A Workgroup Model for Smart Pushing and Pulling

Gail Kaiser, Christopher Vaill and Stephen Dossick
Columbia University
Department of Computer Science
New York, NY 10027
212-939-7100/fax:212-939-7084
psl@cs.columbia.edu
http://www.psl.cs.columbia.edu

Abstract
*Our **Workgroup Cache** system operates as a virtual intranet, introducing a shared cache to members of the same workgroup. Users may be members of multiple workgroups at the same time. Criteria are associated with each workgroup to pull documents from an individual cache to the shared cache, or push from the shared cache to an individual cache. These criteria provide semantics of the workgroup's tasks and interests to reduce latency for its members.*

1. Introduction

Caching reduces average access latency, from registers and memory pages cached by hardware, to the application level such as a web browser retaining retrieved documents. We focus here on potentially shared networked documents and define two terms in relation to this type of caching: *Zero latency* refers to the condition where access to a document produces a cache hit on the local machine, that is, there is little or no latency due to the network (we assume that latency due to local disk and memory access is insignificant in comparison to network latency). A document with zero latency is usually saved in the cache after an explicit access, or is pulled there through some prefetching mechanism. *Negative latency* refers to automatic presentation, or push, of a document to a user based on a prediction that the user will want that document. With an ideal system, a user would be presented with documents either that she was about to request, or that she would not know to request but that would be immediately useful to her.

We also distinguish between individual and shared caching. An individual cache, such as built into a web browser, stores documents for one user for use by that same user. A shared cache, such as a web proxy cache, stores documents accessed by many users, and can provide a document to a user with relatively small latency even if that particular user has never accessed the document before. If that shared cache had pushed the document to appropriate individual caches, then those users can enjoy zero latency if they happen to later access that document.

The central problem in caching normally lies in the cache replacement algorithm, typically LRU, where the trivial cache fill algorithm is to add every newly accessed document to the cache. Simple algorithms like these are routinely used when nothing is known about the semantic content of the accesses or the tasks the user will perform utilizing those accesses, so predictions about the future must necessarily be rather generic. However, in order to achieve zero latency in a shared cache system, we must take a broader view - and devise more sophisticated algorithms for both cache fill (pull) and replacement (save). Negative latency also needs a compatible algorithm for cache-based recommendation (push).

Our Workgroup Cache system can (in principle) leverage any knowledge available about the semantic content and pragmatic usage of documents as a basis for prediction of future accesses. We focus the potential semantics and pragmatics with respect to a "workgroup", here a set of users working on the same task or related tasks. Workgroup membership can be determined in a number of ways: The users can be specified in advance, such as a software development team working closely together (although they might be physically dispersed); or determined by dynamically including users whose document accesses match patterns associated with the workgroup, such as amateur programmers actively working on the same subsystem of an open-source project like Linux. Or any other method that groups users according to the task(s) they are working on or likely to work on in the near future – such as inferring a Web community from link topologies [1].

Various cache fill and replacement criteria, as well as recommendation criteria (to serve documents with negative latency), may be defined separately and then associated on the fly with a given workgroup. Although obviously any run-time realization of this Workgroup Cache model might be limited in what criteria can be applied, the criteria are intended to be fully configurable and our proof-of-concept implementation supports dynamic "plug-in" of criteria objects. Criteria might be based on software process or workflow routing among workgroup members, document access patterns of workgroup members (e.g., if my supervisor keeps returning to such and such technical report then I want to read it too), or with XML metadata associated with or embedded in accessed documents. Criteria might be defined via simple filter rules, like Cisco firewalls or Web search engine queries, or via a very elaborate event/data pattern notation.

Many large software development projects, particularly in the growing open-source software community [2], rely on the coordination of multiple developers distributed over a worldwide geographic area. Numerous project coordination difficulties can arise due to the relatively little contact the developers may have with each other, they might never even have met! A system for recommending documents (such as source files) to these developers based on the nature of their present tasks and the document content might help ease some of these difficulties.

