Web Server

Experimental Setup
Workload Characterization

Prof. Domenico Cotroneo

Prof. Pietro Liguori

pietro.liguori@unina.it

Objectives

1. Experimental Setup
   - Setup Server
   - Jmeter and httperf
   - Parameters collection

2. Capacity Test and Performance Analysis

3. Hypothesis Tests in Matlab

4. Workload Characterization

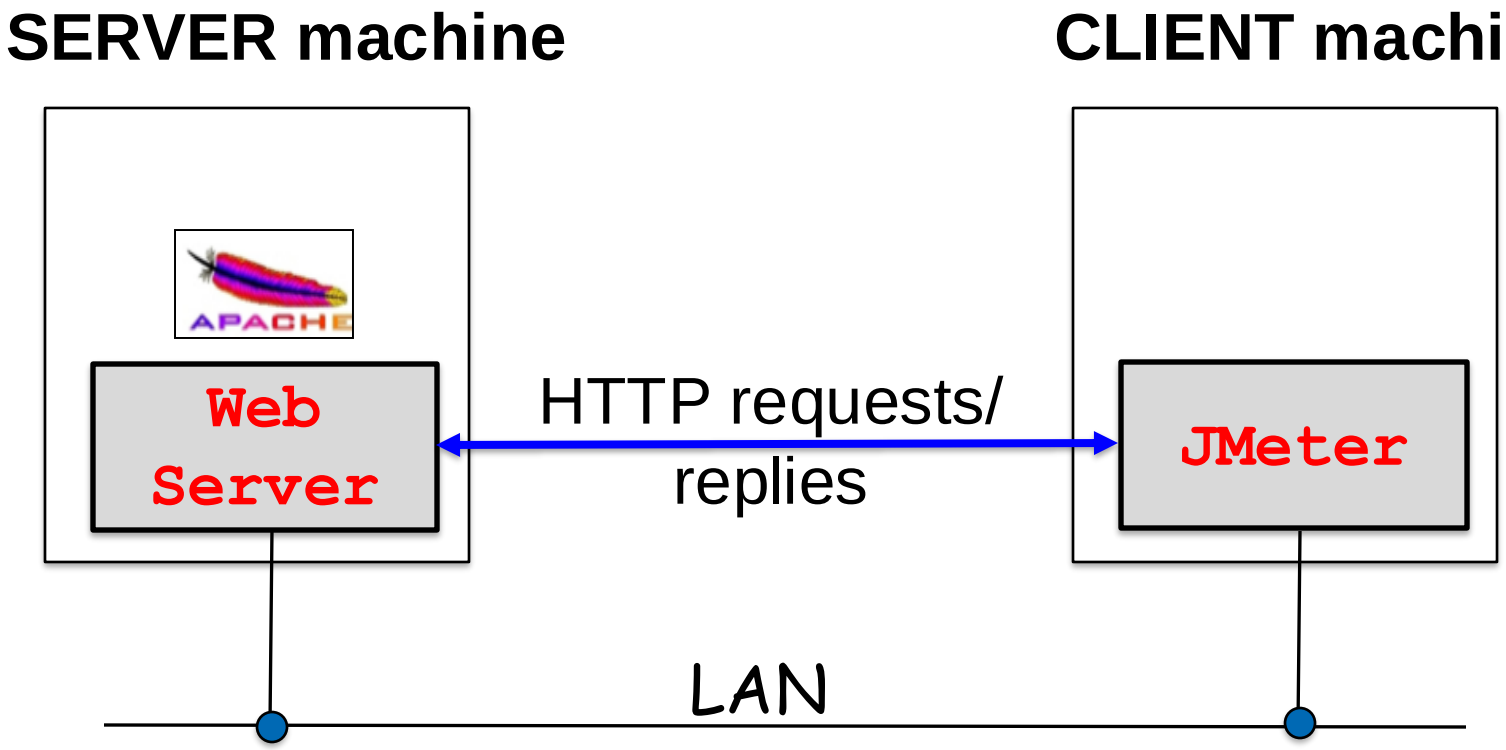
5. Experimental Design and Analysis

1. Experimental Setup

Experimental Setup

- **Apache Web Server**
- Load Generator:
  - *Apache Jmeter*
  - *httperf*
- Low-level data collection by *unix* utility
- Linux machines
  - either a physical or virtualized environment

Experimental Setup (cont.)

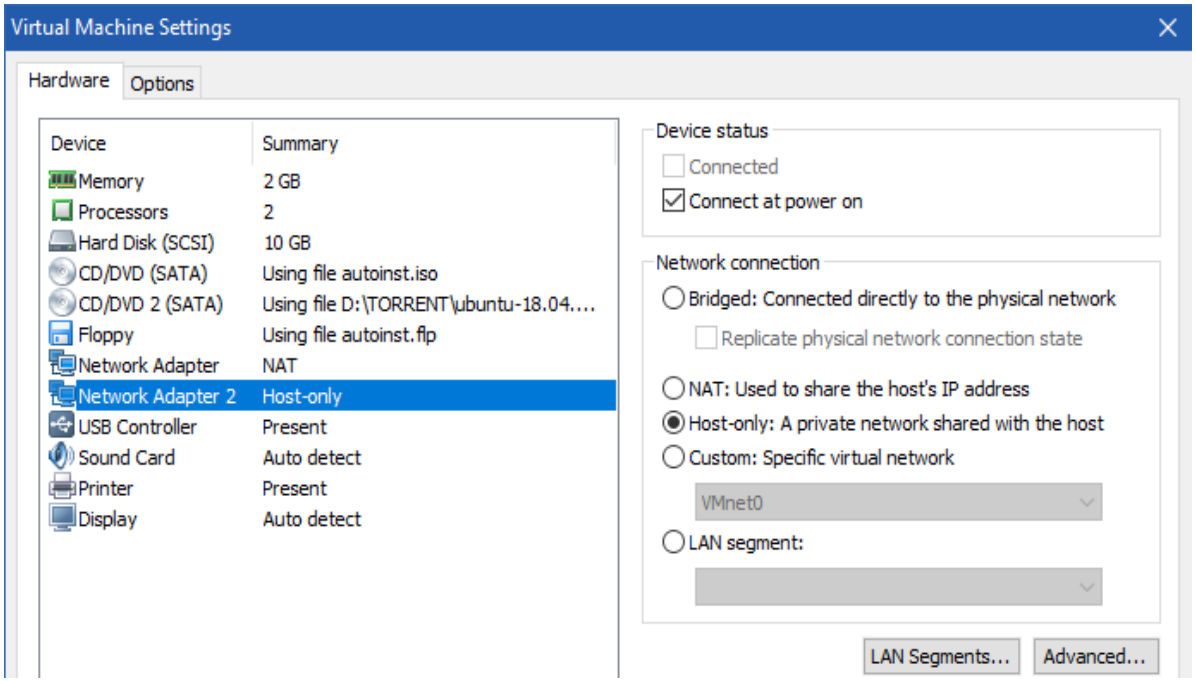


- We use virtualized environments in this study

Experimental Setup (cont.)

