# 🔥 The Ultimate Guide to Performance Engineering: Bottleneck Patterns, Diagnostics, and Solutions Across Oracle, Java, Linux, AWS Microservices, and Distributed Systems



Santhosh Kumar Jampala
https://www.linkedin.com/in/santhoshjampala/

| 🔑 Symptom | 🧩 Likely Bottleneck | 🛠️ Linux/Java/Oracle Clues | 🚀 Action Plan |
|---|---|---|---|
| **High CPU Usage** (80–100%) | - Tight loop- GC Overhead- Inefficient code- Oracle parsing overhead | - top, pidstat -u: Java PID at top- top -H: one thread near 100%- jstack: RUNNABLE thread(s) in same method- AWR: High “parse CPU” | - Capture multiple jstack dumps- Optimize logic, eliminate loops- Use bind vars to reduce Oracle parse load- Use jfr or perf to trace CPU usage |
| **High Memory Usage** (steady increase) | - Memory leak- Cache overflow- Long-lived object graphs | - free -m: swap increasing- jmap -histo / jmap -dump- GC logs: frequent FGCs- Oracle PGA/SGA nearing OS limits | - Dump heap, analyze with MAT- Tune caches- Fix long-lived references- Tune Oracle SGA/PGA to leave headroom |
| **High CPU + Memory** | - GC Thrashing- Memory leak + promotion failures | - jstat -gcutil: OldGen > 90%- jstack: GC threads RUNNABLE- GC logs: frequent FGCs, CMS failures | - Tune GC (G1GC, CMS, ZGC)- Adjust heap size- Reduce allocation rate- Fix object retention |
| **High I/O Wait** (%wa) | - Disk bottleneck- Oracle table scan- Log/trace overload- NFS delay | - vmstat: high wa, low id- iostat -x: %util > 90%, await > 20ms- AWR: db file sequential/scattered read high | - Use SSDs- Optimize queries with proper indexes- Reduce disk logging- Check NFS/storage config |
| **High Thread Count** | - Thread leak- Blocking operations spawning threads | - ps -eLf: nlwp very high- jstack: many similar threads- ulimit -u close to max | - Cap thread pools- Use executor reuse- Identify thread leaks- Close file handles/sockets properly |
| **Many BLOCKED Threads** | - Lock contention- Synchronized hotspots- Oracle TX enqueue locks | - jstack: many BLOCKED on same monitor- One thread holding monitor- Oracle: enq: TX - row lock contention | - Refactor code for fine-grained locking- Use ConcurrentHashMap etc.- Identify hot rows in DB and avoid them |
| **Threads WAITING/TIMED_WAITING** | - DB/API slowness- I/O bound calls- | - jstack: threads in socketRead- Oracle: slow | - Use async I/O- Tune DB/APIs- Add timeout + circuit breaker- Use connection pooling |

| | | | |
|---|---|---|---|
| | Socket read timeout | queries in AWR- Logs: external call slowness | |
| **GC Threads Using CPU** | - Over-sized Eden- Promotion failure- Memory churn | - jstack: GC threads RUNNABLE- jstat: high Eden + frequent FGC- GC logs: CMS/G1 failures | - Tweak GC generation sizes- Tune CMSInitiatingOccupancyFraction or G1 region size- Switch to ZGC/G1 if using old collector |
| **OOM Errors** | - Memory leak- Poor heap sizing- Unclosed buffers/caches | - Logs: OutOfMemoryError- heapdump.hprof- jstat -gcutil: FGCTime spikes | - Analyze heap dump (MAT/VisualVM)- Fix leaks- Tune JVM heap (-Xmx)- Free non-heap (buffers, file I/O) |
| **Load Avg > CPU Cores** | - CPU pressure- I/O delay- Thread pool exhaustion | - sar -q: runq > cores- top: high load- iostat: high await | - Profile CPU vs I/O source- Tune thread pools- Add more compute nodes (scale out) |
| **Oracle Wait Events** (high wait time) | - Slow I/O- Row lock- Latch/cache issues- Redo bottlenecks | - AWR/ASH: db file seq/scat read, log file sync, latch free, buffer busy- v$session/v$event | - Add missing indexes- Move logs to SSD- Reduce commit rate- Tune SGA pools (library, shared) |
| **Connection Pool Starvation** | - DB not responding- JDBC leaks- Pool too small | - Logs: ConnectionTimeoutException - jstack: waiting on getConnection()- Oracle: v$session shows maxed connections | - Increase pool size (carefully)- Tune SQL for speed- Fix JDBC leaks (unclosed connections)- Use pool monitoring (HikariCP etc) |
| **Slow External Calls (APIs)** | - DNS resolution delay- Remote API latency- TLS handshake overhead | - jstack: waiting on HTTPClient/Socket- Logs: endpoint latency spikes- dig, ping, curl -w slow | - Use persistent HTTP connections- Pre-resolve DNS- Use local cache- Tune HTTP client (timeouts, TLS reuse) |
| **DB Slow Queries** | - Missing indexes- High cardinality | - AWR: Top SQL by elapsed- v$sql shows high | - Add or rebuild indexes- Use bind vars for plan stability- Analyze stats (DBMS_STATS) |

| | | | |
|---|---|---|---|
| | joins- Bad stats | elapsed_time- EXPLAIN PLAN is suboptimal | |
| **High GC Frequency** | - Short-lived object spam- Small young gen- Memory leak | - GC logs: frequent Pause Young- jstat -gc shows frequent minor GC- Memory not recovered | - Increase young gen- Pool objects- Use escape analysis-friendly patterns |
| **High Redo/Log Write Wait** | - Too frequent commits- Slow disk for redo- Low redo buffer | - AWR: log file sync, log file parallel write- Redo write time high | - Batch commits- Move logs to fast disk- Increase LOG_BUFFER |
| **Excessive Hard Parses** | - Literal SQL- Poor cursor reuse- High parse CPU | - AWR: "parse count (hard)" high- library cache lock waits- v$sqlarea has many similar SQLs | - Use bind variables- Set cursor_sharing=force (carefully)- Increase shared_pool_size |

| 🔑 **Symptom** | 🧩 **Likely Bottleneck** | 🛠️ **Linux/Java/Oracle Clues** | 🚀 **Action Plan** |
|---|---|---|---|
| **High Context Switches** | - Thread contention- Lock starvation- Blocking I/O overload | - pidstat -w: high cswch/s (>10k/sec)- vmstat: high cs- jstack: many threads BLOCKED or WAITING | - Reduce locking (use ReentrantLock, LongAdder)- Replace blocking I/O with async- Profile code for hot spots |
| **High Steal Time (%st)** | - Virtualization CPU overcommit- Noisy neighbor VMs | - mpstat: %steal > 5%- iostat: normal but app slow | - Contact cloud/infra team- Migrate instance- Isolate workloads |
| **High Run Queue (r in vmstat)** | - CPU saturation- Lock contention- Thread starvation | - vmstat: r > #vCPUs- top: CPU near 100% | - Optimize threads, locks- Scale out horizontally |
| **High Minor Page Faults** | - Memory fragmentation- Frequent object creation | - pidstat -r: minflt/s very high- vmstat: high pgfault | - Pool/reuse objects- Tune JVM GC for allocation patterns |

| | | | |
|---|---|---|---|
| **High Major Page Faults** | - Memory under-provisioned- Swap usage | - pidstat -r: majflt/s > 0- vmstat: si/so > 0 | - Increase RAM- Reduce heap size- Avoid swap-intensive workloads |
| **High ksoftirqd CPU** | - Network packet storm- Interrupt coalescing issues | - top: ksoftirqd at top- mpstat -I S: softirq high- /proc/softirqs counters | - Tune NIC interrupt coalescing- Distribute IRQs with irqbalance |
| **High Disk Queue Depth** | - Disk under-provisioned- IOPS saturation | - iostat -x: avgqu-sz > 1- High await (>10ms) | - Add SSDs- Tune I/O scheduler (noop, deadline) |
| **Buffer Cache Contention (Oracle)** | - Hot blocks- Poor index design | - AWR: buffer busy waits high- v$bh: same block# hotspots | - Rebuild/index hot tables- Use hash partitioning or reverse keys |
| **Redo Log Contention (Oracle)** | - Frequent small commits- Slow disk for redo logs | - AWR: log file sync wait event high | - Batch commits- Move redo logs to SSD/NVMe- Tune LOG_BUFFER |
| **High log file parallel write** | - Redo log I/O bottleneck | - AWR: log file parallel write wait high | - Separate redo logs from datafiles- Fast storage for redo logs |
| **Oracle enq: TX - index contention** | - Index block hot spots | - AWR: TX - index contention wait- Hot index blocks | - Rebuild index- Use reverse key/index partitioning |
| **Library Cache Mutex Contention** | - Hard parsing- Frequent SQL invalidation | - AWR: library cache: mutex X waits- v$sqlarea: many unique SQLs | - Use bind variables- Increase shared_pool_size |
| **Checkpoint I/O Spikes (Oracle)** | - Log switches too frequent- Checkpoint tuning issues | - AWR: checkpoint complete messages- log file switch timeouts | - Increase redo log size- Tune checkpoint intervals |
| **File Descriptor Exhaustion** | - Resource leak (unclosed files/sockets)- High concurrency | - lsof -p <pid>: open files near ulimit -n- ulimit -n low | - Close file handles in code- Raise nofile limit |
| **Thread Pool Exhaustion** (App Layer) | - Too small pool- Blocking I/O- Downstream slowness | - JMX: ActiveThreads == MaxThreads- Thread dump: many WAITING | - Increase pool size- Use async patterns- Tune DB/API calls |
| **GC Time Spikes** | - Tenuring threshold too low- | - GC logs: long Pause Full- jstat: OldGen spikes | - Tune MaxTenuringThreshold- Pre-size young gen- Analyze object allocation |

