

WCDP: A protocol for web cache consistency

Renu Tewari, Thirumale Niranjan[†], Srikanth Ramamurthy[†]

IBM Almaden Research Center, [†]IBM Software Group
{tewarir, niranjan, ramamur}@us.ibm.com

Abstract

Cache consistency at web intermediaries is required for scalable web content delivery. In this paper we describe the Web Content Distribution protocol (WCDP), which is an invalidation and update protocol to provide cache consistency for a large number of frequently changing web objects. WCDP supports different levels of consistency: strong, delta, weak, and explicit. It supports atomic invalidates and mutual consistency among objects and handles multiple deployment architectures. WCDP achieves scalability by grouping objects and messages together and by using a hierarchical organization for message delivery. WCDP operates between the origin server, mirror sites, and the participating web intermediaries. It is not, however, targeted for inter-CDN operations.

1 Introduction

With caches being widely deployed for scalability and performance improvements, the issue of cache consistency and a protocol to support it remains ad hoc at best. In the current mode, consistency management is controlled by the intermediary relying on a client driven mechanism called client polling. An intermediary based on its optimization criteria can poll the server using the HTTP if-modified-since (IMS) request to check if an object is fresh. This on-demand polling by the intermediary reduces the benefit of caching by adding to the latency observed by the users and reduces the scalability of servers due to the extra message overhead especially when the objects are updated infrequently. The content provider can provide hints based on HTTP cache control headers with expiration times or a max-age directive. However, these are coarse mechanisms and do not control the level and degrees of consistency that may be required by content providers for dynamic content. Most expiration times are not known a priori and content providers resort to setting very small values to control consistency or simply mark the object as uncacheable.

The aim of the WCDP invalidation protocol is to enable server-driven consistency where the content provider can dynamically control the propagation and visibility of an object update. In server-driven consistency, the origin server invali-

dates or updates the data when it changes without the intermediary resorting to frequent polling. WCDP supports multiple levels of consistency: (1) strong consistency for mirror sites, (2) delta consistency for participating intermediaries, and (3) explicit consistency as a default. It supports atomic invalidates for maintaining mutual consistency among objects. The intermediaries can explicitly subscribe to receive invalidation (or update) notifications or can rely on an implicit subscription set up by an administrator for mirror sites or partnering intermediaries. WCDP enhances scalability by grouping objects together as an addressable unit and also by grouping messages together. It supports distribution hierarchies for scalable message delivery. WCDP assumes a reliable underlying transport mechanism for in-order message delivery in its normal operation. Failure scenarios are discussed in Section 3.7. Although not part of the protocol, WCDP supports authorization and authentication for different levels of security support. The WCDP protocol is based on implementation experiences with the Content Distribution Framework of the IBM Websphere Edge Server Version 2.0. The following sections describe the design features of WCDP and the protocol highlights. More details can be found in the IETF submission [2].

2 Design Issues

2.1 Scalability

For the invalidation protocol to be scalable it should be able to scale with respect to the number of objects for which consistency has to be maintained, the number of messages sent and received, and the number of clients to which invalidations are sent. Scalability is achieved in WCDP by aggregating objects into *object-groups* and aggregating multiple messages into a single message-group. The protocol can invalidate an individual object or a set of objects addressed as a unit in an object-group. Clients can be organized into a distribution hierarchy where the WCDP server only communicates with the WCDP clients at the top level of the hierarchy, thereby, limiting the number of clients it communicates with. In the terminology used in the rest of the paper the WCDP server refers to the origin server and the WCDP client refers to a

caching intermediary.

2.2 Consistency levels

WCDP supports multiple consistency levels to control how and when the object changes are notified to the WCDP clients. In the simplest case a WCDP client relies on explicit consistency based on the HTTP cache-control headers and Expires tag. In another case, WCDP supports best-effort invalidations providing weak consistency. Weak and explicit consistency are supported by default. More stringent forms of consistency such as delta and strong consistency have to be explicitly requested. The multiple consistency levels that are supported by WCDP include: i) *strong consistency*: where a read at the WCDP client reflects the last committed update at the origin server [4], ii) *delta consistency*: where the read at the WCDP client cache can be up to "delta" time units stale with respect to the last committed update at the origin server, iii) *weak consistency*: where the read at the WCDP client does not necessarily reflect the last update at the origin server but some correct previous value, iv) *explicit consistency*: where an expiration time of an object is provided or a time-to-live (TTL) value is provided by the origin server given some a priori knowledge. v) *mutual consistency*: where a group of objects are mutually consistent with respect to each other. In this case some objects in the group cannot be more current than the others.

