Maintaining Mutual Consistency for Cached Web Objects

Bhuvan Urgaonkar, Anoop George Ninan, Mohammad Salimullah Raunak
Prashant Shenoy and Krithi Ramamritham

Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003
{bhuvan,agn,raunak,shenoy,krithi}@cs.umass.edu
http://lass.cs.umass.edu/projects/broadway

Abstract

Existing web proxy caches employ cache consistency mechanisms to ensure that locally cached data is consistent with that at the server. In this paper, we argue that techniques for maintaining consistency of individual objects are not sufficient—a proxy should employ additional mechanisms to ensure that related web objects are mutually consistent with one another. We formally define the notion of mutual consistency and the semantics provided by a mutual consistency mechanism to end-users. We then present techniques for maintaining mutual consistency in the temporal and value domains. Our techniques provide several tunable parameters that allow a tradeoff between network overhead and the fidelity of consistency guarantees. A novel aspect of our techniques is that they can adapt to the variations in the rate of change of the source data, resulting in judicious use of proxy and network resources. We evaluate our approaches using real-world web traces and our results show that (i) careful tuning can result in substantial savings in the number of polls incurred without any substantial loss in fidelity of the consistency guarantees, and (ii) the incremental cost of providing mutual consistency guarantees over mechanisms to provide individual consistency guarantees is small.

1 Introduction

Web proxy caching is a popular technique for reducing the latency of web requests. By caching frequently accessed objects and serving requests for these objects from the local cache, a web proxy can reduce user response times by up to 50% [12, 16]. However, to fully exploit this benefit, the proxy must ensure that cached data are consistent with that on servers. Several techniques such as time-to-live (TTL) values [10], client polling [8] and leases [6, 18] have been developed to maintain consistency of cached web objects. In this paper, we contend that maintaining consistency of individual objects at a proxy is not sufficient—the proxy must additionally ensure that cached objects are mutually consistent with one another. The need for mutual consistency is motivated by the observation that many cached objects are related to one another and the proxy should present a logically consistent view of such objects to end-users. Consider the following examples that illustrate the need for mutual consistency.

- **News articles**: Most newspaper web sites carry breaking news stories that consist of text (HTML) objects accompanied by embedded images and audio/video news clips. Since such stories are updated frequently (as

*Also affiliated with the Dept. of Computer Science and Engg., Indian Institute of Technology, Powai, Bombay.
additional information becomes available), a proxy should ensure that cached versions of such stories and the accompanying embedded objects are consistent with each other.

- **Sports/financial information**: Proxies that disseminate sports information such as up-to-the-minute scores also need to ensure that cached objects are consistent with each other. For instance, a proxy should ensure that scores of individual players and the overall score are mutually consistent. Similarly, a proxy that disseminates financial news should ensure that various stock prices as well as other financial information such as stock market indices are consistent with one another.

To quantitatively demonstrate the need for mutual consistency mechanisms, we considered two time-varying web objects from two newspaper web sites. We assumed that the web proxy employs a cache consistency mechanism to ensure consistency of each individual object\(^1\) but does not employ any additional mechanisms for maintaining mutual consistency. We computed the number of occasions on which the cached versions of the objects were out-of-sync by more than ten minutes with one another. Figure 1 depicts our results. We found that even when each object was polled approximately every 15 minutes, the two objects were inconsistent with one another on about 80 occasions over a 45 hour period. Furthermore, the number of violations increased as the frequency at which the proxy polled the server was decreased. Since our results indicate that more frequent polls result in fewer mutual consistency violations, one technique to provide mutual consistency guarantees might be to simply poll each object more frequently. However, such an approach can be wasteful in the number of polls, especially if each individual object is changing less frequently. This motivates the need to develop efficient mutual consistency mechanisms that augment existing consistency mechanisms for individual objects.

In this paper, we present adaptive techniques for maintaining mutual consistency among a group of objects. The contributions of our work are three-fold: (i) we identify the need for mutual consistency among web objects, (ii)
we formally define the semantics for mutual consistency, and (iii) we propose solutions to provide such consistency guarantees.