- In the virtualized environments:
  - Add the *Host-only Network Adapter* to enable the communication between connected guest systems and the host system

- The configuration of a static IP may be useful!



server IP address

```
server@server-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::aada:9c80:c1f6:2441  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:45:97:a0  txqueuelen 1000  (Ethernet)
        RX packets 1165  bytes 1096869 (1.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 509  bytes 50361 (50.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.101  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::7ecb:b31c:d41b:6526  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:25:1b:de  txqueuelen 1000  (Ethernet)
        RX packets 3  bytes 993 (993.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 61  bytes 6440 (6.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Apache Web Server

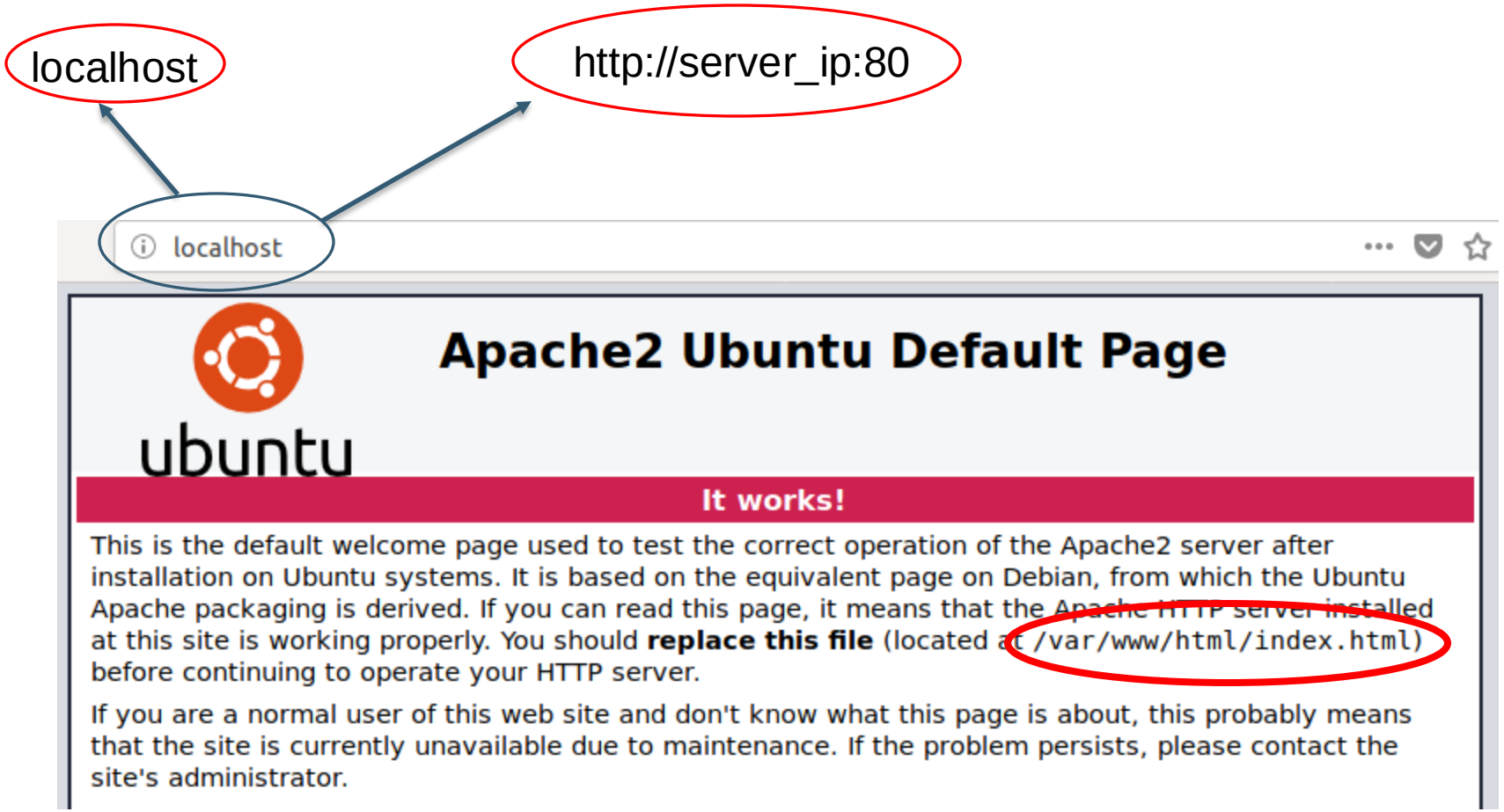
- Apache Web Server:
  - version 2 is used in this study
  - Install it on server machine

  (e.g., on Debian-based distribution: $ sudo apt-get install apache2)

- Main commands :
  - $ service apache2 start: starts the web server
  - $ service apache2 stop: stops the web server

- Testing the installation:
  - Default port: 80

Testing the installation



Apache JMeter

Apache JMeter

- The **Apache JMeter™** application is an open-source software, a 100% pure Java application designed to load test functional behavior and measure performance.

- Apache Jmeter
  - https://jmeter.apache.org/download_jmeter.cgi
  - binaries: apache-jmeter-version.zip

N.B.: Check out the manual at http://jmeter.apache.org/usermanual/

Apache Jmeter (cont.)

- **Installation**: unzip the zip/tar file into the directory where you want to install JMeter (e.g., home directory)

- Java Runtime Environment (JRE) required:
  - *https://www.java.com/it/download/*
  - $ sudo apt-get install default-jre

    (on Debian-based distribution)

- Run *jmeter.sh* in the bin directory
