### 📘 Performance Testing FAQ Document

#### ❓ What do we monitor in LoadRunner Controller?

**Answer:** In LoadRunner Controller, we monitor the following key aspects during a performance test:

**1. System Resource Monitors:** - CPU usage per server - Memory consumption - Disk I/O rates - Network throughput

**2. Application Server Metrics:** - Hits per second - Throughput (KB/sec) - Connections/sec - Thread utilization - Garbage Collection (GC) activity

**3. Database Metrics:** - Number of active DB sessions - Query execution times - Lock waits and deadlocks - Connection pool utilization

**4. Vuser Metrics:** - Number of running, passed, failed users - Iteration counts per script - Vuser response times - Vuser status during ramp-up/down

**5. Transaction Metrics:** - Average, minimum, maximum response times - Transactions per second (TPS) - Transaction pass/fail counts - 90th percentile response time

**6. Error Logs & Custom Monitors:** - HTTP/DB/Application errors - SNMP/JMX/custom monitors - APM-integrated metrics (Dynatrace/New Relic)

#### ❓ What happens in load test if only 50% JVM memory is available?

**Answer:** If JVM has only 50% of its maximum memory available during a load test:

* **Frequent Garbage Collection (GC):** The JVM will invoke GC more often, especially Full GC, which can lead to longer pause times.
* **Higher CPU Usage:** More frequent GC leads to more CPU being consumed.
* **Memory Pressure:** Application threads might wait longer for memory allocation.
* **Risk of OutOfMemoryError (OOM):** If memory isn’t released fast enough, the app may crash.
* **Performance Degradation:** Increased latency, timeouts, and failed transactions.
* **Throughput Impact:** Lower TPS due to GC overhead.

➡️ It’s critical to monitor GC, heap usage, and tune memory settings or scale up/down JVMs accordingly.

#### ❓ Difference between lr\_abort() and lr\_exit() in LoadRunner?

**Answer:** Both functions are used to control script execution but differ in behavior:

| Feature | lr\_abort() | lr\_exit() |
| --- | --- | --- |
| Purpose | Immediately stops the script execution | Gracefully exits at a given point |
| Result | Marks iteration as failed | Can exit as pass or fail depending on argument |
| Execution Flow | Skips remaining actions and test logic | Allows some post-code execution if needed |
| Use Case | On critical failure (e.g., no token from login) | For conditional exits (e.g., skip rest on condition) |

Examples: - lr\_abort(); → used when critical data is missing - lr\_exit(LR\_EXIT\_ACTION\_AND\_CONTINUE, LR\_AUTO); → used when we want to skip further actions

#### ❓ What metrics are shared in the final performance test report?

**Answer:** In the final report we include both technical and business-facing metrics:

**1. Test Summary:** - Test goal - Scenario type and duration - Number of Vusers

**2. Throughput Metrics:** - Hits/sec - Throughput (KB/sec) - Bandwidth utilization

**3. Transaction Metrics:** - Avg/Min/Max response time - 90th percentile - Pass/Fail counts - TPS

**4. Error Metrics:** - HTTP/500/504 errors - Application errors - Script-level errors

**5. Vuser Metrics:** - Running vs. completed users - Iteration summary

**6. System Resource Metrics:** - CPU usage - Memory utilization - Disk I/O - Network stats

**7. APM/Infra Metrics:** - GC activity - Heap usage - DB query time - External services

**8. Visuals/Graphs:** - TPS vs. response time - Errors over time - Memory usage trends

**9. Summary & Observations:** - Bottlenecks - Recommendations - SLA status

#### ❓ What are the different types of performance testing?

**Answer:** Performance testing is an umbrella term for various testing approaches:

1. **Load Testing:** To test system behavior under expected load.
2. **Stress Testing:** To test limits by increasing load until system breaks.
3. **Spike Testing:** To evaluate performance with sudden traffic spikes.
4. **Endurance Testing:** (a.k.a Soak testing) to check long-term stability.
5. **Scalability Testing:** To test performance when scaling up/down resources.
6. **Volume Testing:** Focuses on large amounts of data, not users.
7. **Configuration Testing:** Checks behavior with different environment configs.
8. **Baseline Testing:** Establishes a known performance level.
9. **Benchmark Testing:** Compares against SLAs or competitors.
10. **Failover Testing:** Tests system recovery mechanisms.

