The HASE++ Experimentor

Fred Howell

March 4, 1997

1 Introduction

Making repeated runs of a simulation, varying the input parameters and plotting graphs of some output parameters is a standard requirement of simulation. However this aspect is not generally well supported, and most users end up writing their own scripts to do this simple task.

This guide describes a simple set of perl scripts for experimenting with hase++ simulation code and generating GNUplot graphs. The scripts could be modified to cope with another simulation environment.

2 The Problem

The problem is to run the simulation with varying input parameters. These are typically set using either command line options or an input parameter file - for HASE++ a file called "input.params" is used. An example is shown below:-

```
100 (pspeed)
100 (nspeed)
100 (mspeed)
10 (nmsgs)
128 (nprocs)
4 (workload)
1 (memmodules)
```

The simulation is started using the command line:-

```
[calves]fwh: run_sim
```

This reads in the "input.params" file, runs the simulation and produces a trace file. The simulation code is responsible for collating results on its own performance, and each entity must write the final result to the trace in the format:

```
sim_trace(1,"C RESULTS total_time %f packets %d\n",total_time, packets);
```

As many "name value" results as needed may be added to this trace line.
Running the simulation produces a trace file looking like:-

```
u:Node1 at 19600.000000: C RESULTS total_time 19600.000000 read_time 19600.000000
u:Node2 at 20400.000000: C RESULTS total_time 20400.000000 read_time 20400.000000
u:Node0 at 22800.000000: C RESULTS total_time 22800.000000 read_time 22800.000000
u:Node3 at 24800.000000: C RESULTS total_time 24800.000000 read_time 24800.000000
u:Mem0 at 24800.000000: C RESULTS utilisation 99.596774
u:Mem1 at 24800.000000: C RESULTS utilisation 99.596774
u:Mem2 at 24800.000000: C RESULTS utilisation 99.596774
u:Mem3 at 24800.000000: C RESULTS utilisation 100.000000
```

3 The tools

There are two phases to experimentation; running the simulation N times, and gathering the required statistics from the runs. These are dealt with by `exp.perl` and `results.perl`.

3.1 exp.perl

exp.perl runs the simulation N times, varying the input parameters across a range. It uses an input file named "range.src" in the format:

```
exp2
run1
nprocs 16 64 8
memmodules 1 4 1
```

Line 1 (`exp2`) is the name of the simulation executable. Line 2 (`run1`) is the name of the directory to place results in. After this are the input parameters to vary (from input.params). In this case `nprocs` varies from 16 to 64 in steps of 8, and `memmodules` varies from 1 to 4 with unit increments.

Running `exp.perl` causes the executable to be run repeatedly, each time modifying the `input.params` file, and each time copying the resulting trace-file into a new subdirectory. For the above example, the following directory structure will be set up:

```
run1/
run1/16/1/tracefile
run1/16/2/tracefile
run1/16/3/tracefile
run1/16/4/tracefile
run1/24/1/tracefile
run1/24/2/tracefile
run1/24/3/tracefile
run1/24/4/tracefile
...
```

```
run1/64/1/tracefile
run1/64/2/tracefile
run1/64/3/tracefile
run1/64/4/tracefile
```

To extract some useful data from all these tracefiles, the next utility is used.

3.2 results.perl

`results.perl` reads in the `range.src` file used by `exp.perl`, and also requires an extra file, `collate.src` which specifies which results from the tracefiles to extract.

An example is:-

```
tracefile
Node[0-4] total_time.avg total_time.max read_time.avg read_time.max
Mem[0-4] utilisation.avg
```

The first line specifies the name of the tracefile (in this case "tracefile"). The next lines have the format (node name; list of variable names). This example calculates the average `total_time` across entities Node0 - Node4. The node name used is a "regular expression", so wildcards may be used to match several entity names. It also calculates the maximum `total_time` and the average and maximum `read_time`. The valid suffixes to the variable names (as given in the RESULTS lines of the tracefile) are: `.min, .max, .avg, .sum`.

The result of running "results.perl" *from within the experiment directory - "run1" in the example above* is the following:-

```
16/                      Mem[0-4].nprocs.gp        collres.tmp
32/                      Node[0-4].memmodules.gp   input.params
48/                      Node[0-4].nprocs.gp       range.src
64/                      collate.data
Mem[0-4].memmodules.gp   collate.src
```

The data file is in `collate.data`, and GNUplot files to display this are in the `.gp` scripts. To display these, start gnuplot and type "`load 'Mem[0-4].nprocs.gp'`" - a sample session is given below, and the resulting graph is shown in figure 1.

```
[calves]fwh: gnuplot

        G N U P L O T
        unix version 3.5
        patchlevel 3.50.1.17, 27 Aug 93
        last modified Fri Aug 27 05:21:33 GMT 1993

        Copyright(C) 1986 - 1993   Colin Kelley, Thomas Williams

        Send comments and requests for help to info-gnuplot@dartmouth.edu
```

```
        Send bugs, suggestions and mods to graphics

Terminal type set to 'x11'
gnuplot> load 'Node[0-4].memmodules.gp
```

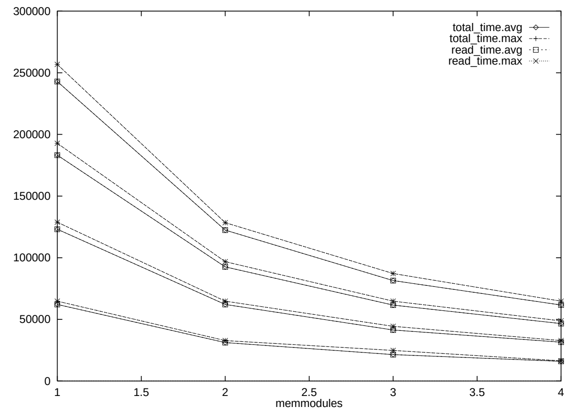

Figure 1: Sample GNUplot graph.

For completeness, an example Gnuplot script file is given below:-

```
set xlabel 'mspeed'
set ylabel ''
plot \
'collate.data' using 2:3 title 'total_time.avg' with linespoints,\
'collate.data' using 2:4 title 'total_time.max' with linespoints
```

4 The user interface

A Motif front end for controlling `exp.perl` is available. It reads in the `input.params` and `range.src` files (if present), displays them in a grid of boxes to allow the user to enter the ranges, and has a "Go" button to run the simulations.

One for controlling `results.perl` would also be useful. This would have (entity name), (variable name).suffix pairs. These could be derived from a single tracefile, or could be typed in manually. This could automate the calling of GNUplot too.

5 Accessing global parameters from the simulation

5.1 From HASE

These are read in automatically by the initialisation code.

5.2 From HASE++

This must be done manually at present. There is a general class in /home/fwh/emin/workloads/interface.c which loads and accesses input.params values. This should be a standard part of the Hase++ package.

6 Future plans

- Parallel experimentation.
- 3D graphs.