ECE 465 Semester Project: Cooperative Web Caching

Prof. Paris
Spring 2007

March 6, 2007

1 Overview

We have discussed caching of web content in class and identified two commonly used caches: the cache built into the browser and a dedicated caching server for a local network.

The purpose of this project is to develop a simplified co-ooperative caching system. You can think of this as a combination of the two caching strategies above. The basic idea for the system is that clients on the local network collaborate and share the contents of their own caches with each other. This provides the same benefits as a dedicated local caching server without the need to deploy such a server.

Specifically, when a client requests a web document it first consults its own cache to determine if the document is available. If it is not found, it proceeds to ask the co-operating hosts if any of them have the document cached. If the document is available it will be served locally. Only if none of the co-operating hosts has the document, will a request be sent to the origin server. Clearly, this description is somewhat simplified and focus on the essentials of the problem.

A detailed description and corresponding requirments are given in the following sections.

2 Principles of Operation

You will implement a simplified version of the problem introduced above. Specifically, your caching system will only need to work for documents with URL of the form:

`http://plano.spec.gmu.edu:8000/xxx.html`,

where `xxx` represent three-digit numbers from `000` to `999`. Thus, there is a total of 1,000 documents in the web-space of interest. I will provide the web server and all the documents. The server will be programmed to be slow to respond to simulate a congested access link and/or internet. This will highlight the benefits of caching.

Your approach to the problem will need to proceed in three steps.

Co-operative cache formation: Any hosts interested in participating in the co-operative cache identify themselves to one of the hosts already in the caching group and request membership. As a result the list of co-operating hosts is updated for all participants including the new member. Provisions must also be made to allow members to leave and to remove non-responsive hosts.

Request and response: If a client makes request for one of the documents in the document space of interest, it will first look for it in its own cache. If it is not found, it will ask all members of the group for a copy. Only if none of the members have a cached copy, will a request be sent to the the origin server.

You should maintain statistics on each transaction, including at least the time it took to satisfy the request, wether the response came from a cache or from the origin server, and the cache hit rate. Each host should also track how many requests it was able to satisfy for other members of the group.

Caching: Clearly, the received document should be placed in the cache for future access. To simulate limited resources, each host is only allowed to store 100 documents in its cache. Thus, you need to define strategies for maintaining the cache of your host.

Notice how the first two steps involve coordination and communication between hosts. Consequently, a suitable set of protocols is required to allow independently designed applications to work together. The caching strategies in contrast are independent of each other. The work products for this project fall into two categories: each group (see below) will have to design and implement a network application (i.e., a program) to participate in the co-operative caching system; all groups together are to design a set of protocols describing the format, contents, and sequence of messages exchanged between participating hosts. Ultimately, the applications developed by different groups are to work together.

3 Requirements

The three stages described above build on each other; formation must occur before other the two steps can be tackled. This dependency dictates the sequence in which the application should be standardized and implemented.

The following provides a closer look at the specific events that must occur in each stage. It also contains some thoughts to get you started designing the protocol and application.

3.1 Step I: Formation

A master server is used to manage the system. The IP address of the server is known to all participants and should be a configuration parameter for the programs. To indicate the intention to participate in the co-operative cache, hosts send a message to the server. The master server will acknowledge the message and update its list of participants; the master server stores the members contact information (IP address, port number) and distributes this information to all members.

Similarly, the master server responds to members' requests to leave the system and removes members who have been unresponsive.

The protocol to be designed for this stage should specify the format of messages used for invitation and responses. It should specify when either of these messages should be sent, including by whom and how often.

The products for the formation phase must be a protocol (agreed upon among all groups) and software implementations demonstrating formation functionality. Specifically, the mechanism for formation must be standardized and implemented. It is *strongly* recommended that you undertake thorough testing within and between groups.

Each group must implement the master server functionality, i.e., each group must be able to serve as the master server.

3.2 Step II: Request and Response

In the operational phase, the co-operating hosts each work like a proxy-server. They look like a server for the original client and other members sending them requests. The act like a client when they send requests to other members, and possibly the origin server, in repsonse to the original client's request.

The protocol to be designed for this stage must support the co-operation between members. It must specify the format and contents of messages to be sent between members to query them about the contents of their caches.