#### ❓ Alternative names for performance testing types?

**Answer:** | Standard Name | Alternate Name | |————————|————————————| | Load Testing | Volume Load Testing | | Stress Testing | Breakpoint/Limit/Torture Testing | | Spike Testing | Burst/Surge Load Testing | | Endurance Testing | Soak/Longevity Testing | | Scalability Testing | Elasticity Testing | | Volume Testing | Flood/Data Load Testing | | Baseline Testing | Reference Load Testing | | Benchmark Testing | SLA Compliance Testing |

#### ❓ How to model workload to achieve 30 TPS?

**Answer:** **Goal:** Achieve 30 Transactions Per Second (TPS)

1. **Understand Transaction Duration**
   * Assume each iteration = 1 transaction = 10 seconds
2. **Calculate Users Required**
   * TPS = (Users × Iterations) / Test Duration
   * Users = (TPS × Duration) / Iterations
3. **Sample Calculation:**
   * If 1 user performs 1 txn every 10s → 0.1 TPS
   * To get 30 TPS → Need 300 users
4. **Use Pacing & Think Time** to control iteration rate
5. **Distribute TPS Across Transactions:**
   * Login = 5 TPS, Search = 10 TPS, Checkout = 15 TPS

#### ❓ What does Transactions per Second (TPS) mean?

**Answer:** **TPS = Total successful transactions / total test duration (in seconds)**

* **What is a transaction?** A business action like login, search, purchase
* **Example:** 100 users × 2 transactions per 10 seconds = 20 TPS
* **Types of TPS:**
  + **Avg TPS**: Overall
  + **Peak TPS**: Max observed
  + **Per Transaction TPS**: Split per action
* **Important:**
  + TPS ≠ Users/sec
  + Only count successful transactions
  + TPS is impacted by think time and pacing

#### ❓ How to monitor Garbage Collection (GC) in APM tools?

**Answer:** **GC monitoring is vital for Java-based applications.**

**Key GC Metrics to Monitor:** - GC count (Young/Old gen) - GC pause time - Memory before/after GC - Frequency of Full GCs

**Tools & Where to Monitor:** - **Dynatrace:** JVM section → GC metrics, heap usage graphs - **New Relic:** JVM tab → GC Time, heap - **AppDynamics:** Memory tab → GC duration, survivor space - **Elastic APM/Prometheus:** Java agent or exporter needed

**When to be concerned:** - GC > 10% of total time - Full GC every few seconds - Heap not reclaiming memory

#### ❓ What metrics do we monitor in APM tools?

**Answer:** **APM (Application Performance Monitoring) Tools** monitor:

**Application Metrics:** - Avg response time - Error rate - Transactions/sec (TPS) - Apdex score

**Infrastructure Metrics:** - CPU/Memory usage - Disk I/O and latency - Network throughput

**JVM Metrics:** - GC activity - Heap/Non-heap memory - Thread states

**Database Metrics:** - Slow queries - Lock time - Connection pool usage

**External Services:** - 3rd party API performance - Call success/failure rates

**Real User Monitoring (RUM):** - Page load time - DOM processing - Frontend/backend split

**Tracing:** - Distributed trace path - Service-to-service timing

**Alerting & Baselines:** - Anomalies - SLA violations - Custom thresholds

#### ❓ Define Pacing in performance testing

**Answer:** **Pacing = Time between end of one iteration and start of the next**

**Why It Matters:** - Controls load generation - Helps simulate real user delays - Used to reach specific TPS

**In LoadRunner:** - Runtime Settings → Pacing tab - Options: Immediately, fixed, random, based on iterations/hour

**In JMeter:** - Use Timers like: - Constant Timer - Uniform Random Timer - Throughput Shaping Timer (plugin)

**Example:** - Script runs in 5 seconds - To run every 15 seconds → Add 10s pacing

**Key Point:** Pacing + Think Time = Controls iteration rate and TPS