DEVELOPING A WEB CACHING ARCHITECTURE WITH CONFIGURABLE CONSISTENCY: A PROPOSAL

Francisco J. Torres-Rojas and Esteban Meneses and Alexander Carballo
Computing Research Center - Costa Rica Institute of Technology
Escuela de Ingeniería en Computación, Instituto Tecnológico de Costa Rica,
POBox 159-7050 Cartago, Costa Rica

Keywords: Web Caching, Consistency Protocols, Timed Consistency.

Abstract: In recent years, Web Caching has been considered one of the key areas to improve web usage efficiency. However, caching web objects proposes many considerations about the validity of the cache. Ideally, it would be valuable to have a *consistent cache*, where no invalid relationships among objects are held. Several alternatives have been offered to keep consistency in the web cache, each one being better in different situations and for diverse requirements. Usually, web cachers implement just one strategy for maintaining consistency, sometimes giving bad results if circumstances are not appropriate for such strategy. Given that, a web cacher where this policy can be adapted to different situations, will offer good results in an execution with changing conditions. A web caching architecture is proposed as a testbed for consistency models, allowing both *timing* and *ordering* issues to be considered.

1 INTRODUCTION

The most important traffic carried by today networks certainly belongs to world-wide-web. Every second millions of users exchange gigabytes of information by browsing, downloading, buying, playing, talking, etc. Much of the current efforts aim to alleviate the delay experienced by users when surfing the web. While increasing the bandwidth available to users helps reducing latency, bottlenecks are found everywhere, user preferences and events (natural, economical, social, etc.) also rule the behavior of network congestion. One solution is to upgrade the congested network components or even performing load balancing techniques on them. However finite speed nature of signals crossing transoceanic or even satellite links, package queuing due finite processing time and storage, etc. impose limits on the expected response time.

Information been locally accessible will clearly improve the user experience while imposing reduced load on the external network. Many of the web objects exhibit access locality meaning that once requested they are most likely to be requested in the near future. So instead of visiting the source server, they could be locally stored and locally served. A large group of users with overlapping interests may request those locally stored objects without the burden of long delays.

However, caching and replication of web objects can produce inconsistencies, as it is known from the distributed systems literature. For example, consider a client requesting two pages: one from ESPN about the NBA playoff final game (which has not yet finished) and another from CNN with a link to the ESPN one (cached on the CNN site). After the match is over, suppose the ESPN page is changed so the result is presented, but the CNN one is using the cached version, thus offering a stale information about the game and making inconsistencies in the whole caching system. On the other side, there are some applications where such requirements are not needed and a more relaxed consistency policy is sufficient.

Several caching strategies have been published mostly based on the distributed nature of web objects (Wessels and Claffy, 1998; Cao and Liu, 1997; Yu et al., 1999; Tewari et al., 2002; Kawash, 2004; Shim et al., 1999). While previous work is mostly based on timing requirements for consistency, to our knowledge, only (Bradley and Bestavros, 2002) addresses both *ordering* and *timing*, fundamental concepts on distributed objects consistency (Ahamad and Raynal, 2003). Our aim in this paper is to present a proposal for an infrastructure allowing several consistency models to be tested, considering time and order.

J. Torres-Rojas F., Meneses E. and Carballo A. (2005).
DEVELOPING A WEB CACHING ARCHITECTURE WITH CONFIGURABLE CONSISTENCY: A PROPOSAL.
In *Proceedings of the First International Conference on Web Information Systems and Technologies*, pages 110-116
DOI: 10.5220/0001231801100116
Copyright © SciTePress

Section 2 introduces the web caching problem, while in section 3 some approaches for this problem are presented. In section 4 the main consistency protocols are reviewed and some useful techniques are presented. Section 5 presents a proposal to introduce consistency models in web caching. Finally, conclusions and future work are left for the last section.

2 WEB CACHING CONSISTENCY

As already mentioned, information being accessible in the cache will improve the user experience. According (Wang, 1999) there are several advantages for using web caching: it reduces bandwidth consumption so network traffic decreases and lessens network congestion; it reduces access latency due frequently accessed objects are fetched by a near proxy and non-cached objects are fetched in a relatively faster network; it reduces the work load of the remote server; if the remote server is no available, a copy of the requested object could be retrieved from a local proxy; it provides an option to monitor web access patterns.