  - $ ./apache-jmeter-version/bin/jmeter.sh

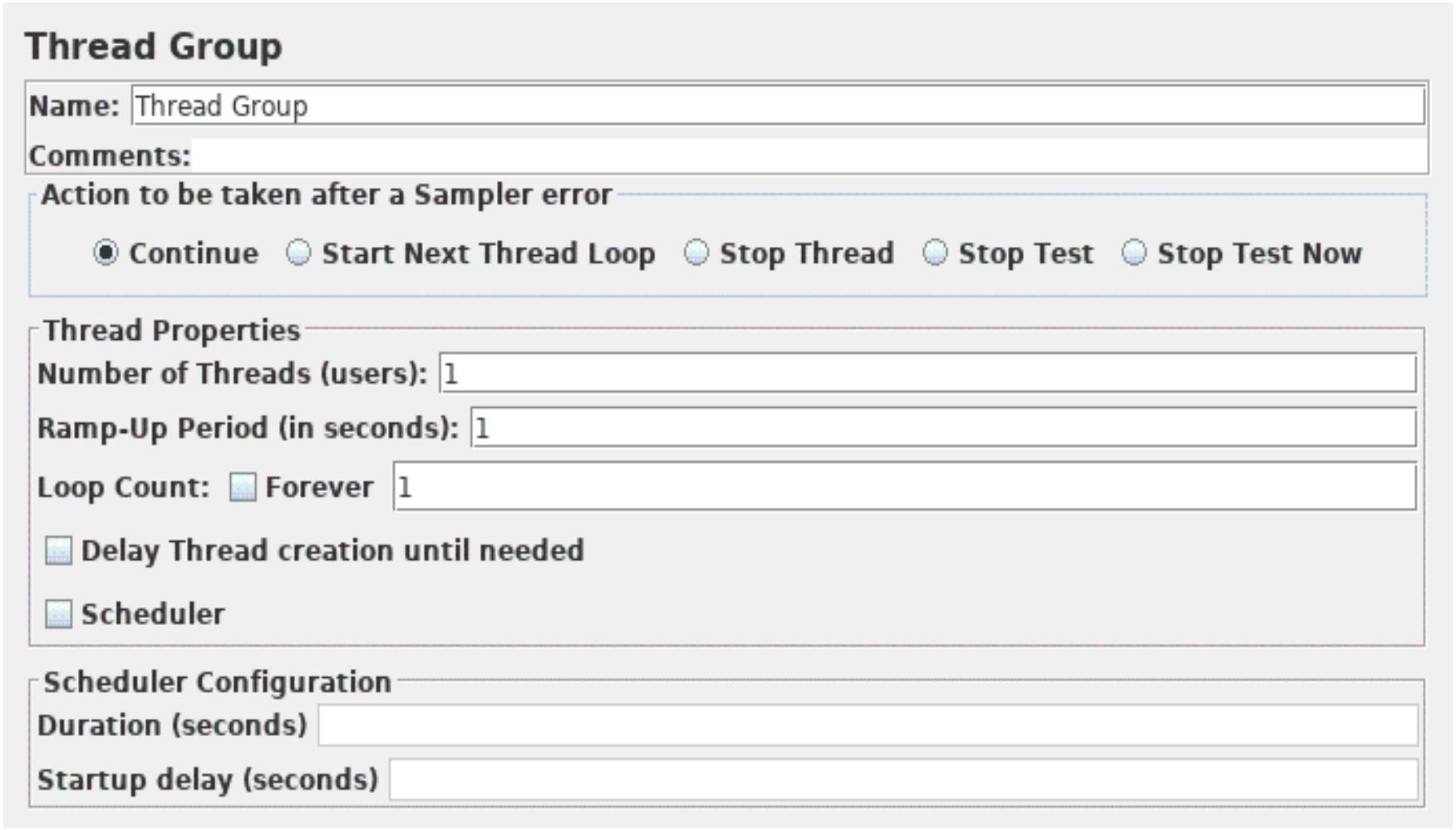
Main elements of a Test Plan

- Thread Group
- Samplers
- Controllers
- Timers
- Listener

Thread Groups

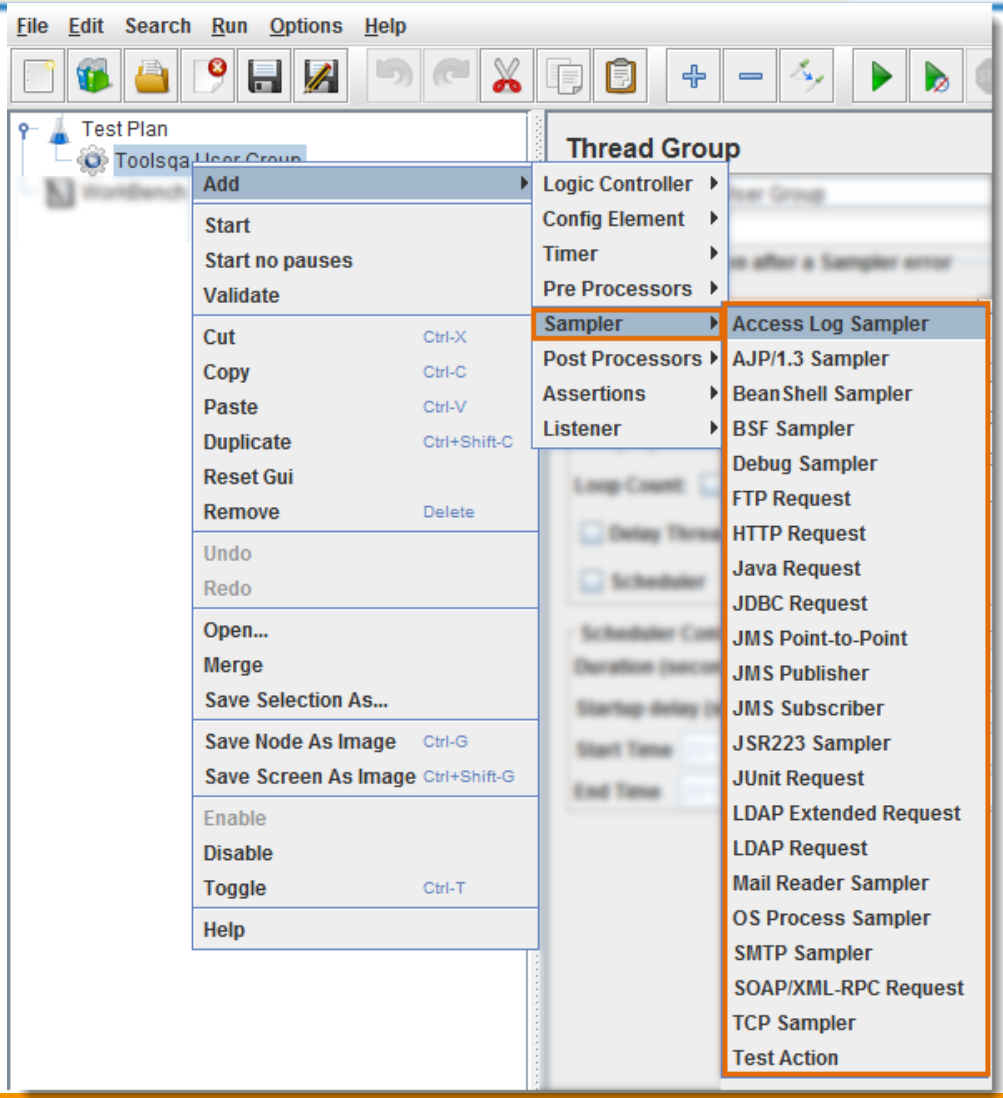
- ***ThreadGroup***
  - It controls the number of threads JMeter will use to execute your test. Allows to:
    - Set the number of threads (users)
    - Set the ramp-up period
    - Set the number of times to execute the test (Loop Count)

Thread Groups



Samplers

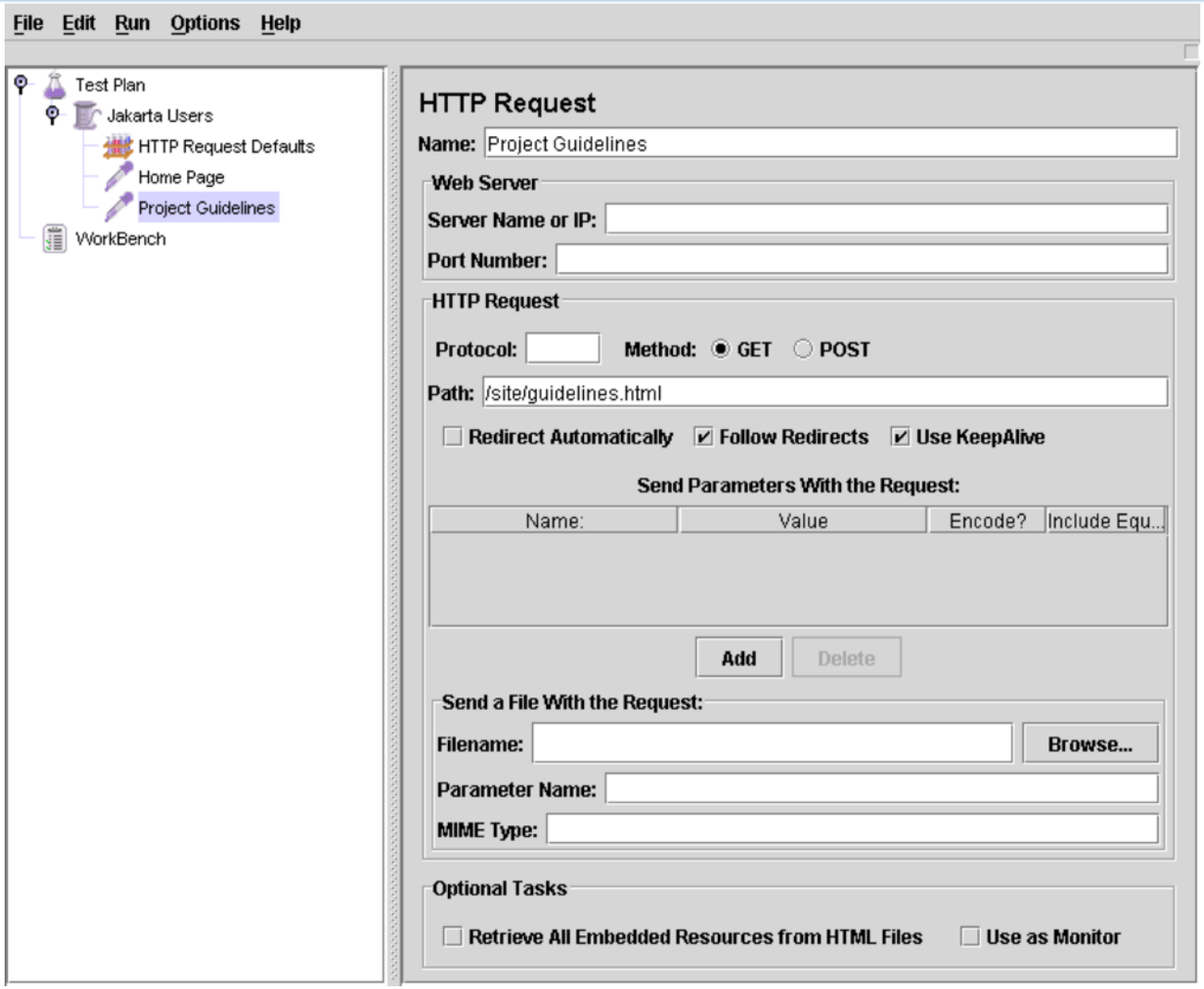
- ***Sampler***
  - Samplers tell JMeter to send requests to a server and wait for a response (e.g., ***HTTP Request Sampler*** to send an HTTP request)



Samplers: HTTP Request

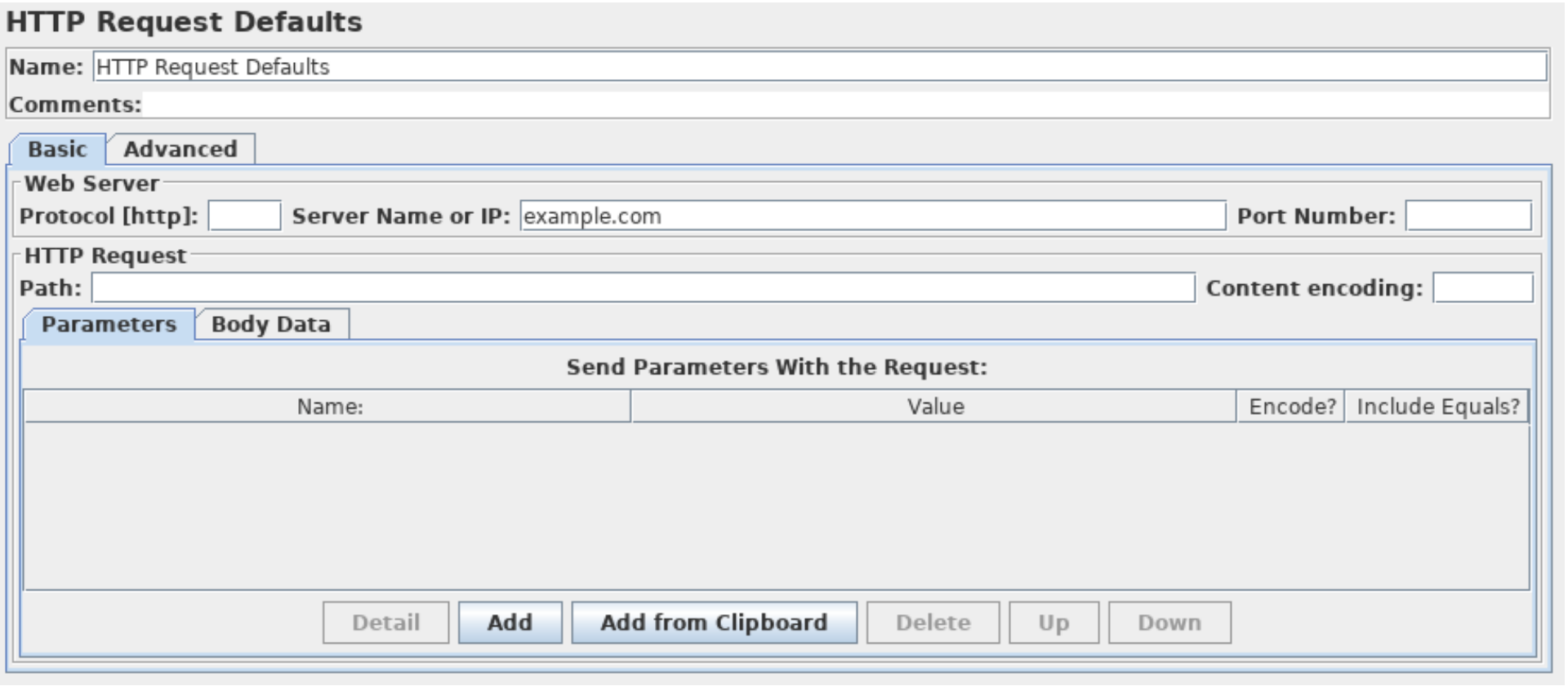
SAMPLER:

Add ***HTTP Requests***: you should add an HttpRequest sampler for each page you want to test. You just need to specify the path in the textbox (relative to the web server' root directory).



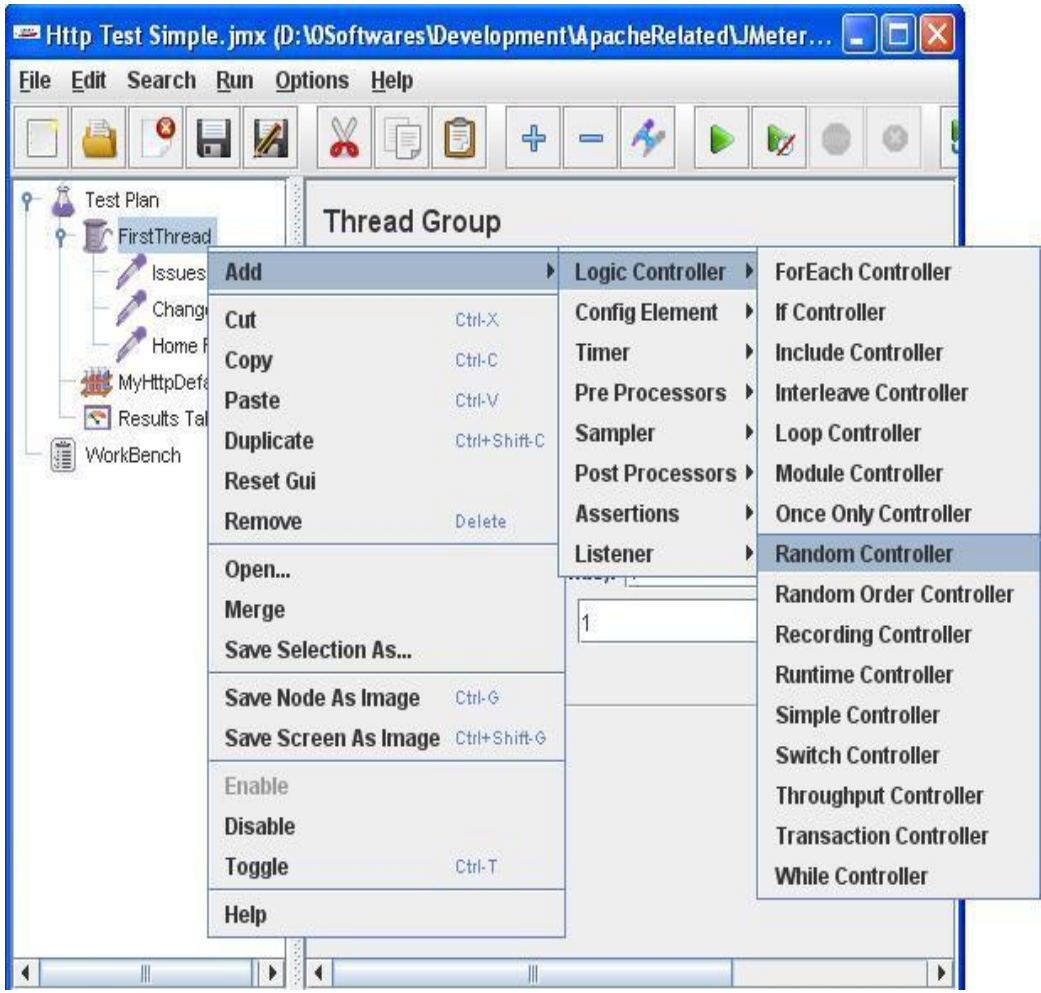
Config Element: HTTP Request Default

- ***HTTP Request Default*** lets you set default values that your HTTP Request controllers use.



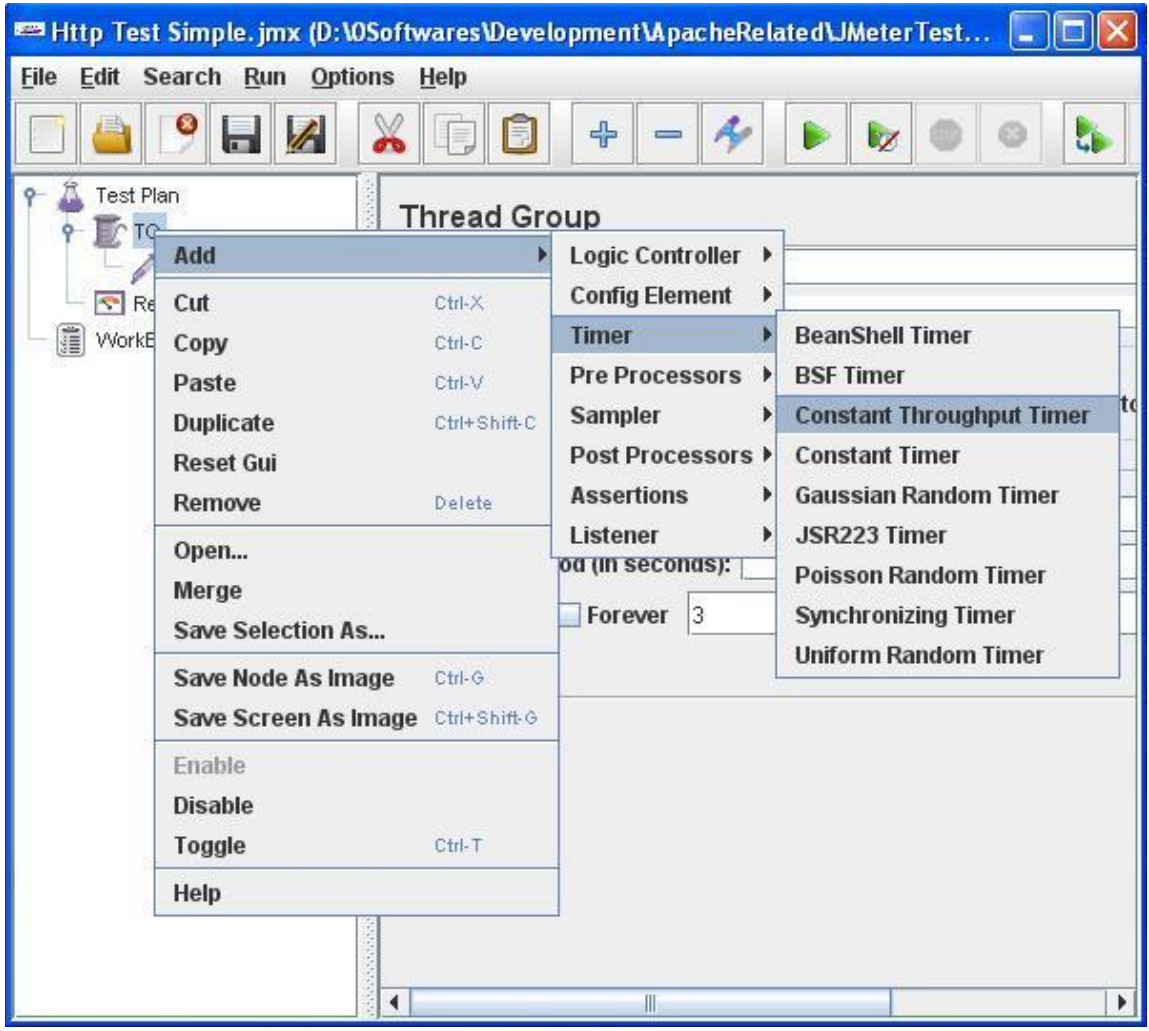
Controllers

- ***Logic Controller***
  - Let you customize the logic that JMeter uses to decide when to send requests (e.g., *Simple Controllers, Loop Controllers, Interleave Controllers,* ***Random Controller)***
  - **Add the HTTP request to the controller**

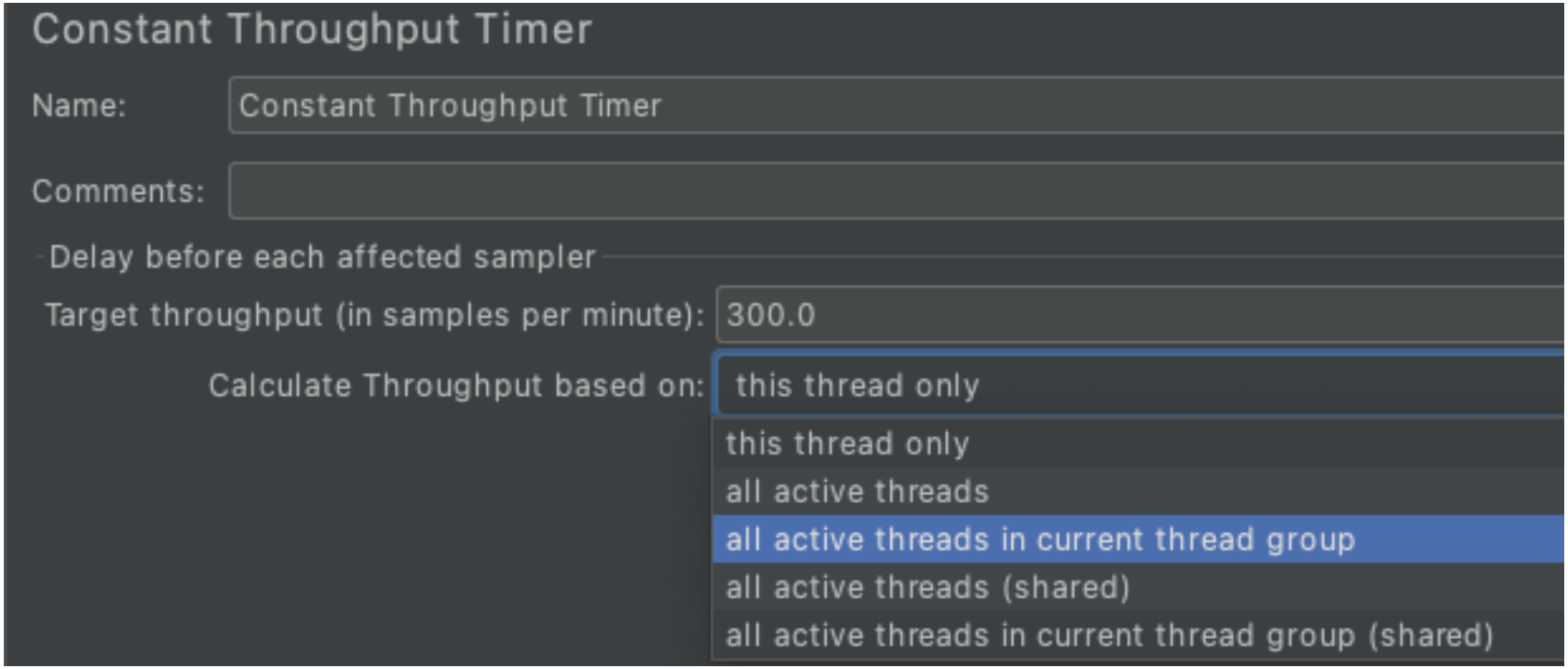


Timers

- ***Timer***
  - Each request in JMeter is made immediately after the another if there is NOT a Timer
  - Timer let you spacing out the samplers requests



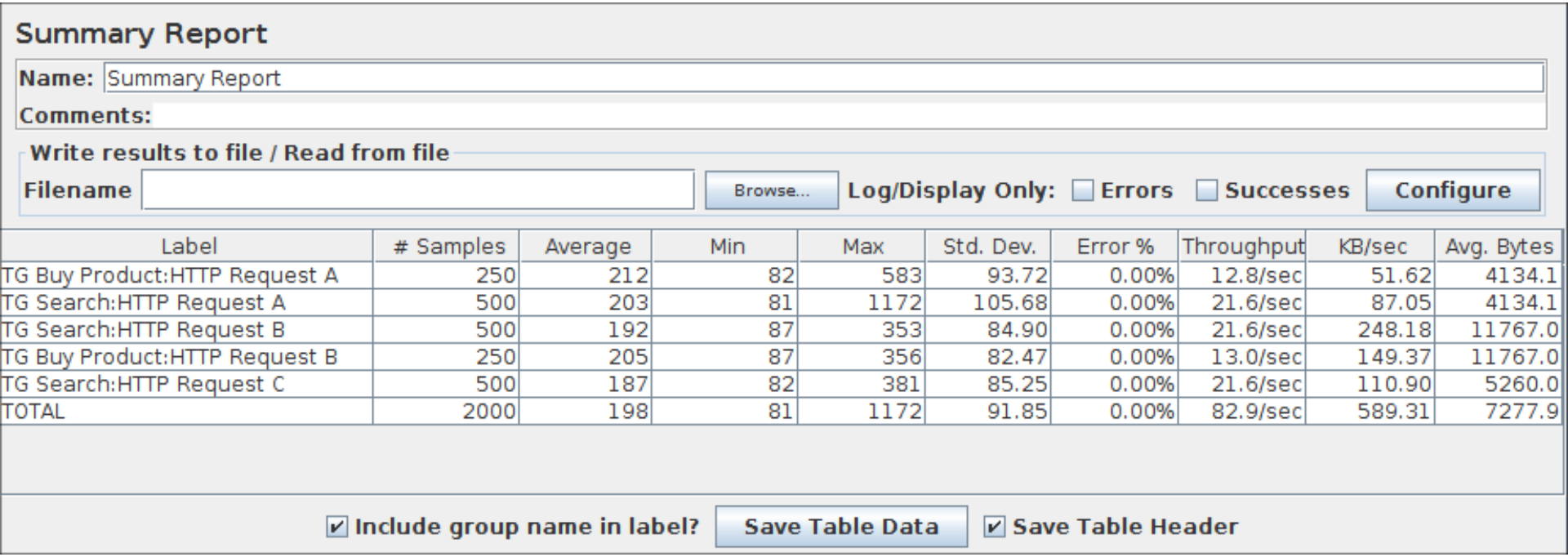
Constant Throughput Timer (CTT)



Listeners

- ***Listeners***
  - Provide access to the information JMeter gathers about the test cases while JMeter runs
  - Display the same response information in different ways (tree tables, graphs)
  - Can collect response time, latency, throughput, #of errors, etc.

