Clique: A transparent, Peer-to-Peer collaborative file sharing system

Bruno Richard, Donal Mac Nioclais, Denis Chalon
HP Laboratories Grenoble
HPL-2002-307
October 28th , 2002*

E-mail: Bruno.richard@hp.com, donal_mac-nioclais@hp.com, denis.chalon@hp.com

peer-to-peer, groupware, file system, replication, optimistic reconciliation

Clique is a HP Labs Grenoble project. The goal is to develop a novel peer-to-peer, server-less distributed file system based on optimistic replication algorithms, which transparently integrates into users' native file systems. Some properties of the Clique system are epidemic replication, a no lost updates consistency model and conflict management, as well as disconnected operation and replica convergence. These properties ensure that updates done by any peer of the group will never be lost, and also that they will converge on all the group member machines. The system is well adapted to highly disconnected environments, network partitions, and variable join/leave rates. Even under adverse connectivity conditions, over time, assuming intermittent point-to-point connectivity between each peer and at least one other peer in the group, the local file system view at each node converges towards a consistent global view. The reconciliation protocol used is stateless and has no notion of group membership, in order to achieve a linear worst-case scalability in the order of N, the number of peers in the network. A lower layer protocol has been developed, which enables one-to-all communications by taking advantage of IP Multicast augmented with network load management and a priority mechanism ensuring liveness of the higher layers of the protocol.

* Internal Accession Date Only
Approved for External Publication

Clique: A transparent, Peer-to-Peer collaborative file sharing system

Bruno Richard
Bruno.richard@hp.com

Donal Mac Nioclais
donal_mac-nioclais@hp.com

Denis Chalon
denis.chalon@hp.com

hp Laboratories
38053 Grenoble CEDEX 9
France

Abstract

Clique is a HP Labs Grenoble project. The goal is to develop a novel peer-to-peer, server-less distributed file system based on optimistic replication algorithms, which transparently integrates into users' native file systems. Some properties of the Clique system are epidemic replication, a no lost updates consistency model and conflict management, as well as disconnected operation and replica convergence. These properties ensure that updates done by any peer of the group will never be lost, and also that they will converge on all the group member machines. The system is well adapted to highly disconnected environments, network partitions, and variable join/leave rates. Even under adverse connectivity conditions, over time, assuming intermittent point-to-point connectivity between each peer and at least one other peer in the group, the local file system view at each node converges towards a consistent global view. The reconciliation protocol used is stateless and has no notion of group membership, in order to achieve a linear worst-case scalability in the order of N, the number of peers in the network. A lower layer protocol has been developed, which enables one-to-all communications by taking advantage of IP Multicast augmented with network load management and a priority mechanism ensuring liveness of the higher layers of the protocol.

1. Problem statement

The development of Clique was motivated by our perceived need for a new file sharing technology which supports collaboration and synchronization in a simple, transparent, reliable manner.

Scenario 1

In the first case, individuals are beginning to use a range of increasingly powerful computing devices with highly varying connectivity patterns. These range from desktop PCs with a permanent, high bandwidth connection to the Internet via the corporate LAN and firewall, through laptops with 802.11 cards that offer good connectivity at medium speeds, but only within certain limited areas, to powerful PDA devices with low bandwidth and sporadic connectivity. A user, then, will soon have a sort of 'personal network' of intermittently connected clients which will increasingly be differentiated less by storage and processing capacity and more by physical mobility, network connectivity and bandwidth.

The user would increasingly like to be able to access a given portfolio of files on any one of this wide range of devices. He will expect modifications made to any of these files on one device to be automatically reflected on another, even when devices are geographically far apart and connected to physically separate networks. He will want all files to be immediately accessible from all devices, even when operating in disconnected mode. Ideally, the system should scale to an arbitrary number of nodes, of shared files and versions.

Scenario 2

In the second case, users will want to share their portfolio of files with groups of other people on other devices. Each of the users would like to have read access to the group files, but also to be able to modify them, to add new files and delete other ones, and they would like to have their changes shared with other users of the group. Typically, Adam, an HP employee using Clique, might wish to share files between 6 machines:

- A desktop PC connected via the HP intranet to the desktop PC of his team colleagues, Bill and Christiana, and separated from the Internet by a corporate firewall.
- A home 802.11 network of two desktop PCs which are connected to an ISP via an ADSL link and have dynamically assigned IP addresses.
- A laptop on which he works while commuting between work and home. This computer is intermittently connected to both the HP internal network and the home office network.

