Scenarios

Notes from

Brendan Gregg - Systems Performance_ Enterprise and the Cloud (2020, Pearson)

Common http codes

- 200 - success
- 201 - created
- 202 - accepted
- 204 - no content
- 307 - temp redirect, 308 - perm redirect
- 301 - moved permanently
- 400 - bad request
- 401 - auth needed
- 403 - auth have, but perm not enough
- 404 - not found
- 405 - method not allowed
- 418 - i'm a teapot
- 500 - server error
- 501 - not implemented
- 502 - bad gateway, proxy is contacting server but server is failing
- 503 - service unavailable (server not ready (down, maintenance))
- 504 - gateway timeout, proxy is contacting server but no response in time

Considering pstack, ptrace

How to Respond to Memory Pressure

Check running processes:

```
ps aux --sort=-%mem | head
```

1.
    - Find memory-hungry processes.

Check swap usage:

```
free -m
```

2.
    - If swap is full and memory is under pressure, consider increasing swap space.

Reduce cache memory (temporary fix):

```
echo 3 > /proc/sys/vm/drop_caches
```

3.
    - Clears page cache but doesn't fix root cause.

Tune system parameters (if necessary)

```
sysctl -w vm.swappiness=10
```

4.
    - Lowering `swappiness` makes Linux less likely to swap.