Strong consistency is useful for mirror sites that need to reflect the current state at the origin. It is also required if WCDP is used to update multiple origin servers that are part of a big cluster. Certain type of applications can tolerate stale data as long as it is within some known time bound. For such applications delta consistency is recommended [1]. Delta consistency assumes that there is a bounded communication delay between the WCDP server and client. Mutual consistency is useful when a certain set of objects at a WCDP client (e.g., the fragments within a sports score page, or within a financial page) need to be consistent with each other. In this case they are atomically invalidated such that they all either reflect the new state or remain in the earlier stale state.

2.3 Invalidates and Updates

If data is changing infrequently then, for small data sizes, sending the updated object instead of an invalidate message improves performance. Sending an invalidate message causes an intermediary to mark the object as invalid; a subsequent request requires the intermediary to fetch the object from the server (or from a designated site). Thus, each request after a cache invalidate incurs an additional delay due to this remote fetch. An invalidation adds to 2 control messages and a data transfer (an invalidation message, a read request on a miss, and a new data transfer) along with the extra latency. No such delay is incurred if the server sends out the new version

of the object upon modification. A drawback, however, is that sending updates incurs a larger network overhead (especially for large objects). Delta encoding techniques have been designed to reduce the size of the data to update by sending only the changes to the object. (Delta encoding is not related to delta consistency) [10]. Updates, however, require better security guarantees and make strong consistency management more complex. Nevertheless, updates are useful for mirror sites where data needs to be "pushed" to the replicas when it changes. Updates are also useful for preloading caches with content that is expected to become popular in the near future.

WCDP supports invalidation by default but also extends it to support updates via a *refresh* directive without resorting to sending data as part of the protocol. In WCDP updates are used to "push" content from the origin server to mirror sites and is handled by combining an invalidation with an *immediate-refresh* directive that causes the WCDP client to send a read (or IMS) request to the origin server to get a new copy of the object. It is the responsibility of the client to load the new version of the object from the origin server. If all clients happen to load immediately, it may cause a load surge at the origin server. The origin server can further extend an invalidate with a *delayed-refresh* directive and a TTL value that defines the duration the client must wait before sending a read request. A WCDP client sends a read request to the origin server only after the TTL time interval. This limits the burst of requests at the origin server.

3 Protocol Details

3.1 Object Invalidation Identity

In WCDP, an object is identified for invalidation by its i) `obj_invalidation_id`, and optionally, ii) a URL.

The `obj_invalidation_id` is a unique opaque string assigned by the origin server for the purpose of explicit invalidation of objects by name. It is useful to not attach any semantic meaning to the `obj_invalidation_id`, but rather to view it as an opaque unique invalidation identifier associated with an object. When the WCDP client cache makes an HTTP GET request to the origin server, it receives an `obj_invalidation_id`, corresponding to the URI requested, embedded in the HTTP response as a private header. The WCDP client then maintains the mapping between an `obj_invalidation_id` and an internal `obj_cache_id`, and the external URL. This mapping is essential, when we consider that an HTTP Server maps external URLs to local filenames, but there is no way to compute a reverse mapping from filenames to URLs. WCDP solves the problem by piggybacking this information in the response from the origin server, allowing for incremental construction of the reverse mappings at the requesting WCDP client.

The internal `obj_cache_id` is the identifier by which the WCDP client cache matches an incoming request with the lo-

cal cached object. For static content, it is typically the same as the external URL. For dynamic content with fragments, the external URL along with other HTTP header tags and cookie values are combined to create the `obj_cache_id` for each fragment. For a given `obj_invalidation_id`, there could be multiple `obj_cache_id`'s. For example, if an object has multiple variants (for different languages, user agent types, etc.) it may use the same `obj_invalidation_id` but will have different `obj_cache_ids` for each variant. The form, specification and interpretation of the `obj_cache_id` is not within the scope of the WCDP protocol and is determined by each individual WCDP client cache implementation.

3.2 Object Grouping

In WCDP objects can be grouped into *object groups* or volumes and addressed as a unit. Object groups enhance scalability by limiting the size and number of messages, and the state at the WCDP server. For example, all objects in a sub-directory can belong to the same object group. Each WCDP client is informed of the object group a requested object belongs to by the origin server. The origin server sends the `object_group_invalidation_id(s)` along with the `object_invalidation_id` in the HTTP response as a private header. Objects can belong to multiple object groups. Invalidation messages can be issued for all the objects in an object group by just naming the group itself.