During work hours, Adam's laptop is connected to the HP intranet. Modifications made to any of the files or sub-directories in the Clique shared directory on one device, for example, Adam's desktop PC, are automatically propagated in the background to other connected members of the Clique group, in this case his laptop and his colleagues' desktop PCs. At the end of the working day, Adam may disconnect his laptop and continue to work on some of the shared files as he commutes. On connecting his laptop to the home network, the modifications he and his colleagues made during the working day are automatically reflected on the home network PCs, and any modifications made, for example, by his wife Diana to the

files on the local network are uploaded to the laptop (and hence may be uploaded to the HP desktop PCs, along with Adam's own modifications, when Adam reconnects to the HP intranet the following morning). As we will show, this epidemic-style replication pattern guarantees that all nodes in the Clique group will eventually achieve a consistent view of the file system state.

For maximum performance in the mobile environment, Clique employs a weakly consistent update policy, which does not place any limits on file modification permissions even when network access is unavailable. This introduces the likelihood of update conflicts, whereby multiple users independently modify a particular file. In this case, multiple distinct, but equally valid, versions of the file temporarily exist in the system. Current groupware solutions often blindly overwrite some of these versions with others according to a simplistic metric such as, for example, a comparison of version timestamps. This technique, known as the *Thomas write rule* [1], can have disastrous consequences from an end-user perspective. To avoid this, Clique uses a 'no lost updates' reconciliation policy, which guarantees that every update made at any node in the group will always eventually be 'seen' by all other nodes in the group, and no file modifications are ever irretrievably lost by the system.

Motivations

Our design motivations can be split into three categories; ease of use, support for real world conditions and some additional desirable system properties.

Ease of use

- *Transparency*: Integration into the user file system. All nodes should ideally have a local copy of every file in the system. From the end-user perspective, the shared files should not appear to be any different from the standard files available on the user's hard disk. File access latency times should not be noticeably different from ordinary files.

- *Self-organization* and *full decentralization*: No server is required, and there is no primary (master) repository for particular files. All nodes 'own' all files, and all share the administrative tasks such as setup.

- *Stability*: At any moment, all files in the local file system should be in a valid, usable state.

- *Mutability*: The files are fully mutable on all nodes, i.e. writeable everywhere.

- *Platform independence*: Files may be shared between disparate platforms.

Support for real world conditions

- *Tolerance of network partitions.* Long-term network partitions are a feature of today's Internet, where NAT boxes, firewalls and wireless radio shadows are commonplace, and short-term partitions can occur when using unreliable IP multicast. Certain highly mobile nodes (e.g. Adam's laptop in the scenario above) intermittently move between network 'islands' and act as a 'bridge' between network partitions.

- *Resilience*: Very high tolerance to node crashes and a dynamically changing group membership, which is an intrinsic characteristic of ad-hoc wireless networking environments.

-*Disconnected operation*: All files should remain accessible while disconnected from the network.

- *Scalability*: Ideally, the Clique system should scale to 1,000 nodes, as well as large file sizes (1 GB) and large numbers of files in the system (up to 10,000).

Additional desirable properties

- *No lost updates* semantics: This prevents the system from losing a modification done on any node. In the worst case, conflicting modifications are saved in alternate locations and notifications are sent to the appropriate node for manual user correction.

- *Any-to-any update transfer*: Each peer must receive all the updates issued by other peers, even if the original issuers are not directly reachable. In other words, we wish to achieve epidemic replication of the volume contents [2].

Convergence: If no updates occur in the system, and if nodes are reasonably interconnected (i.e. there are no partitions in the peer connectivity graph), then all peers will converge to the same replicated state [3].

2. Earlier work

Collaboration tools

Clique belongs to a class of applications called *collaboration tools*. This term refers to software that enhances communication, collaboration and co-ordination between people by enabling them to share information in a real-time, dynamic fashion. The earliest commercial effort in this area was *Lotus Notes* [4], a personal information management utility which today offers group messaging, calendar and scheduling facilities with advanced security and scalability features for enterprise use. However, it has very limited support for update reconciliation. As it is based on a client-server model, it generally requires dedicated support staff and currently offers no support for client mobility.