The products of this phase must be a protocol (agreed upon among all groups) and software implementations demonstrating interaction between members of the co-operating caching system.

The set of statistics to be maintianed by each host should be standardized as well.

3.3 Step III: Caching

Caching strategies do not need to be co-ordinated between members. Obvious strategies might include:

- add items to the cache until it is full and then stop updating, or
- maintain the most recently requested documents in cache.

3.4 Demonstration and Testing

The operation of the system must be tested and demonstrated. Each implementation should provide for two modes of operation:

- **Automatic:** the program runs autonomously and generates requests to random documents in the document space.
- **Manual:** the program accepts user input for a specific document to be requested.

3.5 Bells and Whistles

While not strictly required, the following provide ideas for "nice-to-have" additions.

- Show the contents of your cache: add the ability to look at the contents of your cache, how long each document has been there, and how it got there.
- Make it work with your browser: configure your browser to use your program as a proxy. If a document in the document space of interest is requested it co-operates with the other caches to fill the request. Otherwise it forwards the request and relays the response.

4 Organization

4.1 Groups

You will work on this project in groups; the group assignments are given below (see section 5). Each group is responsible for organizing its work. It is recommended that responsibilities are assigned directly to group members; such responsibilities should include at least:

- Group leader
- Software developers
- Testers
- Standards/Protocol Designers
- Documentation Writers and Manager

Clearly, each group member will likely have multiple responsibilities.

4.2 Standards and Protocols Design

As explained above, you will be responsible for designing most of the protocols to be used. These will be relatively simple. Nevertheless, the groups will have to work together to develop these standards. Otherwise, it is highly unlikely that the peers from different groups will work together. For this purpose you should form standards committees to which all groups contribute. The work products of these standards groups are documents that specify the protocols to be used. These committees should also coordinate interoperability testing.

You should begin with the design of the protocols for the applications needed in step I and later proceed to the step II protocols.

4.3 Schedule

The project begins immediately and will run through the end of the semester. Groups will present their projects and conduct interoperability demonstrations during the last class of the semester. You should plan to complete the formation phase the week of March 27, and the other two steps by April 26.

I will schedule group progress reports and reports from the standards committee (alternating) every week during class. The first progress report from groups will be on March 6. The first report from the standards group will be on March 8.

4.4 Deliverables

Each group must maintain a project web site that documents the group's progress. At the very minimum that web site should document the group's plan, including assigned responsibilities, milestones, and a list of current problems. Group "proprietary" information should not go on the web site unless it is password protected. Each group will have to submit a final report that documents the design and implementation.

4.5 Programming Languages and Computers

You should be able to use any computer for this project as long as it is able to connect to the Internet. You may have problems if your host is behind a firewall. Each group is to field at least one cache.

The programming language to be used is your choice. Use the programming language that you are most comfortable with. Both C/C++ and Java are good choices. If there is expertise in your group, Perl, Ruby, or Python may work even better. It may be easier to find a good compiler to work on Unix, but with the availability of CygWin you have access to the same tools on Windows. Perl, Ruby, and Python also work both on Unix and Windows.

4.6 Grading

In addition to the final report, each group member will turn in a confidential evaluation of all the other team members. This peer evaluation will be combined with my overall evaluation of the group's work to arrive at a final grade for the each student. Hence, both individual efforts as well as the group's performance will factor into the final grade for the project.

5 Group Assignments

5.1 Group I

- Berhane, Dawit B.
- Gharavi, Mahdi
- Noboa Heredia, Mauricio A.
- Sarang, Bora
- Weidman, Andrew C.

5.2 Group II

- Diala, Jovette B.
- Kim, Byung U.
- Man, Michael Y.
- Nguyen, Thang V.
- Sood, Karan

5.3 Group III

- Gallahan, Kevin M.
- Misleh, Robert P.
- Nonaka, Justin H.
- Roberts, Kristin M.
- Singh, Rajendra M.

5.4 Group IV

- Alam, Shafia A.
- Baker, David C.
- Mahmood, Rehan S.
- Manthena, Mohan V.
- Sullivan, Michael B.
- Thall, William P.

5.5 Group V

- Esteki, Danesh J.
- Jung, Joseph K.
- Nah, Jongbong
- Reichert, Eric D.
- You, Hye-Jin