The increased performance or efficiency of introducing some caching level could be measured by counting how many objects were locally accessed (from the cache) among the total number of requested objects, the cache hit rate. Another useful indicator of cache efficiency is the byte hit rate, the number of byte actually provided from the cache as a percentage of the total number of bytes transferred. The higher the byte hit rate the less traffic is injected on the external network or the backbone saving bandwidth.

Nevertheless, some disadvantages emerge as web caching is implemented (Wang, 1999). One of these is cache consistency. This issue has been considered in previous work (read section 3). Those schemes can be categorized into three different philosophies (Kawash, 2004):

- **Expiration Protocol** A client p requests object X from its owner q at time t, q sends a copy of the value of X with a *time-to-live* value Δ. So, every requests for X could be satisfied by the local cache of p during the interval $[t, t+\Delta]$. After that interval, the copy of X in the cache expires and the next request must be transmitted into the remote server.
- **Polling Protocol** A client p must be checking object X from its owner periodically in order to see if X has changed.
- **Invalidation Protocol** The owner q of X remembers all the clients p of object X so when X changes q sends an invalidation message to all its users. The next read for X at a specific client p_i suppose downloading a fresh value for X from q.

These policies have several performance and scalability implications (Kawash, 2004; Gwertzman and Seltzer, 1996; Cao and Liu, 1997). But, more important, it is often obviated what order protocol these models induce. Particularly, expiration and polling protocols implement *coherence* (a relaxation of sequential consistency model) and invalidation protocol implements *Sequential Consistency* or **SC** (read section 4).

Dilley *et al* (Dilley et al., 1999) recognize four levels where caching plays an important role in the web information infrastructure: (1) at the browser level, catching user preferences as to aid the rendering process of web contents; (2) at LAN level, collecting requested objects of a community of users; (3) at ISP or carrier level, reducing the injection of unnecessary traffic on the backbone; (4) at the source site, reducing the server load by caching the most requested objects avoiding unnecessary delays recomputing already served data. Figure 1 depicts those scenarios.

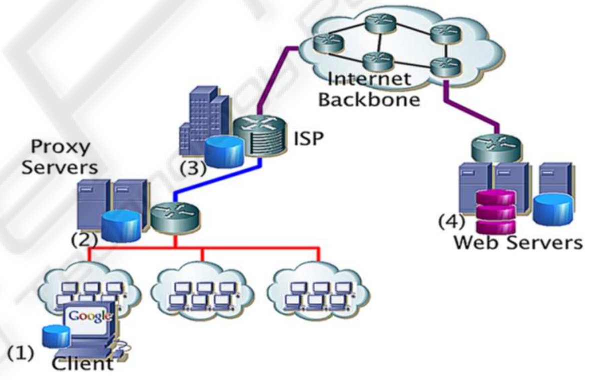

Figure 1: Web caching levels

The **HTTP/1.0** specification (Berners-Lee et al., 1996) comes with basic features helping web caches. For example the uniform resource locator (**URL**) provides a mean of object identification, when a client issues a **GET** command the cache server easily verifies the object existence using URL as key in a flat or a hash table. The *time-to-live* (**TTL**) value established a simple mechanism of object freshness and if coupled with the *If-Modified-Since* header more consistent results could be achieved.

Figure 2 clarifies the use of the TTL value in web caching. The client issues an object X read $r(x)$ which is attended by the proxy, since no cached version of X exists then the proxy issues the request to the source server returning a copy of the web object valid through the TTL value. Future requests from the client will be served from the proxy server until the expiration of the TTL when a new read operation to the source server is issued, thus TTL establishes a basic level of consistency.