The current market leader in collaboration software is *Groove Workspace*, developed by Ray Ozzie, the former designer of the Lotus Notes System. This supports an intuitive notion of *shared spaces*, through which users interact with one another, sharing and even co-editing documents in real time. Groove leverages its tight integration with common Microsoft Office utilities to support automatic resolution of conflicting updates and can be deployed in a lightweight peer-to-peer mode. However, a centralized server is required to support advanced functionality such as offline operation and security management, and there is no support for network partitioning.

Collaborative interactions can only be conducted through the Windows-only Groove application interface.

Distributed file systems

Clique can alternatively be seen as a lightweight *distributed file system*. The first such system, the Network File System [5], [6] developed at Sun Microsystems allows hosts to share files across a network using a pessimistic replication model. A single replica is stored on a central server and other devices access the file remotely as required. NFS is unsuited to mobile environments as it assumes permanent LAN connections between the server and client nodes. The server represents a single point of failure and a performance bottleneck in the system.

The Coda file system [7] was designed with the primary goal of providing *high availability* of shared information by replacing the single central server with a group of optimistically replicated servers. The drawback of this client-server approach is that individual nodes cannot synchronize in a pair-wise manner. Rather, all inter-node communications must pass through a central server. Using Clique, two PDAs could synchronize automatically on coming into range of a short-range wireless link such as 802.11. This is not possible with a client-server model.

The ROAM file system [8] was the first to tackle the problem of geographic mobility of users. Each ROAM node is a peer in a specific *ward* (which may overlap), similar to a Clique group. Wards elect a ward master and are connected in an adaptive ring topology using point-to-point links. In turn, ward masters form a 'super-ward'. All nodes are permitted to issue updates at any moment. Updates are propagated to other ward members, and through the ward master to achieve eventual reconciliation across all wards. However, because the reconciliation mechanism used is based on the standard version vector mechanism introduced by Parker *et al.* [9], all ward members are required to keep a list of peer nodes who are also members of the ward, which potentially limits the scalability of the system. Additionally, the mechanism by which peers dynamically discover and join a particular ward is not described.

A recent trend has been the emergence of a second generation of fully peer-to-peer platforms. These systems such as Freenet [10], Chord [11], Pastry [12], Tapestry [13], CAN [14], Hypercast [15] typically construct a server-less *overlay network* at the application level that implements a *distributed hash table*. For each file to be stored, a unique key is produced by, for example, hashing the file contents. Each node in the overlay stores files relating to a particular range of key values. The system supports *put* and *get* operations, which store and retrieve files at the corresponding node. These overlay networks exhibit very high levels of fault-tolerance and generally scale well, with *O(logN)*.

A number of file systems, such as CFS [16], PAST [17] and Oceanstore [18] have been implemented on top of these systems. PAST and CFS are designed for use as read-only archival and publishing platforms. Oceanstore adds a client-server layer with strong security and scalability features in order to provide a global storage repository distributed across a vast number of untrusted network nodes. However, these systems do not support traditional file system semantics, such as delete and rename operations.

Other efforts [19], [20], [21], [22], [23] have focused on replication policies, such as gossiping, rumoring and anti-entropy algorithms. In the Clique case, we took an approach relying on a one-to-all communication channel.

Reconciliation

As optimistic replication systems allow replica contents to diverge in the short term, they require a method for *reconciling* conflicting updates which may be generated concurrently by different users. Reconciliation protocols have been categorized [24] into two general classes, namely *state transfer* and *operation transfer* protocols.

State transfer protocols involve the transfer of the current object state from the node with the most recent object version to other objects. In the case of a file system application such as Clique, the objects to be transferred are files. As such, the entire file is transferred from the node with the most recently updated file to other nodes, which reconcile their states by overwriting their local file copy with the newer version. Several methods have been devised to synchronize replicas, such as the *Thomas' write rule* [1], the *two-timestamp* algorithm [25], *version vectors* [9] and *version histories* [26], as used in Clique.

Operation transfer protocols such as those used by Bayou [27], Icecube [28], or Ficus [29] potentially offer significant gains in bandwidth usage by transferring only the semantic operations performed on an object since a previous reconciliation, rather than the object itself. Objects are assumed to be synchronized (or null) across all replicas at a given time, T_0, and each node maintains an ordered log of all operations performed on a particular object since T_0. As operations performed on an object are application-specific, operation transfer protocols generally require knowledge of the semantics of the application performing each update. In the case of a distributed file system, each node in the system would also require a consistent profile of applications.