Listener Example: Summary Report



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
TG Buy Product:HTTP Request A	250	212	82	583	93.72	0.00%	12.8/sec	51.62	4134.1
TG Search:HTTP Request A	500	203	81	1172	105.68	0.00%	21.6/sec	87.05	4134.1
TG Search:HTTP Request B	500	192	87	353	84.90	0.00%	21.6/sec	248.18	11767.0
TG Buy Product:HTTP Request B	250	205	87	356	82.47	0.00%	13.0/sec	149.37	11767.0
TG Search:HTTP Request C	500	187	82	381	85.25	0.00%	21.6/sec	110.90	5260.0
TOTAL	2000	198	81	1172	91.85	0.00%	82.9/sec	589.31	7277.9

Include group name in label? | Save Table Data | Save Table Header

httperf

httperf

- httperf is a tool for measuring web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring server performance.

- Installation:
  - $ sudo apt-get install httperf

  (on Debian-based distribution)

- Example of use:
  - $ httperf --hog --server 192.168.185.143 --port 80 --uri /index.html.it --rate 1 --num-conn 1 --num-call 1 --timeout 10

httperf

$ httperf --hog --server 192.168.185.143 --port 80 --uri /index.html.it --rate 1 --num-conn 1 --num-call 1 --timeout 10

- hog: allows httperf to request as many TCP ports it needs
- sever: IP or hostname of the server
- port: port of the server
- rate: rate at which connections are created (conns/second)
- num-conn: total number of connections
- num-call: number of calls per connection
- timeout: the time (in seconds) httperf will wait for a server response

httperf example

$ httperf --hog --server 192.168.185.143 --port 80 --uri /index.html.it --rate 10 --num-conn 100 --num-call 10 --timeout 10

Total: connections 100 requests 1000 replies 1000 test-duration 9.943 s

Connection rate: 10.1 conn/s (99.4 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 8.2 avg 43.3 max 144.4 median 36.5 stddev 23.7
…
Request rate: 100.6 req/s (9.9 ms/req)
Request size [B]: 81.0

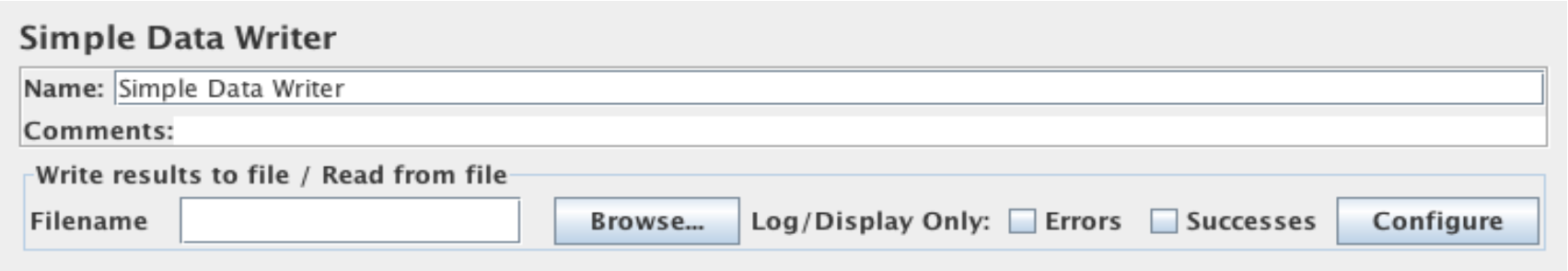
Reply rate [replies/s]: min 100.0 avg 100.0 max 100.0 stddev 0.0 (1 samples)
Reply time [ms]: response 3.8 transfer 0.1