| | | | |
|---|---|---|---|
| | Large object allocation | | |
| **Connection Leaks (DB/JDBC)** | - Unclosed connections- No timeout settings | - HikariCP/DBCP metrics: active == max- Oracle: many inactive sessions | - Enforce try-with-resources- Set timeouts- Leak detection (leakDetectionThreshold) |
| **Oracle latch: row cache objects** | - Heavy DDL- Frequent object invalidation | - AWR: latch: row cache objects waits | - Reduce DDL at runtime- Use static object structures |
| **Excessive log file switch (archiving needed)** | - Archiving backlog- Small redo logs | - Alert log: frequent switches- v$log: status ACTIVE/CURRENT | - Increase redo log size- Tune archive destinations |
| **High Network Retransmissions** | - Packet loss- NIC issues | - netstat -s: retransmits > 0- ifconfig: RX/TX errors | - Check cables, NICs- Adjust MTU- Optimize TCP stack |
| **Frequent ORA-01555: Snapshot Too Old** | - Undo retention too low- Long-running queries | - AWR: ORA-01555 errors- v$undostat | - Increase UNDO_RETENTION- Break large queries into chunks |
| **High TCP TIME_WAIT Count** | - Short-lived connections- No keep-alive reuse | - ss -s: TIME_WAIT > 10k- netstat -an | - Enable keep-alive- Use connection pooling- Tune tcp_fin_timeout |
| **High Oracle cursor: pin S Waits** | - Frequent hard parses- SQL not shared | - AWR: cursor: pin S wait on X | - Use bind vars- Tune cursor sharing- Increase shared_pool_size |
| **Network Latency Spikes** | - External API slowness- DNS resolution delays | - ping, mtr, traceroute to APIs- App logs: response time high | - Use persistent connections- Pre-resolve DNS- CDN/proxy caching |
| **High Java Finalizer Activity** | - finalize() abuse- Unreleased resources | - jstat: high FGC- Thread dump: Finalizer threads RUNNABLE | - Remove unnecessary finalizers- Use try-with-resources- Consider PhantomReferences |
| **Long GC Stop-the-World Pauses** | - Large heap + Full GC- CMS fragmentation | - GC logs: “Pause Full” > 500ms- jstat: high FGC | - Switch to G1GC/ZGC- Reduce OldGen occupancy- Tune CMSInitiatingOccupancyFraction |
| **High SoftIRQ CPU Usage** | - Network flood- Interrupt handling bottleneck | - mpstat -I S: high softirq- /proc/softirqs counters | - Use irqbalance- Offload NIC features (GRO/TSO) |

## 🔎 Oracle AWR Analysis – Bottleneck Patterns & Resolutions

| 🔎 Symptom/Pattern (AWR Clues) | 🧩 Likely Bottleneck | 🧪 How to Detect (AWR/ASH) | 🛠️ Resolution Plan |
|---|---|---|---|
| **Top SQL by Elapsed Time / Gets** | - Expensive queries- Missing indexes- Large table scans | - AWR SQL ordered by elapsed_time / buffer_gets- High disk_reads, rows_processed | - Optimize SQL (joins, predicates)- Add/rebuild indexes- Use partitioning |
| **High db file sequential read** | - Slow single-block I/O- Index scans on large tables | - AWR: db file sequential read top wait- Top SQL: high disk_reads | - Check IOPS/disk latency- Add indexes- Tune storage layer |
| **High db file scattered read** | - Full table scans- Poor SQL design | - AWR: db file scattered read wait- Top SQL: full scans (rows_processed high) | - Rewrite SQL- Add indexes- Avoid full scans where possible |
| **High buffer busy waits** | - Hot blocks- Block contention | - AWR: buffer busy waits high- Segment statistics: hot objects | - Reorganize data- Use hash partitions/reverse key indexes- Tune block sizes |
| **High log file sync** | - Frequent commits- Redo log I/O bottleneck | - AWR: log file sync in top waits- Average sync time > 5ms | - Batch commits- Optimize redo logs (size, placement)- Tune storage IOPS |
| **High log file parallel write** | - Slow redo log writes- Storage I/O limits | - AWR: high log file parallel write wait- High redo write times | - Use fast disks (SSD/NVMe)- Tune log buffer size- Reduce commit frequency |
| **High free buffer waits** | - DBWR unable to keep up- Slow disk writes | - AWR: free buffer waits in top waits- High dirty buffer count | - Increase DBWR I/O capacity- Add disks / use faster storage- Check db_writer_processes |
| **High library cache lock/mutex** | - Cursor contention- Frequent hard parses | - AWR: library cache waits- High parse count in AWR | - Use bind variables- Increase shared pool- Reduce hard parsing SQLs |
| **High latch: cache buffers chains** | - Hot block in buffer cache | - AWR: latch: cache buffers chains- Segment stats: hot object | - Reorganize data- Partition tables- Tune SQL to avoid hot blocks |
| **High row cache lock** | - Data dictionary contention- DDL during runtime | - AWR: row cache lock in top waits | - Avoid DDL during peak- Tune shared_pool_size |
| **High enq: TX - row lock contention** | - Hot rows- Update contention | - AWR: enq: TX in top waits- Blockers in v$session | - Optimize transactions- Reduce row-level contention- Use |

| | | | |
|---|---|---|---|
| | | | SELECT FOR UPDATE SKIP LOCKED |
| **High enq: TX - index contention** | - Hot index blocks | - AWR: enq: TX - index waits | - Rebuild index- Use reverse key/index partitioning |
| **High db file parallel write** | - Slow datafile writes- High I/O workload | - AWR: db file parallel write top event- I/O timing > 10ms | - Optimize disk layout- Tune dbwr processes- Use faster disks |
| **direct path read/write high** | - Direct I/O operations (e.g., large queries, bulk loads) | - AWR: direct path read or write top waits | - Optimize bulk operations- Use parallel DML carefully |
| **latch free high** | - Latch contention (shared resource hot spot) | - AWR: latch free top wait- v$latch stats high | - Tune latch-intensive code- Reduce shared resource usage |
| **High Parse CPU Usage** | - Literal SQL- Excessive parsing | - AWR: high parse time CPU- High parse count (hard) | - Use bind variables- Tune cursor sharing (FORCE, SIMILAR) |
| **High cursor: pin S Waits** | - Pin contention in library cache | - AWR: cursor: pin S wait- High mutex waits | - Use bind variables- Tune shared pool size |
| **High db file sequential read in Index Reads** | - Inefficient index access- B-Tree depth too high | - AWR SQL plan shows many index leaf blocks | - Rebuild index- Consider index compression or partitioning |
| **High undo segment waits** | - Long transactions- Undo retention too low | - AWR: undo segment waits- v$undostat metrics | - Increase UNDO_RETENTION- Commit frequently |
| **High gc Waits in RAC** | - Global cache transfer latency | - AWR: gc cr request, gc buffer busy waits- High interconnect latency | - Optimize interconnect (InfiniBand, RoCE)- Tune cache fusion settings |
| **SQL Execution Time > Expected** | - Suboptimal execution plans- Stale stats | - AWR: SQL with high elapsed time- v$sql_plan shows bad plan | - Gather fresh stats- Use SQL Plan Baselines- Tune optimizer parameters |
| **cell single block physical read (Exadata)** | - Smart scan not used | - AWR: cell single block top event- Exadata-specific metrics | - Enable smart scans (direct path reads)- Use full scans when beneficial |

## 🔑 Cache-Related Performance Bottlenecks – Advanced Guide

### 🧱 Cache Layers Involved

| Layer | Cache Type | Example |
|---|---|---|
| Application Layer | In-memory caches | Ehcache, Guava, Caffeine, Hazelcast |
| JVM Layer | Code cache, object heap | JIT-compiled methods, permgen/metaspace |
| Oracle DB | Buffer cache, shared pool, library cache, dictionary cache | Data blocks, SQL cursors, PL/SQL, data dictionary |
| OS Layer | Filesystem cache, page cache | Linux page cache, disk read-ahead |
| Network Layer | DNS cache, connection pools | DNS resolver cache, HTTP keep-alives |

