Case study and experimental setup

Objectives

- Workload Characterization
- Capacity Test and Experimental Design and Analysis
- Performance Degradation Analysis

Workload characterization

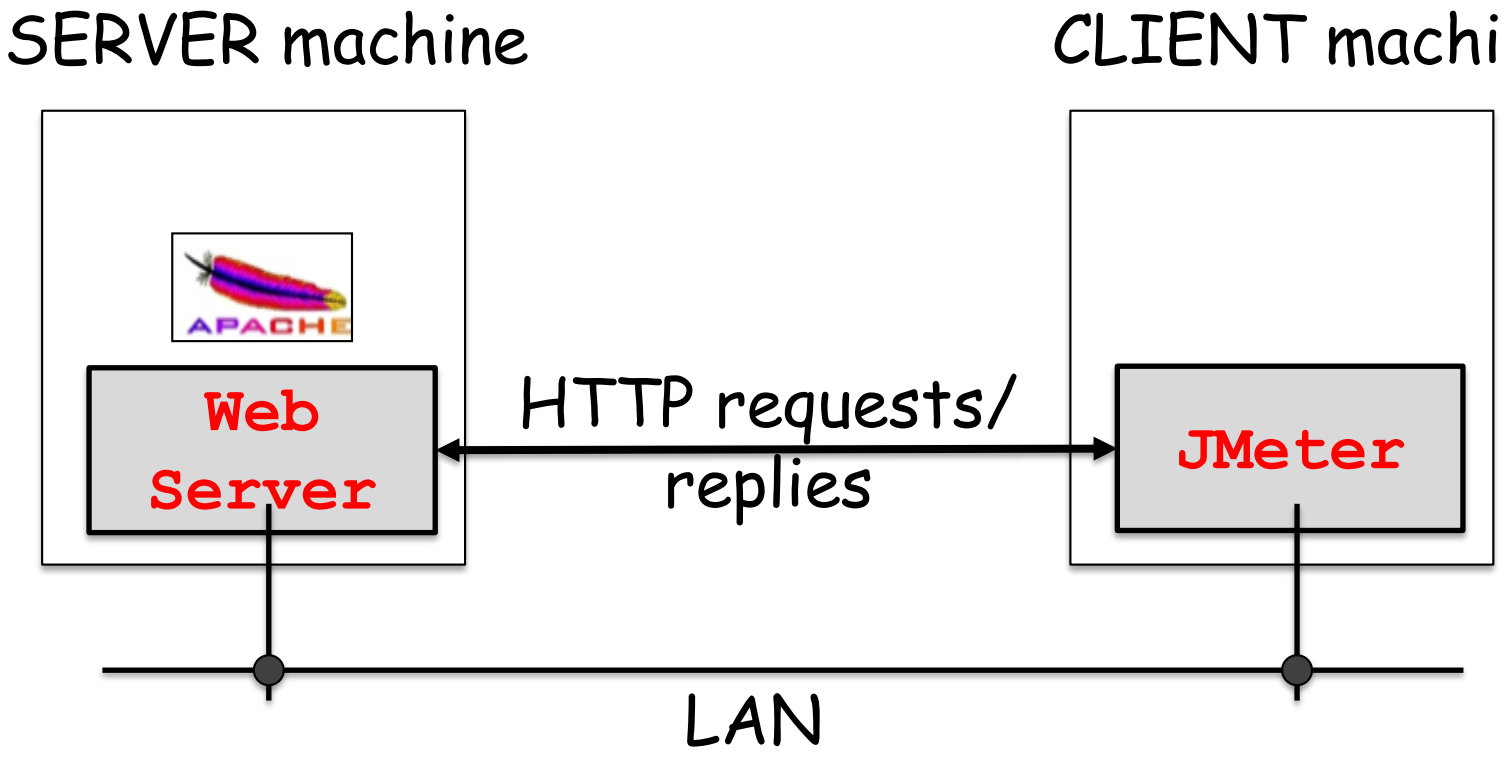
- Setup: client-server installation emulating a web server and a set of users requesting for resources to it

- Objectives
  - Monitor the workload
  - Characterize the observed workload by several techniques, comparing them
  - Characterized workload can be used to subsequent performance analysis

Experimental Setup

- Apache Web Server
- Load Generator:
  - Apache Jmeter 5.0
  - httperf
- Low-level data collection by unix utility
- Linux machines
  - either a physical or virtualized environment

Experimental setup (cont.)

