Chapter 1

System Performance Reporting

1.1 Understanding System Performance Parameters

The definition of performance of a system is dependent upon the expectations from a system. For example, **low latency** is desired from *database servers*, while **high throughput** is needed from *file servers*.

Actual performance has to be judged on the basis of performance level agreements. This has to be clearly defined for anyone - "*The web server should always react within 10 seconds*" is better than "*generic load should be less than 60%*", because that's what the end user will care about!

Thus, first we need to decide upon which metrics we want to measure, and then collect baseline data for it via monitoring systems.

1.1.1 Typical Performance Focus Areas

Factor	Description
Memory	The single most important factor that affects server performance. When enough memory isn't available, swap has to be used to house the excess pages and then the IO performance suffers, thus bogging down the entire system. It even affects the network throughput.
Disk	Another very important factor in overall server performance. When the disk is slow, too much memory is wasted to buffer data that's waiting to be written to disk. Processes will also have to wait longer to access data from the disk.
Network	Network is no longer a significant bottleneck, since most network connections aren't 10Mbps anymore - enterprise infrastructure uses Gigabit connections as a standard.
CPU	While the CPU has many tunables, in general it is not a very significant factor in performance deterioration. It is only for certain workloads that CPU becomes a factor in performance. The gain from CPU optimizations can be expressed in nanoseconds.

1.1.2 Common Performance Monitoring Tools

Terms	Description
top	While it's a very basic tool, it's also very rich in features. It provides an excellent generic overview of everything going on in the system. Typical use case for `top` is to detect problems and then use a more specialized tool to diagnose further.
iostat	A dedicated tool to detect Input/Output problems. It shows statistics about I/O. To detect which process is creating a high I/O load, a valuable tool is **iotop**.
vmstat	This tool shows statistics about virtual memory usage.
sar	The **System Activity Reporter** specializes in providing long-term data about what the system has been doing and long term performance statistics.

1.2 Understanding top

This is perhaps the single most important performance monitoring utility due to the kind of data it provides. There are alternatives to top such as `htop`, but top is programmed efficiently and doesn't have too much overhead. Comparing the two - htop uses about 5 times as much system resources as top!

The first feature of interest in the output of top is the **load average**, which consists of three numbers: the load average for the last 1, 5 and 15 minutes. The load average is the average of the number of processes in a runnable state, i.e., currently being executed by the CPU or waiting for CPU, over the concerned period of time. Optimally, all CPUs should be utilized as much as possible, but no process should be waiting for the CPU. The output of the `nproc` command tells us the effective number of CPUs available (= Physical CPUs × logical cores per CPU).

The individual CPU utilization per CPU core can be shown by pressing the `1` key. A typical output is:

```
# top
%Cpu0  :  5.5 us,  3.3 sy,  0.0 ni, 90.7 id,  0.0 wa,  0.5 hi,  0.0 si,  0.0 st
```

Here, the number after the CPU indicates the core number. The *us* value refers to CPU usage in percentage in user space, i.e., by processes started by the end user without administrative privileges. The *sy* does the same, but for processes started by the users with root privileges. The *id* value is the percentage of time the processor remains idle. The next important metric is the number before *wa* which represents the waiting time, i.e., percentage of time processes spend waiting for I/O. A high value here indicates that the there's something wrong with the I/O channel and may indicate imminent disk failure.

Next, the memory statistics are shown, which includes the amount of memory completely free and amount of memory used to cache files that are frequently requested. Buffers contain data that needs to be written to disk during high I/O loads. While these are technically *non-essential*, it's suggested that 30% of the total memory be dedicated to buffers/cache usage.

We can also toggle the fields being shown by pressing the `f` key. If we quit top using the `q` key, the edits to the configuration are gone the moment we quit. However, if we quit using `Shift + W`, then the configuration is written to the `.toprc` file.

1.3 Understanding iostat

The iostat tool is a part of the `sysstat` package, which needs to be installed to use the `iostat` command. The command by itself provides a snapshot of the I/O statistics at the time of the invocation of the command. However, it takes two arguments in the syntax: `iostat <interval> <loops>`. The interval refers to the gap between displaying statistics and the loops refer to the number of times the command should show its output. Typical output for the command is:

```
# iostat 3 2
Linux 3.10.0-693.17.1.el7.x86_64 (vmPrime.somuVMnet.local)        Tuesday 27 February
 ↪  2018         _x86_64_        (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
0.50    0.00    0.64    0.49    0.00   98.37

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               1.20        54.56         3.33     584199      35622
scd0              0.00         0.10         0.00       1054          0
dm-0              1.11        51.31         3.13     549442      33537
dm-1              0.01         0.21         0.00       2228          0
sdb               0.00         0.10         0.00       1044          0
sdc               0.00         0.10         0.00       1044          0
sdd               0.00         0.03         0.00        336          0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
5.44    0.00    1.36    0.00    0.00   93.20

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.68         0.00        10.88          0         32
scd0              0.00         0.00         0.00          0          0
dm-0              2.04         0.00        32.48          0         95
dm-1              0.00         0.00         0.00          0          0
sdb               0.00         0.00         0.00          0          0
sdc               0.00         0.00         0.00          0          0
sdd               0.00         0.00         0.00          0          0
```