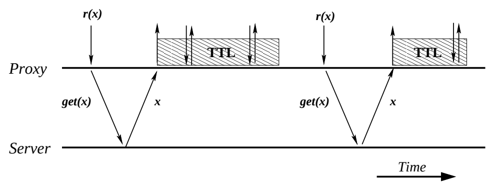

Figure 2: Basic consistency by means of TTL values

3 RELATED WORK

In the last 10 years, several models have been proposed for consistent web caching (Bradley and Bestavros, 2002; Cao and Liu, 1997; Cate, 1992; Duvvuri et al., 2000; Gwertzman and Seltzer, 1996; Mikhailov and Wills, 2003; Rabinovich and Spatscheck, 2002; Tewari et al., 2002; Yin et al., 1999; Yu et al., 1999), some of them are presented in (Kawash, 2004). In this section some ideas of these models are briefly reviewed.

Adaptive TTL is one of the first models, introduced in the Alex file system (Cate, 1992) was favored by the results of Gwertzman and Seltzer (Gwertzman and Seltzer, 1996). They found this protocol reduces network bandwidth consumption and server loads in comparison to other protocols. On the other hand, Cao and Liu (Cao and Liu, 1997) advocate a consistency protocol based on invalidations of outdated cached web documents initiated at the server side.

Yu, Breslay and Shenker (Yu et al., 1999) offer a hierarchical architecture for web caching which is scalable and that uses an invalidation approach. Yin *et al* (Yin et al., 1999) developed a different technique by offering object *leases*. If a client p request an object X from a server q then q grants a lease to this request for X. The lease duration denotes the interval of time during which the server agrees to notify the client if the object is modified. After the lease expires, it must be renewed. Volume leases groups some objects with the same lease. Based on this idea, Duvvuri, Shenoy and Tewari (Duvvuri et al., 2000) developed *adaptive leases* that improves the efficiency by adapting the lease duration considering some characteristics of the network.

Basis token consistency, developed by Bradley and Bestavros (Bradley and Bestavros, 2002) is a mechanism that attach a header to each object in order to keep consistency in the cache. Finally, Tewari, Niranjan and Ramamurthy (Tewari et al., 2002) developed *WCDP*, a protocol where several levels of time consistency are permitted.

4 CONSISTENCY MODELS

Although web applications have very different requirements than other areas of distributed computing, there is still room to introduce some ideas born in distributed objects consistency protocols. In this section, basic notions for consistency in distributed systems, including time and order considerations, are presented.

4.1 Order Models

A distributed system consists of *N* user processes and a distributed data storage. Because of caching and replication, several, possibly different, copies of the same data objects might coexist at different sites of the system. Thus, a consistency model, understood as a contract between processes and the data storage, must be provided. There are multiple consistency models (Ahamad et al., 1995; Ahamad and Raynal, 2003; Herlihy and Wing, 1990; Lamport, 1978; Torres-Rojas et al., 1998; Torres-Rojas et al., 1999).

The *global history* $\mathcal{H}$ of this system is the partially ordered set of all operations occurring at all sites. $\mathcal{H}_i$ is the total ordered set or sequence of operations that are executed on site *i*. If **a** occurs before **b** in $\mathcal{H}_i$ we say that **a** precedes **b** in *program order*, and denote this as **a** $<_{\texttt{PROG}}$ **b**. In order to simplify, we assume that all operations are either **read** or **write**, that each value written is unique, and that all the objects have an initial value of zero. These operations take a finite, non-zero time to execute, so there is a time elapsed from the instant when a **read** or **write** “starts” to the moment when such operation “finishes”. Nevertheless, for the purposes of this paper, we associate an instant to each operation, called the *effective time* of the operation. We will say that **a** is *executed* at time *t* if the effective time of **a** is *t*. If **a** has an effective time previous to the effective time of **b** we denote this as **a** $<_{\texttt{E-T}}$ **b**. Let $\mathcal{H}_{i+w}$ be the set of all the operations in $\mathcal{H}_i$ plus all the **write** operations in $\mathcal{H}$. The partially ordered *happens-before* relationship “$\rightarrow$” for message passing systems as defined in (Lamport, 1978) can be modified to order the operations of $\mathcal{H}$. Let **a**,**b** and **c** $\in \mathcal{H}$, we say that **a** $\rightarrow$ **b**, i.e., **a** happens-before (or *causally precedes*) **b**, if one of the following holds:

1. **a** and **b** are executed on the same site and **a** is executed before **b**.
2. **b** reads an object value written by **a**.
3. **a** $\rightarrow$ **c** and **c** $\rightarrow$ **b**.