---

### 🔍 Cache Bottleneck Patterns & Resolutions

| 🔑 Symptom | 🧩 Likely Cache Issue | 🧪 Detection/Diagnostics | 🛠️ Resolution Plan |
|---|---|---|---|
| **High GC/Memory Usage in Java App** | - Cache too large (unbounded growth)<br>- Inefficient eviction | - jmap -histo shows large cache classes<br>- jstat -gc shows OldGen growth<br>- JVM metrics: cache size | - Configure max size/TTL<br>- Use LinkedHashMap/Caffeine with eviction<br>- Tune eviction policies (LRU, LFU) |
| **Frequent Full GCs** | - Cache churn creating objects<br>- Poor object pooling | - GC logs: frequent Full GC<br>- jstat -gcutil: high OldGen | - Reduce cache churn<br>- Pool objects<br>- Pre-warm caches |
| **Slow Queries in Oracle** | - Low buffer cache hit ratio<br>- SQLs causing physical I/O | - AWR: Buffer Cache Hit Ratio < 90%<br>- v$sysstat/v$sysmetric: cache misses | - Increase db_cache_size<br>- Tune SQL for index usage<br>- Keep hot tables in memory |
| **Library Cache Latch Contention (Oracle)** | - Excessive hard parsing<br>- Literal SQL (no bind vars) | - AWR: library cache lock/library cache mutex waits<br>- v$sqlarea: many similar SQLs | - Use bind variables<br>- Increase shared_pool_size<br>- Avoid dynamic SQL in loops |

| | | | |
|---|---|---|---|
| **Shared Pool Fragmentation (Oracle)** | - SQLs > 4KB<br>- Poor cursor sharing | - AWR: high parse time CPU<br>- v$sgastat: free memory low in shared pool | - Increase shared_pool_size<br>- Use CURSOR_SHARING wisely<br>- Pin PL/SQL packages |
| **High buffer busy waits (Oracle)** | - Hot blocks in buffer cache<br>- Contention on frequently accessed data | - AWR: high buffer busy waits<br>- Segment statistics: hot objects | - Reorganize tables<br>- Use hash partitioning or reverse key indexes |
| **Poor File System Cache Performance** | - Inefficient OS page cache usage<br>- Double buffering | - vmstat: high page-in/out<br>- sar -B: high page faults<br>- iostat: high disk reads | - Use direct_io (DB side) where applicable<br>- Tune Linux dirty ratios (/proc/sys/vm/dirty_*) |
| **DNS Resolution Delays** | - DNS cache miss at app layer<br>- Frequent external lookups | - dig, nslookup: slow response<br>- Logs: connection delays on first API call | - Use persistent connections<br>- Pre-resolve DNS entries<br>- Local DNS caching (e.g., dnsmasq) |
| **High Connection Pool Churn** | - No connection caching<br>- Low pool sizes | - Connection pool metrics: high borrowed but no reuse<br>- Logs: frequent connection opening/closing | - Use proper pooling (HikariCP, c3p0)<br>- Set minIdle, maxPoolSize<br>- Reuse connections across requests |
| **Cache Invalidation Storm** | - Sudden eviction of large portions of cache (e.g., cache.clear())) | - Logs: bulk cache evictions<br>- Spikes in DB or backend calls after | - Use selective invalidation (by key)<br>- Staggered TTLs ("jittered cache")<br>- Versioned cache keys |
| **Cache Stampede Effect** | - Many threads request same data simultaneously (cold cache) | - Logs: many requests for same key after expiry<br>- Spikes in backend load | - Add request coalescing ("single-flight" logic)<br>- Use per-key locking<br>- Stagger cache expirations |
| **Hot Key Contention** | - One key heavily accessed by all threads (e.g., a global config) | - Cache metrics: hot key hit rate disproportionate<br>- Thread dump: threads waiting on cache fetch | - Shard data if possible<br>- Add randomization to keys<br>- Use per-tenant cache |
| **Cache Eviction Too Aggressive** | - Low max size / short TTL<br>- Premature eviction of useful data | - Metrics: low hit rate despite high cache usage<br>- Logs: frequent refetch from DB | - Increase cache size<br>- Adjust TTLs<br>- Review eviction policy (LRU vs. LFU) |

| | | | |
|---|---|---|---|
| **Oracle latch: cache buffers chains Waits** | - Hot block access in buffer cache | - AWR: latch: cache buffers chains top wait<br>- Segment stats: hot object | - Partition or hash table<br>- Reduce block-level contention |
| **SGA Memory Pressure (Oracle)** | - SGA too small for workload<br>- Fragmented pools | - AWR: high free memory fragmentation<br>- v$sgastat: low free memory | - Increase SGA (sga_target)<br>- Use sga_target vs. manual sizing based on workload |
| **High cache buffer chains latch misses** | - Large block sizes<br>- Hot objects | - AWR: latch misses > 0<br>- Hot segment analysis | - Use smaller block size if possible<br>- Optimize schema design |

---

### 🧬 Advanced Debugging Techniques for Cache Issues

| Tool/Command | Purpose | Example |
|---|---|---|
| jmap -histo | Find largest objects in JVM | jmap -histo:live <pid> |
| jstack | Find threads blocked on cache | Look for BLOCKED on cache class |
| jstat -gcutil | JVM GC trends (churn) | jstat -gcutil <pid> |
| AWR SQL ordered by Gets | High cache usage SQLs | Check top SQL by buffer_gets |
| v$sysstat | Buffer cache hit ratios | select name, value from v$sysstat where name like '%cache%'; |
| sar -B, vmstat | OS cache activity | sar -B for page faults, vmstat for si/so |
| lsof, ulimit -n | Check file descriptor leaks | lsof -p <pid> |
| dtrace/perf | Trace kernel-level caching | Use for syscall-level I/O |
| Cache Metrics (Caffeine, Guava) | Monitor hit/miss rates | Expose via JMX or logs |

## ☁️ AWS Cloud Microservices Performance Issues – Deep Dive

| 🔎 Symptom/Pattern | 🧩 Likely Bottleneck | 🔬 Detection Methods | 🚀 Resolution Strategy |
|---|---|---|---|
| **High Latency in API Gateway/ALB** | - Backend slow responses<br>- Cold starts (Lambda)<br>- TLS handshake delays | - AWS X-Ray traces: high API latency<br>- ALB target response times<br>- CloudWatch metrics: TargetResponseTime, 5XX errors | - Enable caching in API Gateway<br>- Use provisioned concurrency for Lambda<br>- Optimize TLS handshake (TLS 1.3, keep-alives)<br>- Reduce backend response times |
| **Container CPU Spikes (ECS/EKS)** | - Inefficient app code<br>- Thread leaks<br>- GC pauses | - CloudWatch: CPUUtilization spikes<br>- kubectl top pods in EKS<br>- JVM jstat + GC logs | - Optimize code/hotspots<br>- Adjust CPU limits/requests<br>- Tune GC (G1GC, ZGC) |
| **Container Memory Usage Spikes** | - Memory leaks<br>- Unbounded cache growth<br>- Large object creation | - CloudWatch: MemoryUtilization trends<br>- JVM heap usage (Prometheus, JMX)<br>- Container OOM kills | - Set memory limits + requests<br>- Profile heap usage (jmap, VisualVM)<br>- Fix leaks, cache sizing |
| **Cold Start Latency (Lambda)** | - Unprovisioned Lambda<br>- Large package size<br>- VPC-enabled Lambda | - CloudWatch: Duration, InitDuration metrics<br>- X-Ray traces | - Use provisioned concurrency<br>- Minimize deployment package size<br>- Use Lambda SnapStart (Java)<br>- Avoid VPC unless necessary (or use ENI reuse) |
| **High API 5XX Errors** | - Backend failures<br>- Rate limits<br>- API Gateway integration timeouts | - CloudWatch: 5XXError metrics<br>- X-Ray: backend service faults | - Set retries with backoff<br>- Handle errors gracefully<br>- Optimize downstream services |

| | | | |
|---|---|---|---|
| **High Latency Variability (P99 Spikes)** | - Noisy neighbor effect<br>- Network bottlenecks<br>- GC pauses | - CloudWatch: P99Latency spikes<br>- VPC Flow Logs: high retransmits<br>- JVM GC logs | - Use multi-AZ deployments<br>- Tune GC for low-latency<br>- Use HTTP/2 multiplexing |
| **Throttling in AWS Services (429 Errors)** | - Exceeding AWS API limits<br>- Unthrottled burst traffic | - CloudWatch: Throttles metrics<br>- AWS SDK retries/exceptions | - Implement client-side throttling<br>- Use service quotas APIs to monitor<br>- Stagger workloads |
| **High DB Latency (Aurora/DynamoDB)** | - Query inefficiency<br>- Hot partitions<br>- IOPS saturation | - RDS Performance Insights: Top SQLs<br>- DynamoDB CloudWatch: ThrottledRequests | - Optimize SQL (Aurora): indexes, partitioning<br>- DynamoDB: Use partition keys, adaptive capacity<br>- Scale read replicas (Aurora) |
| **Slow Microservice-to-Microservice Calls** | - Network latency<br>- TLS handshake overhead<br>- DNS resolution delays | - X-Ray: high service latency<br>- dig, mtr, curl -w tests | - Enable HTTP/2/gRPC<br>- DNS caching<br>- Enable TLS session reuse |
| **Service Discovery Delays** | - DNS TTL too low<br>- Overloaded CoreDNS | - kubectl logs coredns in EKS<br>- High DNS resolution time in X-Ray | - Increase DNS TTLs<br>- Scale CoreDNS replicas<br>- Use service mesh sidecar DNS caching |
| **High Inter-Zone Latency** | - Cross-AZ traffic<br>- Unoptimized routing | - VPC Flow Logs: cross-AZ traffic<br>- X-Ray traces: high latency spans | - Use same AZ placement where possible<br>- Optimize inter-service communication |
| **Frequent Pod Evictions (EKS)** | - Node resource exhaustion<br>- Misconfigured resource limits | - CloudWatch: Evictions metric<br>- kubectl describe pod | - Set proper CPU/memory requests<br>- Use Cluster Autoscaler<br>- Right-size nodes |