A related concept is that of *atomic invalidations* to support mutual consistency. Objects can be related to each other due to references (hyperlinks) between them or due to inclusion (multiple dynamically computed objects are assembled to form a personalized page). In an atomic invalidate/update, the objects are invalidated/updated using lock semantics. The objects are not accessible to the user agents (locked) until all the objects in the set are invalidated/updated.

3.3 Content Groups and Subscription

Subscription to notifications by the WCDP clients enhances the scalability of the system by reducing the number of messages transmitted. Subscriptions are at the granularity of *content groups*. A content group is a large aggregation of objects. Objects can belong to multiple "content groups". Each content group represents objects that are related by user interest. For instance, content groups can be topic-based, e.g., sports, news, sports/baseball, etc. Objects can be classified into content groups by the content creator at the origin server. Similar to the object group, a content group becomes metadata with which the object is associated, again returned as part of the HTTP response private header. The WCDP client sends an explicit `register` message to the WCDP server indicating the content groups that it is interested in. Once that message is acknowledged, the origin server will send invalidations to

objects belonging to those content groups.

3.4 Consistency support

WCDP by default supports explicit and weak consistency. With strong consistency and invalidations, the WCDP server waits (or times out) until it receives the invalidation responses from all the participating WCDP clients that were sent the invalidation requests. After the responses are received the object can be made "live" at the origin server. When strong consistency is desired with updates (that is, invalidates with the `immediate-refresh` directive), the updates are propagated in a two-phase manner. During the first phase, the WCDP server sends the invalidation notification with an `immediate-refresh` directive to all subscribing WCDP clients. Upon its receipt, each WCDP client pulls the content from the origin server and stores it a temporary location, and then sends an invalidation response to the WCDP server. When all WCDP clients have responded, the second phase `commit` message is sent by the WCDP server to all the WCDP clients, which causes them to make the new content "live"; the origin server also makes the content "live". The WCDP clients respond with a `commit` response. While the WCDP client caches are waiting for a `commit` request, any user agent request for the object is forwarded to the origin server and not cached. Note that the strong consistency requirement for updates does not imply that all WCDP client caches are tightly synchronized. All it implies is that the WCDP client would either have the new object or have the older one marked as invalid. Failure scenarios are discussed in section 3.7. The underlying assumption made is that the messages are sent over a reliable transport such that the WCDP client and servers can determine if there is a failure of client, server or the network.

WCDP provides delta consistency if the message latency is bounded and is less than delta. The WCDP server sends a heartbeat messages with a period smaller than delta. The delta value can be defined per content group or explicitly per object or object group. However, different values of delta at fine-granularity adds a lot of overhead. It is recommended to have a common delta for a content group. In case a WCDP client does not receive a heartbeat for more than delta time units since the last heartbeat or request, it marks the corresponding content group as temporarily invalid till it resolves with the WCDP server again. The result of this action is to revert to an explicit consistency mode for all objects at the WCDP client that belong to the content group.

3.5 Message Types and Formats

Messages in WCDP are either request or response messages which are sent and received by the WCDP clients and servers. Message can be grouped together into a message group to batch multiple messages together.

There are 9 types of messages in WCDP: i) Invalidation request, ii) Invalidation response, iii) Register request, iv) Register response, v) Join request, vi) Join response, vii) Commit request, viii) Commit response, ix) Heartbeat request.

- *Invalidation request:* The invalidation request is sent by the WCDP server to the (subscribing) WCDP clients. The invalidation request consists of: 1) list of <identifiers>, 2) the invalidation action, 3) the invalidation type, 4) the consistency level. Each request is also tagged with a unique, monotonically increasing, request sequence number.

The identifiers consist of `object_invalidation_id(s)` and/or `obj_group_invalidation_id(s)` along with an optional external URL. An invalidation request may contain multiple identifiers in order to perform multiple invalidations together using a single message or for requiring an atomic invalidate of the identifiers in the request.

The invalidation action consists of either: 1) `immediate-invalidate`, 2) `delayed-invalidate` at a specific time or interval, 3) `immediate update` (invalidate with an `immediate-refresh` directive), 4) `delayed update` (invalidate with `delayed-refresh` after a specified interval) and possible combinations. The refresh directive is used to implement content "updates" by requiring the WCDP client cache to pull the content from the origin server. However, a WCDP client may not comply with the refresh directive and ignore pulling the content from the origin server. The delayed refresh is useful to stagger the requests at the origin server to avoid a surge of requests. It can also be used to schedule an update at a given time.