For example, say a primary developer is making major changes to one file in a module, and a secondary developer attempts to start adding a small feature to another file in the same module. If both developers are already or automatically become (due to their apparently shared interests) members of the same workgroup, and are using Workgroup Cache powered by semantics drawn from software configuration management, the system would "know" about the dependencies between the two files, and could push (recommend) the changing source file to the secondary developer. Then she could check if the code she was adding would be incompatible with the likely new version of the module. Or pull criteria might continuously update the feature self-assignments records, to also inform that another secondary developer had already started adding a similar feature, preventing duplication of effort.

2. Related Work

Cache prefetching as a means of performance improvement is a familiar subject in the fields of computer architecture, operating systems and compilers. McIntosh [3] describes methods for a compiler to insert instructions to prefetch data that is needed later in the code. Of course, all the code to be executed in the future can be seen when the prefetches are inserted, so the prediction can be based on a concrete analysis of what data will be needed. This prediction is similar to the workflow analysis used (optionally) by Workgroup Cache. However, a compiler inserting prefetching instructions has no notion of configurable prefetch criteria based on functional groups, i.e., prefetch criteria that change depending on the type of program being compiled (or in the Workgroup Cache case, the tasks to be performed).

Prefetching of data for performance enhancement has been applied to streaming multimedia applications, but this is often limited to special-purpose buffering. For example, the **Nemesis** system [4] must only predict how much data will be needed from a single source, depending on current frame rates. There is one data source, and one recipient.

Speculative cache prefetching has also been explored for improving access times in low and intermittent bandwidth networks. Tuah *et al.* [5] conduct a quantitative investigation into access time improvements gained by a prefetching model. They classify multiple prefetches as either *mainline* prefetches, which are prefetches of documents that are most likely to follow one another, or *branch* prefetches, which are prefetches of a number of alternative documents, any one of which may be accessed in the future. Branch prefetching can be expected to produce better access time than mainline prefetching, but at a higher retrieval cost, i.e., greater bandwidth devoted to retrieving documents that may never be accessed. Tuah *et al.* concludes that an ideal prefetcher would adapt its strategy depending on available resources and target performance.

Caubweb [6] prefetches subsets of the WWW document space, called ``weblets'', while the user is still connected, for the user to browse after network disconnection. Prefetching is configured and executed explicitly by the user - document parameters and starting document are set, and Caubweb follows hyperlinks and caches all linked information conforming to the given parameters. The document parameters include pattern matching on the URL or hyperlink text, depth from the start document, and file properties such as size and MIME type. This system basically requires a user to know and specify precisely which documents she will need after disconnection; no "knowledge" of access patterns or tasks is employed.

A variety of large scale, typically hierarchical shared cache systems have been developed for WWW, e.g., **Harvest** [7], to enable users to obtain documents from a “nearby” proxy cache rather than a distant server. Tewari *et al.* [8] studied such architectures’ performances on three “benchmark” trace workloads to arrive at these design principles:

- Share data among many clients
- Minimize the number of hops to locate data on hits or misses
- Cache data close to clients.

They found that most deployed and proposed architectures violate some of these principles, reducing the miss rate in preference to reducing hit time and miss time. They promote a new variant of the "hint cache" data-location metadata hierarchy, which is then exploited by a "push cache" mechanism. Their simulations show 1.27 to 2.43 times speedup compared to other large scale cache architectures. The results are expected to be better when clients of a leaf cache are grouped intelligently, e.g., according to workgroups. However, Tewari *et al*. still base predictions of future accesses primarily on past patterns, and do not prefetch from primary sources but only from other pushcashes in the system. Their open source **Cuttlefish** implementation is now available at **PushCache.com**. It is possible that their **PushAp Kit** product could be used to implement our Workgroup cache architecture on top of Cuttlefish.

The **Coda** filesystem [9] includes extensive provisions for caching files on a client for use when disconnected from the server; indeed one of the main features of the Coda system is improved accessibility via caching, or "hoarding" as Coda calls it. A Coda client caches files periodically or at user request, using recent file accesses in concert with a hoard profile, which is a configuration created by the user to specify which files are to be cached, and what the hoard priorities of those files are. This approach is more advanced than Caubweb's, but still requires the user to specify what is to be cached, using a simple decaying priority algorithm to cache files that are accessed but not specified in a hoard profile, and cannot take into account the file accesses of other same-workgroup users when determining hoard priority.