Reply size [B]: header 251.0 content 1788.0 footer 0.0 (total 2039.0)
Reply status: 1xx=0 2xx=1000 3xx=0 4xx=0 5xx=0

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Parameters Collection

Collecting high-level parameters

- You can collect the high-level parameters by using JMeter listener
  - Use a ***simple data writer*** listener (the others consume a lot of memory).
  - Save on a file (see image below), and after the test, you can use other listeners, by loading the written result file (graph, table, result in table, summary report)

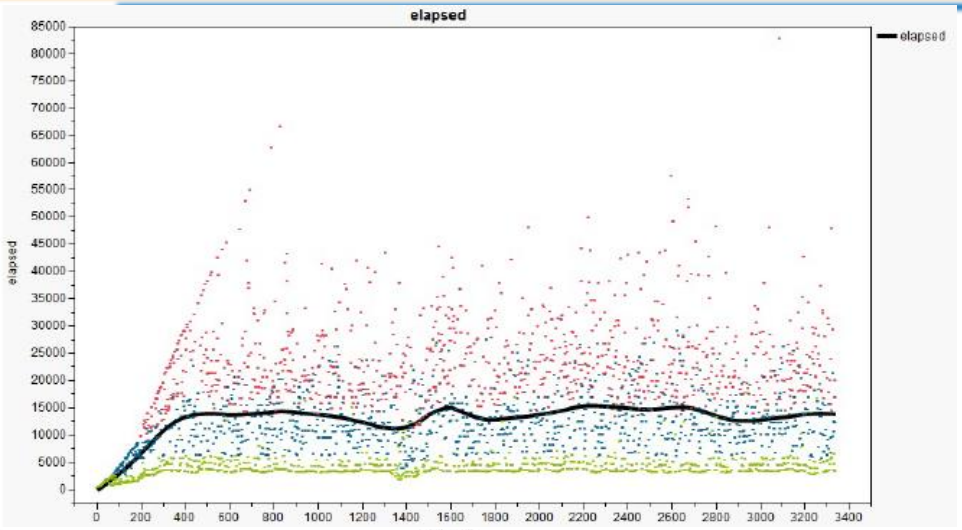


Collecting high-level parameters

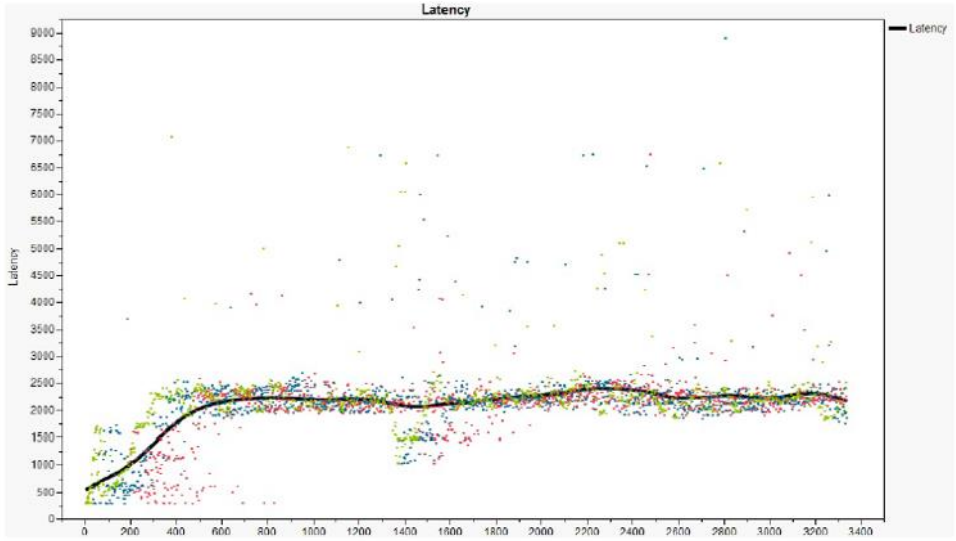
- Use “Configure” to choose format (use *csv*) and fields
- Important parameters related to the request. At least:
  - Timestamp: when the request is made
  - Thread name: who made the request
  - Page (Label): what page (type) required
  - Byte count: how many bytes exchanged

- Choose also parameters related to the response, for successive performance analysis. At least:
  - Latency, elapsed time, success/errors, for throughput and response time analysis

Elapsed vs Latency



Elapsed



Latency

Small
Medium
Large

Simple Data Writer

A	B	C	D	E	F
timeStamp	elapsed	label	responseCode	responseMessage	threadName
1,64E+12	15	HTTP Request 10	200	OK	Thread Group 1 1-4
1,64E+12	29	HTTP Request 19	200	OK	Thread Group 1 1-1
1,64E+12	105	HTTP Request 7	200	OK	Thread Group 1 1-3
1,64E+12	10	HTTP Request 2	200	OK	Thread Group 1 1-6
1,64E+12	13	HTTP Request 8	200	OK	Thread Group 1 1-7
1,64E+12	22	HTTP Request 3	200	OK	Thread Group 1 1-26
1,64E+12	16	HTTP Request 7	200	OK	Thread Group 1 1-8
1,64E+12	44	HTTP Request 6	200	OK	Thread Group 1 1-2
1,64E+12	26	HTTP Request 18	200	OK	Thread Group 1 1-10
1,64E+12	30	HTTP Request 16	200	OK	Thread Group 1 1-9
1,64E+12	24	HTTP Request 4	200	OK	Thread Group 1 1-13
1,64E+12	19	HTTP Request 15	200	OK	Thread Group 1 1-29
1,64E+12	17	HTTP Request 13	200	OK	Thread Group 1 1-30
1,64E+12	19	HTTP Request 12	200	OK	Thread Group 1 1-15