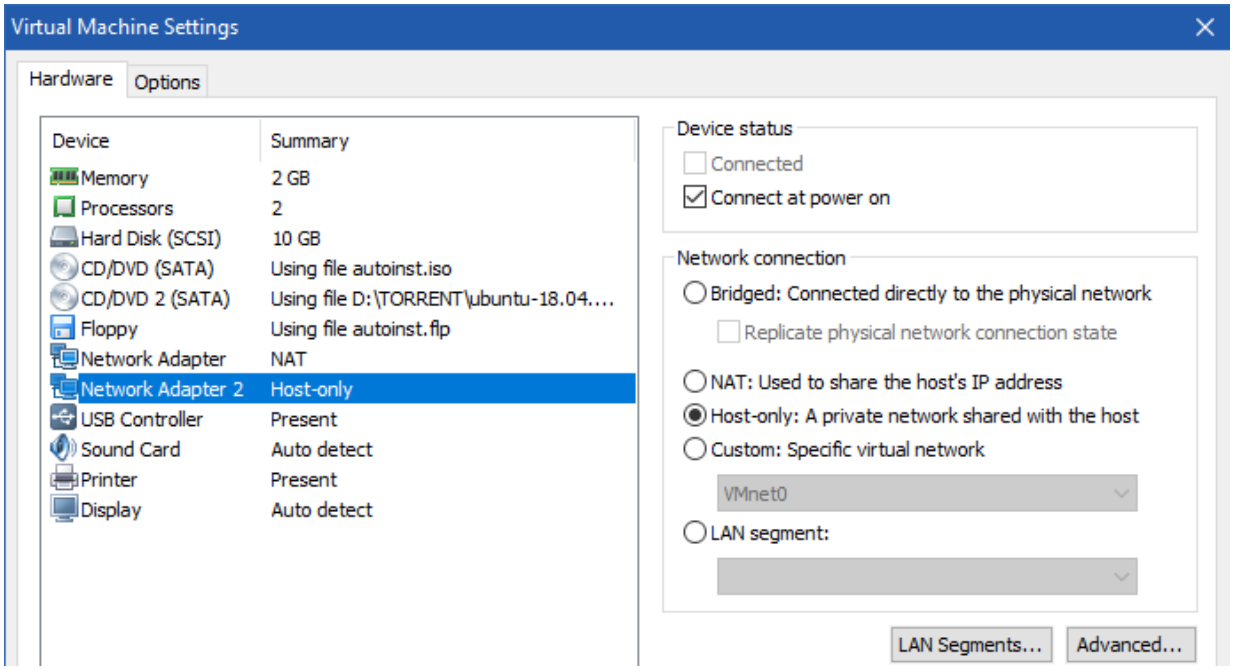


- In this study:
  - Two virtualized environment
  - Ubuntu 18.04 is used in this study

Experimental setup (cont.)

In the virtualized environments:

- add the *Host-only Network Adapter*: in this mode, the communication between connected guest systems and the host system is possible
- it can be useful to configure a static ip



