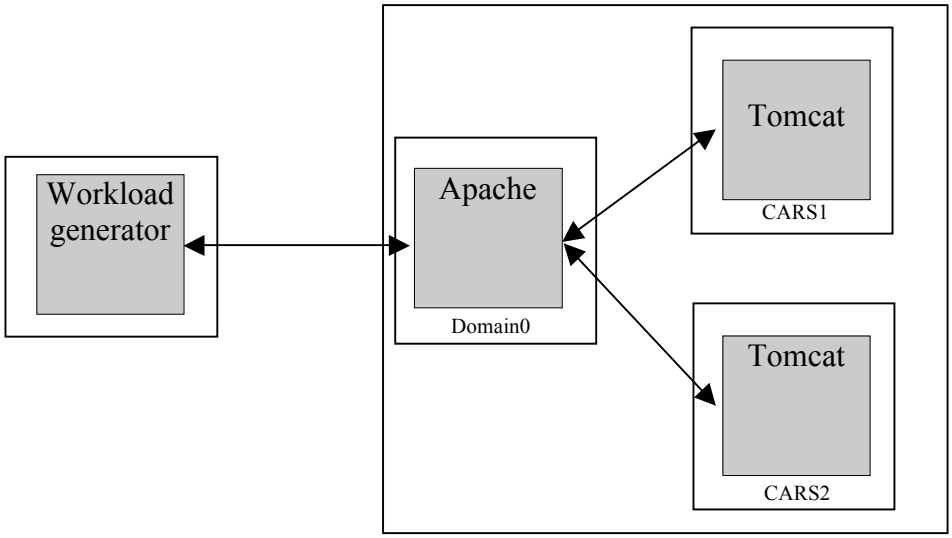
Configuració i Avaluació del Rendiment de Servidors (CARS)

Lab 7

Objective of the session:

Benchmark a dynamic web environment like:



where an Apache HTTP sever serves static contents, Tomcat runs dynamic web contents (JSP and Servlets). All the application data is stored in a backend DBMS (emulated in our lab). A workload generator emulates the load produced by a set of active web clients. Apache and Tomcat run in the same machine, while the workload generator resides in a different node.

Note: All the required files to complete this session can be found in
http://people.ac.upc.edu/dcarrera/cars/

The required files are:
cars_**client**_sessio1_0809_final.tgz and
cars_**server**_sessio1_0809_final.tgz

Note: Most of the activities involved in this session take $CARS_HOME as the root folder (corresponding to /home/alumne).

Part 0: Start the CARS01 virtual machine

As root, in the /root directory of the server machine run:

[$ROOT] xm list

To list the currently running VMs.

Now, change the allocated memory (300 Mb) for the Domain-0 in Xen.

[$ROOT] xm mem-set 0 300

It's time to start the first VM, that will boot with 2 virtual CPUs and 300MB of RAM.

[$ROOT] xm create CARS01.cfg vcpus=2 memory=300

After that, you can check that the VM is running with

[$ROOT] xm list

Notice that if you look for the network interfaces in the domain-0 (using ifconfig) you'll find that for the domain X, there's a vifX.0 interface with an assigned IP. The command "route –n" will show the routes to access the VMs. In the case of CARS01, the address is 10.0.0.1

You can log into the VM using an ssh session, with any of the following users/passwords:

alumne/sistemes
root/cars

In the domain-0:
Uncompress the server files downloaded into $CARS_HOME, using:

[$CARS_HOME] tar xvf cars_server_sessio1_0809_final.tgz

Copy the web application to the VM CARS01:

[$CARS_HOME] scp –r app1 alumne@10.0.0.1:.

