Highly Available Caching

Peter B. Danzig
Network Appliance
2770 San Thomas Expressway
Santa Clara, CA 95051
danzig@netapp.com
(408)367-3402 (voice) (408)367-3451 (Fax)
Katia Obraczka
Information Science Institute
University of Southern California
4676 Admiralty Way suite 1001
Marina del Rey, CA 90292
katia@isi.edu
(310)822-1511 (voice) (310)823-6714 (Fax)

Abstract

This paper presents techniques for improving availability, fault tolerance, and load balancing in information caching systems. Our approach uses a *cluster* of two or more cache servers. The cache cluster appears as a single virtual cache to the outside world. Each cache periodically checks the availability of the current cluster *representative*. In case the representative is down, one of the *backup* caches becomes the cluster representative and starts servicing requests on the cluster's behalf. When the representative comes back up, it re-assumes its role.

Our current design and implementation uses a cluster size of two cache servers but can be extended to clusters of arbitrary sizes.

1 Introduction

Large Internet Service Providers (ISPs) rely heavily on caching for improved performance. Caches are also used to implement company's firewalls. Caching can considerably improve response time, save network bandwidth, and offload servers. Unfortunately, caches constitute a single point of failure. Cache failures can cause significant service performance degradation and must be avoided.

The goal of highly-available caches is to address fault tolerance in caching architectures. A similar solution has been proposed for achieving fault tolerance in routers [?].

The idea is to organize caches in *clusters* of two or more cache servers. At any given time, one of the cluster members is operating as the cluster *representative*, who is responsible for answering requests received by the cluster. The remaining members of the cluster operate as *backups*. If for any reason the cluster's representative becomes unavailable, a backup will automatically take on the representative role and will start answering requests on behalf of the cluster.

Clients reference a cache cluster using the cluster's IP address, an IP address to which the virtual interfaces of all cluster members are bound. Cache cluster clients such as web browsers will be configured to point to a cluster the same way as they point to regular caches.

In the remainder of this document, we assume clusters of two caches. The cache cluster protocol we describe below can be extended to handle clusters of more than two caches.

2 The Cache Cluster Protocol (CCP)

The cluster pair runs the cache cluster protocol (CCP) which allows the backup cache to detect failure of the cluster representative and to take on its role. For load balancing, the representative role may alternate periodically between the two caches.

At startup, one of the caches in a cluster is configured as the cluster representative. Alternatively, the cluster can go through an *initialization* phase, where the cluster caches start up as backups. Depending on their configured *priority*, one of the caches assumes the role of the cluster representative.

The backup cache periodically polls the representative by sending standard ICMP request messages (or `ping`'s) and waiting for replies. If for some reason the representative does not respond, the backup assumes it is down and becomes the cluster representative. Once the default representative comes back up, it resumes its functions and the backup cache goes back to its default backup state.

At any given time, there is only one cluster representative who responds to requests destined to the cluster. Requests are addressed to a virtual IP address, or the cluster's IP address, which is bound to the virtual interface of both cluster caches. The representative has that interface turned on, while the backup cache has it turned off. When the backup cache takes on the representative role, it turns on the virtual interface bound to the cluster's IP address. It also sends a gratuitous ARP broadcast to directly connected routers and gateways. The ARP broadcast causes the appropriate routing table MAC-IP adress mapings to be updated. This way future client requests will be delivered to the current representative.

When the default representative comes back up, it sends an ARP broadcast to update directly connected routers. The representative also sends periodic

ARP broadcasts to ensure ARP records are up-to-date.

3 Implementation

We implemented the representative and backup side of the protocol using a single perl script. Through a command line paramenter, the cluster administrator configures a cluster cache as representative or as backup. Figure 1 shows the protocol's pseudocode.

Besides configuring a cache as representative or backup, the cluster administrator can also use command line parameters to configure the parameters below. Otherwise, they take on their default values.

- Path names for ping, and ifconfig. Their default values are `/usr/sbin/ping`, and `/sbin/ifconfig`.
- Name of virtual interface bound to cluster IP address.
- Cluster IP address.
- Cache server IP address.
- Ping timeout period, with default value set to 3 seconds.
- Poll interval with default value set to 3 seconds.