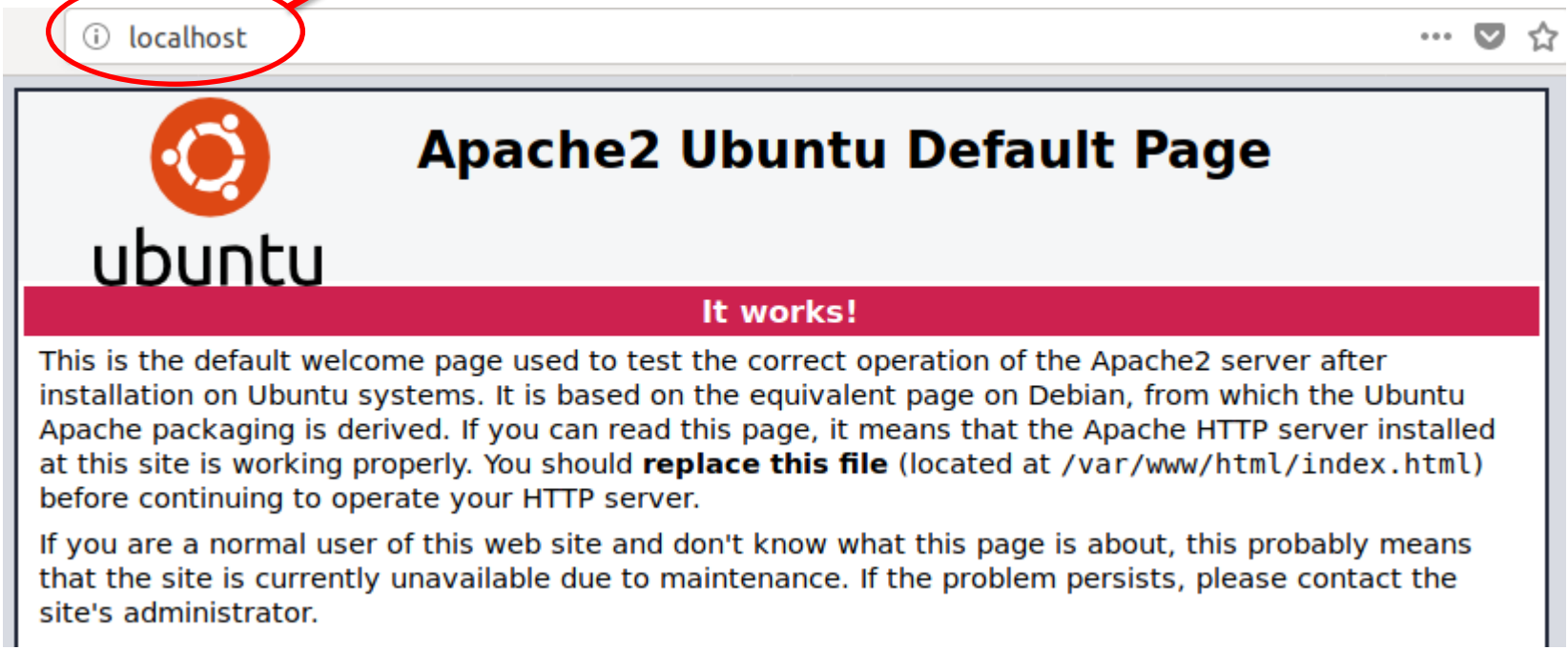
Apache Web Server

- Apache Web Server:
  - version 2 is used in this study
  - $ 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

localhost

http://server_ip:80



Apache JMeter

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

- Apache Jmeter 5.0
  - https://jmeter.apache.org/download_jmeter.cgi
  - binaries: apache-jmeter-5.0.tgz

Apache JMeter (cont.)

- To install it, simply unzip the zip/tar file into the directory where you want JMeter to be installed (e.g., home directory)
- We need to install Java Runtime Environment (JRE):
  - $ sudo apt-get install default-jre
- To run JMeter, it is sufficient to run *jmeter.sh in the bin directory*
  - $ ./apache-jmeter-5.0/bin/jmeter.sh

Main Elements of a Test Plan

- *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)

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

Main Elements of a Test Plan

- *Controllers*
  - 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)
  - 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

Main Elements of a Test Plan

- *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.