Bayou [27] is a framework for asynchronous collaboration applications which uses an operation transfer reconciliation system. Each replicable object is associated with a primary node, which is uniquely responsible for assigning total ordering between operations. Updates are propagated in an epidemic fashion between nodes. Primary nodes are defined to be well-connected, stable

nodes, which function as a cluster of servers for less well-connected non-primary devices as in the Coda [7] model.

3. Approach

Terminology

We first define some terms used in the remainder of this paper. We distinguish between a *peer*, a physical device on which Clique is installed, and an *agent*, a particular instance of the Clique application running in a separate Java Virtual Machine on a peer. When launching the Clique agent, the user supplies a pathname which points to the root of the local Clique *volume*. Over time, the contents of this directory will converge towards a global view of the volume in a particular *Clique group* (i.e. among all nodes that share a common communication channel). As a single peer may simultaneously be running a number of agents, it may consequently be a member of a number of Clique groups.

Any file or sub-directory of an active volume is replicated automatically across all currently active agents in the group. A *version* of a local file is an object which represents the state of the file's contents (expressed in terms of a unique hash of its contents) at a particular moment in time, namely the time of its last modification. A given copy of a file corresponding to a particular version on a given machine is called a *replica* of the file. Clique does not currently have any notion of file ownership of a particular file; all replicas are equal and an update made to any replica will have the same 'weight' as an update made at any other replica.

Network *partitions* can be of a temporary or a long-term nature. Temporary partitions, commonly caused by crashes or congestion at the network layer, generally heal quickly, although in a wireless environment temporary partitions should be considered more the rule than the exception. Long-term partitions, on the other hand, never heal. They may be the result of physical disconnections in the network or network management tools that disrupt IP routing such as NAT boxes and firewalls. The epidemic replication protocol used by Clique can exploit physical device mobility to allow convergence of volume state even across long-term partitions.

Local agent system

The Clique system is implemented using agents running on each peer of a group. An agent has the following responsibilities:

- Implementing the protocol for communication with the other peers from the group.
- Scanning the local volume for any local modifications such as file addition, deletion, update or renaming.
- Communicating any performed modifications, whether locally or remotely, to the other peers in the group.
- Resolving update conflicts.
- Updating the local storage device with the appropriate global modifications.

Architecture

Clique relies on a one-to-all group communication channel. In our initial implementation, we use IP multicast as the transport layer. The *channel protocol* builds a communication channel on top of this layer (See Figure 1). This protocol tolerates the three types of channel unreliability – (limited) loss, (finite) duplication, and reordering – which are inherent to IP multicast.

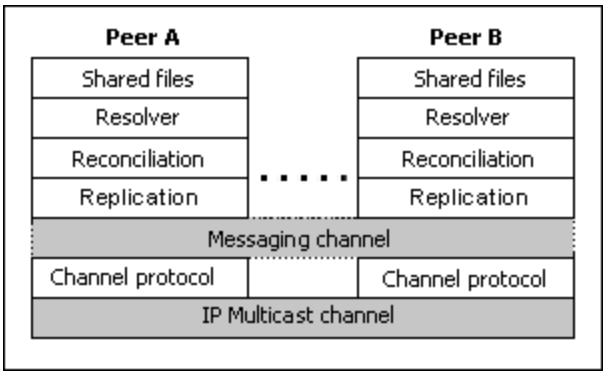

Figure 1: The Clique architecture

Additionally, IP multicast effectively limits the maximum packet size to 1472 data bytes over Ethernet links. To handle these limitations conveniently, our channel protocol masks message buffering, chunking and reconstruction as well as low-level packet duplication and reordering. This layer is also responsible for encryption of the network traffic. A flow control mechanism based on carrier-sensing and random back-off times has been implemented to reduce the probability of broadcast collisions and prevent saturation of the multicast channel. The *replication layer* sees a (lossy) messaging channel offering one-to-all communication with client-side buffering in the form of input and output queues. It is responsible for advertising the available local files and directories as well as their properties, and for file transfer. Some details are given below. The *reconciliation layer* ensures that proper file versions are kept during synchronization between peers. Finally, the *resolver layer* handles conflicting updates made by different peers.

Channel protocol

The lower-level channel protocol is an optimistic variation on the *Stenning protocol* [30]. Stenning defined a protocol whereby a high-level message is divided into blocks. These blocks are sent indefinitely, until a message receipt acknowledgement is received from a higher layer. In our case, the blocks are sent only once. If blocks are lost during transmission, a *resend* command will implicitly be issued at the higher layer.