Two distinct operations **a** and **b** are *concurrent* if none of these conditions hold between them.

If $\mathcal{D}$ is a set of operations, then a *serialization* of $\mathcal{D}$ is a linear sequence S containing exactly all the operations of $\mathcal{D}$ such that each **read** operation to a particular object returns the value written by the most recent (in the order of S) **write** operation to the same object. If $\prec$ is an arbitrary partially ordered relation over $\mathcal{D}$, we say that serialization S *respects* $\prec$ if $\forall$ **a**, **b** $\in \mathcal{D}$ such that **a** $\prec$ **b** then **a** precedes **b** in S.

Intuitively, one would like that any **read** on a data item **X** returns a value corresponding to the results of the most recent **write** on X. In some systems this could mean that after making an update, all other processes may be notified about the change as soon as it is required. Assuming the existence of absolute global time, this behavior can be modeled with *linearizability* (Herlihy and Wing, 1990):

Definition 1 *History* $\mathcal{H}$ *satisfies* Linearizability (**LIN**) *if there is a serialization* S *of* $\mathcal{H}$ *that respects the order* $<_{\texttt{E-T}}$ (Herlihy and Wing, 1990).

A weaker, but more efficient, model of consistency is *sequential consistency* as defined by Lamport in (Lamport, 1978):

Definition 2 *History* $\mathcal{H}$ *satisfies* Sequential Consistency (**SC**) *if there is a serialization* S *of* $\mathcal{H}$ *that respects the order* $<_{\texttt{PROG}}$ *for every site in the system* (Lamport, 1978).

SC does not guarantee that a **read** operation returns the most recent value with respect to real-time, but just that the result of any execution is the same as if the operations of all sites were executed in some sequential order, and the operations of each individual site appear in this sequence in the order specified by its program.

An even weaker model of consistency is *causal consistency* (Ahamad et al., 1995).

Definition 3 *History* $\mathcal{H}$ *satisfies* Causal Consistency (**CC**) *if for each site i there is a serialization* S_i *of the set* $\mathcal{H}_{i+w}$ *that respects causal order "$\rightarrow$"* (Ahamad et al., 1995).

Thus, if **a**,**b** and **c** $\in \mathcal{H}$ are such that **a** writes value **v** in object X, **c** reads the same value **v** from object X, and **b** writes value **v'** into object X, it is never the case that **a** $\rightarrow$ **b** $\rightarrow$ **c**. **CC** requires that all causally related operations be seen in the same order by all sites, while different sites could perceive concurrent operations in different orders.

CC is a model of consistency weaker than **SC**, but it can be implemented efficiently (Ahamad et al., 1995; Torres-Rojas et al., 1998). Such implementation requires keeping track of which processes have seen which **write** events. In fact, there is dependency graph for determining which operation is dependent on which other operations. So, this data structure must be built and maintained. For fulfilling this need, vector clocks (Torres-Rojas et al., 1998) can be used.

4.2 Time Models

In neither **SC** nor **CC** real-time is explicitly captured, i.e., in the serializations of $\mathcal{H}$ or $\mathcal{H}_{i+w}$ operations may appear out of order in relation to their effective times. In **CC**, each site can see concurrent **write** operations in different orders. On the other hand, **LIN** requires that the operations be observed in their real-time ordering. Ordering and time are two different aspects of consistency. One avoids conflicts between operations, the other addresses how quickly the effects of an operation are perceived by the rest of the system (Ahamad and Raynal, 2003).

Timed consistency (**TC**) as proposed in (Torres-Rojas et al., 1999) requires that if the effective time of a **write** is t, the value written by this operation must be visible to all sites in the distributed system by time $t + \Delta$, where Δ is a parameter of the execution. It can be seen that when $\Delta = 0$, then **TC** becomes **LIN**. So, **TC** can be considered as a generalization or weakening of **LIN**.