| | | | |
|---|---|---|---|
| **Lambda API Gateway Timeouts** | - Lambda exceeding API Gateway 29s limit | - CloudWatch: IntegrationLatency near 29s<br>- X-Ray traces | - Optimize Lambda logic<br>- Break large tasks into smaller functions<br>- Use Step Functions for orchestration |
| **Cache Miss Spikes (ElastiCache/DAX)** | - TTL expiries<br>- Hot key eviction | - CloudWatch: CacheHits/Misses metrics<br>- Application logs | - Increase TTLs<br>- Use distributed locking (e.g., Redlock)<br>- Avoid sudden bulk invalidations |
| **High EBS Latency (GP2/GP3)** | - Disk IOPS limits<br>- Burst balance depletion | - CloudWatch: VolumeQueueLength, VolumeThroughputPercentage | - Use provisioned IOPS (io1/io2)<br>- Monitor burst balance<br>- Distribute workload across volumes |
| **High Lambda Concurrent Executions** | - Traffic spike<br>- No reserved concurrency | - CloudWatch: ConcurrentExecutions metric<br>- Throttling errors in logs | - Set reserved concurrency<br>- Use weighted routing for canary deployments<br>- Enable async event sources |
| **VPC Network Bottlenecks** | - High ENI usage<br>- Insufficient bandwidth | - VPC Flow Logs: retransmits, high latency<br>- CloudWatch: Network metrics | - Use Nitro-based instances<br>- Scale horizontally<br>- Enable Enhanced Networking (ENA) |
| **Service Mesh Overhead (App Mesh/Istio)** | - Sidecar CPU/memory usage<br>- mTLS encryption cost | - CloudWatch: sidecar metrics<br>- Istio metrics: pilot, envoy stats | - Optimize Envoy configs<br>- Reduce mTLS if not needed for internal traffic<br>- Offload metrics collection |
| **X-Ray Trace Data Missing** | - Sampling too low<br>- Misconfigured SDKs | - No traces for requests in X-Ray | - Adjust sampling rules<br>- Instrument SDKs properly |

| | | | |
|---|---|---|---|
| | | | - Use OpenTelemetry for custom metrics |
| **High DynamoDB Throttling** | - Hot partitions<br>- Exceeded RCU/WCU | - CloudWatch: ThrottledRequests, ProvisionedReadCapacityUnits | - Use adaptive capacity<br>- Distribute keys<br>- Switch to on-demand mode |
| **Inefficient Autoscaling Response** | - Cooldown delays<br>- Metric lag | - CloudWatch: delayed scale-in/scale-out | - Tune scaling policies<br>- Use predictive scaling<br>- Leverage step scaling with proper thresholds |

---

### 🧨 Advanced Tools for AWS Microservices Performance Analysis

| 🔧 Tool | Purpose | Example |
|---|---|---|
| AWS X-Ray | Distributed tracing | Find service-level bottlenecks |
| CloudWatch Logs Insights | Query logs at scale | Analyze latency spikes |
| CloudWatch Metrics | Infrastructure + app metrics | CPU, memory, latency, errors |
| AWS Distro for OpenTelemetry | Custom metrics + tracing | Ingest into CloudWatch/Prometheus |
| VPC Flow Logs | Network diagnostics | Detect packet loss, latency |
| kubectl top, kubectl logs | EKS performance | Resource usage per pod |
| JMX Exporter + Prometheus | JVM metrics | GC, heap, thread pool analysis |
| Load Testing (k6, JMeter) | Synthetic load | Test under production-like conditions |
| CloudTrail | API activity tracking | Detect throttles, access patterns |

### 🚧 Additional AWS Cloud Microservices Performance Bottlenecks

#### 🥷 Hidden/Advanced Patterns

| 🔍 Bottleneck | 🧩 Details | 🛠️ Detection & Tools | 🚀 Resolution Strategy |
|---|---|---|---|
| **API Gateway Header Bloat** | - Large HTTP headers cause latency<br>- Custom headers | - ALB logs: ClientClosedConnection errors<br>- CloudWatch 5XX spikes | - Minimize JWT/claims size<br>- Compress headers |

| | | | |
|---|---|---|---|
| | (JWTs, tracing IDs) can cross 8KB+ | | where possible<br>- Move metadata to payload |
| **Lambda Execution Environment Leaks** | - Static variables persist across invocations, leading to state leaks | - Unpredictable behavior in cold vs warm starts<br>- Hard-to-debug issues in stateful logic | - Avoid static mutable state<br>- Use idempotent functions<br>- Isolate logic in handlers |
| **Excessive CloudWatch Logs Latency** | - High log volume can throttle ingestion<br>- Delayed metric generation | - Delayed CloudWatch metrics/alarms<br>- High CloudWatch ThrottledLogEvents | - Filter logs (reduce verbosity)<br>- Use Kinesis Firehose for high-throughput logs |
| **Unbalanced Load in ALB/EKS** | - Sticky sessions or DNS caching causes traffic skew | - TargetResponseTime high for subset of pods<br>- X-Ray: some pods overloaded | - Use round-robin or least-connections<br>- Use service mesh load balancing |
| **Intermittent Network Latency (AWS Global Network)** | - AWS backbone congestion<br>- Cross-region replication delays | - VPC Flow Logs: retransmits, high RTT<br>- Ping/traceroute: latency spikes | - Use Global Accelerator for low-latency routing<br>- Localize traffic where possible |
| **Lambda Package Size Bloat** | - Large dependencies inflate package >250MB | - Deployment errors, slow cold starts<br>- S3 deployment requirement | - Use Lambda layers<br>- Minimize dependencies<br>- Tree-shake unused libraries |
| **EBS Burst Balance Exhaustion** | - GP2 volumes exceed burst IOPS | - CloudWatch: BurstBalance metric = 0<br>- High VolumeQueueLength | - Switch to GP3/io1/io2 with provisioned IOPS<br>- Distribute load across volumes |
| **Undersized NAT Gateway** | - NAT GW bandwidth bottleneck (~5 Gbps limit) | - Slow outgoing traffic<br>- CloudWatch NAT metrics: BytesOut at limit | - Use multiple NAT GWs per AZ<br>- Use Egress-Only Internet Gateway if IPv6 |

| | | | |
|---|---|---|---|
| **Unoptimized Step Functions Execution** | - Long-running states cause timeouts<br>- High state transitions cost | - Step Functions metrics: ExecutionTime, BilledDuration | - Split long states<br>- Reduce state transitions<br>- Consider Lambda batching |
| **Excessive SNS/SQS Fan-Out Latency** | - Many subscribers cause delivery delays | - CloudWatch: NumberOfMessagesSent high, delayed NumberOfMessagesReceived | - Use FIFO queues to preserve order<br>- Consider Kinesis if high fan-out required |
| **Noisy Neighbor in Multi-Tenant K8s Cluster** | - One namespace hogs resources | - EKS metrics: node CPU/mem skewed<br>- kubectl top pods imbalance | - Enforce resource quotas<br>- Use LimitRanges per namespace |
| **Container Image Bloat** | - Large images (>1GB) cause slow pulls | - EKS: ImagePullBackOff errors<br>- Long pod startup times | - Use minimal base images (distroless, alpine)<br>- Multi-stage Docker builds |
| **High Service Mesh CPU/Memory (Envoy)** | - Sidecar proxies consume significant resources | - Prometheus: envoy_cluster_upstream_rq_time spikes<br>- CPU/memory usage of sidecars | - Optimize Envoy config<br>- Consider eBPF-based alternatives (Cilium) |
| **Inefficient VPC Peering Architecture** | - Cross-VPC traffic via peering scales poorly | - VPC Flow Logs: high cross-VPC traffic<br>- Latency variance in cross-service calls | - Consider Transit Gateway or PrivateLink |
| **RDS Connection Pool Saturation** | - High connection churn from Lambda/ECS | - RDS metrics: DBConnections at max<br>- App logs: Too many connections errors | - Use RDS Proxy<br>- Reuse connections<br>- Tune pool size per workload |
| **Inefficient Data Fetch in Microservices** | - N+1 query problem across services | - X-Ray: repetitive service calls<br>- Logs: many API calls for single request | - Batch APIs<br>- GraphQL federation<br>- CQRS patterns |
| **Overly Chatty Microservices** | - High inter-service call rates cause latency | - X-Ray: many small calls<br>- Network metrics: high packet rates | - Aggregate API calls<br>- Use event-driven design (SNS/SQS, EventBridge) |

| | | | |
|---|---|---|---|
| **Elastic IP Address Reassign Delays** | - Failover IP takes time to propagate | - Failover tests: EIP takes ~60s to reattach | - Use Route 53 health checks + multi-value answers<br>- Use Global Accelerator for low-latency failover |
| **CloudWatch Alarms Lag** | - Delayed metrics cause late alerts | - Alarm triggered after >5 min delay | - Use high-resolution metrics (1s granularity)<br>- Leverage real-time monitoring tools (Datadog, New Relic) |

---

## 🧵 Thread Dump Analysis – The Ultimate Guide

### 🏗️ Thread Dump Basics

- A **Thread Dump** is a **snapshot of all JVM threads**, showing:
  - **Thread state**: RUNNABLE, WAITING, BLOCKED, TIMED_WAITING
  - **Stack traces**: What the thread is doing
  - **Locks held/waiting on**: monitor, ownable synchronizer
  - **Thread names**: http-nio-8080-exec-1, ForkJoinPool.commonPool-worker-3
  - **Daemon/Non-daemon status**