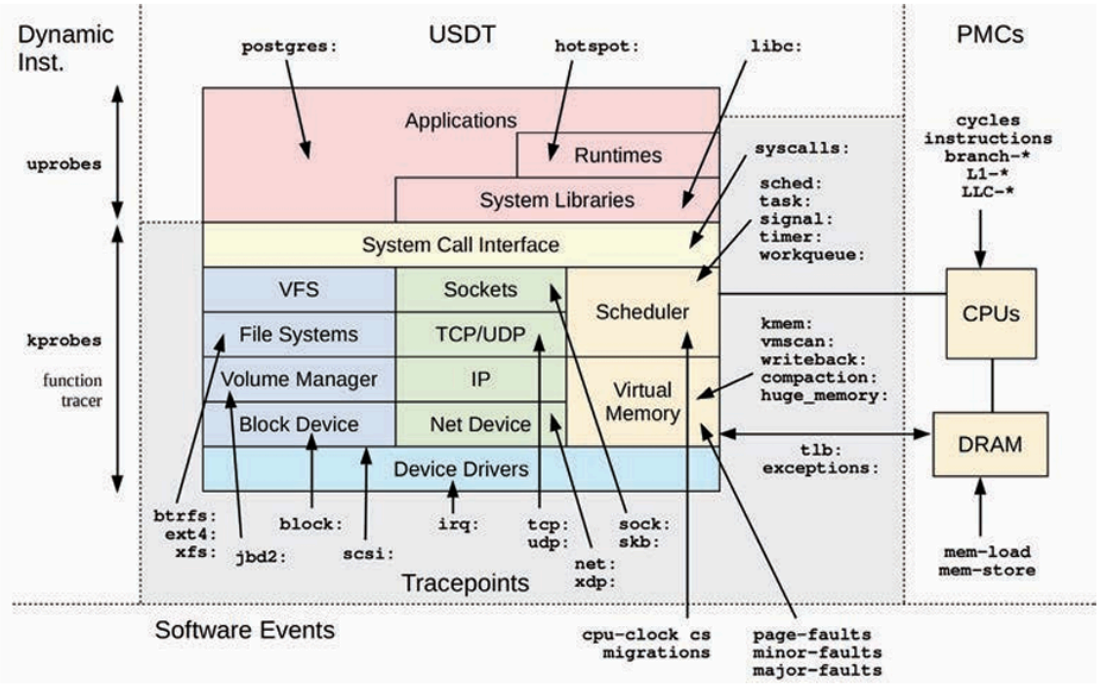
Topics to review

Table 4.2 **Linux observability sources**

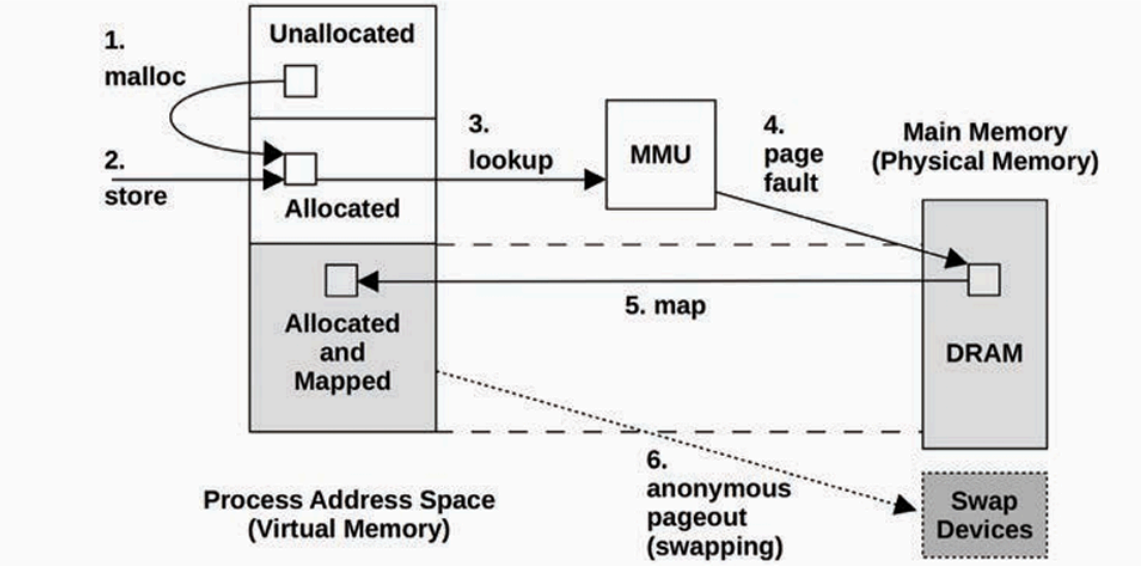
Type	Source
Per-process counters	/proc
System-wide counters	/proc, /sys
Device configuration and counters	/sys
Cgroup statistics	/sys/fs/cgroup
Per-process tracing	ptrace
Hardware counters (PMCs)	perf_event
Network statistics	netlink
Network packet capture	libpcap
Per-thread latency metrics	Delay accounting
System-wide tracing	Function profiling (Ftrace), tracepoints, software events, kprobes, uprobes, perf_event

-

points, including blk:block_rq_issue.



-
- Kprobes, uprobes
- Sys
- Cpu ticks
- Memory reclaim
- Delay accounting
- Netlink (special socket family af netlink for fetching kernel)
- Bpftrace
- -e on perf list and starace
- Pmcs - overflow sampling, availability in cloud
- Oversubscription, swap device


1.
malloc
2.
store
Unallocated
Allocated
Allocated
and
Mapped
3.
lookup
MMU
4.
page
fault
Main Memory
(Physical Memory)
5. map
DRAM
6.
anonymous
pageout
(swapping)
Swap
Devices
Process Address Space
(Virtual Memory)

4.3.10 Other Observability Sources

Other observability sources include:

- **MSRs**: PMCs are implemented using model-specific registers (MSRs). There are other MSRs for showing the configuration and health of the system, including the CPU clock rate, usage, temperatures, and power consumption. The available MSRs are dependent on the processor type (model-specific), BIOS version and settings, and hypervisor settings. One use is an accurate cycle-based measurement of CPU utilization.
- **ptrace(2)**: This syscall controls process tracing, which is used by gdb(1) for process debugging and strace(1) for tracing syscalls. It is breakpoint-based and can slow the target over one hundred-fold. Linux also has tracepoints, introduced in Section 4.3.5, Tracepoints, for more efficient syscall tracing.
- **Function profiling**: Profiling function calls (mcount() or __fentry__()) are added to the start of all non-inlined kernel functions on x86 for efficient Ftrace function tracing. They are converted to nop instructions until needed. See Chapter 14, Ftrace.
- **Network sniffing (libpcap)**: These interfaces provide a way to capture packets from network devices for detailed investigations into packet and protocol performance. On Linux, sniffing is provided via the libpcap library and /proc/net/dev and is consumed by the tcpdump(8) tool. There are overheads, both CPU and storage, for capturing and examining all packets. See Chapter 10 for more about network sniffing.
- **netfilter conntrack**: The Linux netfilter technology allows custom handlers to be executed on events, not just for firewall, but also for connection tracking (conntrack). This allows logs to be created of network flows [Ayuso 12].
- **Process accounting**: This dates back to mainframes and the need to bill departments and users for their computer usage, based on the execution and runtime of processes. It exists in some form for Linux and other systems and can sometimes be helpful for performance analysis at the process level. For example, the Linux atop(1) tool uses process accounting to catch and display information from short-lived processes that would otherwise be missed when taking snapshots of /proc [Atoptool 20].
- **Software events**: These are related to hardware events but are instrumented in software. Page faults are an example. Software events are made available via the perf_event_open(2) interface and are used by perf(1) and bpftrace. They are pictured in Figure 4.5.
- **System calls**: Some system or library calls may be available to provide some performance metrics. These include getrusage(2), a system call for processes to get their own resource usage statistics, including user- and system-time, faults, messages, and context switches.

