ECE 313 MW: EWS Measurements

In this experiment, you will use `vmstat` to collect some performance statistics on the engineering workstations while running the SPEC '95 benchmarks. System administrators primarily use `vmstat` to monitor system memory use; in particular, `vmstat` can be used to identify memory bottlenecks and, to a lesser extent, I/O bottlenecks.

`vmstat` is available on all the engineering workstations, but our experiments will be confined to the Solaris platform. The syntax for `vmstat` is:

```
vmstat interval [count]
```

where *interval* is the number of seconds between reports, and *count* is the total number of reports to generate. If *count* is not specified, then `vmstat` will continuously produce reports event *interval* seconds until you kill it. Typical output resembles:

```
$ vmstat 1

 procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr s0 s6 -- --   in   sy   cs us sy id
 0 0 0    720  1536   0   1  1  0  0  0  0  0  0  0  0  155   36  264 35  2 63
 0 0 0 304888 56680   0   1  0  0  0  0  0  0  0  0  0  137   13   20  0  0 100
 0 0 0 304888 56680   0   0  0  0  0  0  0  0  0  0  0  134   14   22  0  0 100
 0 0 0 304888 56680   0   0  0  0  0  0  0  0  0  0  0  135    9   18  0  0 100
 0 0 0 304888 56680   0   0  0  0  0  0  0  2  0  0  0  149    9   19  0  0 100
 0 0 0 304888 56680   0   0  0  0  0  0  0  0  0  0  0  137    9   17  0  1 99
 0 0 0 304888 56680   0   0  0  0  0  0  0  0  0  0  0  135    9   15  0  0 100
```

Here, `vmstat` produces a "report" every second (each line constitutes a report of system activity). You should always ignore the first line of the `vmstat` output; it contains average statistics since the system was last booted and is essentially useless.

We will walk through some of the more important numbers in the `vmstat` output. For a more thorough discussion of `vmstat`, please see the man page.

memory

> The most important figure here is the *free* column. This keeps track of the number of memory pages that are available (`vmstat` uses the term "page" to refer to 1KB even though most microprocessors use 4KB pages). This number will drop as programs `malloc()` additional memory.

page

> These numbers reflect the paging activity for the system. UNIX makes use of a swap file (a.k.a., paging file) to increase the amount of memory available to processes.
>
> The *po* column indicates the number of 1KB pages that have been swapped out to disk, presumably to free up additional physical memory for another use. Being that the engineering workstations have a fairly large amount of physical memory, paging out memory should only occur under heavy loads where there are several processes contending for the memory.

Similarly, the *pi* column reports the number of pages that have been swapped into memory from disk. Activity in the pi column does not necessarily represent a memory problem, however. Programs typically make use of memory-mapped files, so when the process first accesses the memory location mapped to the file, UNIX must page in the appropriate pages (see the `mmap()` function for more details).

It should be noted that all of these statistics in the paging section are totals for the last reporting interval (for the last second in the earlier example).

faults

in reflects the number of device interrupts that were generated during the last reporting interval (this excludes the clock interrupt, but includes things like keyboard and mouse interrupts).

sy reflects the number of system calls during the last reporting interval. When a process makes a system call, it essentially makes a call to the operating system. Whenever you call `malloc()` to allocate memory or `open()` to open a file or `select()` to block on I/O, you generally make a system call.

cs represents the number of context switches that have occurred during the last reporting interval. When the scheduler in UNIX decides to execute another process (either because the current process has blocked or because the current process exhausted its timeslice), the scheduler must reload the processor with the previous state of the process so that it may resume execution (this includes, among other things, restoring the register file). Depending on the amount of information stored in each context, this operation may become fairly expensive.

cpu

This section gives a breakdown of the average CPU activity during the last reporting period. The breakdown is along the lines of user time, system time, and idle time. System time is the amount of time that the CPU spends executing operating system and privileged code. For example, device drivers run in system mode, and the kernel code that gets executed when you make TCP/IP networking calls would fall under the system time category.

User time is the amount of time that the CPU spends executing the "ordinary" part of most programs. Most CPU-intensive programs will typically spend about 70% of their time in user mode--this is when the do all of their application-specific stuff.

When the CPU is not executing user code or system code, it is idle. A process may be idle when it is blocked waiting for I/O, for example.

1) Outline of the Experiment

You will monitor an engineering workstation's activity while it is executing the SPEC '95 floating point benchmarks. These benchmarks are an industry standard measure of floating point performance across computing platforms. As such, these benchmarks place significant stress on a computer's CPU. Their execution also has side-effects on memory usage, paging activity, disk I/O, etc.

