# Ichnaea: Statistical Parallel Profiling for Clusters

Matthew Redmond
Massachusetts Institute of Technology
mrdmnd@mit.edu

16 Dec 2011

## Abstract

In this paper, the current state of hardware profiling on networked clusters is examined in some depth, with some conjecture about the near future. Next, an implementation of a simple Unix command-line tool *Ichnaea* is given. Ichnaea automates the process of cluster-wide profiling. Finally, we examine opportunities for extension and further work.

## Profiling Background

Traditionally, a *profiler* is a piece of code that runs alongside a target program while logging the interesting actions (for some user-defined concept of interesting action) of the target program. Many profilers also include a tool that generates a human-readable report from these log files.

There is an important distinction to be made between *deterministic* profilers and *statistical* profilers: a deterministic profiler records each action taken by a program (variables initialized, memory allocations, function calls, exceptions, etc.) with full precision and consistency, whereas a statistical profiler uses a sampling-based method to log snapshots of the stack frame and instruction pointer during program execution.

Profilers have historically been used in a number of ways, but the two most common use cases are debugging and performance optimization. Debugging profilers are (by necessity) deterministic, but there exist both deterministic and statistical profilers for optimization. It is usually the case that deterministic profilers incur greater overhead costs than statistical

profilers (roughly 5%-8% overhead with a deterministic profiler, compared to 0.1% for a statistical profiler) due to the amount of instrumentation required for determinism, so for time-critical programs that need to be profiled, choosing a statistical profiler is often the best decision. On the other hand, a notable disadvantage to statistical profiling is that if multiple programs are running at the same time, the profiler has no way of separating them - the results will include samples from both programs. This is particularly noticeable when run on a cluster - other users may interfere with a particular profiling session.

The class of deterministic profilers includes the Python profilers cProfile [1] and hotshot [2], as well programs like Valgrind [3] that are run on binaries compiled with debug symbol info in GCC [4]. Some examples of statistical profilers are OProfile [5], Shark (OS X) [6], perf [7], vTune (Intel) [8] and gprof [9]. Most of these profilers are able to generate *call-graph* output data, which is an abstract representation of a program's execution as a directed graph with nodes containing binary symbol data, and edges between parent nodes and their callees with weights proportional to the call counts (or sample counts, for a statistical profiler).

A technical detail that merits some discussion is how the profiler associates binary symbol data with source code. In most Unix-based systems, the Unix kernel itself has a set of functions with some built-in symbol tables that must be extracted with various ELF binary tools before they can be identified. Before the profiler can present a nicely formatted output, it must look up the names of the functions associated with the samples it took. This is adequate for most compiled or interpreted languages, but presents a unique challenge for JIT-compiled languages like Java (and other JVM langauges) as well as some LLVM code (Julia), where the symbols that are executed during runtime aren't necessarily visible to the reporting tool ex post facto. Some tools have managed to work around this limitation, but as industrial use of JVM languages for large-scale data processing (Hadoop [10], Cassandra [11], HBase [12], etc.) increases, the importance of accurate data collection for JIT languages increases as well. OProfile supports JIT execution for the JVM with a simple extension.

# Ichnaea Goals

Currently, profiling tools are very well incorporated into the single-machine programming ecosystem. What's lacking is a simple extension of serial profiling tools to a networked cluster setting. A few large scale companies have attempted real-time cluster analytics (Amazon Web Services [13] and Google Wide Profiling [14] are the two largest) but these systems are too complicated for use in a small- to medium- sized cluster like the kinds found in research universities and academic institutions worldwide. What is needed is a better solution; a "unixy" command that does one simple task well: profile every machine in the cluster simultaneously, generating log files on each machine, then aggregate the local results in a MapReduce phase to produce one output file detailing the profile of the cluster on the whole. Ideally, the overhead from the profiler is minimal.

# Ichnaea Implementation

## OProfile

The tool OProfile [5] was chosen as the base profiler, because it is both very well documented and has very low overhead. OProfile is an event-driven statistical profiler with a wide range of supported kernels and processor architectures, and it's available on every major Unix distribution. OProfile allows the user to select a hardware event (supported events depend on the processor architecture, but generally include Instructions Retired, CPU Cycles, Cache Misses, Floating Point Operations, etc.) and a sampling rate value.

Whenever the processor encounters the specified hardware event, it increments the corresponding performance counter. When that performance counter reaches the rate value, OProfile takes a snapshot of the stack frame and the instruction pointer of the machine, logging both pieces of data before resetting the performance counter to zero. In this manner, the highest resolution comes from low rate values, as snapshots will be taken more frequently.

## Ichnaea Architecture

Each machine in the cluster is required to have Ichnaea installed (or be able to run the Ichnaea scripts). The head machine will need a copy of the Parallel-SSH tools [15]. Each machine will need to have passwordless ssh set up (copy the host's key to the allowed keys file on each machine). Each machine will also need a copy of OProfile. Each machine must be configured so that the invoking account has passwordless sudo access (it's possible, and might be a good idea, to create a separate user with root privileges for profiling purposes only). To achieve passwordless sudo, the fastest method is to edit the file /etc/sudoers with visudo and add the string NOPASSWD after (ALL), like so:

```
root                ALL = (ALL) ALL
some_sudoer         ALL = (ALL) ALL
my_invoker          ALL = (ALL) NOPASSWD: ALL
other_sudoer        ALL = (ALL) ALL
```

A typical workflow for profiling programs on a single machine is to turn the profiler on, run the program, turn the profiler off, and analyze the data. We take the same approach here to maintain a similar cognitive model for users, but automate each step to run in parallel across the cluster. Each machine will invoke a script to turn its individual profiler on, then run a copy of the program, invoke a script to turn off its profiler, and output a local file containing the profile log. Ichnaea will then aggregate the individual files and generate a final output report on the head machine.