Let there be two processes, P_1 and P_2, where P_1 is considered the emitter of a message M, and P_2 is the recipient for M. Note that as we are using multicast in the Clique case, there may be multiple P_2 entities receiving the same message at once. On receipt of a high-level message with identifier ID_M, P_1 divides the message M into equally sized chunks $\{m_1, m_2,\ldots m_N\}$, each of which is tagged with a 0-based sequence number and the high-level identifier ID_M. P_1 sends the chunks in sequence to P_2. P_2 receives the chunks and places them in a buffer corresponding to their message identifier ID_M at the correct insertion point indicated by its sequence number. The last chunk m_N contains an end-of-message tag to inform P_2 that the message is complete. Once all the chunks have been received by P_2, the received message M is exposed by placing it in the higher layer incoming message queue. A garbage collector process periodically executes to detect corrupted or incomplete messages. If P_2 has received at least one chunk of a particular message and has not received any new chunks within x seconds (typically 5 seconds), the message M is considered corrupted and all the buffered chunks are discarded.

Replication protocol

The replication protocol has the following properties:

- Individual nodes have no knowledge of group membership. All messages broadcast by any group participant are received by all currently reachable group members at no additional cost in network bandwidth. Nodes are not required to maintain tables of current or past neighbors, removing a significant barrier to scalability inherent to traditional distributed file systems.

- In order to guarantee high tolerance to the lossy nature of the communication channel and the variable rate of node disconnections, the higher-layer messaging protocol is entirely asynchronous and sessionless. Nodes make no assumptions as to which other nodes may or may not respond to a given message and as no acknowledgements are required, message implosion is avoided. At any moment, participants may disconnect from the group, without any formal sign-off procedure.

- The broadcast nature of all inter-node communications permits efficient use of the network bandwidth. When a file is updated at one node, another random group member node will initiate the update transfer sequence. Other currently connected nodes may take advantage of these broadcasts to update the contents of their local volume at no cost in network bandwidth terms.

- Updates are propagated through the system in an epidemic fashion, i.e. two nodes do not have to be directly connected in order to achieve eventual synchronization. In particular, this allows updates to jump across islands of nodes that are physically disconnected from one another, using a bridge node which is intermittently connected to both partitions, so that global consistency can be achieved through strictly local operations.

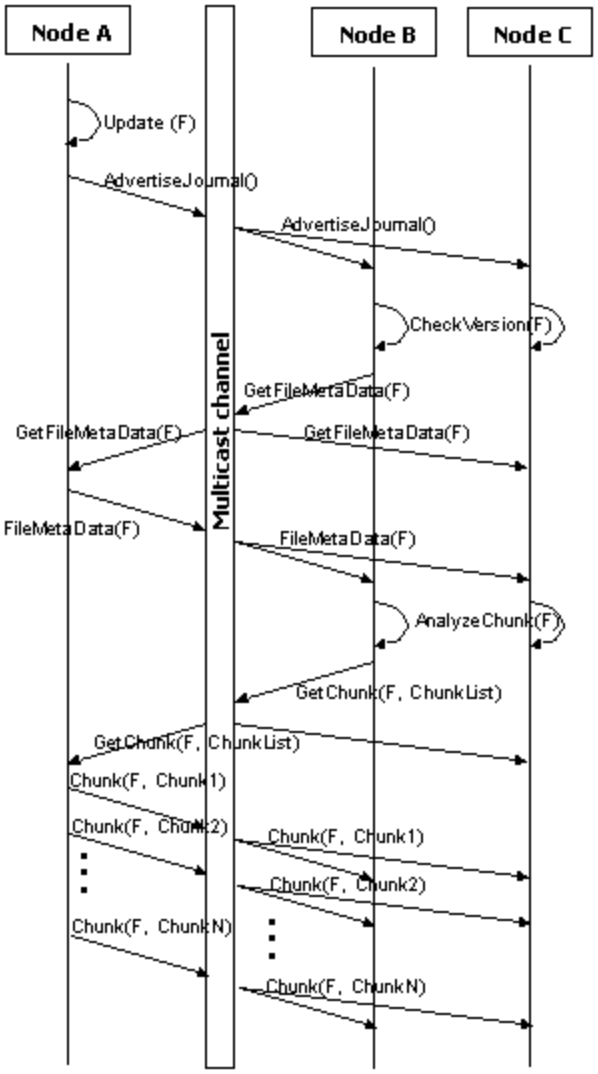

Figure 2: The messaging protocol

The following describes a typical Clique replica reconciliation process;

1. A file, *F*, is updated at node A.
2. Node A periodically broadcasts a *Journal* message, which contains a description of the version history of all files in the local volume.
3. Nodes B and C run the version history check procedure locally and independently establish that A has a more recent version of *F*.
4. Node B broadcasts a *GetFileMetaData(fileHash)* message for *F*. Node C notes that Node B has made a *GetFileMetaData* request and suppresses its own such message.