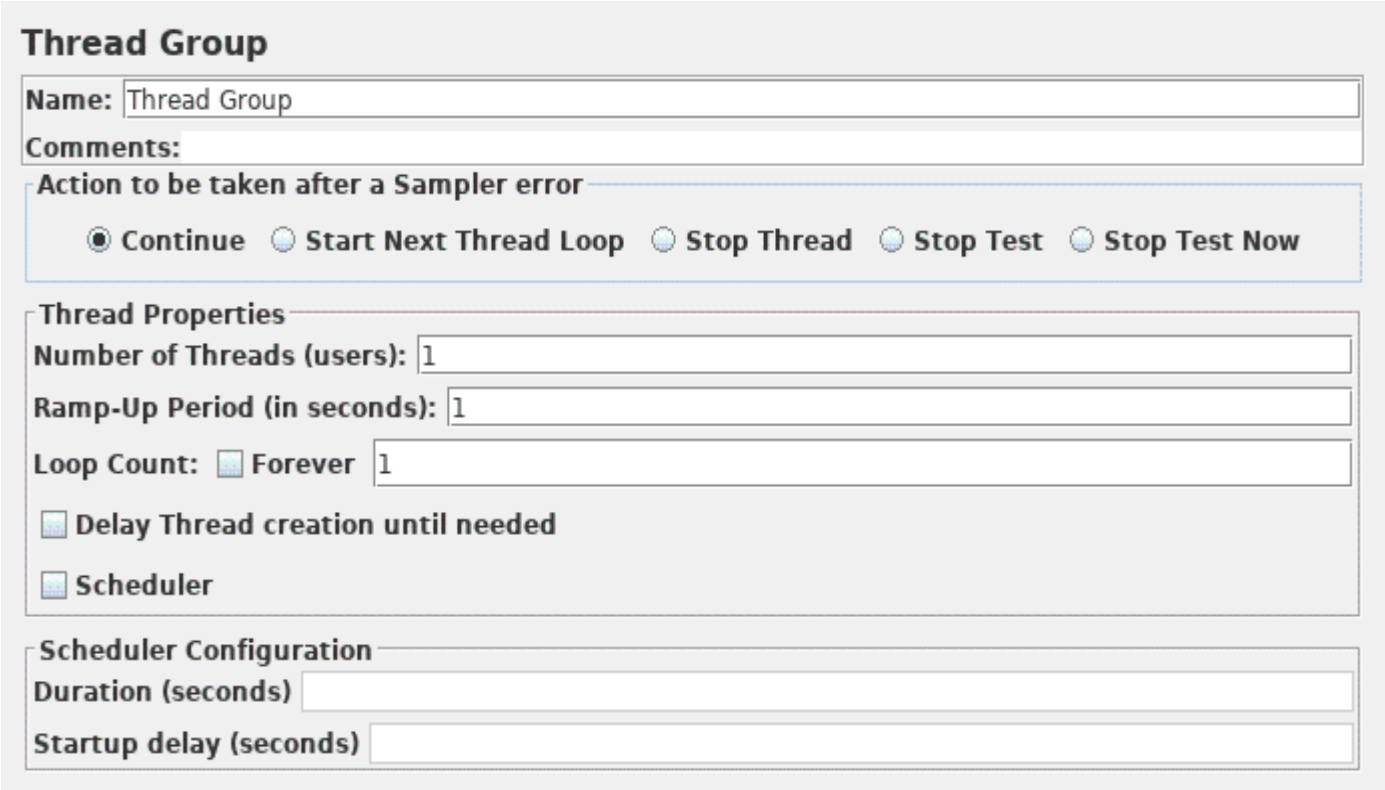
JMeter: Running a simple test

The minimal operations are:

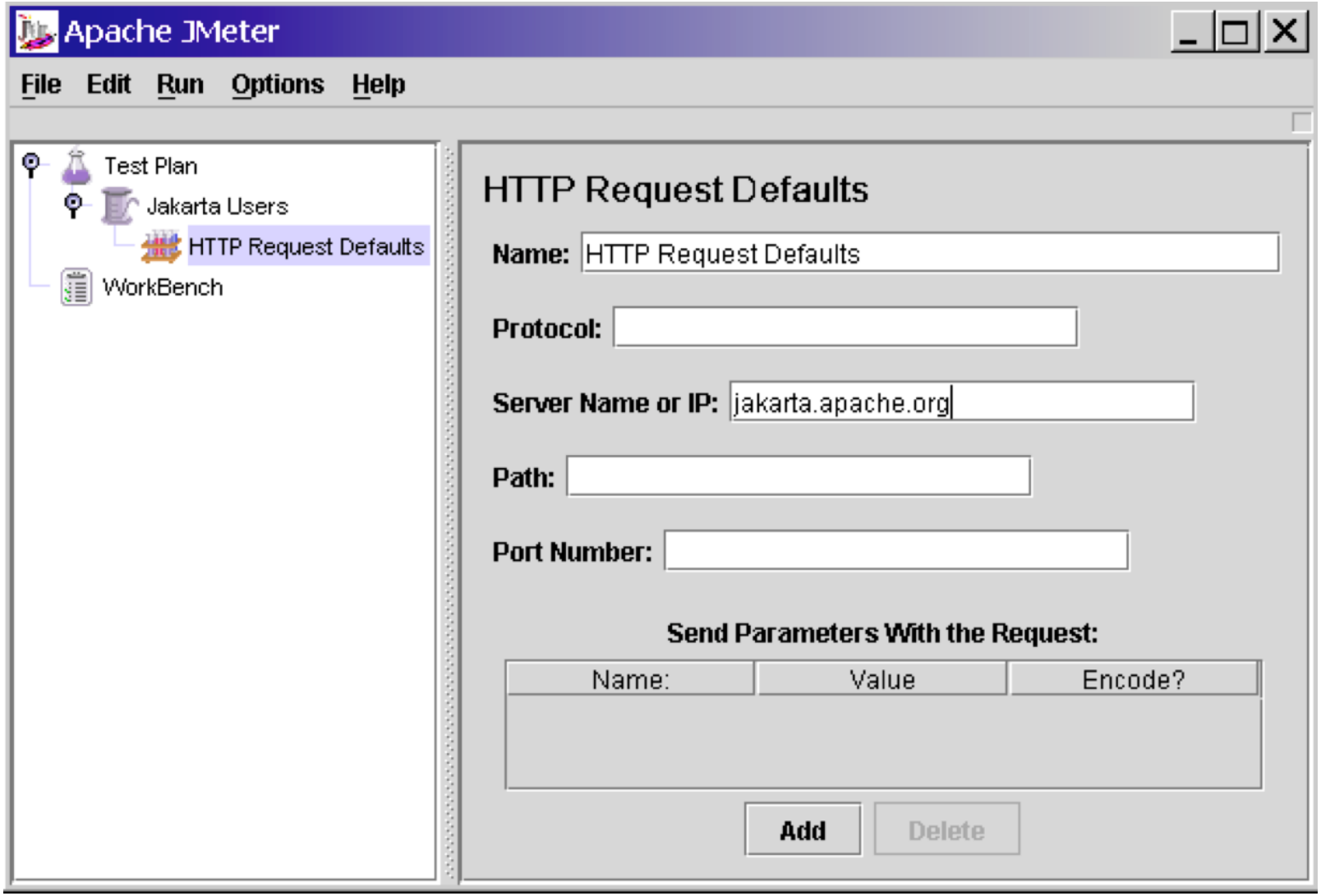
- Add a *Thread group*: this simulates the presence of *n* users submitting requests to the server.
  - To repeat the test for an indefinite time, select the "Loop Forever" checkbox.
  - If you want to simulate a given request rate (R), set the **constant throughput timer** ("right click" on thread group, then ->add->timer). Select "all active threads"

- Add an *HTTPRequestsDefault* ("right click" on thread group, then ->add->Config Element), specifying the server name (localhost, if you are running client and server on the same machine).