In our previous work on **Laputa** [10], prefetching was also proposed to support network disconnection. We identified three types of criteria for determining what documents to prefetch: manual, heuristic and process-based. While manual and heuristic methods were seen in Coda's "hoarding", the process-based method was new. Since Laputa was meant for disconnected software development, information about software processes would be used to determine what documents to fetch. Laputa might fetch all documents necessary for the completion of a selected task, plus documents necessary for tasks expected to soon follow the current task in the process. Workgroup Cache similarly considers workflow semantics to predict future data need, but extends beyond Laputa by including the work processes of multiple users, i.e., multiple participants in the workflow, in its document prefetch criteria.

The **Remembrance Agent** [11] augments human memory by displaying a list of documents that might be relevant to the user's current context. Unlike most information retrieval systems, the Remembrance Agent runs continuously without user intervention. Its unobtrusive interface allows a user to pursue or ignore its suggestions as desired. The implementation currently available for the **emacs** text editor continuously watches what the user types and reads, and finds old email, notes files, and on-line documents apparently relevant to the user's context. One-line suggestions at the bottom of the display buffer, along with a numeric rating indicating how relevant it thinks the document is. The frequency with which the front end provides new suggestions, the number of suggestions, and whether to look at text notes files, old email, or other document sources is customizable for each user. An extension to a workgroup Remembrance Agent is briefly postulated, but apparently has not yet been pursued. Instead a "wearable" version is being investigated [12]. A group-oriented "wearable" is, however, under development in the **Factoid** project at Compaq (http://www.research.digital.com/wrl).

Group-oriented recommender systems are familiar to anyone who makes purchases via the World Wide Web. Retailers such as **Amazon.com** and **CDNow.com** use such tools to suggest future purchases to customers based on their history of previous purchases. Of course, the algorithms used in these two cases are proprietary, so we can only guess at how they actually work, but Amazon states that buying patterns of other customers are used in the determination of recommendations. The criteria for recommendation, then, are based on data from a functional group, but that group is universal; apparently, it contains all Amazon customers. One result of this "universal group" strategy is that specialized recommendations can be given only based on per-customer data; recommendations arising from other customers' buying patterns may not be very useful.

CDnow's system recommends albums bought by other customers with buying patterns similar to the current user's. This allows for some specialization based on the automatic clustering of users by music preferences. The criteria for recommendation in this case are attached to an implicit group of sorts, the group of customers with "similar buying patterns". This group is defined only vaguely, however, and again in practice does not seem very useful.

The **Alexa** package (http://www.alexa.com) is a kind of recommender system for the Web, in the form of "related sites". In newer versions (4.06 or later) of **Netscape Communicator**, Alexa's recommendations are presented through the "What's Related" control in the location toolbar. Again, the system is proprietary, but Alexa makes several statements hinting at how the system works. The **Internet Explorer** version watches its users' browsing behavior, such as links followed and time spent at a site, and uses this information, collected from all Alexa users, to infer relations between sites. These related sites are then listed. Alexa's recommendations are similar to those of Amazon in that they are based on data collected from an effectively universal set of users, although the universal group for Alexa is apparently much larger, including statistics mined from browsing patterns of all Web users. As far as we know, Alexa has no notion of groups to which users belong, in order to further specialize recommendations.

Fab [13] also recommends Web sites, but bases its recommendations on a personal profile that becomes adapted to the individual user over time. The user's recommendations are initially random, but the user can manually rate the pages recommended to her. These ratings then are used to alter the user's profile, so that the next group of sites recommended to that user are more personalized. There is also a provision for "parasitic" users: users that do not create profiles of their own but use the profiles of others to get recommendations. The group model lies at the opposite extreme from the ones we have seen so far. Instead of a universal set of users, the set contains only one user, and recommendations are based on the profile of that one user. This allows for very specialized recommendations, but is not applicable to recommendations as needed in a collaborative environment of shared media.