### ✅ When to capture?

- During high CPU usage
- During latency spikes
- When blocked threads suspected
- During memory leaks (GC threads active)

### ✅ How to capture?

- jstack <pid> (Linux/Unix)
- kill -3 <pid> → Outputs to stdout
- JVM tools: VisualVM, Java Mission Control, JFR
- Cloud: ECS/EKS sidecar jstack, AWS Lambda SnapStart diagnostics

### 🚦 Thread States & What They Indicate

| Thread State | Meaning | Possible Bottleneck |
|---|---|---|
| RUNNABLE | Actively executing on CPU | Hot code path, tight loops, infinite loops, GC overhead |
| WAITING / TIMED_WAITING | Waiting for external events (I/O, locks, sleeps) | Network/API delays, DB calls, thread pool starvation |
| BLOCKED | Waiting for a monitor/lock | Lock contention, deadlocks, synchronized bottlenecks |
| TERMINATED | Completed thread | Normal or unexpected (abrupt termination) |
| GC Threads | JVM GC activity | High CPU, GC pressure, object churn |

### 🔍 Bottleneck Patterns from Thread Dumps

| 🔎 Pattern | 🧩 What to Look For | 🚀 Action Plan |
|---|---|---|
| **CPU Bottleneck (High RUNNABLE Threads)** | Many threads in RUNNABLE at top of dump, in same code (e.g., parsing, loops, serialization) | Optimize code, algorithm, data structures; Profile with -XX:+UnlockDiagnosticVMOptions -XX:+PrintCompilation; Use perf, jfr |
| **Lock Contention (BLOCKED Threads)** | Many threads BLOCKED on same lock/monitor; One thread holding the lock | Refactor locking code; Reduce critical sections; Use ReentrantLock, StampedLock; Consider lock-free structures (ConcurrentHashMap, LongAdder) |
| **Thread Pool Starvation** | All threads in pool in WAITING; No available worker threads | Increase thread pool size; Analyze task execution time; Consider queue capacity |
| **External I/O Blocking** | Threads WAITING in socketRead, read0, poll, accept | Add timeouts; Use async/non-blocking I/O (NIO, Netty); Cache external calls |
| **Deadlock Detected** | Found one Java-level deadlock in dump | JVM reports deadlock with threads and locks; Break cycle; Redesign lock acquisition order |

| | | |
|---|---|---|
| **GC Overhead** | GC threads in RUNNABLE; frequent GC logs | Tune heap size, GC algorithm; Reduce object allocation rate |
| **ForkJoinPool/Parallel Streams Saturation** | ForkJoinPool worker threads in BLOCKED/WAITING | Reduce parallelism; Avoid nested parallel streams; Control task size |
| **High Thread Count (Leak)** | Thousands of threads; common prefix (e.g., http-nio-8080 or Executor-) | Check thread creation in code; Limit thread pools; Use close() on resources |
| **Thread Waiting on Database Calls** | Threads in WAITING in JDBC driver calls (e.g., socketRead) | Optimize SQL; Use connection pool; Check DB performance (AWR, slow queries) |
| **Async Task Backlogs** | Threads in WAITING in ScheduledThreadPoolExecutor/CompletableFuture | Monitor queues; Increase pool sizes; Apply backpressure strategies |
| **Finalizer/Reference Handler Threads Active** | Threads in ReferenceHandler, Finalizer | Avoid finalize(); Use AutoCloseable; Reduce phantom/weak references |

---

**Typical Stack Frames in Thread Dumps**

| Method | What It Indicates | Related Bottleneck |
|---|---|---|
| java.net.SocketInputStream.socketRead0 | Waiting for I/O | Network API delays |
| sun.nio.ch.EPollArrayWrapper.epollWait | I/O multiplexing wait | Normal; may indicate thread pool starvation if too many |
| java.util.concurrent.ThreadPoolExecutor.getTask | Waiting for task | Normal unless pool too small |
| java.util.concurrent.locks.ReentrantLock$NonfairSync | Locking bottleneck | Lock contention |
| java.lang.Thread.sleep | Sleeping (deliberate delay) | May cause latency spikes if overused |
| java.lang.Object.wait | Synchronized wait | Inter-thread communication or resource lock |
| org.apache.catalina.connector.Request.doRead | Servlet input blocking | HTTP request I/O bottleneck |
| org.hibernate.loader.Loader | Hibernate ORM data loading | N+1 query issues, lazy loading |

---

### 🛠️ Advanced Tools for Thread Dump Analysis

| Tool | Purpose | Usage |
|---|---|---|
| Thread Dump Analyzer (TDA) | Visual thread dump analysis | Highlight threads by state, identify locks |
| fastThread.io | Online thread dump viewer | Detect deadlocks, contention hotspots |
| jClarity Censum / GCEasy | GC + thread dump correlation | Visualize GC impact on threads |
| JFR (Java Flight Recorder) | Profiling + thread activity | Correlate CPU, threads, GC events |
| async-profiler + Flamegraphs | CPU hotspots | Flamegraph visualization of code execution |

---

### 🧬 Thread Dump Analysis Workflow

1️⃣ Capture **3-5 dumps at 10s intervals** during issue.
2️⃣ Filter by thread **state**: focus on RUNNABLE, BLOCKED.
3️⃣ Group by **thread name prefix**: e.g., http-nio, ForkJoin.
4️⃣ Identify **hot code paths**: same method in multiple threads.
5️⃣ Look for **locks held/waited**: detect contention or deadlocks.
6️⃣ Cross-check with **GC logs**, **CPU usage**, **external systems**.
7️⃣ Form **hypotheses** and **test fixes** in dev/stage.

---

## 🗑️ GC Log Analysis – The Ultimate Guide for Java Performance

### 🚦 Why Analyze GC Logs?

GC logs reveal:

✅ **Allocation pressure**
✅ **Memory leaks**
✅ **Pause times affecting latency**
✅ **Throughput vs. responsiveness trade-offs**
✅ **GC algorithm effectiveness**

✅ They **bridge the gap** between **code behavior** and **JVM internals**:

- GC logs = "heartbeat" of the JVM.

---

### 📊 GC Log Essentials: Key Fields to Watch

| Field | What It Shows | Impact on Performance |
|---|---|---|
| **Pause Time** (Pause Young, Pause Full) | How long app is paused | Directly affects P99 latency |

| | | |
|---|---|---|
| **GC Type** (Young, Old, Mixed) | Which generation collected | Helps identify which space causes issues |
| **Before/After Heap** | Heap size before/after GC | Reveals allocation rate & retention |
| **Frequency of GCs** | How often GC runs | Too frequent = high churn, too rare = risk of OOM |
| **Promotion Rate** | Rate of objects moving to OldGen | High rate → risk of OldGen fill-up |
| **Allocation Rate** | How much memory app allocates per second | High rates = GC pressure |
| **Concurrent Phase Durations** | Time spent in background GC tasks | Long phases = background work stalling app |
| **Finalizer Activity** | Finalizer threads usage | Excessive use → leaks, GC delays |

---

### 🦠 Common GC Bottleneck Patterns

| 🔎 GC Pattern | 🧩 Symptoms in Logs | 🚀 Root Cause & Fix |
|---|---|---|
| **GC Thrashing (Frequent Minor GCs)** | Many **Pause Young** eventsLow pause times but very frequentOldGen stable | High object churn in EdenObject allocation too fast |
| **Promotion Failure** | Frequent Full GC (promotion failed) | OldGen too smallObjects live too long |
| **Long GC Pauses (Full GC)** | Few but long Pause Full eventsApp freezes | Heap too largeCMS fragmentationOldGen leaks |
| **Concurrent Mode Failure (CMS)** | concurrent mode failure in logs | CMS can't finish before OldGen fills |
| **GC Overhead Limit Exceeded** | JVM exits with GC overhead limit exceeded | JVM spends >98% time in GC |
| **High GC CPU Usage** | GC threads in RUNNABLE in thread dumpGC CPU >30% | Too frequent GCsLarge heapsCMS scanning cost |
| **Metaspace OOM** | java.lang.OutOfMemoryError: Metaspace | Excessive class loadingClassloader leaks |
| **Frequent G1 Mixed GCs** | Pause Mixed (G1 Evacuation Pause) events frequent | OldGen has high live dataMixed collections trigger often |
| **Unusual Pause Distribution** | Long tail latencies (high P99) | Some pauses 5-10x longer than median |

---

### GC Log Formats & Enabling Logging

| JVM | GC Log Option | Notes |
|---|---|---|
| Java 8 | -XX:+PrintGCDetails -Xloggc:<file> | Standard format |
| Java 9+ | -Xlog:gc*:file=<file> | Unified Logging |
| Java 11+ | -Xlog:gc*:stdout:time,uptime,level,tags | More granular |
| Enable GC Log Rotation | -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=20M | Avoid huge logs |

---

### GC Algorithm-Specific Insights

#### G1GC Patterns

| Pattern | Clue in Logs | Action |
|---|---|---|
| Excessive Mixed GCs | Pause Mixed every few seconds | Reduce -XX:InitiatingHeapOccupancyPercent |
| Pause Time Spikes | Long Pause Young/Pause Mixed | Tune -XX:MaxGCPauseMillis |
| High Allocation Rate | Eden fills up fast | Increase YoungGen size (via -XX:NewRatio or G1NewSizePercent) |

