CMSC 621
Fall 2018
Project 2

The primary goal of the project is to build a distributed web-services framework. The goal is to make you familar with the tools of the trade being used to implement many distributed systems today. They are represented by an increasingly large alphabet soup -- XML, SOAP, REST, JAX, Ajax, Axis2, ...

Requirements -- The Barebones Infrastructure -- 100 points

This describes the underlying distributed infrastructure you will create to access and invoke web services. There will be multiple sites hosting a collection of webservices. Each site will provide a mechanism for the user to ask for a certain service. To invoke a service, a site needs to first "discover" all sites that host the webservice that it is looking for, and then invoke the webservice so as to balance the load on each site. Service descriptions should be provided in WSDL 2.0. You may assume that there is no disambiguation involved -- all services called "foo" offer the same service and interface. There should be minimal centralized components (single points of failure) in your system. You may want to develop a GUI for your system, however you will be graded based on the overall functionality and not on whether you have a GUI. Similarly, we do not care what the services actually are -- they could be something simple like adding and subtracting or echoing or chargen. Each service should run on multiple sites, so your discovery/load balancing can actually show results. You should have a few different services that load machines differently. These loads need not be real -- you can associate notional loads with each service and as a machine runs a service, the corresponding load should be assumed.

Extra Credits -- Upto 25 Points

All sites are required to participate in a blockchain that logs transactions. Every service invocation should be treated as a transaction and logged in the blockchain. Look at HyperLedger fabric. There are many ways to build this. You can simply had the node executing the service affirm the transaction. You can use majority consensus. Requests for service should go to all nodes, including those that won't run the service. Each then validates the transaction to the blockchain.

Discussion

The description of the system above, like in real life, is underspecified. There are a variety of ways to implement the ideas described above. For instance, for service discovery you may want to build a distributed mechanism based on algorithms like "Chord" or use a completely replicated registry at each site or create a centralized registry which is cached at sites and so on. Similar ideas can be used for finding the least loaded site. For example, you could maintain a centralized list of loads, and always pick the "least" loaded site that offers a service. You could maintain an "average" load of the system and pick any site that is below a certain threshold from the average (Kruger/Finkel Algorithm)? Perhaps even the load and service information can be merged ? Should request forwarding be allowed -- if site A requests B for some service, B may in turn request C for the same service if it is overloaded and return results to A transparently

You may need to make additional assumptions to come up with a concrete design and then a reference implementation. Hence, state your assumptions clearly in your design, and make sure you justify your design

choices. This will help us grade your contribution accurately. Your assumptions will be evaluated based on how it helps your design and how "real" they are. We will also look at your design rationale

Experiments

A part of your project is to design and carry out an experimental validation to convince us that your system works. These should be test scenarios that show the functionality of your system (unit/functional tests). This is true for both the base task as well as the extra credit. You should also analyze your system for scalability -- number of hosts, number of tasks, etc, reliability -- how does the system behave on random site failures, performance -- can you tune some parameters to make the system perform better. You should experiment with different workload mixtures. Document your results, graph scalability, show tables with averages/variances etc. Comment on the results you get -- try to identify what assumptions you made or implementation mechanisms of your system cause particular scalability patterns. Try and identify both the strengths and weaknesses of your system. If you use your initial experimental results to refine your design, make sure you bring this out in your report.

Mechanics

You will do this project in groups. The group shall have **two** members. Permission to do the project individually will be granted rarely. In any case, such individuals will still be judged against the same criteria as regular groups.

Deadlines

- On or before **November 8th**, you will submit a document which provides preliminary system specifications and design, assumptions you have made, design justifications, and a plan of execution, proposed timeline and testing strategy. This should be submitted on blackboard under "First Project Report"
- On **November 21**, you will submit a brief report of your progress to date, changes in the design if any, as well as an updated timeline. Again, this will be via blackboard , under "Second Project Report".
- The project report detailing your work (the system design, experimental studies etc.) as well as submission of your code, will be due on **December 12** via blackboard - under "Final Project Report". Code and report should be attached as a tarred, gzipped file. Please note that pdf and text (formatted to 72 columns) are the only formats we will accept for report. The report should discuss your design, detail your implementation, and also say which partner did what work. It should also have the results/discussion of the validation and scalability experiments.
- You will also arrange to demonstrate your project to the instructor and TA between **December 13** and **December 19**. Along with the December 12 deadline, note that the report and code submission must precede the demonstration by **48 hours**. (This means that if your demo is on December 13, your deadline for code and report submission is December 11).

Some general suggestions

Start now! Even if you have more than a month to do this project, it is complex! As should be evident to most of you, it is imperative for a project of this complexity and involving teams that you design your system before you code! In your design, you will need to make assumptions as you flesh in the details of the system. Please make sure that you state them in your design document. Make a timeline for your work, and try and

stick to it. Where you divide tasks, make sure you clearly define points of articulation and interfaces between modules. As you form groups, please make sure that you can find a common time to meet. This might sound mundane, but such details clarified ahead of time help avoid problems at the end. This is especially true for those who are part time students and hold jobs which will restrict your schedule. Please comment your code well -- it will help both you and us. You in figuring out code your partners have written, us in grading it. Also, at the minimum use some form of revision control on your source tree. Machines have systems such as CVS, RCS, and GIT available for your use. This will help if lightening strikes, UPC fails and machines/disks crash, making your recent changes disappear! Please do create makefiles/ant scripts -- you may in fact prefer to use freeware IDEs such as Eclipse.

You are allowed to discuss the project across groups. Feel free to discuss approaches, designs, pros and cons of various approaches, etc. Clearly, you are not allowed to share solutions and code. Don't directly reuse code you find on the web without attribution, and be judicious in how much you reuse. If your entire project is reused code, then it is hard to judge what your contribution is, except for Googling! You may read papers and textbooks in this area as well -- some pointers are provided in this document. However, you should cite the sources you have consulted. Also, be sure to mention any students (in or out-of class)/professors/individuals you consulted regarding the project. We strongly suggest that unless you are a very proficient programmger with deep XML knowledge, you look at tools to create and manage web services. We would like to see your final demo on multiple machines and not as multiple instances of your application on a single machine.The Grad lab provides an excellent environment for this

References

1. Web services: Introduction to webservices, WSDL, SOAP, REST, REST and WSDL

2. Service Discovery : Jini , Service Location Protocol , UDDI , Chord

3. Policy Frameworks: http://www-128.ibm.com/developerworks/library/specification/ws-polfram/, http://www.w3.org/Submission/WS-Policy/

4. Frameworks for deploying webservices : Apache Axis2 (Java), Tungsten, Apache Axis (C++), Spring Web Services

5. Miscellaneous: Eclipse, GitHub, JAX WS and RS, Apache Ant, CVS

Anupam Joshi