Ant World [14] uses what it calls "digital information pheromones" to produce website recommendations. The Ant World system appears at the top of the user's browser while the user performs a search on the Web. After each link is traversed, the user must rate the link's usefulness to her search. This intrusive interaction enables the server to build up a database of weighted graphs representing the paths taken by users during their searches. These paths are meant to be analogous to the pheromone trail left by ants searching for food. If a user happens upon a path that was previously rated strongly by another user, the next recommended link in this strong path is marked. Thus, information from other users is used to recommend Web search paths, but again, the set of users from which this information is obtained is universal **--** there is no way to specify what other users are likely to be most relevant to one's own search.

The **ReferralWeb** [15] recommender system explores and maps the user's "social network", and an expert in a requested subject can potentially be found and recommended (referred). The social network is used to find experts socially "near" to the user, since a closer node in the network is more likely to share common interests and is therefore more likely to respond usefully to questions about the requested subject. This project focuses mainly on the construction of the social network, which is another type of implicitly defined workgroup. In the ReferralWeb study, co-authorship on a paper was considered an association between users, so the large bibliography databases already in existence could be used to construct the network. Central to the project's viability is the idea that information from those in a group with whom a user associates is more likely than otherwise to be relevant to that user.

3. Design and Architecture

The Workgroup Cache design is based on a model of small distributed components interacting with each other over a network. There are three main components:

Client: This includes the user-client program itself, plus a Workgroup Cache interface to provide the user with controls. The client typically includes facilities for pulling (requesting) documents directly from the network, and may also support direct pushing (recommendation), but neither is mandatory. Legacy clients are supported on a necessarily ad hoc basis through wrapping, applets, etc.

Personal Cache Module (PCM): This component is associated with a single client, and automatically saves all documents received by the client (until the cache is full, when prioritization criteria are invoked to determine replacement). PCM handles criteria for pushing documents to the shared cache and receives documents pushed to it by the shared cache. It also decides whether to present such pushed documents immediately to the user, and optionally also supports pull criteria. Each PCM may be connected to any number of shared cache modules.

Shared Cache Module (SCM): This component is associated with a workgroup and contains criteria for sending and receiving documents to and from the personal caches of users in that workgroup. This module provides the major caching functionality of the system, and is the main center for task-based document pull (e.g., to retrieve documents likely to be needed "soon" for the in-progress workflow or otherwise predefined task).

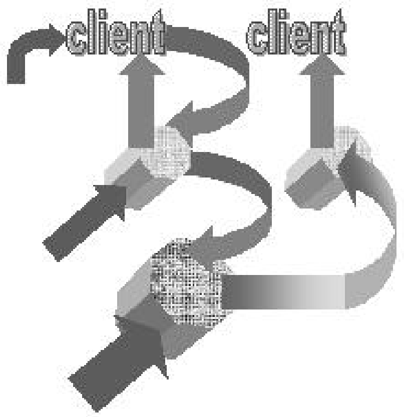

Figure 1. Architecture

The client, PCM and SCM components interact as shown in figure 1. The client can push documents to its PCM, and pull documents from its PCM or from an outside source (for example, the World Wide Web). The central facet of the client component is its cache control interface to the PCM. The criteria for the client to execute any of its interactions with PCM are completely under the user's control. This is meant in part to protect the security and privacy of the user, regarding accessed documents, as well as to permit avoidance of potentially annoying popup pushes if that user prefers. The user cache controls allow the user to specify to which workgroups any documents accessed by that client are relevant, if any, and to accept or decline inclusion in automatically configured workgroups. The user can also turn off caching entirely or temporarily, if, for example, when accessing sensitive documents that should not be cached publicly. The user controls could also provide configuration options for handling this automatically, by, for instance, turning off caching automatically for any document accessed via SSL. The client controls can also be used for manual recommendation of documents: if a user has accessed a document that she thinks might be interesting to her co-workers, she can push it to the SCM, where everyone in the workgroup could then access it.

