Adjusting Throughput in JMeter as per Requirements and Execution Model

Throughput is one of the **most critical performance parameters** in **Apache JMeter**, defining how many requests are successfully processed per unit of time. **Optimizing throughput** involves carefully configuring JMeter to match the test requirements while considering factors like test execution models, server capacity, concurrency limits, and network constraints.

This guide covers **detailed technical steps** on how to **adjust throughput in JMeter** based on **different execution models** and **testing requirements**.

---

1. Understanding Throughput in JMeter

📌 What is Throughput?

Throughput in JMeter is measured as **the number of successful requests per second (or per minute) processed by the server**.

💡 **Formula for Throughput in JMeter**:

Throughput = (Total Requests) / (Total Time)

📌 Throughput vs. Response Time vs. Concurrency

- **Throughput**: Number of requests processed per second.
- **Response Time**: Time taken for a single request.
- **Concurrency (Threads)**: Number of virtual users (VUs) running simultaneously.

🚀 High Throughput ≠ High Concurrency

- A system **may handle 500 TPS (transactions per second)** with just **50 concurrent users** if response times are fast.
- Conversely, a **slow server** could require **500+ concurrent users** to achieve the same throughput.

---

2. Execution Models in JMeter for Throughput Control

The **execution model** determines how JMeter generates and controls throughput.

Execution Model	Use Case	Key Strategy for Throughput Control
Open Model	**API Performance Testing (SLA Validation, Load Testing)**	Use **Throughput Shaping Timer** or **Constant Throughput Timer** to **control requests per second**.
Closed Model	**User Flow Testing (E2E, UI-based Load Tests)**	Use **Thread Groups** to **control concurrency and pacing**.
Hybrid Model	**Complex Workloads (Mix of UI + API Requests)**	Use a **combination of timers, pacing, and concurrency control**.

---

3. Techniques to Adjust Throughput in JMeter

To adjust throughput in JMeter, you can use the following **four primary techniques**:

📌 3.1 Using Throughput Timers (Best for Open Workloads)

JMeter provides **timers** that can be used to **throttle request rates**.

🛠 3.1.1 Constant Throughput Timer

✅ **Best for:** API Testing, SLA Compliance, Fixed TPS
🛠 **How It Works:**

- Ensures a fixed **Transactions per Second (TPS)**, **irrespective of the number of threads**.
- Works **per thread group** and applies delays to maintain the desired throughput.

⚙️ **Configuration Steps:**

1. Add **Constant Throughput Timer** (Test Plan > Thread Group > Add > Timer > Constant Throughput Timer).
2. Set **Target Throughput (Requests per minute)** in Throughput (in samples per minute).
   - Example: For **50 TPS**, set **3000 samples per minute**.
3. Set **“Calculate Throughput Based On”** to:
   - **All Active Threads** *(Recommended)*
   - **Current Thread Group** *(Use when multiple thread groups exist)*

📝 **Example:** Set **Constant Throughput Timer = 6000 samples/min** to achieve **100 TPS**.

🚨 **Key Considerations:**

- If JMeter **cannot achieve** the requested throughput (due to **low thread count or slow response time**), it will fail to meet the target.
- **Does NOT control concurrency**; only regulates request rate.

🛠 3.1.2 Throughput Shaping Timer (Ultimate Control Over Load Patterns)

✅ **Best for:** Load Tests with increasing TPS (Ramp-Up)
🛠 **How It Works:**

- Allows **precise control of throughput over time** (e.g., ramp-up from 10 TPS → 100 TPS → 200 TPS).
- Helps simulate real-world traffic patterns.
- Works **better than the Constant Throughput Timer** for **dynamic workloads**.

⚙️ **Configuration Steps:**

1. Install the **Throughput Shaping Timer Plugin** from **JMeter Plugins Manager**.
2. Add **Throughput Shaping Timer** (Test Plan > Thread Group > Add > Timer > Throughput Shaping Timer).
3. Define the **Target Request Rates (TPS)** for different time intervals:

Time (Seconds)	**Target TPS**
0-30s	50 TPS
30-60s	100 TPS
60-90s	200 TPS

4. JMeter will **automatically adjust the request rate** to match the defined load profile.

📝 **Example:**

Ramp-Up: 10 TPS → 50 TPS → 100 TPS → 200 TPS

🚨 **Key Considerations:**

- Needs **Concurrency Thread Group** for fine-tuned load execution.
- Ensures **load patterns follow real-world scenarios**.

📌 3.2 Using Concurrency Thread Groups (Best for Closed Workloads)

If your **test is user-flow based**, controlling **active users** is more relevant than raw request count.

🛠 3.2.1 Concurrency Thread Group

✅ **Best for:** Simulating real users, UI Load Testing
🛠 **How It Works:**

- Ensures **a steady number of concurrent users** rather than controlling raw TPS.
- Can be used **with pacing** to achieve a target throughput.

⚙️ **Configuration Steps:**

1. Install the **JMeter Plugins Manager** and add **Concurrency Thread Group**.
2. Configure the following:
    - **Target Concurrency** = Total number of users in steady state.
    - **Ramp-Up Time** = Time taken to reach full load.
    - **Hold Time** = Duration for which the test will maintain the load.

📝 **Example:**

Target Concurrency: 200 users

Ramp-Up Time: 5 min

Hold Time: 30 min

🚨 **Key Considerations:**

- Works well for **UI-based load testing** but does not guarantee a specific **TPS**.

📌 3.3 Pacing to Control Requests Per User

For tests that **simulate real user behavior**, the **pacing strategy** is critical.

🛠 3.3.1 How to Use Pacing

✅ **Best for: Workloads where each user sends requests at regular intervals**
🛠 **How It Works:**

- **Pacing = Think Time + Delay Between Iterations**
- Ensures **each user sends a limited number of requests per minute**, instead of firing continuously.

⚙️ **Configuration Steps:**

1. Add a **Test Action** sampler (Thread Group > Add > Sampler > Test Action).
2. Set a **Pause Time** to control pacing.

3. Use **Uniform Random Timer** (Timer > Add > Uniform Random Timer) to introduce realistic variation.

📝 **Example:**

User sends 1 request every 5 seconds → 12 requests per minute

🚨 **Key Considerations:**

- Required for **realistic user simulation**.
- Works **only when using a user-driven model**, not for raw throughput.

---

4. Choosing the Right Throughput Adjustment Strategy

Use Case	Best Strategy
API Performance Testing (Fixed TPS)	✅ **Constant Throughput Timer**
Load Testing with Increasing TPS	✅ **Throughput Shaping Timer**
Simulating Real User Behavior	✅ **Concurrency Thread Group + Pacing**
Mix of APIs + UI Traffic	✅ **Hybrid Model (Combination of Above)**

---

5. Key Performance Tuning Considerations

- 💡 **Monitor Server Response Time**: If response times are high, increasing throughput may lead to bottlenecks.
- 💡 **Tune JMeter's Engine**: Use -Xms4G -Xmx4G for high-load tests.
- 💡 **Correlate With Server Metrics**: Use **Grafana, Prometheus, or APM tools** to monitor CPU, memory, and database load.

🚀 **Summary**

✅ **Use Constant Throughput Timer for Fixed TPS tests**
✅ **Use Throughput Shaping Timer for Dynamic Load tests**
✅ **Use Concurrency Thread Group + Pacing for UI/UX-based tests**
✅ **Monitor, Tune, and Optimize JMeter Execution**