

Problem Statement

Cache coherency is a necessary building block for implementing distributed systems. We're going to build a cache coherency protocol to coordinate a web tier.

Overview: We have a set of N web servers. Users connect to those servers and perform searches. The data for the searches comes from back end databases, which they cache. Periodically, this data is updated, and so those caches must be invalidated.

Exercises

- 1) Describe what cache coherency is
- 2) What would happen if we didn't invalidate updated entries? How would the user perceive this?
- 3) Why cache data in the first place?

Understanding Cache Invalidation

Objective: The objective of our scheme works like this when a request arrives:

- 1) the web server makes a connection to the database and requests whatever the user requested
- 2) it stores the results of that query into a local cache
- 3) in the future, when requests come in, they are served by the cache

Example:

- 1) User requests the price of an item on Amazon.com
- 2) The web server keeps a record consisting of $\langle \text{item\#}, \text{price} \rangle$

The Design Process

Why multicast?

The motivation for multicast stems from the fact that we have to communicate the same piece of information to numerous end points. Recall that each web server (from our set of size N) is basically interchangeable with other web servers. They don't authoritatively have data on their own—they act as front ends to the database. So when we update the

database, all the web servers must be updated. One option is to tell each one individually that they need to invalidate that entry. We chose to do it all at once—hence, multicast.

UDP or TCP?

We chose UDP. The reason for this is twofold: first, TCP doesn't really make sense in a one-to-many communication (see the dozens of papers about this). Secondly, we're going to assume that messages don't get lost (that the loss rate is basically 0). This requirement wouldn't necessarily work in a large scale production system, but let's go with it for now.

What information needs to go into the protocol?

Well, a cache invalidation protocol needs to invalidate some entry in a cache... So we have to identify what item is invalidated. In our system items are identified by unsigned integers.

Additionally, we're going to add a **sequence number**. Sequence numbers are extremely useful in networking, but in our case it will serve at least one purpose. The sequence number will let us know if any messages got lost.

So far, we have the following two pieces of information that we need to convey:

- 1) the item that is to be invalidated
- 2) a sequence number (starting at 0)

What will our packet format be?

In general, packet formats can be quite complex. In our case, we need to convey two unsigned integers. Thus, our format is very simple:

```
+-----+-----+-----+-----+
| item number                               |
+-----+-----+-----+-----+
| sequence number                           |
+-----+-----+-----+-----+
```

(each dash represents one bit of data)

What will be the protocol state machine on the database?

Recall from lecture that a protocol state machine is a finite state machine representing the state of a protocol on a particular host. In our case, the state machine is quite simple:

State 0: normal db processing

Transition 1: (from state 0 to state 0) **item updated**

New invalidation packet sent out. Counter updated.

The counter mentioned is copied into the sequence number field of the packet

For the client, the state machine is as follows:

State 0: normal web surfing

Transition 1: (from state 0 to state 0) **receive packet**

Delete cache entry for item indicated in packet. Update counter. If internal counter value differs from value in packet, signal error to user.

Exercises

1) What additional fields do you think might be necessary in the future?