1,64E+12	41	HTTP Request 17	200	OK	Thread Group 1 1-11
1,64E+12	21	HTTP Request 1	200	OK	Thread Group 1 1-17
1,64E+12	21	HTTP Request 7	200	OK	Thread Group 1 1-20
1,64E+12	13	HTTP Request 4	200	OK	Thread Group 1 1-27
1,64E+12	21	HTTP Request 12	200	OK	Thread Group 1 1-5

How to compute throughput and response time?

- $Throughput = \frac{total\ number\ of\ requests\ correctly\ served}{duration\ (seconds)}$

  - Total number of requests is the number of rows of the file (Response message OK)

  - $Duration = \frac{\max(timestamp) - \min(timestamp)}{1000}$

- Response time = average of elapsed

Collecting Low-Level Info

Information to characterize the system's resources usage

Some useful information

- Total Memory Used/Free
- Resident set Size (Real used memory)
- VM Size
- Used SwapSpace
- Buffer
- Cache
- Shared Memory

Linux Utilities to capture

Top
Free
Ps
Vmstat
…

Workload-related measurements:

- CPU utilization
- Page-in/out
- Disk Activity

vmstat example

- vmstat (virtual memory statistics) is a valuable monitoring utility, which also provides information about block IO and CPU activity in addition to memory.

- $ vmstat –n 1 10 > output.txt

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  1      0 8393504 106984 2372420    0    0   938   753  310  884 10  3 54 34  0
 0  1      0 8390172 108044 2374520    0    0  1192     0 1116 3624  8  3 67 22  0
 0  1      0 8385140 110748 2377560    0    0  2948    16 1414 3868  8  3 67 21  0
 0  1      0 8258408 114888 2499896    0    0  4524     0 1385 2757  6  3 72 20  0
 0  1      0 8242244 119248 2511516    0    0  4744     0 1423 3366  5  3 70 21  0
 0  1      0 8324984 122800 2424924    0    0  3808     0 1392 3755  5  3 72 20  0
 0  2      0 8332164 127256 2413380    0    0  4968    12 1381 2925  1  1 75 23  0
 0  1      0 8328532 129328 2415024    0    0  2192     0  811 1480  0  2 77 21  0
 0  1      0 8312084 134208 2426764    0    0  5264     0 1401 2705  1  1 74 24  0
 0  2      0 8300944 139492 2432556    0    0  5796  4464 1454 2528  0  2 75 24  0
```