We begin by formally defining consistency semantics for individual objects and groups of objects. We argue that mutual consistency semantics are not intended to replace existing cache consistency semantics; rather they augment consistency semantics for individual objects provided by web proxies. Since a mutual consistency mechanism builds upon that for individual consistency, we first propose an adaptive technique for maintaining consistency of individual objects. A novel aspect of our technique is that it deduces the rate at which an object is changing at the server and polls the server at approximately the same frequency (thereby reducing the number of polls required to maintain consistency guarantees). Next, we show how to augment this technique with a mechanism to maintain consistency among a group of objects. Our approach can bound the amount by which related objects are out-of-sync with one another and thereby provide mutual consistency guarantees. Our technique provides tunable parameters that allow network overhead (i.e., number of polls) to be traded off with the fidelity of consistency guarantees.

We demonstrate the efficacy of our approaches through trace-driven simulations. Our simulations are based on real-world traces of time-varying news and financial data and show that: (i) careful tuning can result in substantial savings in the number of polls incurred without any substantial loss in fidelity of the consistency guarantees, and (ii) the incremental cost of providing mutual consistency guarantees over mechanisms to provide individual consistency guarantees is small (even the most stringent mutual consistency requirements result in less than a 20% increase in the number of polls).

The rest of this paper is as structured as follows. Section 2 formally defines the notions of individual and mutual consistency semantics used in this paper. Section 3 presents individual and mutual consistency techniques for the temporal domain, while Section 4 presents these techniques for the value domain. Design considerations that arise when implementing our techniques are discussed in Section 5. Section 6 presents our experimental results. Section 7 presents related work, and finally, Section 8 presents concluding remarks.

2 Individual and Mutual Consistency: Definitions and Approaches

In this section, we first define consistency semantics for individual web objects and then formally define the notion of mutual consistency for web objects.

Consider a proxy that services requests for web objects. To improve the latency for such requests, assume that the proxy caches frequently accessed objects. Cache hits are then serviced using locally cached data, while cache misses require the proxy to fetch the requested object from the server. Typically, the proxy employs a cache consistency mechanism to ensure that users do not receive stale data from the cache. To formally define consistency semantics provided by such a mechanism, let $S^t_a$ and $P^t_a$ denote the version of the object $a$ at the server and proxy, respectively, at time $t$. The version number is set to zero when the object is created at the server and is incremented on each subsequent update. The version number at the proxy is simply that of the corresponding version at the server. We implicitly require all cache consistency mechanisms to ensure that $P^t_a$ montonotically increases over time. That is, updates from a server should never arrive out-of-order at a proxy and a proxy should never replace a cached object with an older version.
In such a scenario, a cached object is said to be strongly consistent with that at the server if the version at the proxy is always up-to-date with the server. That is,

$$\forall t, \quad S^a_t = P^a_t$$

This definition of strong consistency ignores network delays incurred in propagating updates to the proxy. Network delays can be accounted for by modifying the above definition as $$\forall t, \quad S^a_{t-d} = P^a_t$$, where $$d$$ denotes the network delay. Strong consistency is difficult to achieve on the Internet due to two reasons. First, due to the large and unbounded message delays on the Internet, a proxy can never strictly guarantee that its version is consistent with the server. Second, network partitions can isolate the proxy from the server and violate consistency guarantees. Furthermore, strong consistency requires that every update to the object be propagated to the proxy. This is not only expensive but also wasteful if the proxy is not interested in every single update. The advantage though is that strong consistency does not require any additional mechanisms for mutual consistency, since the definition of strong consistency implies that objects will always be consistent with one another.

Since many web applications are tolerant to occasional violations of consistency guarantees, we can relax the notion of strong consistency as follows. A cached object is said to be $$\Delta$$-consistent if it is never out-of-sync by more than $$\Delta$$ with the copy at the server. Unlike strong consistency, $$\Delta$$-consistency allows an object to be out of date with the copy at the server, so long as the cached object is within a bounded distance (i.e., $$\Delta$$) of the server at all times. An important implication of $$\Delta$$-consistency is that it does not require every update to be propagated to the proxy—only those updates that are essential for maintaining the bound $$\Delta$$ need to be propagated. $$\Delta$$-consistency can be enforced in the time domain or the value domain. In the time domain, it requires that the copy at the proxy be within $$\Delta$$ time units of the server version at all times. That is,

$$\forall t, \quad \exists \tau, \quad 0 \leq \tau < \Delta, \quad \text{such that} \quad S^a_{t-\tau} = P^a_t$$