First: Setup the environment in the server running:

[$CARS_HOME] source env.sh

Part 1: Setting up httperf (workload generator)

Requirement:
- Httperf (httperf-0.9-cars.tgz)

1. Preparing the workload generator

Download the workload generador into the client machina:

[$CARS_HOME] tar xvzf cars_client_sessio1.0708Q1_final.tgz

A “httperf-0.9” folder is created. To be sure the environment works properly, configure and compile the application:

[$CARS_HOME/httperf-0.9] ./configure; make

The test can be run using:

[$CARS_HOME/httperf-0.9] ./starthttperf.sh <load level>, for application 1

[$CARS_HOME/httperf-0.9] ./starthttperf2.sh <load level>, for application 2

After executing this command, the workload generator will be running for 120s and will report, the data collected for the last 5s period.

Note: The load level represents the number of **new** sessions started per second.

Example: If the life-cycle time for each client is 100 sec. and 10 new seconds are generated per second, a maximum number of 1000 simultaneous clients will be observed, and 30 seconds after starting the test, 300 concurrent clients will be connected to the server.

Part 2: Testing

1. Set-up the server (Tomcat inside CARS01, apache inside HOST)

Before running a test, make sure the servers (Apache and Tomcat) are configured as expected .

The parameters to configure for **Apache** are:
:

- **Thread pol size in** $APACHE_HOME/conf/httpd.conf:
- Section "<IfModule **worker.c**>"

StartServers:	10
MaxClients:	<num threads>
MinSpareThreads:	<num threads>
MaxSpareThreads:	<num threads>
ThreadsPerChild:	<num threads / 10>
MaxRequestsPerChild:	0

- **Connection timeout in** $APACHE_HOME/conf/httpd.conf :
  :
  KeepAliveTimeout: <seconds>

- **Number of connections between Tomcat and Apache in** $APACHE_HOME/conf/workers.properties:

  worker.worker1.cachesize = <número de threads>

The parameters to configure for **Tomcat** are:

- **Thread pool size in** $TOMCAT_HOME/conf/server.xml (AJP 1.3 connector)

  maxThreads = <#threads>
  minSpareThreads = <#threads>
  maxSpareThreads = <#threads>

To start the servers, you must run:

$APACHE_HOME/bin/apachectl start | stop, to start and stop Apache

$TOMCAT_HOME/bin/catalina.sh run, to start Tomcat (it is stop pressing Ctl+C)

2. Run the experiments:

EXPERIMENT 1: Determining the maximum load for the system

Imagine a system that is running behind a system that acts as a load balancer that offers overload protection: that is, it will redirect to each server the maximum load that the system can process without being force to queue client connections.

In summary, imagine a stable system: the load balancer is guaranteeing that the input rate will not exceed the maximum achievable output rate.

In this experiment we want to determine such a maximum load. To achieve this objective we have some data (obtained from a workload model):

- Each client session is composed of 10 requests and some off-times between requests
- The baseline time to process each request in the session and the he baseline sessions lifetime (to be calculated in question a)
- 11% of the time required to completed a single request is spent consuming CPU resources

and a methodology:

- Run the workload generator at different load levels
- For each load level, we must find the server configuration that maximizes the load without queuing client connections in excess
    - The configuration will consist on a number of threads in
- For each load level, we'll monitor the CPU utilization:
    - If the CPU is underutilized, and the response time exceeds the baseline time in excess, we're queuing client connections and need more "processing units"
    - If the CPU is underutilized and there's no queuing, we've reached the maximum output for the current input load.

The average response time observed by the workload generator is shown in the line:

Reply time [ms]: **response XXXms**

The average session time observed by the workload generator is shown in the line:

Session lifetime [s]: **XX.X**

You're requested to:

a) Complete the following steps:

- Set an initial configuration for the server
  - [Apache pool size=50, Apache-Tomcat connections=50, Tomcat pool size=100, Keep-alive timeout=15]
- Run the workload generator with load 1
- Take the observed avg. response time and avg. session lifetime as the baseline values for questions b) and c).

b) Fill a table with the following data, starting from a load level of 1:

- Load level
- Sever configuration
  - [Apache pool size, Apache-Tomcat connections, Tomcat pool size]
  - Keep-alive timeout set to 15 in all tests
- Response time
- Session lifetime
- Avg. CPU utilization (observed during the test, running top)

remarking the row containing the highest load supported for each server configuration.