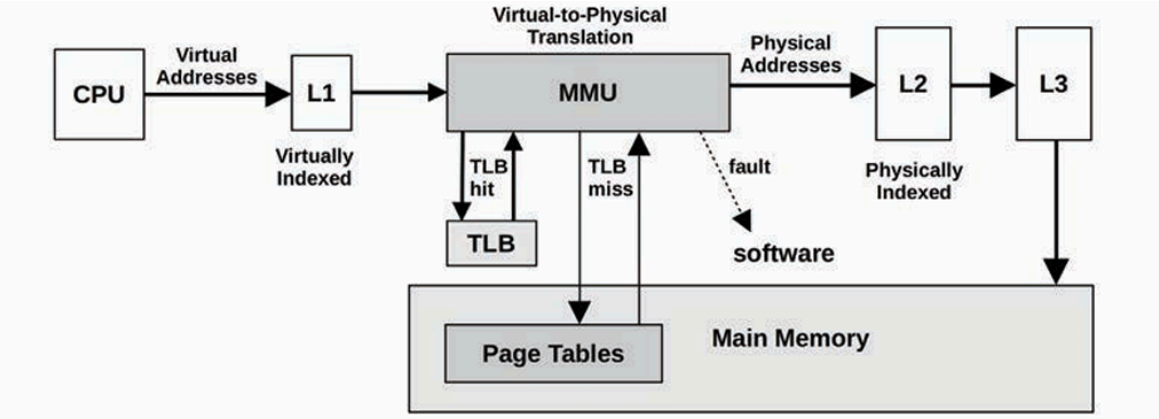
If you are interested in how each of these works, you will find that documentation is usually available, intended for the developer who is building tools upon these interfaces.

And More

Depending on your kernel version and enabled options, even more observability sources may be available. Some are mentioned in later chapters of this book. For Linux these include I/O accounting, blktrace, timer_stats, lockstat, and debugfs.

-
- Commo multithreading types
- Types of locks + midpath, etc. of spin

- Pss
- Page fault flame graph
- Rpc
-
- wss



- Memory queues
- Troubleshooting frameworks
- 8,11,13 in linux os book. Networks just copy paste toc into gpt
- Regular os by robert love - scan: 13,8,15,7, 12,16,5 copy toc into gpt: 3, 4
- Cache sizes and policies
    - Cache line size
- LLC penalties, cache coherency
- Qpi
- Floating point unit
- Optimize space (use smaller set size) and use parallelism for better time
- Common okay values for stats

Problem solving frameworks

General

- App version and dependencies? Are there newer versions? Any performance notes from creator for dependencies?
- Known performance issues? Bug db for this?
- How is app configured?
    - reasoning
- Cache? How big?
- Concurrent? If so, how is its thread pool size + config?
- Debug build on? (slower usually)
- System libraries used? What versions?
- Memory allocator used?
- App configured to use large pages for heap?

- Compiled? What are its compile options + compiler version
- Does the native code include advanced instructions? (Should it?) (For example, SIMD/vector instructions including Intel SSE.)
- Has the application encountered an error, and is it now in a degraded mode? Or is it misconfigured and always running in a degraded mode?
- Are there system-imposed limits or resource controls for CPU, memory, file system, disk, or network usage? (These are common with cloud computing.)

Network

Use

- Utilization - throughput over bandwidth, time interface was busy sending/receiving frames
- Saturating - queuing
- Errors - collisions, frame too short, bad checksum
- One direction matters most - servers are transmit heavy; clients are receive heavy
-
- Tools
    - Dropped, out of order
        - netstat/netstat -s
        - Ip - s link/netstat
    - Ss -tiepm
        - Limiter flag for important sockets to see bottleneck, show socket health
    - overruns.”
    - nicstat/ip -s link: Check the rate of bytes transmitted and received.
    - tcplife: Log TCP sessions with process details, duration (lifespan), and throughput
    - statistics.
    - tcptop: Watch top TCP sessions live.
    - tcpdump: While this can be expensive to use in terms of the CPU and storage costs. Help discover unusual network patterns
    - perf(1)/BCC/bpftrace: Inspect selec
    - High tcp retransmists as part of network saturation`
        -
        - Usage of TCP (socket) send/receive buffers
        - Usage of TCP backlog queues
        - Kernel drops due to the backlog queue being full
        - Congestion window size, including zero-size advertisements
        - SYNs received during a TCP TIME_WAIT interval
    - Tcp out of order packets`

The following are the most basic characteristics to measure:

- **Network interface throughput**: RX and TX, bytes per second
- **Network interface IOPS**: RX and TX, frames per second
- **TCP connection rate**: Active and passive, connections per second

-
- What is rx,tx`

The tools in this section are listed in Table 10.4.