vmstat example (cont.)

- **Procs**
  - **r**: The number of processes waiting for run time.
  - **b**: The number of processes in uninterruptible sleep.
- **Memory**
  - **swpd**: the amount of virtual memory used.
  - **free**: the amount of idle memory.
  - **buff**: the amount of memory used as buffers.
  - **cache**: the amount of memory used as cache.
  - **inact**: the amount of inactive memory. (-a option)
  - **active**: the amount of active memory. (-a option)
- **Swap**
  - **si**: Amount of memory swapped in from disk (/s).
  - **so**: Amount of memory swapped to disk (/s).

vmstat example (cont.)

- **IO**
  - **bi**: Blocks received from a block device (blocks/s).
  - **bo**: Blocks sent to a block device (blocks/s).
- **System**
  - **in**: The number of interrupts per second, including the clock.
  - **cs**: The number of context switches per second.
- **CPU**
  - **us**: Time spent running non-kernel code. (user time, including nice time)
  - **sy**: Time spent running kernel code. (system time)
  - **id**: Time spent idle. Prior to Linux 2.5.41, this includes IO-wait time.
  - **wa**: Time spent waiting for IO. Prior to Linux 2.5.41, included in idle.
  - **st**: Time stolen from a virtual machine. Prior to Linux 2.6.11, unknown.

2. Capacity Test and Performance Characterization

Capacity Test

- **Capacity test**: characterize performance in the limit
  - Objective: find the limits of the system under study
- Useful to capacity planning, management, and performance tuning
- More generally, performance analysis refers to characterizing performance under several working conditions
- Performance can be characterized by several indicators
  - In the web server example, typical indicators are: *Response time*, *Response rate* (*throughput*), *Latency*, *Number of errors* (i.e., *requests failed*), …