---

#### ZGC/Shenandoah

| Pattern | Clue | Action |
|---|---|---|
| Large Heap + Low Pause | Pauses <10ms at large heaps | Expected behavior—good! |
| Pauses >10ms | Unexpected long pauses | Check allocation rate, GC threads |

---

### GC Log Analysis Tools

| Tool | What It Does | Usage |
|---|---|---|
| GCViewer | Visual GC graphing | Parse -Xloggc files |
| GCEasy.io | Online GC log analyzer | Upload logs, get insights |
| JClarity Censum | Commercial GC analyzer | Advanced analysis, not free |

| | | |
|---|---|---|
| [Eclipse Memory Analyzer (MAT)] | Heap dump analysis | Find memory leaks |
| [JFR + JMC (Java Mission Control)] | Real-time GC + thread correlation | Enable with -XX:+FlightRecorder |

---

🧭 **GC Tuning Workflow (Battle-Tested)**

✅ **Capture GC Logs** → 10-30 mins under load
✅ **Correlate with App Latency** (CloudWatch, X-Ray, Prometheus)
✅ **Identify Patterns**:

- GC frequency
- Pause time distribution
- OldGen/YoungGen occupancy
  ✅ **Heap Dump at High GC** → Analyze leaks
  ✅ **Adjust GC Settings**:
- **G1**: MaxGCPauseMillis, InitiatingHeapOccupancyPercent
- **ZGC/Shenandoah**: Heap size, thread count
  ✅ **Test Under Load** → Rinse & repeat

---

## 🧠 UNIX Server Logs & Trace Analysis – Advanced SRE Guide

📁 **Key System Logs to Monitor**

| Log File | Path | Purpose |
|---|---|---|
| syslog / messages | /var/log/syslog or /var/log/messages | Kernel + general system messages |
| dmesg | dmesg or /var/log/dmesg | Boot time + hardware/kernel errors |
| auth.log | /var/log/auth.log | SSH, sudo, login failures |
| secure | /var/log/secure | PAM, security events |
| kern.log | /var/log/kern.log | Kernel errors, OOM killer, CPU stalls |
| journalctl | journalctl -xe | Combined systemd logs |
| application logs | /var/log/<app>/ | App-specific or via journald |
| audit.log | /var/log/audit/audit.log | SELinux, permission audits |
| gc.log | Custom JVM GC log file | GC pause and allocation patterns |
| access.log, error.log | /var/log/nginx/, /var/log/httpd/ | Web server latency, errors |

---

### System Performance Tracing: Core Commands

| Command | Use Case | Sample |
|---|---|---|
| top, htop | CPU/mem usage per process | top -H -p <pid> shows thread CPU |
| vmstat 1 | CPU, run queue, memory | Detect CPU saturation, context switches |
| iostat -x 1 | Disk I/O bottlenecks | Look for await, %util > 90% |
| pidstat -p <pid> -u -r -w -d | Per-process CPU, mem, I/O | High %usr or minflt/majflt |
| free -m | Memory, swap usage | Used/free/buffered swap |
| dstat -tcmnd | Realtime combined stats | See CPU, mem, disk, network |
| netstat -anp | Open ports and connections | Detect socket exhaustion |
| lsof -p <pid> | Open files/sockets | Detect FD leaks |
| `ps -eLf | grep ` | Thread count |
| strace -p <pid> | Syscall-level trace | Trace I/O, read/write, sleep |
| perf top / perf record | CPU profiling | Hotspots at syscall/function level |
| sar -u, sar -r, sar -B | Historic CPU/mem | Show trends in CPU, paging |
| journalctl -u <service> | Systemd app logs | Debug startup/shutdown failure |

---

### Log Patterns That Indicate Bottlenecks

| Pattern | Where to Find It | Implication |
|---|---|---|
| OutOfMemoryError, Killed process | gc.log, kern.log, app logs | JVM killed due to memory limits |
| Segmentation fault, core dumped | syslog, dmesg | Native crash, bad C extension |
| blocked for more than 120 seconds | dmesg, kern.log | I/O lockup, filesystem stall |
| oom-killer events | dmesg, /var/log/kern.log | Memory exhaustion, swap mismanagement |
| connection reset by peer, broken pipe | App logs, network layer | Unhandled exception, retry storm |
| resource temporarily unavailable, EMFILE | dmesg, app logs | File descriptor exhaustion |
| too many open files | ulimit, lsof | Leaky sockets, handle leaks |
| permission denied, SELinux audit | audit.log, secure | Access issues |

| | | |
|---|---|---|
| slow query, timeout | App/DB logs | Downstream slowness, API/DB |
| REJECT, DROP | iptables, auditd | Firewall or security blocking traffic |

---

## 🧪 How to Approach a Unix-Level RCA (Root Cause Analysis)

**1. Capture Key Metrics**

```
uptime
vmstat 1 5
iostat -x 1 5
free -m
ps aux --sort=-%mem | head
```

**2. Check for Kernel-Level Clues**

```
dmesg | egrep -i 'error|warn|oom|blocked|segfault'
journalctl -xe
```

**3. Zoom into Specific Process**

```
pidstat -p <pid> -u -r -w -d 1
strace -tt -T -p <pid> -o strace.log
lsof -p <pid> | wc -l
```

**4. Check Network/Port Exhaustion**

```
netstat -an | grep ESTABLISHED | wc -l
ss -s
```

**5. Correlate with App Logs**

```
grep -i 'exception' /var/log/<yourapp>/app.log
grep -i 'ERROR' /var/log/<yourapp>/*
```

---

## 🛠️ Tools for Advanced Log & Trace Analysis

| Tool | Usage | Ideal For |
|---|---|---|
| Logrotate | Manage log sizes & rotation | Prevent disk fill-up |
| logwatch, goaccess, lnav | Log summarization | Quick daily summaries |

| | | |
|---|---|---|
| ELK Stack (Elasticsearch, Logstash, Kibana) | Centralized logging | Production log analysis |
| Fluentd, Filebeat | Shipping logs | K8s + cloud-native logs |
| bpftrace, eBPF | Kernel-level tracing | syscall/network profiling |
| perf, ftrace | High-res CPU tracing | Profiling Java/C binaries |
| systemtap, bcc-tools | Dynamic kernel probes | Low-level debugging |
| atop | Historic resource tracking | Find slow trends |
| Glances | All-in-one resource monitor | CLI + Web dashboard |
| Loggly, Datadog Logs, CloudWatch Logs | Cloud log aggregation | AWS-specific logging |

---

**Correlating UNIX Logs with Java Performance**

| Log/Event | Java Symptom | Unix Indicator |
|---|---|---|
| High GC time | CPU spikes, pause times | top, jstat, jstack, GC logs |
| Thread lock | High latency, timeouts | jstack, ps -eLf, strace shows blocking |
| I/O bottlenecks | API lag, DB slowness | iostat, dmesg, lsof |
| Memory leak | OOM errors, swap | dmesg, free -m, heap dump |
| File handle leak | Too many open files | lsof, ulimit -n exhaustion |
| Network stalls | Connection timeouts | netstat, tcpdump, ss |

---

**Recommended Aliases for Daily Use**

```
alias top10cpu='ps aux --sort=-%cpu | head -n 11'
alias top10mem='ps aux --sort=-%mem | head -n 11'
alias threads='ps -eLf | wc -l'
alias fdcount='lsof -p $1 | wc -l'
alias memlog='free -h && vmstat 1 5'
alias cpuload='mpstat 1 5'
```

---

# 🧠 Heap Dump Analysis – The Ultimate Guide

### 🧩 What is a Heap Dump?

A **Heap Dump** is a **binary snapshot of the JVM heap memory**, containing:

- All live objects (and optionally GC roots, class metadata, thread stacks)
- Object types, counts, sizes, references

### ✅ Use Cases:

- Memory leaks
- OOM errors (java.lang.OutOfMemoryError: Java heap space)
- Cache growth analysis
- Object retention patterns
- ClassLoader leaks

---

### 📦 How to Capture a Heap Dump

| Method | Command/Tool | Notes |
|---|---|---|
| On-Demand (Linux) | jmap -dump:format=b,file=heap.hprof <pid> | May cause pause; works in prod if stable |
| Signal Trigger | kill -3 <pid> (for thread dump) + jcmd <pid> GC.heap_dump <file> | Safer than jmap in prod |
| At OOM | -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/path/to/dump | Automatically on OOM error |
| JConsole/VisualVM | UI-based trigger | Easy for dev/stage; not ideal for prod |
| Lambda SnapStart | Part of snapshot; no manual dump | Use AWS Lambda tuning |
| EKS/ECS (Java in container) | docker exec <container> jcmd ... | Heap dump inside container |
| Kubernetes | kubectl exec <pod> -- jcmd <pid> GC.heap_dump /tmp/heap.hprof | For Java apps in K8s |