In the output, **tps** refers to the number of transactions per second. The *kB_read/s* and the *kB_wrtn/s* values are self explanatory. The next two columns show the total kBs read and written respectively.

1.3.1 Usage scenario

Let us consider a scenario where top shows us that processes spend 60% of their execution time waiting for I/O. Let us consider that the concerned server is connected to 6 different disks or other storage devices. We can use the output of the iostat command to determine which disk is so slow.

If we consult the output from the command, we can see that dm-0 has the greatest tps. To find out which device is dm-0, we can simply go to the `/dev/mapper` directory and see what links to it:

```
# \ls -l /dev/mapper
total 0
crw-------. 1 root root 10, 236 Feb 27 20:53 control
lrwxrwxrwx. 1 root root       7 Feb 27 20:53 rhel-root -> ../dm-0
lrwxrwxrwx. 1 root root       7 Feb 27 20:53 rhel-swap -> ../dm-1
```

1.3.2 iotop

The **iotop** command needs to be installed using `yum -y install iotop`. It shows the processes that are doing the most amount of I/O in descending order. Typical output looks like:

```
# iotop
Total DISK READ :      45.37 M/s | Total DISK WRITE :       0.00 B/s
Actual DISK READ:      45.37 M/s | Actual DISK WRITE:       0.00 B/s
TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
5696 be/4 root       45.37 M/s    0.00 B/s  0.00 % 73.97 % dd if=/dev/sda of=/dev/null
5450 be/4 root        0.00 B/s    0.00 B/s  0.00 % 12.66 % [kworker/0:2]
1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % systemd --switched-root --system
 ↪  --deserialize 21
...
```

Here we can see that the `dd if=/dev/sda of=/dev/null` is performing the most amount of I/O by copying the entire hard disk to `/dev/null`.

1.4 Understanding vmstat

1.4.1 Virtual Memory

Let us consider the typical output of top sorted on the basis of the Virtual Memory being used:

```
# top
 PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
1920 somu      20   0 1901016 215552  47856 S  0.7 11.6   0:47.79 gnome-shell
...
```

We can see that the `gnome-shell` is using 1901016 KiB of virtual memory, which is $\approx$ 1.82 GiB of virtual memory. Virtual Memory in Linux is memory that doesn't really exist. If we take a look at the `/proc/meminfo` file, we see:

```
VmallocTotal:   34359738367 kB
```

If we convert the VmallocTotal (Total amount of virtual memory that is possible for the kernel to allocate) to human readable units, it comes up to 32PB! That's not possible on most enterprise gear, let alone consumer hardware. Thus, the memory here doesn't really exist.

The key point here is that the kernel frequently needs to dish out unique memory address pointers to programs that demand it, but not actually assign any real memory till it's needed, i.e., the program tries to write to that location.

The kernel, instead of assigning real memory locations to programs, assigns memory in a virtual address space, which it then maps on to real memory on demand. The program itself remains blissfully oblivious to the knowledge of whether the memory it is referencing is virtual or real. All the trouble of fetching data on requirement and saving data falls on the kernel.

1.4.2 Resident Memory

A much more important concept is that of Resident Memory. Contrastingly to the Virtual Memory, the Resident memory is really used and is the total amount RAM being assigned to the process.

1.4.3 vmstat

The `vmstat` command when used without arguments shows various statistics pertaining to the resource consumption on the system:

```
# vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 3  0      0 269904   2116 860396    0    0   176    25  124  129  2  2 95  2  0
```

The significance of each is:

Terms	Description
proc	This part shows information about the processes: the **r** shows the number of running processes, **b** shows the number of blocking processes. A blocking process is a process that's waiting for something (e.g., I/O).
memory	This is the total amount of memory in swap, as well as real physical memory (RAM) used for buffers and cache.
swap	The two sub-categories are *swap-in*(`si`) and *swap-out*(`so`). If at any time we see that the system is utilizing swap memory, we can use `vmstat` to find out if the swap is being used actively, i.e., whether data is being written to or read from it actively.
io	The IO section deals with the number of blocks of I/O that's being performed - *blocks-in*(`bi`) and *blocks-out*(`bo`) provide a way to measure the real I/O activity at the moment, thus helping us discern if the server is spending a lot of time reading or writing during high I/O waits.
system	The metrics shown are *interrupts*(`in`) and *context switches*(`cs`). Interrupts are generally generated when a piece of hardware demands CPU attention. Context switches occur when the CPU switches the present task it's working on after being triggered by the scheduler. It is critical to the multi-tasking ability of a server since multiple processes need to coordinate and divide the CPU cycles. A high number of context-switches would indicate that the CPU isn't getting enough time per process.
cpu	These metrics refer to the percentage of CPU time spent executing programs in the *user-space*(`us`), *system space*(`sy`), *idle*(`id`) or *waiting* (`wa`).

Just like `iostat`, the `vmstat` provides an option to show the information at multiple points in time - the first argument is the time delay and the second the number of loops. To re-run vmstat every 2 seconds for 5 times we use:

```
# vmstat 2 5
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0      0 255824   2116 877492    0    0   135    20  118  122  1  1 96  2  0
 0  0      0 255576   2116 877492    0    0     0     0  174  178  6  3 91  0  0
 0  0      0 255576   2116 877492    0    0     0     0  152  145  6  2 92  0  0
 1  0      0 255384   2116 877492    0    0     0     0  179   161   8   2 90   0   0
 0  0      0 255452   2116 877492    0    0     0     0  256   406   9   8 83   0   0
```

Just like in `iostat` output, the first line has higher values of certain metrics because it gives a generic overview of the system operations where applicable. The next lines portray the activity within the delay time frame.

For detailed memory utilization statistics, we use `vmstat -s`:

```
# vmstat -s
1865964 K total memory
730672 K used memory
902516 K active memory
466968 K inactive memory
255684 K free memory
2116 K buffer memory
877492 K swap cache
1048572 K total swap
0 K used swap
1048572 K free swap
6813 non-nice user cpu ticks
1125 nice user cpu ticks
7128 system cpu ticks
543996 idle cpu ticks
8383 IO-wait cpu ticks
0 IRQ cpu ticks
180 softirq cpu ticks
0 stolen cpu ticks
734741 pages paged in
106795 pages paged out
0 pages swapped in
0 pages swapped out
667859 interrupts
685758 CPU context switches
1519798095 boot time
4208 forks
```

To change the display unit being used, we use the `-S (k/K/m/M)` option to change units. (where K=KiB and k=kB).

1.5 Understanding sar components

`sar` stands for the **System Activity Reporter**. It is a part of the *sysstat* package (like `iostat` and `vmstat`), and it collects data on an interval of 10 minutes by default. However, it can also be used to collect instantaneous data about the system as well.

What truly distinguishes `sar` from the other tools is the fact that it can be tasked to data collection for an extended period of time and the queried for information about a very specific period.

To make sorting and finding data in `sar` easier, it is recommended to set `LANG=C` before starting sar. Every Linux OS has an environment variable called **LANG** that affects the behavior of several utilities as well as setting the language. POSIX standard mandates that a locale called either POSIX or C be defined. Thus, it disables localization and makes the output predictable. Unless the option is set, timestamps are formatted in AM/PM which makes filtering said data harder. With the `LANG=C` option however, the timestamps use the military time format (24-hour format). A handy option is to generate an alias such as:

```
# echo "alias sar='LANG=C sar'" >> /etc/bashrc
```

sar data is collected via cron jobs in `/etc/cron.d/sysstat`. The collected data is written to `/var/log/sa`. The file `/etc/sysconfig/sysstat` has a `HISTORY` variable which dictates how long data should be stored. Typically, it's on a monthly basis.

1.5.1 /etc/cron.d/sysstat

This cronjob launches two different jobs are launched: **sa1** and **sa2**. The *sa1* job is for collecting short term statistics data while the *sa2* job executes once a day to collect data for long term statistics. Both of these write the results of their monitoring in a file in the `/var/log` directory.

1.5.2 /var/log/sa/sa[dd]

These are actually a bunch of files that start with the prefix `sa` and end with the date in `dd` format. Thus, typical file names are: *sa01, sa25, sa31,* etc. These files are unreadable by typical pagers like `less` and needs to be read by using the `sar` utility itself, by issuing commands like `sar -q` to get information about disk statistics, etc. One common mistake while accustoming to `sar` is to forget to start the sysstat services, since without them the data for the sar log files aren't populated and the utility has no data to work with.