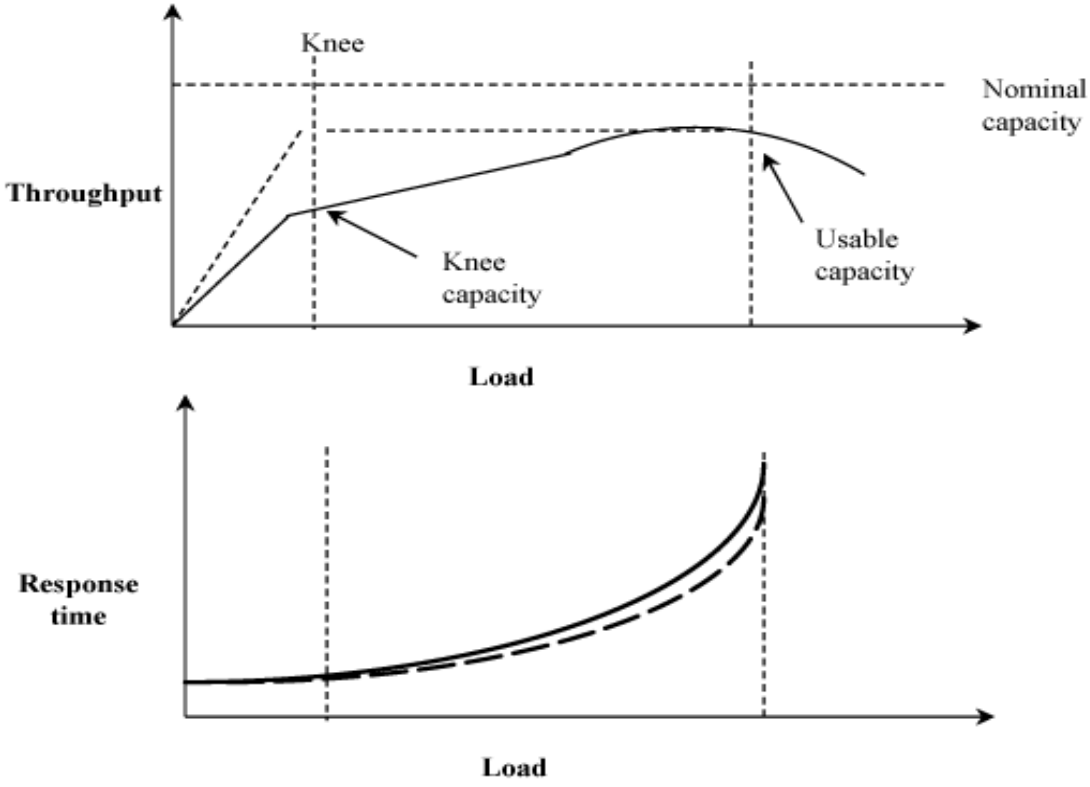
Commonly Used Performance Metrics

- **Response time**: the interval between a user's request and the system response
  i. the interval between the end of a request submission and the beginning of the corresponding response from the system
  ii. or the interval between the end of a request submission and the end of the corresponding response from the system
  - The response time of a system generally increases as the load on the system increases.

- **Throughput**: the rate (requests per unit of time) at which the requests can be serviced by the system
  - e.g., it is measured in MIPS for CPU
  - The throughput of a system generally increases as the load on the system initially increases. After a certain load, the throughput stops increasing; in most cases, it may even start decreasing

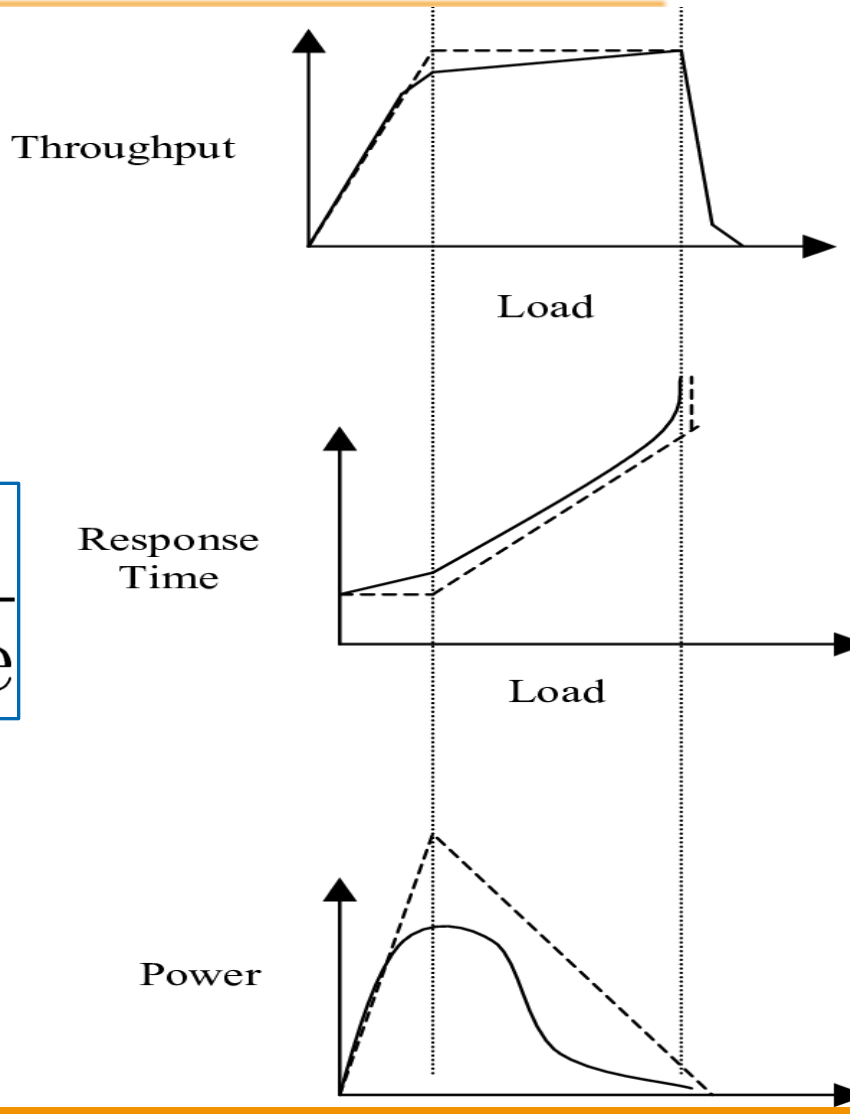
Capacity

- **Nominal Capacity**: maximum achievable throughput under ideal workload conditions
- **Usable Capacity**: maximum throughput achievable without exceeding a prespecified response time limit
- **Knee Capacity**: the point beyond which the response time increases rapidly as a function of the load but the gain in throughput is small Before the knee, the response time does not increase significantly but the throughput rises as the load increases.



Power

$$\text{Power} = \frac{\text{Throughput}}{\text{Response Time}}$$



Capacity Test: procedure

1. Analyze throughput and response time with respect to increasing values of the load applied to the web server for a given time
   - In JMeter, you can use a Constant Throughput Timer (CTT)
   - Keep constant the configuration of the test plan (except for the load of the CTT)
2. Collect several observations (**repetitions**) for each value of the load condition
   - Use the average of the throughput and response time
3. Build the graphs of the throughput and response time when the load increases
   - *X axis*: Load
   - *Y axis*: Average Throughput and Response Time
4. Find the **knee** and the **usable** capacity
   - You can use the power to identify the knee capacity
   - **Make a "conservative" choice for the knee**

Capacity Test: procedure (cont.)

- Since different limits can be reached with different request types, capacity tests can be performed for each request type

- For example, you can perform a different capacity test for each page type:
  - Small Pages
  - Medium Pages
  - Large Pages
  - Random Pages (using a Random Controller)

Exercise 1: Capacity Test and Performance Analysis

- **Client**: With Jmeter, set the request rate, and test the server for a given time, and repeat for increasing request rates
  - At least 3 repetitions
  - Set 5 minutes per measurement
  - Do not perform **too many measurements** (it is a waste of time)
  - **Pay attention to the load value in the CTT**
- **Server:** Use a machine with low hw requirements (to reach the limit of the server)

Exercise 1: Capacity Test and Performance Analysis

- Perform the test and determine the usable capacity and the knee capacity with the **random controller** considering all the pages (i.e., our “request type”)
- Use at least 5 page types with different dimensions to create a real use case scenario
- **Jmeter Reminder**: “add” the requests to the controller

Exercise 2: Bottleneck Performance

- During your experiments, collect server data (e.g., mem, proc, disk, etc.) with vmstat or similar tools

- Plot the **average** values* (y axis) for each load (x axis) and make your considerations (bottleneck, differences between knee and usable capacity, etc)
  - *Similar to the throughput and response time in the Capacity Test

Fairness Index: example

- Measured Throughput: (50,30,50)
- Given the Fair Throughput (50,10,10)
- Normalized Throughput: $x_i = T_i / O_i$

$$f(x_1, x_2, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2}$$

- Example: 50/50, 30/10, 50/10 -> 1,3,5

- **Fairness Index = $(1+3+5)^2$ / $(3(1^2+3^2+5^2)$ = 0.77**

Exercise 3: Fairness Index

- Create a test plan with different thread groups requesting concurrently a (or a set of) resource(s) to the same server
- Use the constant throughput timer to specify the ***fair throughput*** for each thread group
- Assess the ***measured throughput*** (jmeter listener) and compute the ***normalized throughput***
- **Exercise**: *Compute **fairness index.** Any further considerations? (e.g., number of threads, type of request/resource, load, etc.)*