Table 10.4 **Network observability tools**

Section	Tool	Description
10.6.1	ss	Socket statistics
10.6.2	ip	Network interface and route statistics
10.6.3	ifconfig	Network interface statistics
10.6.4	nstat	Network stack statistics

Section	Tool	Description
10.6.5	netstat	Various network stack and interface statistics
10.6.6	sar	Historical statistics
10.6.7	nicstat	Network interface throughput and utilization
10.6.8	ethtool	Network interface driver statistics
10.6.9	tcplife	Trace TCP session lifespans with connection details
10.6.10	tcptop	Show TCP throughput by host and process
10.6.11	tcpretrans	Trace TCP retransmits with address and TCP state
10.6.12	bpftrace	TCP/IP stack tracing: connections, packets, drops, latency
10.6.13	tcpdump	Network packet sniffer
10.6.14	Wireshark	Graphical network packet inspection

-
- Packet sniffing
- Produce log for timestamp, packet + headers, metadata, interface name

-

Os

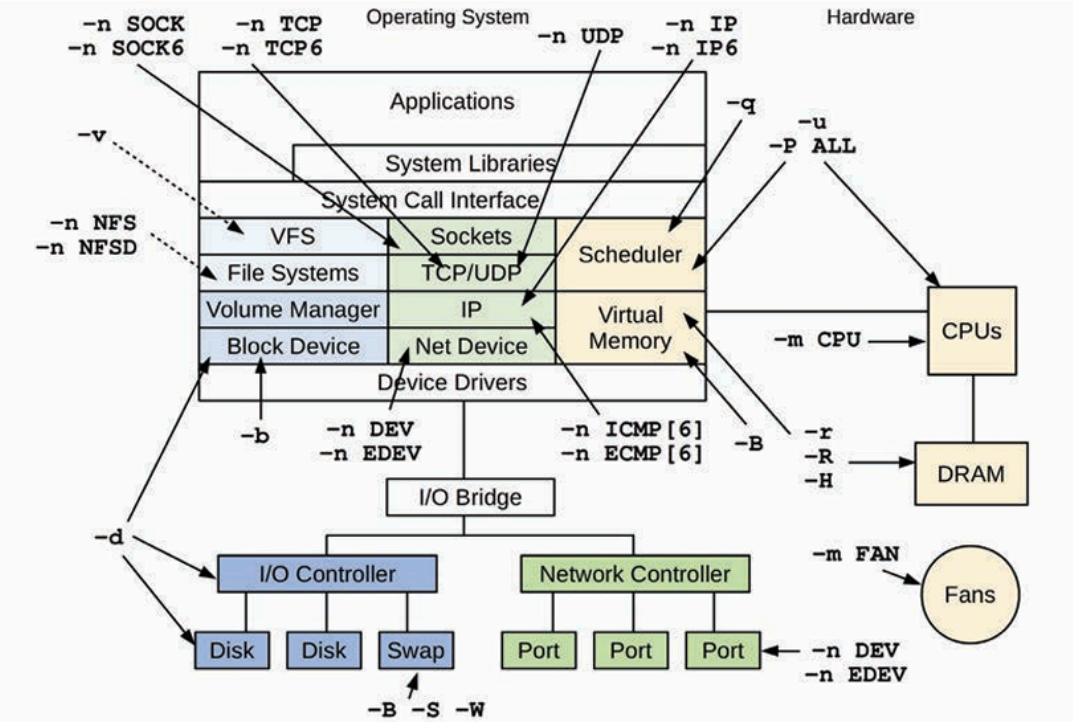
Common optimizations

- In general: response Latency, response throughput, resource utilization, price
- ELIMINATE EXTRA WORK
- Apdex - want closer to 1 (all satisfied); the following are customer ratings
    - Apdex = (satisfactory + 0.5 × tolerable + 0 × frustrating) / total event
- Good tips
    - Optimize common case (where code goes a lot/common tools)
    - Choose observability over faster application (unless difference is massive)
- Good ways to optimize in OS
    - Optimize io size
    - Use buffer
    - Use caching
    - Decrease poling
    - Add concurrency + parallelism (app executes on multiple cpu's at a time via multiprocessing or multithreading)
        - Fibers - lightweight threads (user version); is a schedulable program
        - Co-routine - schedulable by user; lighter than fiber
        - Event-based concurrency - programs split up into events; this is handled by event queue
- Io still handled by kernel
- Use compiler over interpreter
    - Interpreter compiles at runtime and hard to perf due to no mapped functions (basically since isn't compiled)
    - Compilers come with optimizations (at sacrifice of debugging)
- Garbage collection can consume cpu cycles
- Microbenchmarking - test how long simple tasks take to execute over high number of times
- Could do cycle analysis to see at architecture level how many branch predictions are wrong, etc.