The ping timeout interval is the interval in seconds the backup waits for an answer from the cluster representative before declaring it to be down. We set the current default value to 3 seconds [1]. The frequency at which the backup cache polls the representative is given by 1/poll interval. The poll interval's default value is currently set at 3 seconds. The cluster administrator can configure the timeout and poll interval depending on how responsive to cache failures the cluster should be.

4 Usage and Testing

The cluster protocol can be run by using the following command:

```
clusterd -s state -i virtual_interface -c cluster_IP -p cache_IP
              [-I ifconfig_path] [-P ping_path] [-D polling_duration]
              [-T ping_timeout] [-d logging flag]
```

[1] Note that our default value overwrites `ping`'s default value of 20 seconds.

```
for(;;)
{
   /* Backup cache */
   if cache is BACKUP
   {
        if (ping is successful)
        /* Representative cache is up. */
        {
           /* Switch from representative to backup or stay as backup. */
           ifconfig virtual interface DOWN
        }
        elseif (ping timed out)
        /* Representative cache is down. */
        {
           /* Switch from backup to representative. */
           ifconfig virtual interface UP
        }
        else (ping failed)
           exit
   }
   /* Representative cache */
   else
   {
        /* Send gratuitous ARP broadcast. */
        ifconfig virtual interface UP
   }
   sleep POLL_INETRVAL;
}
```

Figure 1: Cluster protocol pseudocode

Note that the last five parameters are optional. If they are not specified in the command line, they take on their default values as specified above.

We are currently testing our protocol using Sun SPARCstation 20 machines running Solaris 2.5.1. The first set of experiments was conducted using a single, multihomed host. Both interfaces (`le0:1` and `le0:2`) were bound to the cluster IP address. Then, the protocol was configured to run in backup mode using `le0:1`, while `le0:2` was configured as the representative cache's interface. We used `ifconfig` to turn `le0:2` up or down and to verify that the other interface `le0:1` was being turned off or on accordingly. We used `traceroute` to verify that the backup cache sends an ARP broadcast when it becomes the cluster representative.

The experiments based on real practically environment was done using two Solaris 2.5.1 hosts. Each host were configured to have cluster IP binded to their virtual interface (`le0:1`). Then, the protocol was configured to run as initially representative cache on one machine, and as backup on the other. We unpluged the ethernet cable from representative cache to assume the representative cache was unable to repond requests or down and re-pluged it to assumimg the default representative cache was comes back up. The configured client was used to verify the responding of the cache by using reload command to request data from cache. Some of data were losted due to the switching from backup to representatie cache between transfering data but the new request were responed completely from the new comming up representative cache machine. We also used `tcpdump` to verify that the backup cache sends an ARP broadcast when it becomes the cluster representative.

5 Summary and Future Work

We described cache clustering, a technique for improving fault tolerance in information caching architectures. Caches in a cluster run the Cache Cluster Protocol (CCP) which allows them to detect failure of the current cache representative and select a new one.

We implemented and tested a prototype of the proposed cache clustering protocol. Below, we list items for future work.

Checking Cache Status

We determine whether the representative cache is up or down based on the response from `ping`. However, the cache program could be down and the `ping` would still be successful. We need a more effective way of assessing whether the representative is up or down.

Clusters of Arbitrary Size

We implemented the protocol for a cluster of size two. An item for future work is to extend the protocol to handle clusters of arbitary sizes.

Load Balancing Among Cache Clusters

One way of balancing the load within a cluster is by periodically rotating the representative role around participating caches.

We also propose combining cache clustering with object space partitioning across clusters using name-based mapping for improved fault tolerance and load balancing in caching architectures. Partitioning the object space across participating servers is a well-known technique to achieve load balancing in distributed information services. The Domain Name System (DNS) [?] organizes its name space hierarchically and partitions it according to administrative domain boundaries.

A similar approach can be used to distribute the load in information caching systems. Take for example a hierarchical cache architecture like the one proposed in [?]. The set of cached objects can be partitioned among cache clusters in the same hierarchical level.

A name-based mapping scheme like the one proposed in [?] could be used to map objects to clusters. Name-based mapping can be implemented using a hash function where the *key* is the object's name and *buckets* are the clusters. Since the an object is always mapped to the same cluster, requests for the same object will point to the same cluster. Name-based mapping schemes perform particulary well when objects are likely to be referenced multiple times.