Actions taken by the SCM are not based on direct per-user controls, but rather on configurable criteria, represented essentially as rules consisting of condition-action pairs. The conditions must be highly flexible, to allow as much freedom as possible in customizing the behavior of the system to fit the behavior of the workgroup. Possible conditions (that might lead to smart push, pull or save actions) could include frequent accesses to a document by several members of the workgroup, as determined by analyzing access history logs, explicit recommendations from more than one user or from one high-priority user, or any other detectable condition that may indicate a document's importance (or non-importance - e.g., a condition may, if satisfied, cause a document to, say, be discarded from the cache). If a workflow system is attached to an SCM component, a condition-action pair could specify that if the first document in a workflow is accessed by a user, the next document or documents in the flow should be pushed to that user's PCM - or to another user's PCM if the workflow specifies routing to another user in the workgroup. (The workflow system must supply an API suitable for implementing a conduit from the workflow engine to the SCM.)

The PCM component's actions include pushing and pulling documents to/from the client and to/from any associated SCM. It can also save documents in its own cache, if it has one. (If the PCM does not have its own cache, it exists solely for its pushing and pulling abilities, in contrast to the SCM, which must always have its own cache to be shared among clients in the workgroup.) The purpose of the PCM is to allow for some criteria to be configured on a per-user basis. A user who is working especially closely with another user in the workgroup may wish to be informed of all document accessed by that second user. That user's PCM would then be configured to pull from the SCM and push to the client any document accessed by the other user. (In principle, any pair of PCM's with their own caches could interact directly, forming their own mini-workgroup and by-passing the SCM's criteria and its cache, but we do not intend to support this initially due to the relatively higher complexity.)

The normal flow of document access generally works as follows:

1. The user requests a document via the client program.
2. The client component tries to pull the document from its PCM.
3. The PCM looks for the document in its cache. If it exists there, that copy is returned to the client, and a record of the access is passed to the SCM. If not, the PCM tries to pull the document from the SCM. In any case, all the criteria rules are checked, and actions are performed for any conditions that are satisfied by the new access.
4. The SCM looks for the document in its cache. If it exists there, that copy is returned, and the access is recorded in the history. If not, either the client or the SCM (determined by the client's controls) retrieves the document from its outside source; when SCM performs the outside pull, its criteria rules are then checked and any applicable actions are performed.

Because each client is associated with exactly one PCM and vice versa, it would be possible to collapse the two components into one. The PCM is separated from the client component in the Workgroup Cache design, however, in order to create as little disturbance as possible to the normal operation of the client. It can run on a separate machine (presumably on the same fast local network as the client) while the client needs only a small amount of user interface and PCM-interface code, ideally as unobtrusive as a Java applet.

4. Realization

Our proof-of-concept implementation of Workgroup Cache is a collection of Java applications, based largely around a middleware framework called the Groupspace Controller ([16] describes **GC**'s predecessor but most aspects still apply). GC provides a flexible and robust environment that ties together various parts of a Workgroup Cache component, and includes built-in support for event publish/subscribe, including "request" as well as conventional "notification" style events, and for communication between GCs on different machines.

The client we have chosen to focus on initially is the University of California at Irvine's **Chimera** open hypermedia system [17]. Chimera includes a linkbase that represents external (to the media) N-ary hyperlinks among diverse types of documents, such as GIF images and FrameMaker documents, whose "viewers" interface to Chimera through its API. We plan to employ Workgroup Cache ourselves for our own software development, where Chimera will store links between source code files and other types of media such as design documents and email archives, as well as among source code files (e.g., identifier definitions and uses). Thus Chimera, or actually its viewers, makes for ideal end-user clients for applying Workgroup Cache to distributed software development projects (although our own development is admittedly not distributed any further than the various participants' homes in addition to the Columbia campus).

The implementation of the client component required small alterations to the Chimera code to interface with GC, and hence with the PCM. When the user follows a link in Chimera, the code inserted into Chimera's native event handling mechanism intercepts that Chimera event and translates it into a GC event, and fires the event in the controller. The PCM interface picks up that event, and requests the document from the PCM. The added Chimera interface code also provides the user with the cache controls.

The PCM component is implemented as another GC with a personal cache controller service and a cache interface service. The personal cache controller handles all the local cache criteria, and does the sending and receiving of access requests. The cache interface is a simple implementation of the Internet Cache Protocol (**ICP** [18]), so any external web proxy cache that speaks ICP (we're using **Squid**, see http://squid.nlanr.net) can be used to handle the actual storing and retrieval of files.