In *timed* models, the set of values that a **read** may return is restricted by the amount of time that has elapsed since the preceding **writes**. A **read** occurs *on time* if it does not return stale values when there are more recent values that have been available for more than Δ units of time. This definition depends on the properties of the underlying clock used to assign timestamps to the operations in the execution. Let T(**a**) be the real-time instant corresponding to the effective time of operation **a**.

Definition 4 *Let* $\mathcal{D} \subseteq \mathcal{H}$ *be a set of operations and* S *a serialization of* $\mathcal{D}$. *Let* **w**, **r** $\in \mathcal{D}$ *be such that* **w** *writes a value into object* X *that is later read by* **r**, *i.e.,* **w** *is the closest* **write** *operation into object* X *that appears to the left of* **r** *in serialization* S. *We define the set* $\mathcal{W}_{\mathbf{r}}$, *associated with* **r**, *as:* $\mathcal{W}_{\mathbf{r}}=\{\mathbf{w'} \in \mathcal{D} \mid (\mathbf{w'}$ *writes a value into object* X$) \wedge (T(\mathbf{w}) < T(\mathbf{w'}) < T(\mathbf{r}) - \Delta)\}$. *We say that operation* **r** ***occurs*** *or* ***reads on time*** *in serialization* S, *if* $\mathcal{W}_{\mathbf{r}} = \varnothing$. S *is* **timed** *if every* **read** *operation in* S *occurs on time.*

Definition 5 *Let* **a**, **b** $\in \mathcal{D} \subseteq \mathcal{H}$ *with effective times* t_1 *and* t_2, *respectively, be two operations over the same object* X. *We say that* **a** $<_\Delta$ **b** *if:*

1. *Both* **a** *and* **b** *are* **write** *operations and* $t_1 < t_2$, *or*
2. **a** *is a* **write** *operation,* **b** *is a* **read** *operation and* $t_1 < (t_2 - \Delta)$.

Definition 6 *History* $\mathcal{H}$ *satisfies* Timed Consistency (**TC**) *if there is a serialization* S *of* $\mathcal{H}$ *that respects the partial order* $<_\Delta$ (Torres-Rojas et al., 1999).

Now, we combine the requirements of well-known consistency models such as **SC** and **CC** with the requirement of reading on time.

Definition 7 *History $\mathcal{H}$ satisfies* Timed Sequential Consistency (**TSC**) *if there is a serialization S of $\mathcal{H}$ that simultaneously respects the partial order $<_{PROG}$ and the partial order $<_\Delta$* (Torres-Rojas et al., 1999).

Definition 8 *History $\mathcal{H}$ satisfies* Timed Causal Consistency (**TCC**) *if for each site* i *there is a timed serialization S_i of $\mathcal{H}_{i+w}$ that simultaneously respects causal order $\rightarrow$ and the partial order $<_\Delta$* (Torres-Rojas et al., 1999).

Figure 3 presents the hierarchy of these sets. If an execution satisfies **LIN**, it satisfies **SC** as well, but the contrary is not always true. If a set of operations $\mathcal{D}$ satisfies **LIN**, then it is always possible to produce a serialization S of $\mathcal{D}$ such that all the operations are ordered by the real-time instants when each operation was executed. In turn, S satisfies Definition 4, even for Δ = 0. Then, **LIN** is a case of **TSC** where Δ = 0, and therefore **LIN** $\subset$ **TSC**. It is easy to see that **SC** $\subset$ **CC**.

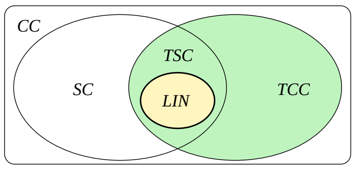

Figure 3: Consistency Criteria Hierarchy

4.3 Lifetime based model

This technique (Torres-Rojas et al., 1998) provides consistency across different but related set of objects.

Let C_i denote the cache of site i, which stores copies of objects that have been accessed recently. If a cache miss occurs when accessing object X, some server provides a copy of its current version of X. Once this copy is stored in C_i, we denote it as X_i. The *start time* of X_i, denoted as ${X_i}^\alpha$ is the time when the value of X_i was written. The latest time when the value stored in X_i is known to be valid is its *ending time* and it is denoted as ${X_i}^\omega$. The interval $[{X_i}^\alpha, {X_i}^\omega]$ is the currently known *lifetime* of the value stored in X_i.