### Sample Ichnaea Invocation

```
mrdmnd@barracuda:~$ ssh beowulf1.csail.mit.edu
mrdmnd@beowulf1:~$ ichnaea hosts.txt CPU_CLK_UNHALTED:50153 "julia/julia julia/test/perf.j"
mrdmnd@beowulf1:~$ cat localOutput
Counted CPU_CLK_UNHALTED events, count 50153
samples  %        app name                 symbol name
...      ...      ...                      ...
...      ...      ...                      ...
```

The first argument is a file containing the hostnames of each machine to profile (one per line). The second argument is an OProfile hardware event (must be supported by the architecture) and a count. Last argument is the command to run on each machine. If you want to invoke a program separately, or just profile the entire cluster for a fixed amount of time, you can send a sleep command instead of something more meaningful.

## Future Improvements

Ichnaea was designed and built to benchmark and diagnose the hardware performance characteristics of the Julia language [16] against the hardware performance characteristics of other languages. While it is currently able to take intra-machine measurements, an interesting extension would be to incorporate inter-machine data as a measure of communication. This could be accomplished with some sort of packet-sniffing program.

Currently, Ichnaea will not run at full capacity on Amazon EC2 instances because of the virtualization layer Amazon uses. OProfile does not have access to the performance counter data it needs, and can only run in timer interrupt mode (no hardware events). There is a patch for OProfile called XenOProfile [17] that supposedly enables the tracking of hardware events, but the author was unable to get XenOProfile functional. Enabling these hardware counters would enable full functionality of Ichnaea on EC2.

Finally, the aggregation stage is done as a serial operation for this first version of Ichnaea. If the number of machines scales too large, or the log files for each machine grow too big, it may make sense to implement the key aggregation as a MapReduce on the cluster itself. This could probably be accomplished fairly easily.

# Acknowledgements

The author would like to thank Jeff Bezanson and Alan Edelman for their help in facilitating this project, as well as the generous donation of Amazon EC2 credits and cluster time on the Beowulf and Evolution clusters.

# Code

## prelaunch.sh

```
#/usr/bin/bash
mkdir /root/.oprofile/‘hostname‘_oprofile_session_samples;
opcontrol --init;
opcontrol --no-vmlinux --session-dir=/root/.oprofile/‘hostname‘_oprofile_session_samples/ --event=$1;
opcontrol --start;
```

## postlaunch.sh

```
#/usr/bin/bash
opcontrol --stop;
opreport --session-dir=/root/.oprofile/‘hostname‘_oprofile_session_samples/ -l -D=smart -t=0.05 > ./‘hostname‘_report.txt
opcontrol --shutdown;
rm -r /root/.oprofile/‘hostname‘_oprofile_session_samples/
```

## ichnaea.sh

```
#/usr/bin/bash
mkdir local_cache
parallel-ssh -h $1 "sh prelaunch.sh $2; $3; sh postlaunch.sh;"
parallel-slurp -h $1 -L local_cache ./‘hostname‘_report.txt ‘hostname‘_report.txt
python aggregate.py local_cache $2 > output.txt
rm -r local_cache
```

## aggregate.py

```
import os
import sys
import operator

# We map keys (app name, symbol name) to sample count values in this dictionary.
d = {}

rootdir = sys.argv[1]
evt_string = sys.argv[2]

output = "Counted %s events, count %s \n" % evt_string.split(":")
output += "samples \t\t \% \t\t app name \t\t symbol name \n"

for root, subFolders, files in os.walk(rootdir):
    for filename in files:
     filePath = os.path.join(root, filename)
     with open(filePath, 'r') as f:
         for line in f[3:]
                samples, percent, appname, symbolname = line.split(" ")
                samples = int(samples)
                key = (appname, symbolname)
                val = samples
                if d[key] not None:
                    d[key] += val
                else:
                    d[key] = val

total_samples = 0

# Count total samples for percentage display.
for val in d.itervalues():
    total_samples += val

# Go through dictionary in reverse sorted order by value.
for (key, val) in sorted(d.iteritems(), reverse=True, key=operator.itemgetter(1)):
    output += "%i \t\t %f \t\t %s \t\t %s \n" % (val, 100.0*val / total_samples, key[1], key[2])

print output
```

# References


1 *cProfile*
http://docs.python.org/library/profile.html

2 *Hotshot Profiler*
http://docs.python.org/library/profile.html

3 *Valgrind*
http://valgrind.org/

4 *GCC Debugging*
http://gcc.gnu.org/onlinedocs/gcc/Debugging-Options.html

5 *OProfile*
http://oprofile.sourceforge.net/news/

6 *Shark*
http://developer.apple.com/technologies/tools/

7 *perf*
https://perf.wiki.kernel.org/

8 *VTune*
http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/

9 *gprof*
http://sourceware.org/binutils/docs/gprof/

10 *Hadoop*
http://hadoop.apache.org/

11 *Cassandra*
http://cassandra.apache.org/

12 *HBase*
http://hbase.apache.org/

13 *Amazon Web Services*
http://aws.amazon.com/

14 *Google Wide Profiling*
http://research.google.com/pubs/archive/36575.pdf

15 *Parallel-SSH*
http://code.google.com/p/parallel-ssh/

16 *Julia Language*
https://github.com/JuliaLang/julia

17 *XenOProfile*
http://xenoprof.sourceforge.net/