General checks

- Vmstat and pidstat to see what task is spending so long in
- Remember that off cpu is network, io, lock, idle waiting for work, sleep
-

Sar



formatts are svg, json, csv

System

- Perf list - see tracepoints
- Kprobe - unstable api since is a tracepoint but reveals a lot more about kernel; may differ per kernel
    - Last resort to debug

Table 4.3 **kprobes to tracepoints comparison**

Detail	kprobes	Tracepoints
Type	Dynamic	Static
Rough Number of Events	50,000+	1,000+
Kernel Maintenance	None	Required
Disabled Overhead	None	Tiny (NOPs + metadata)
Stable API	No	Yes

    -
- Usdt - user version of tracepoints

- Uprobe - user version or kprobe
- ## Reasoning for looking at system calls
- Trace execve to see new processes
- Look at read, write, send, recv for io profiling
- Check kernel time (high sys%)
- ## User and system time tools
- User AND system
    - Getrusage
    - proc/pid/stat
    - Pidstat
    - perf
- ## System calls
- Strace
    - -ttt time since epoch show
    - -t give seconds+milliseconds of time spent in call
    - -p PID
    - -f follow child threads
    - -o write to output to file
        - Uses breakpoint based tracing. Is invasive and programs with high system calls may suffer because of this
- Perf
    - Pefrf is better since uses per-cpu buffers to reduce overhead + system wide (strace is only certain processes; less safe due to more overhead)
- Execsnoop
- SyscountStrace
- Execsnoop
- Syscount
- ## Schedule tools
- Schedstats
- Worse:
    - Perf
    - Runqlat
- ## Sleeping tools
- naptime.bt
- ## Locks tools
- Make sure:
    - Check for contention, excessive hold times
    - Cpu profiling does not tell you full story; may have spent time idling waiting for locks
        - Unlike spin locks. Mutex locks
- Klockstat
- bpftrace
- pmlock.bt and pmheld.bt for pthread mutex locks, and

- mlock.bt and mheld.bt for kernel mutexes.
- Thread state
- pstack(1) or gdb(1)

Memory

-

USE + tips

- Increasing tlb size decreases misses of translations
- Saturation - swapping, oom killing
    - Pgscan - when scan takes longer than 10 seconds (looking for memory to reclaim), there's memory pressure - look in sar -B
    - cat /proc/pressure/memory
        - See avg pressure over time, total time with memory pressure, some/full - how many processes are waiting for memory
- Utilization - percent used (and available memory)
    - Per process
    - System wide
    - These types of memory
        - Virtual memory
        - Physical
    - Usage of memory resource controls
    -
- Look for memory usage
    - Top
    - vmstat
- Look for memory pressure
    - Continual page scanning (memory pressure)
    - Vmstat - swapping
    - Perf / bcc / bpftrace - identify cause of memory pressure
    - Psi
    - High scanning (sar -B and check pgscan columns)
    - Look at oom killer logs in var/log/messages

More specific framework

- What is the working set size (WSS) for the applications?

- Where is the kernel memory used? Per slab?
- How much of the file system cache is active as opposed to inactive?
- Where is the process memory used (instructions, caches, buffers, objects, etc.)?
- Why are processes allocating memory (call paths)?
- Why is the kernel allocating memory (call paths)?
- Anything odd with process library mappings (e.g., changing over time)?
- What processes are actively being swapped out?
- What processes have previously been swapped out?
- Could processes or the kernel have memory leaks?
- In a NUMA system, how well is memory distributed across memory nodes?
- What are the IPC and memory stall cycle rates?
- How balanced are the memory buses?
- How much local memory I/O is performed as opposed to remote memory I/O?
-
- Memory leak?
    - Is program not freeing memory?
    - Is memory growth higher than desired?
- Static performance
- how much main memory is there in total?
- How much memory are applications configured to use (their own config)?
- Which memory allocators do the applications use?
- What is the speed of main memory? Is it the fastest type available (DDR5)?
- Has main memory ever been fully tested (e.g., using Linux memtester)?
- What is the system architecture? NUMA, UMA?
- Is the operating system NUMA-aware? Does it provide NUMA tunables?
- Is memory attached to the same socket, or split across sockets?
- How many memory buses are present?
- What are the number and size of the CPU caches? TLB?
- What are the BIOS settings?
- Are large pages configured and used?
- Is overcommit available and configured?
- What other system memory tunables are in use?
- Are there software-imposed memory limits (resource controls)?

tools

- Swapping
    - Vmstat
    - Getdelays.c
- top/htop
- Vmstat
- Pmap
    - Smem - more detailed

-
- Ps aux
- pidstat
-
- Swapon - Swap device usage
- psi
- Slabtop - Kernel slab allocator statistics
- Numastat - NUMA statistics
- Pmap - Process address space statistics
- Perf - Memory PMC and tracepoint analysis
- Drsnoop - Direct reclaim tracing
- Wss - Working set size estimation
- Bpftrace - Tracing programs for memory analysis
- Page fault
- flame graph
- memleak