We refer to these semantics as $$\Delta_t$$-consistency. To define $$\Delta$$-consistency in the value domain, let $$S^a_t$$ and $$P^a_t$$ denote the value of the object $$a$$ at time $$t$$. Then $$\Delta_v$$-consistency requires that the difference in the values between the proxy and the server versions be bound by $$\Delta$$. That is,

$$\forall t, \quad |S^a_t - P^a_t| < \Delta$$

Whereas $$\Delta_v$$-consistency is meaningful only when the cached object has a value (e.g., stock prices, sports scores, weather information), $$\Delta_t$$-consistency can be applied to any web object. A number of techniques can be used to enforce $$\Delta$$-consistency at a proxy.\(^2\) $$\Delta_t$$-consistency, for instance, can be simply implemented by polling the server every $$\Delta$$ time units and refreshing the object if it has changed in the interim. By pulling an update every $$\Delta$$ time units, the proxy can ignore all other updates that occur between successive polls and yet maintain consistency guarantees. A more efficient mechanism requires the proxy to predict future changes based on past history and poll the server accordingly [14]. For $$\Delta_v$$-consistency, a proxy must refresh the cached object every time its value at the

\(^2\) In this paper, we consider only proxy-based approaches. Server-based approaches for enforcing $$\Delta$$-consistency are also possible. In such approaches, the server pushes relevant changes to the proxy (e.g., only those updates that are necessary to maintain the $$\Delta$$-bound are pushed). The study of such server-based approaches is beyond the scope of this paper.
server changes by $\Delta$. To do so, the proxy needs to track both the frequency of changes of an object as well as the magnitude of each change in order to predict the next time the object will change by $\Delta$ [14]. Regardless of the exact approach, all proxy-based mechanisms need to adapt to dynamic changes to the data, since most time-varying web data changes in a random fashion.

Having defined consistency semantics for individual objects, let us now examine consistency semantics for multiple objects. For simplicity, we focus only on two objects but all our definitions can be generalized to $n$ objects. To formally define mutual consistency ($M$-consistency), consider two objects $a$ and $b$ that are related to each other. Cached versions of objects $a$ and $b$ at time $t$, i.e., $P^a_t$ and $P^b_t$, are defined to be mutually consistent in the time domain ($M_t$-consistent) if the following condition holds

$$\text{if } P^a_t = S^a_{t_1} \text{ and } P^b_t = S^b_{t_2} \text{ then } |t_1 - t_2| \leq \delta$$

where $\delta$ is the tolerance on the consistency guarantees. Intuitively, the above condition requires that the two related objects should have originated at the server at times that were not too far apart. For $\delta = 0$, it requires that the objects should have simultaneously existed on the server at some point in the past. Note that mutual consistency only requires that objects be consistent with one another and does not specify any bounds on individual objects and their server versions (i.e., although mutually consistent, the objects themselves might be outdated with their server versions). Consequently, $M_t$-consistency must be combined with $\Delta_t$-consistency to additionally ensure the consistency of each individual object. This clean separation between $M$-consistency and $\Delta$-consistency allows us to combine any mechanism developed for the former with those for the latter. It also allows us to easily augment weak consistency mechanisms employed by existing proxies with those for mutual consistency.

Mutual consistency in the value domain ($M_v$-consistency) is defined as follows. Cached versions of objects $a$ and $b$ are said to be mutually consistent in the value domain if some function of their values at the proxy and the server is bound by $\delta$. That is,

$$\forall t, \ |f(S^a_t, S^b_t) - f(P^a_t, P^b_t)| < \delta$$

where $f$ is a function that depends on the nature of consistency semantics being provided. For instance, if the user is interested in comparing two stock prices (to see if one outperforms the other by more than $\delta$), then $f$ is defined to be the difference in the object values. Like in the temporal domain, the above definition provides a separation between $\Delta_v$-consistency and $M_v$-consistency—the former ensures that the cached value of an object is consistent with the server version, while the latter ensures that some function of the object values at the proxy and the server are consistent. Table 1 summarizes the taxonomy of consistency semantics discussed in this section.