The invalidation type specifies if the object(s) need to be invalidated "atomically" or individually. In an individual invalidate each object or object in an object group is treated individually. The consistency level determines the type of consistency required. Weak and explicit consistency are supported by default.

- *Invalidation response:* The invalidation response is sent by the WCDP client to the WCDP server after receiving and processing the corresponding invalidation request. The invalidation response consists of a status code for each invalidation request. This consists of the 1) request sequence number, 2) status code for each `object_invalidation_id`.

Since multiple requests can be grouped in a message group the request sequence number is useful to match the requests and responses. Examples of status codes are: `SUCCESS_OK`, `OBJECT_NOT_FOUND`, `WAITING_FOR_COMMIT`, `NOT_AUTHORIZED`.

- *Register request:* The register request is sent by the WCDP client to the WCDP server to subscribe to notifications for objects in a content group. The register request consists of: i) a list of <contentgroup_id(s), last invalidate request sequence number if known> and ii) the consistency level supported by the client, iii) request sequence number. Depending on authorization requirements, it could contain credentials and other authorization information.
- *Register response:* The register response is sent by the WCDP server to the WCDP client in response to a corresponding register message. The register response consists of: 1) status code and 2) lease duration (optional), iii) request sequence number, 3) desired consistency level for each content group. The WCDP server initiates a catch-up sequence to get the WCDP client up-to-date with respect to the WCPD server. It computes the last set of invalidate requests for the objects in the content group (that are later than the last invalidation sequence number that the WCDP client has seen) and sends them with the register response.
- *Join request:* The join request is sent by the WCDP client to the WCDP server after recovering from a failure. The purpose of a join request is to initiate a catch-up sequence where the WCDP client can get up-to-date with the WCDP server. The join request consists of 1) list of <content group id, last invalidate request sequence number>, 2) request sequence number.
- *Join response:* The join response is sent by the WCDP server to the WCDP client cache. On receiving a join request, the WCDP server will compute the list of invalidations (only the last invalidation of an object needs to be sent) that need to be sent to the joining client. It will then package the invalidations and send them in the join response. The WCDP server will resume sending heartbeats (if had stopped them due to the failure). The join response consists of 1) status code 2) request sequence number. This is followed with a batch of invalidation requests.

Note that instead of a join request the WCDP client could have just sent another register request. We have tried to distinguish the rejoin after failure and the initial register.
- *Commit request:* When strong consistency is desired, the notifications are sent in a two-phase manner. The commit request is sent by the WCDP server to the WCDP client cache after it receives all acknowledgment responses back from the client caches. The commit request consists of 1) invalidation request sequence number, 2) identifier (used in the original invalidation request).

- *Commit response:* The commit response is sent by the WCDP client cache to the WCDP server and consists of 1) the invalidation request sequence number, 2) status code.
- *Heartbeat request:* If delta consistency or strong consistency is required, the WCDP server sends a periodic heartbeat message to the client. The heartbeat period is determined by the value of delta and should be smaller than delta. Also the heartbeat interval should be smaller than the timeout value used by the strong consistency implementation.
- *Message Format Example:* WCDP messages are in XML and sent using an HTTP POST method, possibly over a persistent TCP connection. Possible extensions include a SOAP-based model, as well as configuring the invalidation server as a Web Service. Table 1 shows the XML-based format of an invalidation request and response.

3.6 Distribution Hierarchies

The distribution of invalidation notifications can be made scalable by constructing a distribution hierarchy. The hierarchy has the WCDP server at the root, WCDP clients (or clients acting as gateways) at the intermediate levels and WCDP clients at the leaf level. A WCDP client could belong to multiple distribution hierarchies, and a distribution hierarchy could propagate invalidations for multiple WCDP servers. A WCDP client at the non-leaf level acts as a proxy for the WCDP server, receiving notifications from the higher level and forwarding them to the WCDP clients in the lower level in its sub-tree. With a tree-like organization the load on the WCDP server can be reduced, providing scalability. Similar approaches have been proposed by [12, 11, 9, 7, 6].