Cpu

Things to look out for + USE

- Spin locks - excessive user/system
- Waiting - need more cpu; fix limits
- Voluntar context switch rate
- Interrupt rate
- User to system time ratio
    - Higher user is usually higher cpu
- Utilization should be in amount of cpu time used within given limit of user/cgroup you are in
    - Vmstat
    - Top for highest cpu used per process
        - Cpu load averages also show utilization not just saturation
    - Pidstat - break down user and system cpu usage %
        - Ps aux
    - Pmcarch - show high level cpu usage
    - Flame graph
        - Height is stack size
        - Width is cpu time
- Saturation - part of load averages, also wait for cpu. Run queue length/latency
    - load averages
        - Top, vmstat, uptime
    - Runqlat - runqueue latency
    - Runqlen - summarize runqueue length
- Errors easiest to check in logs

    - Demesg
    - var/sys/log
- Other
    - Perf
    - Profile
        - Change with -F: sample (hz) and time frame (sec)
- Thread scalability issues, hot cpus - PER CPU STATS
    - Mpstat
- Bad ipc
    - Perf - look for cycle-based inefficiencies
- showboost/turboboost
    - Current cpu clock rates in case is too slow
- Bad caching?
    - Fix cach size
    - Use cpu binding or exclusive sets (like cpu binding but with sets)
- Bad priority?
    - Use RT or deadline
    - Or fix nice values
- Misc
    - time ptime - time command and see its cpu breakdown
    - Turboboost - show cpu clock rate and other stats
    - Showboost - show cpu clock rate and turbo boost
    - Tlbstat - summarize tlb cycles
    - Cpudist - summarizze cpu on time
    - Sortirqs - soft interrupt time
    - Hardirqs - hard interrupt time
    - Bpftrace - cpu analysis + tracing
- Off cpu
- Application perf
- Capture stack trace when block event happens (off cpu)
- Off-cpu Timer-based sampling - basically taking stack snapshot every x freq
    - Must offset from freq of program to ensure don't miss any behavior that happens between freq of program
    - Less invasive, less overhead
- Advanced
- What is the CPU utilization system-wide? Per CPU? Per core?
- How parallel is the CPU load? Is it single-threaded? How many threads?
- Which applications or users are using the CPUs? How much? Which kernel threads are using the CPUs? How much?
- What is the CPU usage of interrupts?
- What is the CPU interconnect utilization? Why are the CPUs being used (user- and kernel-level call paths)? What types of stall cycles are encountered?

Static tuning

6.5.7 Static Performance Tuning

Static performance tuning focuses on issues of the configured environment. For CPU performance, examine the following aspects of the static configuration:

- How many CPUs are available for use? Are they cores? Hardware threads?
- Are GPUs or other accelerators available and in use?
- Is the CPU architecture single- or multiprocessor?
- What is the size of the CPU caches? Are they shared?
- What is the CPU clock speed? Is it dynamic (e.g., Intel Turbo Boost and SpeedStep)? Are those dynamic features enabled in the BIOS?
- What other CPU-related features are enabled or disabled in the BIOS? E.g., turboboost, bus settings, power saving settings?
- Are there performance issues (bugs) with this processor model? Are they listed in the processor errata sheet?
- What is the microcode version? Does it include performance-impacting mitigations for security vulnerabilities (e.g., Spectre/Meltdown)?
- Are there performance issues (bugs) with this BIOS firmware version?
- Are there software-imposed CPU usage limits (resource controls) present? What are they?

The answers to these questions may reveal previously overlooked configuration choices.

The last question is especially true for cloud computing environments, where CPU usage is commonly limited.

Flamegraphs

- Y axis - number of stacks; x-axis -> length of time
- Can reveal system io, network io, lock contention, def cpu
-
- Tips:
    - Large Plateau from left to right -> on cpu for many samples, so high cpu usage
    - Look bottom up to understand hiearchy
    - After look top down for common cpu usage patterns, like small frame lock contention
-
-
-
-

Disk

Metrics

(similar to file system)
I/O rate
I/O throughput
I/O size
Read/write ratio
Random versus sequential

Tools

Section	Tool	Description
9.6.1	iostat	Various per-disk statistics
9.6.2	sar	Historical disk statistics
9.6.3	PSI	Disk pressure stall information
9.6.4	pidstat	Disk I/O usage by process
9.6.5	perf	Record block I/O tracepoints
9.6.6	biolatency	Summarize disk I/O latency as a histogram
9.6.7	biosnoop	Trace disk I/O with PID and latency
9.6.8	iotop, biotop	Top for disks: summarize disk I/O by process
9.6.9	biostacks	Show disk I/O with initialization stacks
9.6.10	blktrace	Disk I/O event tracing
9.6.11	bpftrace	Custom disk tracing
9.6.12	MegaCli	LSI controller statistics
9.6.13	smartctl	Disk controller statistics