The values of X_i and Y_i (cached in C_i) are mutually consistent if $max({X_i}^\alpha, {Y_i}^\alpha) \leq min({X_i}^\omega, {Y_i}^\omega)$, i.e., their lifetimes overlap and, thus, they coexisted at some instant. C_i is consistent if the *maximum* start time of any object value in C_i is less than or equal to the *minimum* ending time of any object value in C_i, i.e., every pair of object in C_i is mutually consistent.

In general, the lifetime of arbitrary object values is not known. When site i updates object version X_i at time t, timestamp t is assigned to both ${X_i}^\alpha$ and ${X_i}^\omega$. It must be discovered as it goes that no object copy X_j ($i \neq j$) has been overwritten and use this information to advance ${X_i}^\omega$.

A local timestamp variable called **Context**$_i$ is associated with C_i. Its initial value is 0, and it is updated with the rules:

1. When a copy of object X is brought into C_i (becoming X_i): **Context**$_i$ $:= max({X_i}^\alpha,$ **Context**$_i)$
2. When object copy X_i is updated at time t: **Context**$_i$ $:= {X_i}^\alpha := t$

Context$_i$ keep the latest start time of any object value that is or has been stored in C_i. When a copy of object X is brought into C_i, its ending time must not be less than **Context**$_i$, if necessary, other servers or client sites are contacted until a version of X that satisfies the condition is found. Furthermore, any object $Y_i \in C_i$ such that ${Y_i}^\omega <$ **Context**$_i$ is invalidated. It is proved in (Torres-Rojas et al., 1998) that this protocol induces **SC** on the execution.

5 PROPOSED WORK

This proposal attempts to modify an existing web cache software in order to implement an architecture able to consider several consistency models by introducing timing and ordering requirements. It emerges as part of the SPREAD research project, which is under development in the Computing Research Center at Costa Rica Institute of Technology. SPREAD is oriented in applying consistency techniques and properties to areas as collaborative software, mobile computing, distributed databases, web caching and others.

We selected the Wessels *et al* SQUID software for several reasons. SQUID is one of the most widely used caching solutions deployed in several huge networks. It supports a simple yet hierarchical cache protocol allowing a testbed for research (for example (Dilley et al., 1999; Wessels and Claffy, 1998; Shim et al., 1999; Bradley and Bestavros, 2002; Duvvuri et al., 2000)). It is Open Source, distributed under GNU-GPL terms, meaning access to source code and to a large community of programmers maintaining this project.

Regarding consistency protocols of section 2, **expiration** approach is achieved via adaptive time-to-live or ATTL (Cate, 1992; Gwertzman and Seltzer, 1996). The ATTL, where the life-span of an object is approximated to better reflect its behavior in time, is implemented in most browser level caches and in proxy cache servers, including SQUID, for it is easy to implement using HTTP headers. **Polling** consistency level, also handled in SQUID, is easily implemented by issuing conditional HTTP GET commands with the *If-Modified-Since* header field, here the server is expected to respond mainly with the "*OK*" code (**200**) with the updated object (when changes were made) or the "*Not Modified*" code (**304**) when object data is still valid. Recent SQUID versions support the HTTP/1.1 specification introduces enhanced directives for caching ("*Cache-Control*" directive): age and expiration calculation, freshness requirements, stale control, cache extensions, cacheable objects, etc. Although the **invalidation** approach is not inherent to SQUID, several invalidation alternatives (Bradley and Bestavros, 2002; Duvvuri et al., 2000) have been implemented.

All these three approaches induce some level of consistency, but they take into account only the time axis. Our objective in this proposal is to introduce in SQUID the lifetime based consistency of section 4, which can be modeled to offer several levels of consistency, using the order of events. These levels can be selected among: sequential consistency (**SC**), coherence, causal consistency (**CC**) and no ordering at all.