3.7 Failure and Recovery

WCDP recovers gracefully from failures. When the WCDP client fails and comes back up, it sends a `join` message to the WCDP server, which contains information about the last notification it acted upon. The WCDP server plays back the last set of notifications that the cache had missed, while being down.

When a WCDP server fails, it stops sending heartbeats to its WCDP clients; this causes the clients to start following the Cache Control headers. The clients will attempt to subscribe to an alternate WCDP server. This can be an administrative task, or automatic. In any case, once it re-establishes a path to a WCDP server, it will send it a register message if it is a new server, or periodically send join messages to the old server.

If a WCDP client has failed, the WCDP server detects it by way of a timeout on an invalidation response, and stops send-

ing it any heartbeats and removes it from the set of registered clients.

When a network partition occurs between a WCDP client and the WCDP server, each behaves as if the other has failed. The WCDP client cache reverts to obeying Cache Control headers. The WCDP server will remove the client from the registered set. The client will then try to register or join again.

4 Related Work

The WCDP protocol has borrowed from the multiple approaches proposed in the research literature. The issue of strong consistency for the web was discussed in [4, 6, 3]. While the focus there was to motivate the need for strong consistency and its performance evaluation, WCDP is an instantiation of a protocol that supports server-driven consistency. [10] describes the use of deltas for sending updates and an application-level multicast framework for internet distribution is discussed in [12]. WCIP, which is another protocol proposal submitted to the IETF for server-driven invalidation is similar in its approach to WCDP [1]. WCDP, however, supports different levels of consistency, can handle invalidates and updates, and is a request-response protocol.

References

- [1] D. Li, P. Cao, and M. Dahlin. WCIP: Web Cache Invalidation Protocol. IETF Internet Draft, November 2000.
- [2] R. Tewari, T. Niranjan, and S. Ramamurthy. WCDP: Web Content Distribution Protocol. IETF Internet Draft, February 2002.
- [3] J. Gwertzman and M. Seltzer. World-Wide Web Cache Consistency. In *Proceedings of the 1996 USENIX Technical Conference*, January 1996.
- [4] P. Cao and C. Liu. Maintaining Strong Cache Consistency in the World-Wide Web. In *Proceedings of the Seventeenth International Conference on Distributed Computing Systems*, May 1997.
- [5] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Hierarchical Cache Consistency in a WAN. In *Proceedings of the Usenix Symposium on Internet Technologies (USEITS'99), Boulder, CO (to appear)*, October 1999.
- [6] H. Yu, L. Breslau, and S. Shenker. A Scalable Web Cache Consistency Architecture. In *Proceedings of the ACM SIGCOMM'99, Boston, MA*, September 1999.
- [7] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering Server-driven Consistency for Large-scale Dynamic Web Services. In *Proceedings of the 10th World Wide Web Conference, Hong Kong*, May 2001.
- [8] C. Gray and D. Cheriton. Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 202–210, 1989.
- [9] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, R. Tewari. Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks. In *Proceedings of the World Wide Web conference (WWW2002)*, May 2002.

POST /invalidate_request HTTP/1.1

Content Length: 512

```
< ?xml version="1.0" >
< !DOCTYPE InvRequest "wcdp.dtd" >
< invalidation sequence_number = 100 >
< identifier >
< object_invalidation_id = 12345 />
< object_invalidation_group_id = 9876 />
</identifier>
< action>
< invalidate =immediate/>
< refresh= yes/>
< delay = 60 sec />
< force = no />
</action>
< type atomic= no />
< consistency require_commit = no />
</invalidation>
```

POST /invalidate_response HTTP/1.1

Content Length: 256

```
<?xml version="1.0">
<!DOCTYPE InvResponse "wcdp.dtd">
<invalidation sequence_number = 100 >
<identifier>
<object_invalidation_id = 12345 />
<status = OK />
</identifier>
<identifier>
<object_invalidation_group_id = 9876/>
<status = OK/>
</identifier>
</invalidation>
```

Invalidation Request

Invalidation Response

Table 1: An example request and response format

- [10] J C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential Benefits of Delta Encoding and Data Compression for HTTP. In *Proceedings of ACM SIGCOMM Conference*, 1997.
- [11] Z. Fei. A Novel Approach to Managing Consistency in Content Distribution Networks. In *Proceedings of the 6th Workshop on Web Caching and Content Distribution, Boston, MA*, June 2001.
- [12] P. Francis. Yoid: Extending the Internet Multicast Architecture. Technical report, AT&T Center for Internet Research at ICSI (ACIRI), April 2000.