- Offcputime
- Bpftrace
- perf
- u
- iostat - service times, utilization, iops
    - -x 1 to see how often linux does front/back merges
- Iotop - see which process is causing disk io
- biolatency: To examine the distribution of I/O latency as a histogram, looking for multi modal distributions and latency outliers (over, say, 100 ms)
- biosnoop: To examine individual I/O
- S
- Iostat (service time)
- Check disk controller
    - Current vs max throughput, op rate (utilization)
    - Io wait due to controller saturation (sat)
- If you only have one controller, then iops from iostat is for that controller
- Otherwise, need to divide by total controllers
- Tips
- Iops doesn't tell you everything; type of disk matters (flash, rotational with sequential/random, etc.)
- Io wait doesn't tell you everything either; high cpu process can make io go down
    - Upgrade in cpu can reveal io issues once cpu time down

    - Only good for identifying cpu idle, disk busy type
- Better than io wait: disk io block
- Io may not always match application's io
    - File system inflation, deflation due to merging io or adding extra metadata
    - Swapping
    - Rounding
    - Raid overhead (checksums, mirroring)
    -
- Copy on write allows for sequential reads but may result in random reads later
- 100% utilization for a few seconds is likely an issue

Performance Characterization

The previous workload characterization lists examine the workload applied. The following examines the resulting performance:

- How busy is each disk (utilization)?
- How saturated is each disk with I/O (wait queueing)?
- What is the average I/O service time?

- What is the average I/O wait time?
- Are there I/O outliers with high latency?
- What is the full distribution of I/O latency?
- Are system resource controls, such as I/O throttling, present and active?
- What is the latency of non data-transfer disk commands?

- What is the IOPS rate system-wide? Per disk? Per controller?
- What is the throughput system-wide? Per disk? Per controller?
- Which applications or users are using the disks?
- What file systems or files are being accessed?
- Have any errors been encountered? Were they due to invalid requests, or issues on the disk?
- How balanced is the I/O over available disks?
- What is the IOPS for each transport bus involved?
- What is the throughput for each transport bus involved?
- What non-data-transfer disk commands are being issued?
- Why is disk I/O issued (kernel call path)?
- To what degree is disk I/O application-synchronous?
- What is the distribution of I/O arrival times?

File system

- File system latency analysis
    - Application
        - Closest measure bu depends on your app
    - Syscall
        - Common tools but flooded with other info
    - Vfs
        - Traces more info than needed
    - Top of file system
        - File system specific; allows you to trace what you want
- percent time in file system = 100 * total blocking file system latency/application transaction time
- Disk io used to be file io since worst io is file io
- Measure operation rate, latency
    - Reads, writes, sync/async, operations/sec, avg read size
        - Steadiness of each

Section	Tool	Description
8.6.1	mount	List file systems and their mount flags
8.6.2	free	Cache capacity statistics
8.6.3	top	Includes memory usage summary
8.6.4	vmstat	Virtual memory statistics
8.6.5	sar	Various statistics, including historic
8.6.6	slabtop	Kernel slab allocator statistics
8.6.7	strace	System call tracing
8.6.8	fatrace	Trace file system operations using fanotify
8.6.9	latencytop	Show system-wide latency sources
8.6.10	opensnoop	Trace files opened
8.6.11	filetop	Top files in use by IOPS and bytes
8.6.12	cachestat	Page cache statistics

Section	Tool	Description
8.6.13	ext4dist (xfs, zfs, btrfs, nfs)	Show ext4 operation latency distribution
8.6.14	ext4slower (xfs, zfs, btrfs, nfs)	Show slow ext4 operations
8.6.15	bpftrace	Custom file system tracing

-

Network (os)

- Tcptop
-

Proc

- Contains dirs relating to each PID; inside there are files with stats about process, mapped from kernel data structure
    - limits: In-effect resource limits
    - maps: Mapped memory regions
    - sched: Various CPU scheduler statistics

    - schedstat: CPU runtime, latency, and time slices
    - smaps: Mapped memory regions with usage statistics
    - stat: Process status and statistics, including total CPU and memory usage
    - statm: Memory usage summary in units of pages
    - status: stat and statm information, labeled
    - fd: Directory of file descriptor symlinks (also see fdinfo)
    - cgroup: Cgroup membership information
    - task: Directory of per-task (thread) statistics
- Also has system wide stats
    - cpuinfo: Physical processor information, including every virtual CPU, model name, clock
    - speed, and cache sizes.
    - diskstats: Disk I/O statistics for all disk devices
    - interrupts: Interrupt counters per CPU
    - loadavg: Load averages
    - meminfo: System memory usage breakdowns
    - net/dev: Network interface statistics
    - net/netstat: System-wide networking statistics
    - net/tcp: Active TCP socket information
    - pressure/: Pressure stall information (PSI) files
    - schedstat: System-wide CPU scheduler statistics
    - self: A symlink to the current process ID directory, for convenience
    - slabinfo: Kernel slab allocator cache statistics
    - stat: A summary of kernel and system resource statistics: CPUs, disks, paging, swap,
    - processes
    - zoneinfo: Memory zone information
-