SCM also uses GC to enable its core service, which we call a Workgroup object, to interact with the cache and (optional) workflow system interface(s). The Workgroup object manages the criteria. Multiple Workgroup objects (effectively multiple SCMs for distinct workgroups) can be connected to a single GC, if desired, to share the same cache and workflow system interfaces. The cache interface is an ICP interface as above. The workflow interface, still in the design stages, is planned to work with **IBM's MQSeries Workflow** product.

We anticipate that our Workgroup Cache model and architecture, if not necessarily our early prototype, will prove useful for intelligently sharing information among distributed software developers. However, this work is in progress, as befitting a workshop submission, and empirical data on practical usage of the system is still to come.

Acknowledgements

We thank John Salasin, Scott Gross, Janak Parekh, Dan Port, Adam Stone, Jack Yang and attendees at the 8th WET-ICE for useful technical discussions and feedback.

The Programming Systems Lab is sponsored in part by the Defense Advanced Research Projects Agency, and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-2-0022, and in part by an IBM University Partnership Program Award. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, the Air Force, the U.S. Government or IBM.

Programming Systems Lab software is available at http://www.psl.cs.columbia.edu/.

References

[1] D. Gibson, J. Kleinberg and P. Raghavan. "Inferring Web communities from link topology". *9th ACM Conference on Hypertext and Hypermedia*, June 1998.

[2] Mann, Charles C. "Programs to the People". *Technology Review*, January/February 1999.

[3] McIntosh, Nathaniel. *Compiler Support for Software Prefetching*. Rice University, PhD Thesis, TR98-303, May 1998.

[4] Katseff, Howard P. and Robinson, Bethany S. Predictive "Prefetch in the Nemesis Multimedia Information Service". *2nd ACM International Conference on Multimedia*, 1994.

[5] Tuah, N. J., Kumar, M. and Venkatesh, S. "Investigation of a Prefetch Model for Low Bandwidth Networks". *1st ACM International Workshop on Wireless Mobile Multimedia*, October 1998.

[6] Lo Verso, John R. and Mazer, Murray S. "Caubweb: Detaching the Web with Tcl". *5th Annual USENIX Tcl/Tk Workshop*, July 1997.

[7] Chankhunthod, A. *et al*. "A Hierarchical Internet Object Cache". *USENIX 1996 Annual Technical Conference*, January 1996.

[8] Tewari, Renu *et al.* "Design Considerations for Distributed Caching on the Internet". *19th IEEE International Conference on Distributed Computing Systems*, May 1999.

[9] Kistler, James J. and Satyanarayan, M. "Disconnected Operation in the Coda File System". *Symposium on Operating Systems Principles*, October 1991.

[10] Skopp, Peter D. and Kaiser, Gail E. "Disconnected Operation in a Multi-User Software Development Environment". *IEEE Workshop on Advances in Parallel and Distributed Systems*, October 1993.

[11] Rhodes, Bradely and Starner, Thad. "Remembrance Agent: A continuously running automated information retrieval system". *1st International Conference on The Practical Application of Intelligent Agents and Multi Agent Technology*, April 1996.

[12] Rhodes, Bradley. "The Wearable Remembrance Agent: A system for augmented memory". *Personal Technologies*, 1:218-224 (1997).

[13] Balabanovic, Marko. "An Adaptive Web Page Recommendation Service". *1st International Conference on Autonomous Agents*, February 1997.

[14] Kantor, P.B., Melamed, B. and Boros, E. *A Novel Approach to Information Finding in Networked Environments*. Rutgers University, 1998.

[15] Kautz, H., Selman, B. and Shah, M. "The Hidden Web". *AI Magazine*, 8(2):27-36, Summer 1997.

[16] Kaiser, Gail E. and Dossick, Stephen E. "Workgroup Middleware for Distributed Projects". *IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 1998.

[17] Anderson, Kenneth M., Taylor, Richard N. and Whitehead, E. James Jr. "Chimera: Hypertext for Heterogeneous Software Environments". *European Conference on Hypermedia Technology*, September 1994.

[18] Wessels, D. and Claffy, K. *Internet Cache Protocol (ICP), version 2*. RFC 2186, September 1997.