1.6 Setting up sar

If the **sysstat** package isn't already installed, we first need to install it using `yum -y install sysstat`. Next, we ensure that the cron job for data collection via *sa1* and *sa2* were set up properly in `/etc/cron.d/sysstat` file, which should have the contents:

```
# Run system activity accounting tool every 10 minutes
*/10 * * * * root /usr/lib64/sa/sa1 1 1
# Generate a daily summary of process accounting at 23:53
53 23 * * * root /usr/lib64/sa/sa2 -A
```

Thus, *sa1* is collecting data every 10 mins and *sa2* executes everyday at 11:53PM to collect long term data for the day. Now sar is ready to collect data, but if we were to query the sar already, we'd come up empty, since `sar` hasn't had the opportunity to log data yet!

Next, we check the config in `/etc/sysconfig/sysstat` file, which typically looks like:

```
# sysstat-10.1.5 configuration file.

# How long to keep log files (in days).
# If value is greater than 28, then log files are kept in
# multiple directories, one for each month.
HISTORY=28

# Compress (using gzip or bzip2) sa and sar files older than (in days):
COMPRESSAFTER=31

# Parameters for the system activity data collector (see sadc manual page)
# which are used for the generation of log files.
SADC_OPTIONS="-S DISK"

# Compression program to use.
ZIP="bzip2"
```

The primary feature of interest in this file is the value of the HISTORY variable which decides how long the collected data is stored.

Now, we have to wait for the sadc (*System Activity Data Collector*) utility to collect data for **sar** to analyse.

1.7 Analyzing sar data

Some of the most common options that print certain categories of the collected data are:

1.7.1 I/O operations

The sar -b command shows us the total transfers per second (tps), read tps(rtps), write tps(wtps), blocks read per second (bread)(1 block = 512B) and blocks written to per second (bwrtn). Typical output of the command looks like:

```
# sar -b
00:00:02          tps      rtps      wtps   bread/s   bwrtn/s
00:10:01         2.70      1.15      1.55     64.96    995.40
00:20:01         0.47      0.27      0.20     12.97      8.26
00:30:01         0.07      0.00      0.07      0.00      0.86
00:40:01         0.08      0.00      0.08      0.00      1.04
...
```

Thus, the use of military time makes the output a lot easier to process.

1.7.2 Processor information

It is possible to get the information about a single processor (i.e., a single logical core on a physical CPU, since linux considers each a separate processor) using the sar -P <processorNumber> command. Typical usage is (to find the usage of processor 0):

```
# sar -P 0
00:00:02        CPU     %user     %nice   %system   %iowait    %steal     %idle
00:10:01          0      5.21      0.00      3.09      0.59      0.00     91.11
00:20:01          0      0.12      0.00      0.24      0.15      0.00     99.48
...
04:10:01          0      0.29      0.00      0.36      0.10      0.00     99.25
Average:          0      1.04      0.00      1.03      0.31      0.00     97.62
```

1.7.3 Network Statistics

The sar -n DEV command shows the network statistics for each interface:

```
# sar -n DEV
Linux 3.10.0-693.17.1.el7.x86_64 (vmPrime.somuVMnet.local)         02/28/18
 ↪          _x86_64_        (1 CPU)

00:00:02        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s
 ↪  rxmcst/s
00:10:01           lo      0.00      0.00      0.00      0.00      0.00      0.00
 ↪  0.00
00:10:01   virbr0-nic      0.00      0.00      0.00      0.00      0.00      0.00
 ↪  0.00
00:10:01       virbr0      0.00      0.00      0.00      0.00      0.00      0.00
 ↪  0.00
00:10:01        ens33     63.83     19.03     90.10      1.19      0.00      0.00
 ↪  0.00
...
16:40:01        ens33      1.28      1.13      0.15      0.36      0.00      0.00
 ↪  0.00

Average:        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s
 ↪  rxmcst/s
Average:           lo      0.00      0.00      0.00      0.00      0.00      0.00
 ↪  0.00
Average:   virbr0-nic      0.00      0.00      0.00      0.00      0.00      0.00
 ↪  0.00
Average:       virbr0      0.00      0.00      0.00      0.00      0.00      0.00
 ↪  0.00
Average:        ens33      1.19      0.92      0.83      0.12      0.00      0.00
 ↪  0.00
```

To view the statistics for just one interface, we can use `sar -n DEV \` grep <interface-Name>|:

```
# sar -n DEV | grep ens33
00:10:01        ens33     63.83     19.03     90.10      1.19      0.00      0.00
 ↪  0.00
...
16:40:01        ens33      1.28      1.13      0.15      0.36      0.00      0.00
 ↪  0.00
Average:        ens33      1.19      0.92      0.83      0.12      0.00      0.00
 ↪  0.00
```