There are several possible mechanisms to implement each of the above consistency semantics. We use a metric referred to as fidelity to determine the efficacy of a particular mechanism. Fidelity is defined to be the degree to which a cache consistency mechanism can provide consistency guarantees to users. For instance, a $\Delta_t$-consistency mechanism that satisfies Equation (2) for 95% of the time is said to have a fidelity of 0.95. Fidelity can be computed in two different ways. It could be computed either based on the number of instances for which consistency guarantees are violated, or based on the total time for which a cached object is out-of-sync with the server. In this paper, we use both measures of fidelity to quantify the effectiveness of our proposed approaches.
### Table 1: Taxonomy of Cache Consistency Semantics

<table>
<thead>
<tr>
<th>Semantics</th>
<th>Domain</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta_t$</td>
<td>temporal</td>
<td>individual</td>
<td>Object $a$ is always within 5 min. of its server copy</td>
</tr>
<tr>
<td>$\bar{M}_t$</td>
<td>temporal</td>
<td>mutual</td>
<td>Objects $a$ and $b$ are never out-of-sync by more than 5 min.</td>
</tr>
<tr>
<td>$\Delta_v$</td>
<td>value</td>
<td>individual</td>
<td>Value of object $a$ is within 2.5 of its server copy</td>
</tr>
<tr>
<td>$M_v$</td>
<td>value</td>
<td>mutual</td>
<td>Difference in values of objects $a$ and $b$ is within 2.5 of their difference at the server</td>
</tr>
</tbody>
</table>

### 3  Maintaining Consistency in the Temporal Domain

In this section, we present adaptive techniques for maintaining consistency in the temporal domain based on the definitions in Section 2. We first present a technique for maintaining consistency of individual objects and then show how to augment it for maintaining mutual consistency.

#### 3.1  Maintaining Consistency of Individual Objects

Consider a proxy that caches frequently changing web objects. Assume that the proxy provides $\Delta_t$-consistency guarantees on cached objects. The proxy can ensure that a cached object is never outdated by more than $\Delta$ with its server version by simply polling the server every $\Delta$ time units (using `if-modified-since` HTTP requests).

Whereas this approach is optimal in the number of polls when the object changes at a rate faster than $\Delta$, it is wasteful if the object changes less frequently—in such a scenario, an optimal approach is one that polls exactly once after each change. Consequently, an intelligent proxy can reduce the number of polls by tailoring its polling frequency so that it polls at approximately the same frequency as the rate of change. Moreover, since the rate of change can itself vary over time as hot objects become cold and vice versa, the proxy should be able to adapt its polling frequency in response to these variations.

We have developed an adaptive technique to achieve these goals. Our technique uses past observations to determine the next time at which the proxy should poll the server so as to maintain $\Delta_t$-consistency. To understand the intuition behind our technique, let us refer to the time between two successive polls as the *Time to Refresh (TTR)* value. Our technique begins by polling the server using a TTR value of $\Delta$. It then uses a linear increase multiplicative decrease (LIMD) algorithm to adapt the TTR value (and thereby, the polling frequency) to the rate of change of the object. If the object remains unchanged between two successive polls, then the TTR value is increased by a linear factor (resulting in less frequent polls). If the proxy detects a violation in the consistency guarantees between successive polls, then the TTR is reduced by a multiplicative factor (causing more frequent polls). If the object is updated between successive polls but does not violate consistency guarantees (indicating the proxy is polling at approximately the correct frequency), then the TTR value is increased gradually until the “correct” TTR is found.

Note that the *Time To Refresh (TTR)* value is different from the *Time to Live (TTL)* value associated with each HTTP request. The former is computed by a proxy to determine the next time it should poll the server based on the consistency requirements; the latter is provided by a web server as an estimate of the next time the data will be modified.

---

3 Note that the *Time To Refresh (TTR)* value is different from the *Time to Live (TTL)* value associated with each HTTP request. The former is computed by a proxy to determine the next time it should poll the server based on the consistency requirements; the latter is provided by a web server as an estimate of the next time the data will be modified.