---

### 🔬 What to Look For in a Heap Dump

| 🔎 Symptom | 📍 Heap Dump Clues | 🚀 Possible Root Cause & Fix |
|---|---|---|
| **Memory Leak (High OldGen Usage)** | Dominating classes in OldGenLarge retained size | Unreleased references; caches without eviction; listeners |
| **Unbounded Cache Growth** | One or few classes with millions of instances | Map/Set/ConcurrentHashMap growth |
| **ClassLoader Leak** | Multiple ClassLoader instances holding references | Web apps, redeploys without cleanup |
| **ThreadLocal Leaks** | ThreadLocalMap holding objects | ThreadLocals not removed |
| **Large Collections** | Collections (List, Set, Map) with many elements | Poor batching, unbounded queues |
| **Finalizer Retention** | Objects in Finalizer queue | Overuse of finalize() |
| **Duplicate Strings** | Many identical String instances | String duplication |
| **Excessive Buffers** | Large arrays/byte[] objects | I/O, buffer leaks |
| **Socket/Connection Leaks** | Socket objects retained | Missing close() calls |
| **Anonymous Inner Class Retention** | Inner classes referencing outer | Large object graphs |
| **Retained by GC Roots** | Object graph leading to GC Roots (static fields, threads) | Strong references to static fields |

---

### 🚀 Heap Dump Analysis Workflow (Battle-Tested)

1. **Capture heap dump** during OOM or memory spike.
2. Open with analysis tools:
   - 🛠️ Eclipse Memory Analyzer (MAT)
   - 🛠️ VisualVM
   - 🛠️ GCEasy.io (for quick insights)
3. **Run Leak Suspect Report (MAT)** → Focus on **Top Dominator Tree**.
4. Check **Retained Size** vs. **Shallow Size**.
5. Find **paths to GC Roots** → Who's holding the object?
6. Investigate suspicious classes (e.g., java.util.HashMap, org.apache.catalina.loader.WebappClassLoaderBase).
7. Correlate with **app code, caches, pools, and external systems**.
8. Test fix in dev/stage → Recapture dump under load → Compare results.

---

### 📊 Heap Dump Tools

| Tool | Strengths | Ideal Use |
|---|---|---|
| **Eclipse Memory Analyzer (MAT)** | Open-source, dominator tree, leak suspect report | Detailed offline analysis |
| **VisualVM** | Lightweight, live heap view, histograms | Quick checks, dev environments |
| **GCEasy.io** | Upload heap dump for instant report | Cloud-native insights |
| **HeapHero.io** | Reports + charts, classloader leaks | Cloud dumps |
| **JProfiler** | Heap + CPU profiling | Paid; great for dev/stage |
| **YourKit** | Advanced leak detection, object inspections | Paid; deep dives |
| **JDK Mission Control (JMC)** | Heap + thread + GC correlation | JFR integration |

---

### 🏚 JVM Heap Structure Cheat Sheet

| Space | What It Holds | Flags to Tune |
|---|---|---|
| **Eden** | New objects, short-lived | -Xmn, G1 -XX:G1NewSizePercent |
| **Survivor** | Objects promoted between GCs | -XX:SurvivorRatio |
| **OldGen** | Long-lived objects | -Xmx, G1 -XX:InitiatingHeapOccupancyPercent |
| **Metaspace** | Class metadata | -XX:MaxMetaspaceSize |
| **Code Cache** | JIT-compiled code | -XX:ReservedCodeCacheSize |

---

### 🧪 Advanced Heap Tuning Flags

| Flag | Purpose |
|---|---|
| -XX:+HeapDumpOnOutOfMemoryError | Dump heap on OOM |
| -XX:HeapDumpPath=/path | Set heap dump location |
| -XX:+UseStringDeduplication | Reduce duplicate String instances (G1) |
| -XX:+PrintClassHistogram | Get class instance histogram |
| -XX:+UnlockDiagnosticVMOptions -XX:+PrintGCApplicationStoppedTime | Correlate GC with app pause time |

---

## 📊 LoadRunner Graph Correlation Matrix (Performance Analysis View)

| 🎯 Primary Graph | 🔗 Correlates With | 🧠 What It Tells You | 🛠️ Typical Root Cause |
|---|---|---|---|
| **Average Response Time** | - Throughput- Hits/sec- Errors/sec- CPU/Memory | If response time rises while throughput drops, likely **server bottleneck** | CPU overload, DB latency, GC pause, resource starvation |
| **Throughput (Bytes/sec)** | - Response Time- Hits/sec- Network I/O | High throughput with low response time = healthy. If **throughput drops with rising response time**, backend stress | Server under stress, DB wait, app server queueing |
| **Hits/sec** | - Concurrent Users- Throughput- CPU- TPS | Drop in hits/sec despite constant users may show **errors, delays, queueing** | Thread pool saturation, backend failure, bottlenecks |
| **Transactions per Second (TPS)** | - Response Time- Errors/sec- Backend logs | Stable TPS = stable system. **Dips with response time spikes = app/DB delay** | Slow SQL, code lock, thread pool exhaustion |
| **Error Rate** | - Response Time- TPS- Backend logs | Spikes in errors with stable load = **functional defect** or **resource outage** | Bad code path, 500 errors, broken dependency |
| **Concurrent Vusers** | - Hits/sec- TPS- Response Time | Constant vusers with rising response time = **scalability bottleneck** | Saturated queues, DB contention, I/O wait |
| **Latency** (Network) | - Response Time- DNS Time- TCP Connect Time | Spikes in latency can inflate response time → **network issues** | DNS, TLS handshake, bandwidth congestion |
| **Pages Downloaded/sec** | - Hits/sec- Throughput- TPS | Drop may indicate **resource wait** or **asset timeout** | CDN delay, image load failure, web server queueing |
| **CPU Utilization (Server)** | - Response Time- TPS- GC Activity | Spikes in CPU with drop in TPS = **CPU saturation** | Tight loops, serialization, poor concurrency |
| **Memory Usage (Server)** | - Response Time- GC logs- Swap Activity | Memory spikes = **leak, cache growth**, bad object lifetime | JVM heap leak, unbounded cache, unclosed objects |

| | | | |
|---|---|---|---|
| **GC Activity / GC Time** | - Response Time- CPU Usage- OldGen usage | High GC time = **memory churn**, causes response latency | Poor object lifecycle, small heap, high promotion |
| **I/O Wait / Disk Usage** | - Response Time- TPS- DB Logs | High I/O wait = **disk or DB bottleneck** | Slow IOPS, bad indexes, verbose logging |
| **Network Usage (bytes in/out)** | - Throughput- Hits/sec- Socket time | Drops or spikes suggest **payload issue, delay, or congestion** | Compression, packet loss, slow client |
| **DB CPU or Time/Call (Oracle, MySQL)** | - TPS- Response Time- GC or thread dump | Correlates high app latency with **DB hotspots** | Slow queries, missing indexes, lock contention |
| **Think Time (Controller)** | - Response Time- TPS | Affects user pacing. If you **remove think time** and response time degrades, system is near saturation | App can't scale linearly |
| **Transactions Passed/Failed** | - TPS- Errors/sec- Hits/sec | Shows functional correctness. Drop in passed + spike in failed = **functional issue** | Broken path, network break, invalid response |
| **Bandwidth Graphs** | - Throughput- Page load- Network usage | Detects **network cap, latency issues, or CDN delays** | Network saturation, AWS egress limit |
| **JVM/GC Logs** (external) | - LoadRunner Response Time- Server CPU | Align GC spikes with **LoadRunner response latency** | GC tuning needed |
| **Application Logs (errors)** | - Error Rate- Response Time- TPS | Correlates **server-side exceptions** with client-side metrics | Code defects, unhandled errors, API downtime |

---

🧪 **Practical Use Cases for Correlation**

🔁 **Scenario 1: Response Time ↑ + Throughput ↓ + CPU 100%**

**Likely Cause**: CPU bottleneck on app servers
**Action**: Profile hot code paths, tune thread pools, optimize GC

---

🔁 **Scenario 2: TPS ↓ + Errors ↑ + DB CPU ↑**

**Likely Cause**: DB saturation or slow SQL
**Action**: Tune queries, check AWR, index, scale DB

---

🔁 **Scenario 3: Hits/sec Stable + Response Time ↑ + GC Time ↑**

**Likely Cause**: GC thrashing or memory leak
**Action**: Heap dump, GC log analysis, tune heap size/GC algorithm

---

🔁 **Scenario 4: TPS ↓ + Hits/sec ↓ + Response Time Stable**

**Likely Cause**: Load Generator saturation or network issue
**Action**: Check LG CPU/mem; monitor network; check LG logs