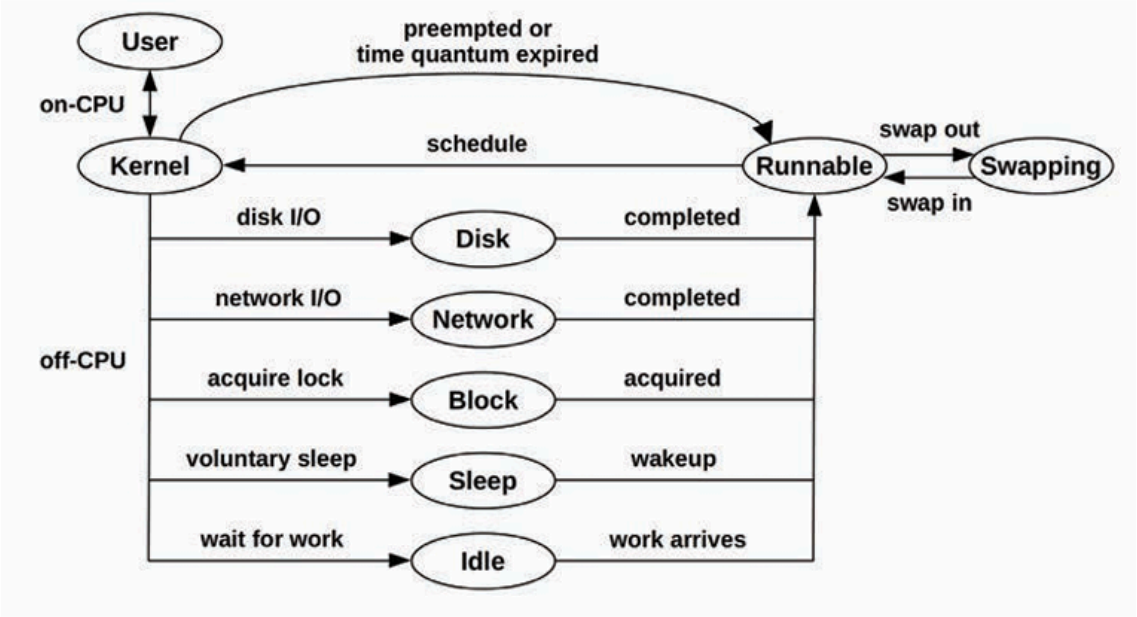
Misc scenarios

- Unexpected outcomes due to symbols missing in binary (from C,C++) due to strip (linux) being used on file to reduce file size
    - This may be due to debugging symbols missing so profilers don't know what's going on (like what address is what ufnction start)
- JIT runtime may be missing symbols in perf due to dynamically creating symbol table. Must provide premade one
    - Java, node
- Missing stacks in perf - frame pointers omitted in compiling, repurposed frame pointer for recyclign

Misc

- Architecture for monitoring -> servers' agents save in db monitoring stats

    - Have ui webserver for analytics dashboard. Devops yser fetches from stats db
- Vmstat reads from
    - Stat, meminfo, vmstat
- Default CONFIG_TICK_CPU_ACCOUNTING is 4 ms for granularity of clock ficks for cpu stats
- Cpu level caches
- Sys
    - Can have writable files for changing kernel state
    - Has cpu core / hardware stats
    - Basically for kernel device and objects management
- CONFIG_TICK_CPU_ACCOUNTING tracks scheduler latency, io wait, swapping, memory reclaim
- Tracepoints add slight cpu overhead to each event
    - Also for post processing events, along with file system management of them
- Linux does not swap processes out at all (for memory)
- Use - utilization (percentage of resource used), saturation (ready queue wait time), errors



-

- Other useful situations (list)

307/308

400

401/403

- Happens when: request requires authentication and you are unauth (401), or you are logged in but don't have valid permissions (403)

404/204/405/501 or any not found/not acceptable request

502

500/503

High memory / crash

High system load, low cpu utilization

Slow disk

Network issue

Slow db

Contention

- Use icicle graph so can see what leaves are pointing at. If same root, likely contention
    - Leaves are at bottom though (scatterred calls)
    - Roots are at top

Cpu

- Faster cpu (lessen memory stalling)
- Caches I1, I2, I3
- Branch prediction
- Prefetching
- Out of order execution
    - Execute other nondependent instructions while waiting for memory
    - Allows for faster cpu processing due to less stalling but a problem for multithreaded

misc

Process wise

Failed batch

Fork bomb

unbootable

-