JMeter: Running a test



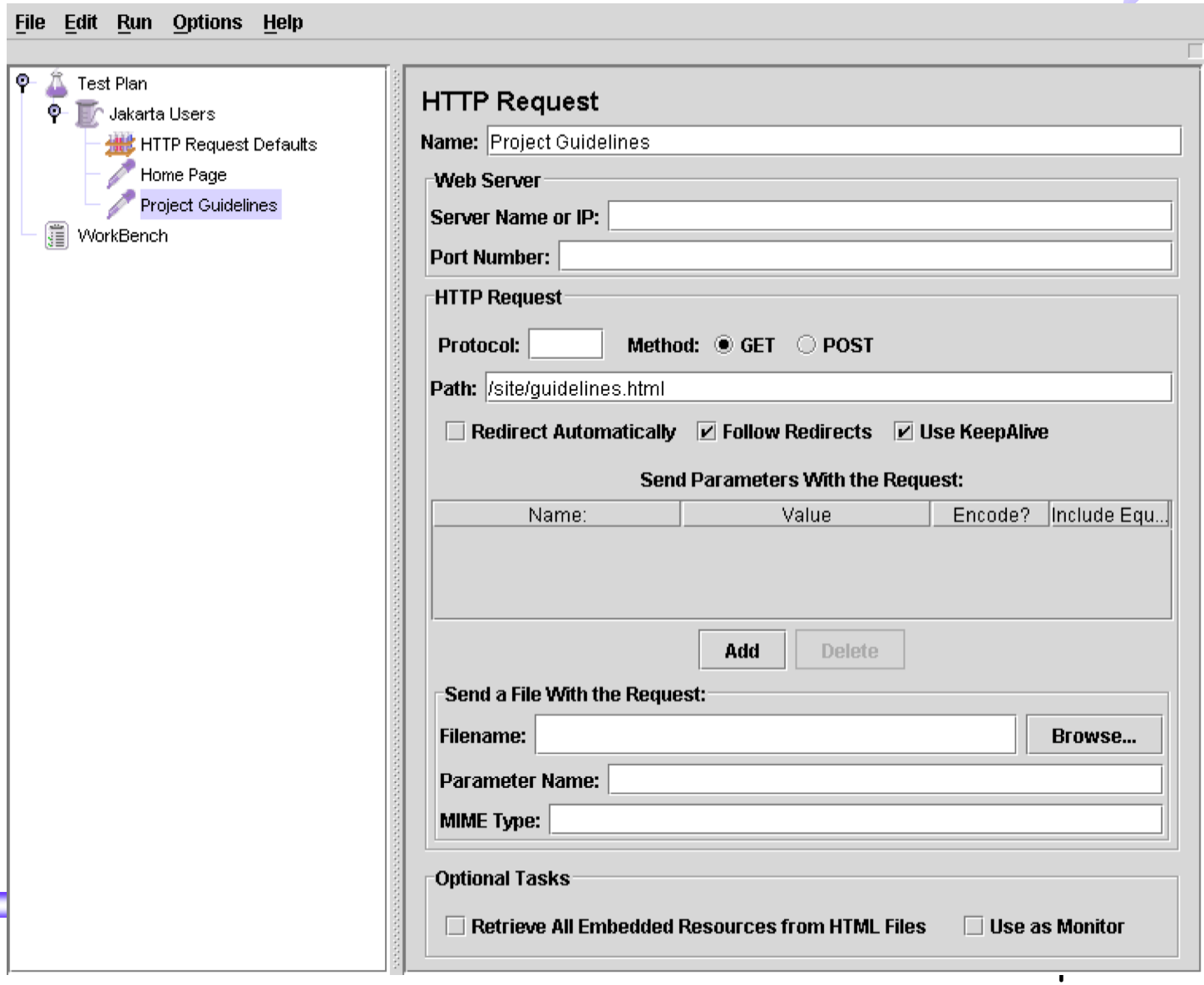
JMeter: Running a test



JMeter: Running a test

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).



