#!/usr/bin/env perl

$path = "/home/utep_1/PLMR/coral-bins/logs/";
$h = `hostname`;
chomp($h);
opendir(DIR,$path);
$c = 0;

while ($f = readdir(DIR)) {
        $f = $path.$f;
        if ( -f $f) {
                open(FILE,$f) || die "Can't open file $f: $!";
                while (<FILE>){
                        if (/^3#/) {
                                @s = split(/;/);
                                $c += $s[10];
                        }
```

```
                }
                close(FILE);
        }
}
closedir(DIR);
print "($h,$c)";
```

Its output could look like this (note that I only made requests to the webproxy on the planetlab-node of the coral network):

```
(planetlab2.hiit.fi,0)
(planetlab1.utep.edu,22)
(planetlab3.mnl.cs.sunysb.edu,0)
(planetlab2.elet.polimi.it,0)
(planetlab-02.ipv6.lip6.fr,0)
(planet4.cs.huji.ac.il,0)
(lsirextpc01.epfl.ch,0)
(planetlab1.csg.unizh.ch,0)
(planetlab2.csg.unizh.ch,0)
(planetlab1.win.trlabs.ca,0)
(sjtu1.6planetlab.edu.cn,0)
```

If you are just interested in the overall hits of the entire network, supply an aggregator that just ignores the hostname part and add all hits up.

Another more interesting way to show the hits per webproxy location is the following:
I wrote a small script that takes the hostname of the planetlab node and returns the <long,lat> coordinate pair for it. That way it can be plotted on a 2-d representation of the world within PLMR, because it is able to set a background picture for the plot. The dots in the following shot represent the servers, the color of the dot represents the Z-Value (the htis in this case), where the color varies from green (lowest) to red (highest - in this particular run). Note: because only planetlab has 22 hits and all others sport 0 hits, only green and red are present.

## 4.2 Example 2

Lastly, I wrote a script that takes the output of the coral-webserver which reflects all incoming and outgoing website requests. This script inspects the output, counts all hits to specific domains and emits them in the form (<domain>,<hits>) This scipt is written in Perl and looks like this:

```
#!/usr/bin/env perl

$f = "/home/utep_1/PLMR/coral-bins/wb/out";
```

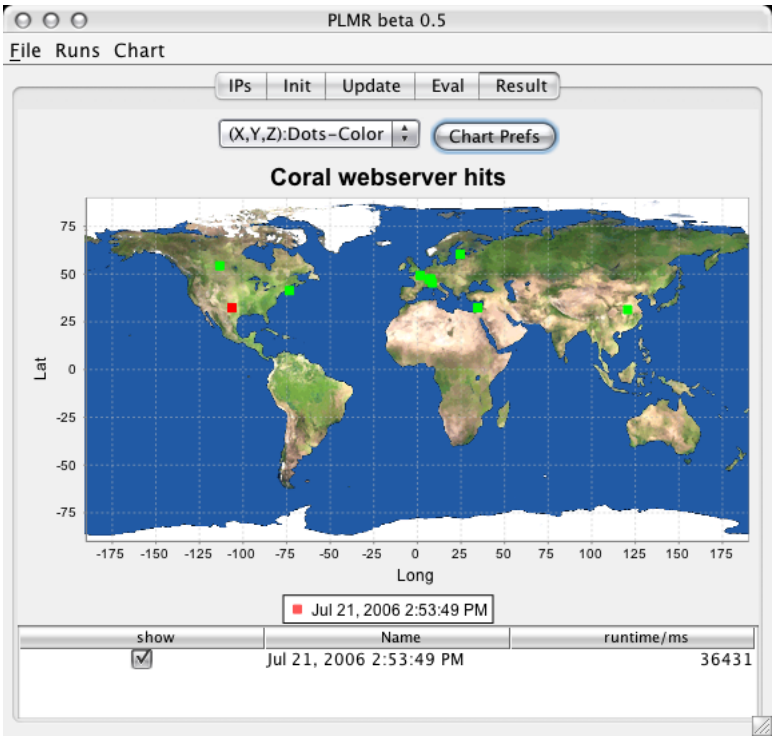


Figure 4.1: (X,Y,Z) Coordinate Plot of Webserver hits on a 2d world map

```
%hits = ();

if ( -f $f) {
        open(FILE,$f) || die "Can't open file $f: $!";
        while (<FILE>){
                if (/"IN"/) {
                        @s = split(/,/);
                        if ($s[4] =~ /"(http:\/\/)?([^\/]+)\/.*"/){
                                $hits{$2}++;
                        }
                }
        }
        close(FILE);
}

foreach $key (keys(%hits)){
        printf "(%s,%s)\n", $key, $hits{$key};
}
```

An output from the node planetlab1.utep.edu could look like this:

```
(www.google.com,10)
(www.tlpd.org,2)
(www.yahoo.com,22)
(www.microsoft.com,5)
(www.faz.net,9)
```

To add the hits per domain up, I wrote a ruby script that does exactly that, it takes two of the inputs from the initializer adds all hits per domain up and emits them. Here it is:

```
#!/usr/bin/env ruby
si = IO.new(0,"r")
i  = si.readline.chomp.to_i
f = si.read(i)
si.read(1)
i  = si.readline.chomp.to_i
s = si.read(i)
hits = { }

pr = proc { |line|
        v = line[1,line.size-2].split(/,/)
        if (hits[v[0]] == nil)
                hits[v[0]] = 1
        else
                hits[v[0]] += v[1].to_i
        end
}

f.split(/\n/).each &pr
s.split(/\n/).each &pr

hits.each { |k,v|
        printf "(%s,%i)\n", k,v
}
```

# Chapter 5

# Conclusion

In conclusion I can say that I did not accomplish what I set out to do, but was still able to implement something of interest and use to the community. I expect researchers who use PlanetLab regularily will be teased by the Coral results and may want to start to use PLMR for their own projects. It also is a great step for me in the undertaking of my thesis project. Thus, the time spent was not in vain.

# Bibliography


[1] CoralCDN. http://www.coralcdn.org/. 1

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, Calif., Dec. 2004. 3

[3] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, Mar. 2004. 1

[4] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, Oct. 2002. URL citeseer.ist.psu.edu/peterson02blueprint.html. 3

[5] PlanetLab. http://www.planet-lab.org. 3