---

## 🌐 Load Balancer Performance Bottlenecks & Correlation – Deep Dive

🧩 **Load Balancer Metrics & What They Indicate**

| Metric | What It Tells You | Symptoms | Root Cause | Action Plan |
|---|---|---|---|---|
| **Request Count / New Connections** | Incoming request volume | Sudden spikes = traffic surge | DDoS, bot traffic, misbehaving clients | Rate limit, WAF rules, scale LB |
| **Target Response Time** (ALB/NLB) | Time backend takes to respond | High = slow backend | Backend bottleneck, DB latency | Profile app, DB, code hotspots |
| **Target Connection Errors** | Failed TCP/SSL connections | Spikes = backend overload | Backend app down, port closed, listener misconfig | Check backend health, port status |
| **HTTP 4XX / 5XX Rates** | Client errors (4XX) / Server errors (5XX) | Spikes in 5XX = backend failure | App crash, thread pool starvation | App logs, thread dumps, GC logs |
| **ELB 5XX Errors (502, 504)** | Gateway timeout, bad response | 504 = backend timeout; 502 = bad response | Backend timeout, bad TLS certs | Tune backend timeouts, fix TLS |
| **Load Balancer Latency** | Time LB takes to route request | Spikes during high load | Queueing at LB, routing delays | Optimize target group health checks, enable keep-alive |
| **Healthy/Unhealthy Targets** | Backend instance health | Drops = failing instances | App crash, unhealthy endpoints | Check app logs, health check config |
| **Connection Reuse / Recycle** | TCP connection handling | High new connections = no reuse | Keep-alive disabled, misconfig | Enable keep-alive; tune idle timeouts |

| | | | | |
|---|---|---|---|---|
| **SSL Negotiation Time** | TLS handshake overhead | Long SSL times = latency | Large certs, no session resumption | Optimize certs, enable session reuse |
| **Active Connections** | Open TCP connections | High with low throughput = slow backend | Backend delay, DB lock, API slowness | Trace backend dependencies |
| **HTTP Backend Errors (ALB)** | App-side errors forwarded by LB | Spikes during traffic peaks | App error under load | Check app metrics, thread pool |
| **Surge Queue (ALB Classic)** | Requests queued at LB | Growing = backend overload | Backend slow to respond | Scale backend, optimize app |
| **Request Processing Time (ALB Classic)** | Time LB spends per request | High = backend delay | DB/API bottleneck, app slowness | Profile backend services |
| **Dropped Connections** | LB rejecting connections | Sudden spikes = overload | LB capacity, backend unresponsive | Scale LB, adjust timeouts |

---

**Load Balancer Bottleneck Patterns – Real-World Scenarios**

| Pattern | Symptom | Detection | Resolution |
|---|---|---|---|
| **504 Gateway Timeout** | User request hangs → error | ALB logs: 504s; X-Ray: backend > timeout | Tune backend timeouts, increase app thread pools |
| **502 Bad Gateway** | App returns malformed response | ALB logs: 502s; App logs: 500 errors | Check app code; TLS handshake issues |
| **Latency Spikes at LB** | Users report slowness | LB metrics: TargetResponseTime ↑ | Backend saturation; thread dump analysis; DB wait |
| **High 4XX Errors** | 403/404/429 rates ↑ | LB logs; WAF logs | Misconfigured routes, rate limiting, auth failures |
| **Unhealthy Targets** | Targets marked unhealthy | LB health check logs | App crash, port closed, misconfig in health check path |
| **Connection Saturation** | LB stops accepting new connections | CloudWatch ActiveConnectionCount at max | Scale LB; adjust connection limits |
| **TLS Handshake Delays** | Users see slow initial load | LB SSL metrics ↑ | Use TLS 1.3; enable session tickets; optimize certs |
| **Cross-AZ Traffic Cost** | Unexpected AWS data transfer costs | VPC Flow Logs; LB metrics | Use same-AZ routing; check target distribution |

| | | | |
|---|---|---|---|
| **Keep-Alive Disabled** | High new connections; high latency | CloudWatch: many NewConnections | Enable keep-alive; tune timeouts |
| **Load Balancer CPU High (Envoy/Nginx)** | LB itself overloaded | Nginx/Envoy stats; OS CPU | Tune worker processes; offload TLS; distribute load |
| **ALB/NLB Limits Reached** | New targets not registering | CloudWatch limits; AWS Service Quotas | Request limit increases; refactor architecture |

---

🔍 **Correlating LB Metrics with Application & Infra**

| LB Metric | Correlates With | Root Cause Example |
|---|---|---|
| TargetResponseTime ↑ | App Response Time ↑, DB Latency ↑ | App slowness, DB locks |
| 5XX Errors ↑ | Thread Pool Saturation, GC Logs | App thread pool exhausted, GC pause |
| NewConnections ↑ | DNS TTL, keep-alive reuse | TTL too low, keep-alive disabled |
| Latency ↑ + ActiveConns ↑ | Backend CPU High | Backend CPU bottleneck under load |
| Unhealthy Targets ↑ | App Crash Logs | App instance crash, port closed |
| TLS Negotiation Time ↑ | Cert Size, Cipher Suites | Large certs, inefficient ciphers |
| Cross-AZ Traffic ↑ | VPC Flow Logs, Cost Explorer | ALB routing to different AZs |
| Surge Queue ↑ | Backend Slow, Thread Dump | Backend app not scaling |

---

## 🧠 Unix Internals: Swapping, Paging, Buffer I/O, and Context Switching

---

🧱 **Core Concepts Cheat Sheet**

| Concept | Description | Impact if Bottlenecked |
|---|---|---|
| **Swapping** | Moving entire memory pages from RAM to disk (swap space) | Severe performance hit; processes freeze |
| **Paging** | Kernel moves memory pages between RAM and swap (demand paging) | Increased page faults; degraded performance |
| **Buffer I/O** | Kernel buffers disk I/O (page cache, dirty pages) | I/O latency spikes; high disk wait time |

| | | |
|---|---|---|
| **Context Switching** | CPU switches from one process/thread to another | High switches = CPU overhead, cache misses |

---

### 📊 Bottleneck Indicators and Metrics

| Symptom | Metric/Command | Interpretation | Root Cause |
|---|---|---|---|
| High Swap Usage | free -m, swapon -s, vmstat si/so | swap used > 0, si/so > 0 = active swapping | Memory pressure; OOM risk |
| High Page Faults | vmstat -s, sar -B | pgfault/s (minor), pgmajfault/s (major) | Excessive memory access; cache misses |
| Disk I/O Wait | iostat -x, vmstat wa | %util > 90%, await > 20ms, wa > 10% | Slow disk; excessive buffer flushes |
| High Context Switch Rate | vmstat cs, sar -w, pidstat -w | >10k/s (app-specific) = potential lock contention | Mutex locks, threads, I/O waits |
| Dirty Pages Accumulating | /proc/meminfo, vmstat | Dirty pages growing; sync lagging | Slow disk writeback; fsync bottleneck |
| High r in vmstat | vmstat 1 | r > CPU cores = CPU saturation | Process queue buildup; CPU bottleneck |

---

### 🚀 Swapping Analysis

| Check | Command | Bottleneck Sign |
|---|---|---|
| Swap Usage | free -m, swapon -s | Swap > 0 MB = memory pressure |
| Swap In/Out Rate | vmstat 1 (si, so) | Non-zero si/so = active swapping |
| Major Page Faults | sar -B | High pgmajfault/s = swap-in delays |
| dmesg Logs | `dmesg | grep -i oom` | |

🚑 **Action**:

- Tune app memory; reduce cache sizes; increase RAM.
- Disable swap (swapoff -a) for performance tests (careful!).

---

🔎 **Paging Analysis**

| Check | Command | Bottleneck Sign |
|---|---|---|
| Page Fault Rate | vmstat -s, sar -B | High pgfault/s + pgmajfault/s |
| Minor vs Major | vmstat -s | High major = slow; minor = normal |
| Page Reclaims | vmstat -s | High reclaims = page pressure |

🚑 **Action**:

- Optimize app memory locality.
- Avoid large random memory allocations.
- Reduce memory-hungry processes.

---

💾 **Buffer I/O Analysis**

| Check | Command | Bottleneck Sign |
|---|---|---|
| Dirty Pages | `cat /proc/meminfo | grep Dirty` |
| Disk Wait | iostat -x 1, vmstat wa | High %util, await, wa |
| Writeback Latency | dstat -d, sar -b | High blocks out; delayed I/O |

🚑 **Action**:

- Tune vm.dirty_ratio, vm.dirty_background_ratio:
- sysctl -w vm.dirty_ratio=10
- sysctl -w vm.dirty_background_ratio=5
- Use faster disks (SSD/NVMe); distribute I/O load.

---

⚙️ **Context Switch Analysis**

| Check | Command | Bottleneck Sign |
|---|---|---|
| Context Switch Rate | vmstat cs, pidstat -w, sar -w | >10k/s sustained = high overhead |
| Thread Count | `ps -eLf | wc -l` |
| Lock Contention | perf top, strace -p <pid> | Threads in futex, epoll_wait |

🚑 **Action**:

- Reduce threads; pool threads effectively.
- Refactor synchronized code; reduce lock granularity.
- Avoid excessive I/O-bound threads.

---

📈 **Correlation Matrix**

| Symptom | Swapping | Paging | Buffer I/O | Context Switch |
|---|---|---|---|---|
| App Latency ↑ | High swap usage | High major faults | High dirty pages, wa | High context switches |
| Disk I/O Wait ↑ | Low RAM = swap-in | High minor faults | High await, %util | Threads waiting on I/O |
| CPU Spikes | Swap-in CPU | Page fault handling | I/O interrupts | High task switches |
| OOM Kills | Yes | Yes | No | Indirect (lock wait stalls) |

---

🧪 **Diagnostic Workflow**

1️⃣ **Check RAM**:

free -m

2️⃣ **Monitor Swapping**:

vmstat 1

3️⃣ **Page Faults**:

sar -B 1 5

4️⃣ **Disk I/O Wait**:

iostat -x 1

5️⃣ **Context Switches**:

pidstat -w 1

6️⃣ **Correlate with App Logs, GC, and Thread Dumps**

---

🛠️ **Advanced Tools**

| Tool | Purpose |
|---|---|
| perf top, perf record | Kernel-level hotspots |
| bpftrace, bcc-tools | Trace syscalls, I/O wait |
| strace -p <pid> | Syscall-level blocking |
| iotop, dstat | Per-process disk I/O |
| sar, atop, glances | Combined metrics overview |

---