JMeter: Running a test

- The workload generator is ready to run (menu "Run")
- Note that you can specify several types of requests (http, ftp, jdbc for DB test, SOAP for web services) by specifying corresponding samplers
- You can customize headers and cookies behavior
- Have several output results (e.g., response time, throughput, error statistics)

Example of Graph Results

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

- 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

Request-oriented workload parameters

```
$ 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

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
- Workload-related measurements:
  - CPU utilization
  - Page-in/out
  - Disk Activity

Linux Utilities to capture

Top
Free
Ps
Vmstat
…

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

```
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).
- IO
  - bi: Blocks received from a block device (blocks/s).
  - bo: Blocks sent to a block device (blocks/s).

vmstat example (cont.)

- 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.

Exercise

- Objective: Characterization of the observed workload

1. Generate a random workload, assuming it is a real field workload…
2. …using JMeter RANDOM contorller or Httperf
3. Collect values of application-level workload parameters
4. Collect values of system-level workload parameters
5. Use techniques explained in the course (compare techniques in both cases)

Example

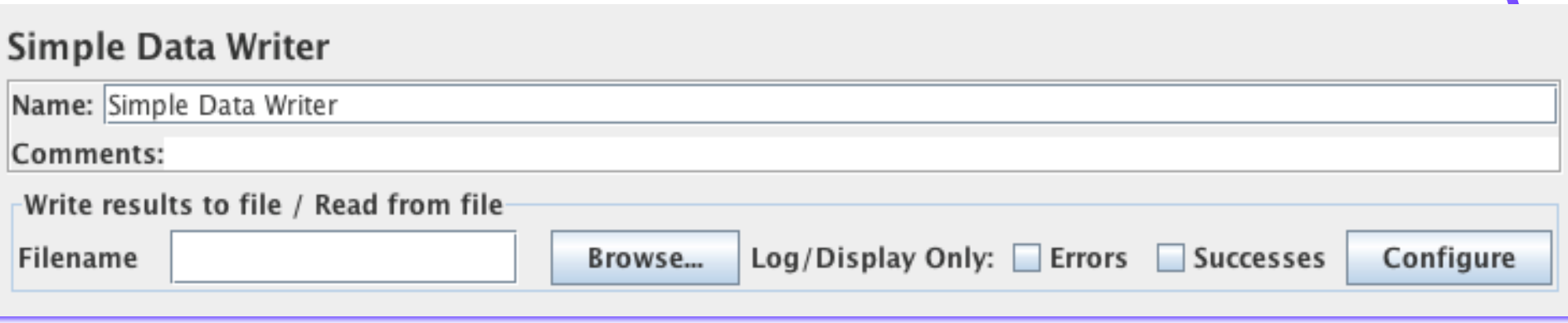
EXAMPLE WITH JMETER

1. Generate the workload

- Suppose to have an average number of concurrent users (e.g., 30)
- Set up 30 threads (set 0 sec. as ramp-up period) in the *threadGroup*
- Set "loop forever" box (stop manually the test after 5 minutes or set the "Duration" in the Schedular Configuration)
- Choose a request rate (e.g., 60 per second)

1. Generate the workload

- Add *httpSamplers* using pages of a websites on your machine (e.g., move the apache manual folder into the root, htdocs) – at least 5 pages
- 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)