Finally, the evaluation criteria to be used is not increasing SQUID performance, but permitting several levels of consistency to be applied dynamically according to web usage requirements.

6 CONCLUSIONS

Several strategies for maintaining cache consistency have been proposed, but they do not consider *order* and *time*, so it is difficult to analyze the properties such strategies offer.

Different web applications require different web cache consistency protocols. More agile web environments need stronger consistency models, while weaker models of consistency are sufficient for less changing systems.

Our attempt is to build a web caching architecture where several consistency models can be used in order to satisfy the needs for different web environments.

REFERENCES

Ahamad, M., Neiger, G., Burns, J. E., Kohli, P., and Hutto, P. W. (1995). Causal memory: definitions, implementation and programming. *Distributed Computing*, 9(1):37–49.

Ahamad, M. and Raynal, M. (2003). Ordering and timeliness: Two facets of consistency? In *Future Directions in Distributed Computing*, Lecture Notes in Computer Science, pages 73–80.

Berners-Lee, T., Fielding, R., and Frystyk, H. (1996). Hypertext transfer protocol – HTTP/ 1.0. RFC 1945, Internet Engineering Task Force. http://www.ietf.orf/rfc/rfc1945.txt.

Bradley, A. and Bestavros, A. (2002). Basis token consistency: Extending and evaluating a novel web consistency algorithm. *Proceedings of the Second Workshop on Caching, Coherence, and Consistency (WC3 '02)*.

Cao, P. and Liu, C. (1997). Maintaining strong cache consistency in the world wide web. *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97)*.

Cate, V. (1992). Alex - a global file system. *Proceedings of the 1992 USENIX File System Workshop*, pages 1–11.

Dilley, J., Arlitt, M., and Perret, S. (1999). Enhancement and validation of the Squid cache replacement policy. *Proceedings of the 4th International Web Caching Workshop*.

Duvvuri, V., Shenoy, P., and Tewari, R. (2000). Adaptive leases: A strong consistency mechanism for the world wide web. *Proceedings of INFOCOM 2000*.

Gwertzman, J. and Seltzer, M. (1996). World-wide web cache consistency. *Proceedings of USENIX Annual Technical Conference*.

Herlihy, M. and Wing, J. (1990). Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492.

Kawash, J. (2004). Consistency models for internet caching. In *WISICT '04: Proceedings of the Winter International Symposium on Information and Communication Technologies*, ACM International Conference Proceedings Series, pages 1–6.

Lamport, L. (1978). How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computer Systems*, 28(9).

Mikhailov, M. and Wills, C. (2003). Evaluating a new approach to strong web cache consistency with snapshots of collected content. *WWW '03: Proceedings of the twelfth international conference on World Wide Web*, pages 599–608.

Rabinovich, M. and Spatscheck, O. (2002). *WEB caching and replication*. Addison-Wesley.

Shim, J., Scheuermann, P., and Vingraleki, R. (1999). Proxy cache algorithms: Design, implementation, and performance. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):549–562.

Tewari, R., Niranjan, T., and Ramamurthy, S. (2002). WCDP: A protocol for web cache consistency. *Proceedings of the 7th International Workshop on Web Content Caching and Distribution*.

Torres-Rojas, F. J., Ahamad, M., and Raynal, M. (1998). Lifetime based consistency protocols for distributed objects. In *Proceedings of the 12th International Symposium on Distributed Computing, DISC'98*, pages 378–392.

Torres-Rojas, F. J., Ahamad, M., and Raynal, M. (1999). Timed consistency for shared distributed objects. In

Proceedings of the Annual ACM Symposium on Principles of Distributed Computing PODC'99, pages 163–172.

Wang, J. (1999). A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communications Review*, 29(5):36–46.

Wessels, D. and Claffy, K. (1998). ICP and the Squid Web cache. *IEEE Journal on Selected Areas in Communication*, 16(3):345–357.

Yin, J., Alvisi, L., Dahlin, M., and Lin, C. (1999). Volume leases for consistency in large-scale systems. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):563–576.

Yu, H., Breslau, L., and Shenker, S. (1999). A scalable web cache consistency architecture. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 163–174.