Finally, remark the maximum observed load supported by the best found configuration. That value is the maximum load that can be attended by our system.

b) Considering the session data provided above, the maximum load determined in "b)", and the baseline response time and session lifetime calculated in "a)", calculate the following data:

- Estimated #concurrent clients at maximum load
- Estimated #requests per second issued by a client
- Estimated #requests per second issued by all the concurrent clients
- Estimated CPU utilization performed by the concurrent clients

Check that the estimated CPU utilization is close to the actual CPU utilization observed during the test (for the configuration with maximum load).

EXPERIMENT 2: Determining the maximum load for the system for a second application

Repeat the steps followed in the experiment 1 for the second application (running in CARS02 VM, in 10.0.0.2):

It's time to start the second VM, that will boot with 2 virtual CPUs and 300MB of RAM.

[$ROOT] xm create CARS02.cfg vcpus=2 memory=300

After that, you can check that the VM is running with

[$ROOT] xm list

In the domain-0:
Uncompress the server files downloaded into $CARS_HOME, using:

[$CARS_HOME] tar xvf cars_server_sessio1_0809_final.tgz

Copy the web application to the VM CARS01:

[$CARS_HOME] scp –r app2 alumne@10.0.0.2:.

First: Setup the environment in the server running:

[$CARS_HOME] source env.sh

Use the following workload generator for this test:

[$CARS_HOME/httperf-0.9] ./starthttperf2.sh <load level>

In this experiment we want to determine the maximum load for this application, considering the following data (obtained from a workload model):

- Each client session is composed of 19 requests and some off-times between requests
- The baseline time to process each request in the session and the he baseline sessions lifetime (to be calculated)
- **2,7%** of the time required to completed a single request is spent consuming CPU resources

EXPERIMENT 3: ALLOCATING RESOURCES TO EACH VM

Now, we need to calculate how much CPU power you need to allocate to each configuration to achieve a particular level of performance.

Use the command

Xm sched-credit –d <domain> -w <weight> –c <cap>

To allocate resources (CPU shares) to each VM and thus, to each application (use 'xm list' to determine the domain ID for each VM).

From "http://wiki.xensource.com/xenwiki/CreditScheduler":

"Weight:
A domain with a weight of 512 will get twice as much CPU as a domain with a weight of 256 on a contended host. Legal weights range from 1 to 65535 and the default is 256.

Cap:
The cap optionally fixes the maximum amount of CPU a domain will be able to consume, even if the host system has idle CPU cycles. The cap is expressed in percentage of one physical CPU: 100 is 1 physical CPU, 50 is half a CPU, 400 is 4 CPUs, etc... The default, 0, means there is no upper cap."

Try to use, for each application the following CAP values: 50, 100, 150 and 200 (that is, 0.5, 1.0, 1.5 and 2.0 times the CPU capacity of a physical processor). Calculate the maximum load level supported for each configuration.

EXPERIMENT 4: MANAGING TWO TOMCAT INSTANCES

In this third experiment you will have two instances of Tomcat running in your server machine, as well as two workload generators running in the client machine. Each server will run a different application, and each workload generator will emulate the clients for one of the applications.

Run in two different shells:
[$CARS_HOME/httperf-0.9] ./starthttperf.sh <load level app1>

and

[$CARS_HOME/httperf-0.9] ./starthttperf2t.sh <load level app2>

Using the data and the calculations obtained in the experiment 1, decide what is the best

configuration to maximize the performance of the system in terms of:

- Session completion rate.
- Request rate

Note: Both of the applications must be run at the same time, and some load must be put on each of them.

Allocate the necessary resources to each application, and remember that the sum of the CAP level for all the applications can't be higher than 200 for a physical system with 2 cores!