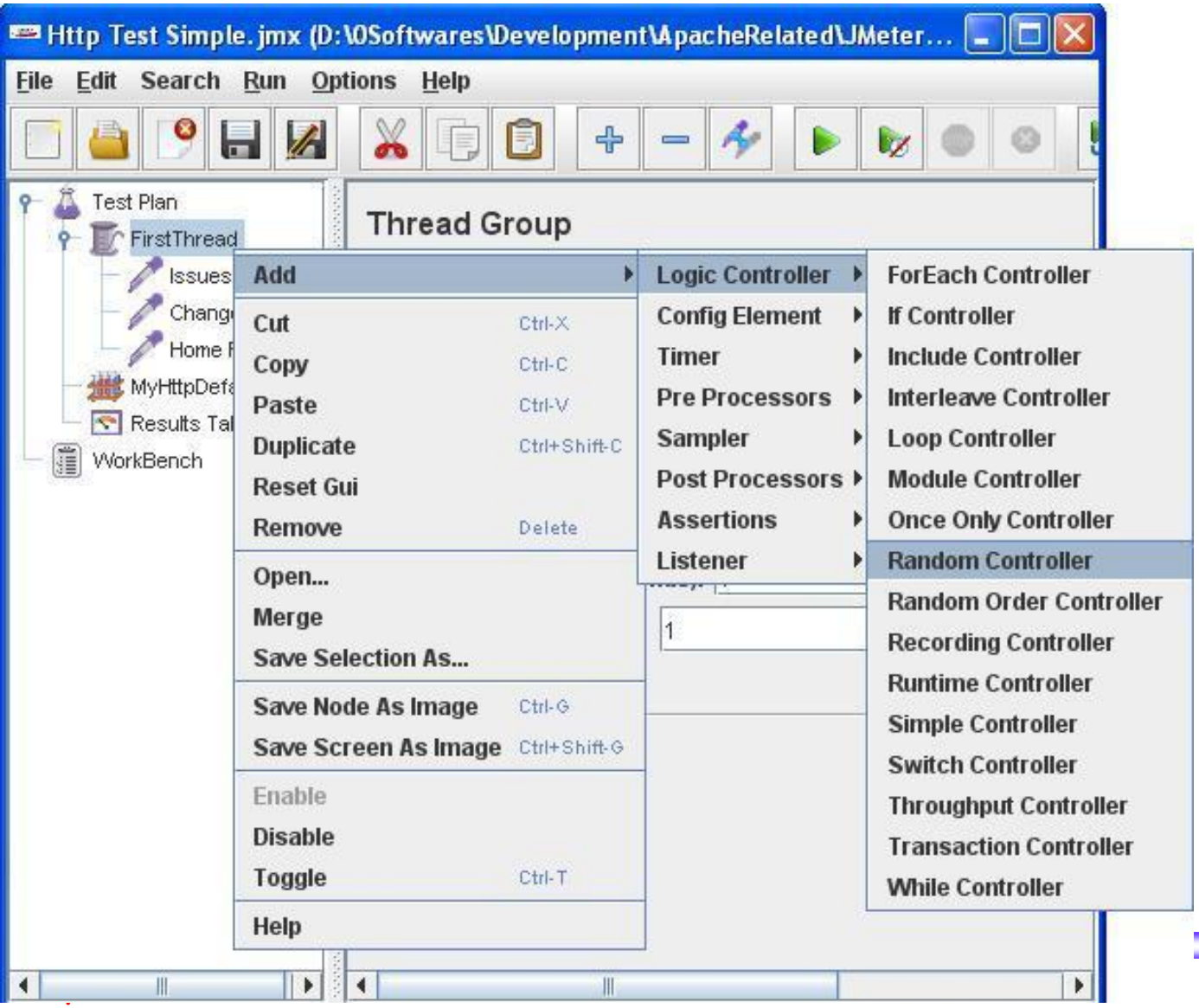


Save Output

- 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

2. Random controller



To generate request randomly among the available pages

3. Collect load parameters

- Simply run the configured test. Results will be saved in the specified file (in the *SampleDataWriter* field)

4. Collect sys-level load parameters

- Using (one or more) utilities (e.g., *top*, *free*, *vmstat*, *iostat*, *ps*) collect at least 15 low-level paramters related to: memory, cpu, and disk utilization.
- To do this, run the utility on the server side while the test has been running and redirect output on a textual file.

Use charachterization techniques (1/2)

- Characterize App-level (controllable) WL using these dimensions:
  - Byte count per request
  - Requested page
  - Thread Name

Use charachterization techniques (2/2)

- Characterize Sys-level (observable) WL using collected parameters
- Techniques:
  - Averaging and dispersion
  - Single/Multiple histograms on parameters
  - Study correlation, use PCA to reduce dimensionality
  - Clustering to reduce samples to consider
- Compare techniques by choosing the most suitable one for your data.
- *Note: not all the techniques make sense for both characterizations and for all the parameters (e.g., some are numerical, some others are categorical)*