You will be collecting your data from two sources: the Sun workstations in Grainger and the Sun workstations in Everitt. Pick one machine from each location and collect data at three different times: one set of data in the morning, one set in the afternoon, and one set in the evening. Hopefully, these three times will capture varying degrees of background usage by other students.

Each experiment will consist of the following tasks:

1. Start `vmstat` to begin taking measurements. Be sure to dump your output to a file for future analysis. A good way to do this is through the `tee` utility:

```
vmstat 1 |& tee vmstat.out
```

This will capture `stdout` to the file `vmstat.out` while still displaying `stdout` on the console.

2. Let `vmstat` execute for about 30 seconds to collect some data for the ambient system state (before starting the SPEC '95 benchmarks).
3. Start the SPEC '95 benchmarks in another window.
4. Terminate `vmstat` once the SPEC '95 benchmarks have completed.

In the next phase of this project, you will analyze the data that you collect using some of the concepts learned in class. We will ask you to define several random variables from the data (such as the number of bytes paged in per second when paging occurs, etc.). You will be asked to plot the distributions for these random variables using the data you have collected and calculate probabilities of certain events from these distributions. Conditional probability will also be used to better understand the workstation's performance given certain background conditions.

2) Running the SPEC '95 Benchmarks

The SPEC '95 benchmarks are installed on the Sun EWS workstations and are available for your use. The setup for the benchmarks is rather peculiar—the SPEC scripts expect their files to be in particular locations within its directory tree. SPEC intended the benchmarks to be run from a single location by a single user—not in a centralized location accessed by several students. To get around this, an EWS administrator created a script that you must run to setup a partial installation of SPEC on the `/tmp` directory exclusively for your use (this tree will contain appropriate links back to the central SPEC distribution for read-only data).

To access the SPEC '95 benchmarks, type `spec` from the Sun workstation. You will see the following message:

```
Sun Microsystems
installing the spec benchmarks in your working directory...
this should take about a minute.  please hang on...
```

Take this time with a grain of salt—these benchmarks are *very* big and copying the appropriate will take quite some time (much longer than a minute in my experience). After `spec` completes, you will be placed in your working directory (off the `/tmp` path). Executing the SPEC benchmark requires yet another script: `runspec`. You should use the following command to execute one iteration of the floating point benchmarks.

```
runspec -a validate -T base -i ref -n 1 -o asc fp
```

Executing all the floating point benchmarks takes about one hour (that's right—one hour). The upshot is that doing this is rather low maintenance on your part. Just start the script and you're essentially done (as long as you've got `vmstat` collecting data in the background). You should see the following lines as `runspec` gets everything ready to run; then, each benchmark (`101.tomcatv, 102.swim,` etc.) will execute in succession.

```
Including file sut.inc
Identifying formats
Checking what benchmarks are around
Verifying Benchmark Sets
We will use: 101.tomcatv, 102.swim, 103.su2cor, 104.hydro2d,
107.mgrid, 110.applu, 125.turb3d, 141.apsi, 145.fpppp, 146.wave5
Checking binaries base base
```

```
Checking input type 'ref'
Setting up working directories
Setting up 101.tomcatv ref base base
Setting up 102.swim ref base base
Setting up 103.su2cor ref base base
Setting up 104.hydro2d ref base base
Setting up 107.mgrid ref base base
Setting up 110.applu ref base base
Setting up 125.turb3d ref base base
Setting up 141.apsi ref base base
Setting up 145.fpppp ref base base
Setting up 146.wave5 ref base base
Running Benchmarks!
Running '101.tomcatv'
  :
  :
```

The SPEC scripts produce a final performance report in the `result/` directory (`*.asc` files). Copy these files to your own home directory and save it for the analysis. The `/tmp` directories frequently get cleaned up by the system administrator, so don't be surprised to see your private copy of the SPEC installation disappear the next time you login.

Since the benchmarks run for quite a long time, you may want to setup `vmstat` to stop taking measurements after a certain period of time (that way, `vmstat` will eventually stop even if you're not there when the benchmarks complete). You can specify the maximum number of "reports" (lines) to generate. To get it to run for 90 minutes with reporting every second (recommended), you would type:

```
vmstat 1 5400
```

When you're done with your SPEC benchmark session, just type "`exit`" at the `spec:` prompt to return you to your regular shell.