Position Summary: Toward a rigorous data type model for HTTP

Jeffrey C. Mogul (Jeffrey.Mogul@Compaq.com)
Compaq Computer Corp. Western Research Lab., 250 University Ave., Palo Alto, CA 94301

Abstract

The HTTP protocol depends on a structure of several data types, such as messages and resources. The current ad hoc data type model has served to support a huge variety of HTTP-based applications, but its weaknesses have been exposed in attempts to formalize and (especially) to extend the semantics of caching within the HTTP distributed system.

1. Introduction

HTTP is a network protocol, but it is also the basis of a large and complex distributed system, with the possibility of caches at many points. An unambiguous and extensible specification of HTTP caching has proved difficult, because HTTP lacks a clear and consistent data type model for the primitive structures of the protocol itself. This is partly a consequence of a conceptual faultline between “protocol designers” and “distributed system designers,” and a failure to meld the expertise of both camps.

2. Problems with the current data model

Every HTTP request operates on a resource and results in a response. HTTP adopted the MIME term entity, defined as “The information transferred as the payload [headers and body] of a request or response ...” HTTP/1.1 added entity tags, used in cache validation. The server may attach an entity tag to a response; a client can then validate the corresponding cache entry by including this entity tag in its request to the server. If it matches the current entity tag, the server can respond with a “Not Modified” message instead of sending the entire entity.

What is the data type of the result of a simple HTTP GET operation? Is it an entity? The attempted analogy between MIME messages and HTTP data types treats the message as the central concern, which is true for MIME (an email protocol that transfers messages) but not for HTTP (a protocol for remote operations on resources). Also, HTTP allows the transmission of subranges of the bytes of a result, or of just the metainformation without the associated body, so the result might span several HTTP-layer messages. Therefore, an HTTP “entity” is merely an ephemeral, and perhaps partial, representation of one aspect of a resource.

So while HTTP has reasonably well-defined terms and concepts for resources and messages, it has no clearly defined term to describe the result of applying an operation to a resource. This might seem like a mere terminology quibble, but the lack of such a term, and the failure to recognize the concept’s importance, has led to a several difficult problems.

In particular, what does an HTTP cache entry store? Clearly not the resource itself (think of a CGI-generated resource). Not a Web “document,” since these are often composites of multiple resources with differing cachability properties. Instead, HTTP caches are currently defined as storing “response messages.” (I.e., an HTTP cache entry does not store what a resource is; it stores what the resource says.) As a result, it is difficult to define precisely what an HTTP cache must do in many circumstances, since the same resource could say two different things in response to two apparently identical requests. The lack of a clear formal specification for caching causes implementors to make guesses. This leads to non-interoperability, because content providers cannot predict what caches do.

It also makes it very hard to extend the protocol to handle partial updates (e.g., delta encoding) or even to define precisely how to combine existing HTTP/1.1 features (e.g., the ability to request a range of bytes and also to apply compression). The current model does not even provide a useful framework to discuss these questions.

3. A better model

We could solve these problems by adding a new data type, the instance. One can think of an instance as a complete snapshot of the current result of applying a GET to the resource. The instance can then be the input to a series of instance manipulations, which can include range selection, delta encoding, and compression.

In this model, HTTP cache entries are defined to store instances (or partial instances). An entity tag is tied to an instance, because it must be assigned prior to any instance manipulations. It is clearly not tied to the “entity” (and would better have been called an “instance tag”). Therefore, a cache can tell that two partial pieces of the same instance may be combined, because they have the same entity tag.

The implications of the new model (necessary protocol changes; the ability to more rigorously define existing and new HTTP features) require a longer writeup. (See research.compaq.com/wrl/people/mogul/hotos8). But it should be clear that the long-term success of a protocol such as HTTP depends on clear definitions that address distributed-systems issues, and on a better dialog between protocol designers and operating systems people.