Dalhousie University, Faculty of Computer Science

CSCI 4145/5408 - Cloud Computing – Summer 2018

Assigned: July 16, 2018

Due: July 24, 2018, 11:59 pm

ASSIGNMENT 5

Load Balancer on a SAAS

This assignment asks you to create a relatively simple web service, deploy it on the cloud, and invoke it from a client application. Your web service will be invoked in different cycles of fixed periods of time. In each cycle the request rate will be different. You will need to determine the average per cycle response time when the web services are invoked while the request rate is varied. You will do this with and without load balancing and report the results.

You can work in groups of two people in this assignment. You do not need to inform us about your groups beforehand, but you have to identify the group members in the submitted document.

A. Create an account on Microsoft Azure, Amazon Elastic Beanstalk or OpenShift.

B. Write and deploy a restful web service, the deployed on the cloud, that has the following input parameters:
   a. Unique Request ID – You can use sequence numbers such as 1, 2, 3, … - or any other unique numbering mechanism of your choice; you may also want to tie the request ID to cycle number or any other parameter if you find it useful.
   b. Payload Size – Integer value to specify the size of data (in Kbytes) that should be returned from web service (Payload size should be in the range of 10 – 100,000 Kbytes.)

C. The web service should return the following for each request in a JSON response message:
   a. Request ID: for the client to match the request and responses whenever needed.
   b. Timestamp taken when request was received on server (there is no obvious way of using this information in the analysis, but it'd be useful for debugging purposes)
   c. Actual Payload: Data generated equal to the payload size received from input attribute. Payload can consist of any combination of characters/bytes (for example: image data in byte array or ASCII characters).
   d. Total web service execution delay: Difference between the start time and the end time of the web service execution in milliseconds, where the end of the web service execution is deemed to be just before the response message is sent.
   e. HTTP Response code: 200 OK, 400 Bad Request, 500 Internal Server Error
      i. HTTP 200 OK: Returned when web service execution ends in success ii. HTTP 400 Bad Request: If any of the input parameter is either out of range or invalid

iii. HTTP 500 Internal Server Error: A general error message returned if any other error occurs while processing the request

D. Test your web service by manually calling it from a browser and show that it works. Please attach snapshots of web service execution simply from a browser – with Google Chrome or Mozilla Firefox with network monitoring enabled.

E. Create a simple client application that takes four input parameters:

a. *Duration*: in seconds (duration of a cycle)
b. *Payload size*: an integer
c. *Number of cycles*: number of times request-response cycle that should be repeated
d. *Cycle request rates*: in requests/second; this value will be different for each of the cycles

In the output, you should produce the following results:

a. Details showing input values to the application.
b. Summary of results
   - Average web service delay (cycle by cycle)
   - For each status code, proportion of responses (# of responses with code / total responses) (cycle by cycle)

Sample Input:

Duration: 5 seconds
# of Cycles: 5
Request rates: Cycle 1: 200 requests/second, Cycle 2: 400 requests/second, Cycle 3: …
Payload Size: 1000

Client Application Program Processing Logic:

For each cycle from 1 TO #noOfCycles
---→Generate parallel threads at a rate and for duration as specified by input
---→For each concurrent thread execution
----------→Generate Unique Request ID
----------→Load payload size as provided in input
----------→Send Web Service request (with Unique Request ID, Payload Size)
----------→Wait until a response is received
----------→Extract the web service delay value from the response packet and save
----------them (also record the HTTP response code)
---→Retrieve the individual delay values and data, calculate the average delay over
---------- all requests/responses of the cycle and produce the required results/output

***Notes*:**

1. Please note that you have to be able create *request rate* amount of requests per second for each of the cycle threads. Thus the total number of requests generated

in one cycle will be *Duration* multiplied by *Request rate* of that cycle. In this way, response delays should be affected by the rate at which requests are generated.

2. Having a measurable delay is very important in this assignment. If you encounter difficulties in creating a measurable delay due to lack of CPU bound tasks, you can add an extra authentication feature to your logic such that the web service takes an extra parameter for Integrity or Authentication Token. This token would be either a 12 digit encrypted "password" key passed inside the header of a request message or a hash code (encrypted) calculated over the message, or both. In this model, the client should send the authentication token to the server in the request header and the web service should check for authentication. Of course, in case of an authentication failure a new response code comes into play (HTTP 401 – Unauthorized). Any password/key used in the authentication mechanisms can be hardcoded in both client and server.
3. Depending on your implementation, the choice of development and deployment environments, different HTTP response codes might have been received. In such case, please record and report all distinct HTTP response codes.
4. During your unit tests, if you encounter some missing responses, or some request drops by the server, you will need to incorporate a timeout mechanism in your design. In this way, you do not wait forever for a response that would never be received. Of course, such a fact also affects the outputs that you are going to report; you should not include such lost requests/responses in delay calculation but you have to report its percentage in your output.

F. Repeat the above experiment while scaling up/load-balancing on your cloud instance or scaling out with more cloud instances. If you want to avoid using Cloud-based load balancing, you can extend your client on your own with following steps:

   a. Create two or three Cloud server instances
   b. Implement Round-Robin technique in a threading queue (you can make use of any external code if required)
   c. Place all parallel threads from your client inside a thread queue that uses RoundRobin technique (implemented in previous step).
   d. Your thread queue will pick one thread at a time using Round-Robin and sends it to different cloud instances as configured in your client.

   Another simple option is for you to modify your code that invokes the web service so that that which instance is used will be based on the generated request ID.

Submission requirements

- A PDF document that includes:
  - Title page with the usual information (identifies course/assignment and your group)
  - Table of Contents
  - URL for the hosted web services for marking (please include the instructions, if necessary, to run the application in your report)

- **Since Cloud Service can be costly, you can pause your service to save some cost after finishing this assignment, TA will inform you to run the web services before marking.**
- Resources used (Cloud instances, Hardware/software platform, including software libraries, used to develop your client, e.g., MacBook Pro with OS X10.9.5 Eclipse SDK v. 3.7.2, … and to develop your web services).
- Identification of any code from other sources that you used (e.g., code snippets form tutorials): Identify source and also where in your code you used the code.
- Description about any framework used to implement web services (like Spring, Jersey, etc.).
- If you have changed the descriptions given in this documents in your implementation or if you have made some significant assumptions or if you have implemented some optional parts (for example, authentication, timeout), please explain them for both client and web service parts separately. If your implementations are not different than the specifications, please also state this.
- Source code and UI snapshots of manual testing of web service application deployed on the cloud server (step D above). Please make sure that browser network monitoring is enabled and persisted. This needs to be included in screen snapshots.
- Source code of load balancing program (if performed by yourself) or description on configuring load balancer (if performed through managed cloud service).
- Performance evaluation tests using several parameters: For at least 3 different sets of parameters, provide both tabular and graphical performance outputs. For each parameter set:
  - For the tabular format, please provide the required outputs (see Part E above) in a reader-friendly way (especially the reader should be able to easily capture the change in the delay and other results with increased load for both loadbalancing and no-load-balancing cases). Each table must be numbered, must have a caption and must be cited in the text with explanations and commentary in sufficient detail so that the reader can understand what has been done.
  - For the graphical output, please provide a 2D line graph (plot) for web service delay. In this graph, the X-axis represents the request rate (i.e. the cycles) and Y-axis represents the delay value in msec. In the graph, you will plot two lines; one with load balancing and one without load balancing. Please do not forget to put axis names and legend. Each figure should be numbered and should bear an explanatory caption. Each figure must be cited in the text and you have to comment on the output results, especially on how the load changes the delay, how load balancing affects the delay.

  As a more general conclusion, please also comment on the effect of the payload size and duration on performance as well.

- Source code (in appropriate folder structure) and documentation sufficient to install and deploy the application to the cloud server and test it. In some cases, you may be invited to a demo.
- All the files should be uploaded in "CSCI4145-A5-{Names}-{B00Numbers}.zip" format.
- Only one submission per group please (it does not matter who submits it provided that both group members are identified in the title page).

Grading Criteria

70%: Correct functionality

30%: Documentation (including testing part and performance evaluation part)