5. Node A divides *F* into a number of chunks and computes an SHA-1 hash of the contents of each chunk. It then broadcasts a *FileMetaData* message which lists each chunk's hash along with a chunk ID.
6. Node B (or C) issues a sequence of *GetChunk(fileHash, startChunk, endChunk)* messages.
7. Node A issues a sequence of *Chunk(fileHash, chunkID)* messages containing the chunk contents.

Intelligent file transfer

The messaging protocol used in Clique is efficient for transfer of large files whose contents have changed only slightly, as only chunks which have changed are requested by the receiving node. The chunk sizes are currently fixed, although we expect that further network bandwidth gains could be achieved by employing a more intelligent, variable chunk size algorithm such as that used in LBFS [38].

Additionally, 'rename' operations are treated as a 'create/delete' combination at the operation level, but the agent can establish, based on a comparison of the hash value of the file, that the operation is merely a 'rename' and will retrieve the deleted file from the Trash folder, again at no cost in terms of network bandwidth.

Replica reconciliation

To synchronize a given object *F* between two sites *a* and *b*, Clique uses a mechanism derived from the lightweight version comparison protocol used by Reconcile [26]. This relies on the knowledge of the past history of the file at each of the sites. For each version of a given file, a signature (based on an SHA-1 hash of the file contents, and a local creation timestamp) is kept. This signature uniquely identifies the file. Let F_a^n be the signature of the n^{th} version of the file seen on site *a*. Each site knows a list of past versions of the file; site *a* knows the history $\{F_a^1, F_a^2, \ldots F_a^n\}$ and site *b* knows the history $\{F_b^1, F_b^2, \ldots F_b^m\}$. Note that it is not necessary to keep the actual previous contents of the file, merely a history of file signatures, which requires very little space.

To synchronize, *b* sends *a* its version history $\{F_b^1, F_b^2, F_b^3, \ldots F_b^m\}$. *a* will compare its latest version's signature F_a^n with *b*'s latest version signature F_b^m. If both signatures are equal, the files are identical and do not need synchronization. If signatures are different, *a* will check for equality between his latest version and one of the past versions of *b*. In other words, it will compare F_a^n with each element in $\{F_b^1, F_b^2, F_b^3, \ldots F_b^{m-1}\}$. If a match is found, i.e. $\$ k \hat{I} \{1, \ldots m-1\}, F_a^n = F_b^k$., this means that *b*'s version of the file is a direct descendant of *a*'s latest version, which implies that *a*'s version of F may be safely overwritten with *b*'s. A similar mechanism applies in the opposite order when *b*'s version of *F* needs to be refreshed with *a*'s. When both comparisons fail, this means that there is no direct ancestral link between the versions, and an *update conflict* is detected.

Deletion detection

Delete operations: These must be treated explicitly in a file reconciliation system. To see why this is so, imagine the following scenario; two users, *A* and *B*, set up a Clique network and synchronize their volume contents. *B* subsequently disconnects from the network and deletes a file *F* while offline. Upon *B*'s reconnection, *B* would falsely establish that *A* had a new file, *F*, which was not in *B*'s volume and would initiate a file transfer, when in fact the correct behavior would be for *A* to delete its copy of *F*.

To avoid this, a deletion operation is recorded in the version history for a file as a new version, with a hash value of 0, and the version history for the deleted file is kept in the local Journal for a long time (see "Garbage collection" below). This allows update/delete and create/delete conflicts to be detected, triggering a warning to the user.

Garbage collection

Version history-based reconciliation algorithms traditionally suffer from garbage collection problems. In a stateless system such as Clique, where nodes have no knowledge of group membership and so cannot establish at any given moment whether they have successfully reconciled with all other nodes in the group, it can be difficult to detect when a version can be safely deleted from a node's version history table. Clique applies a pragmatic localized approach to this problem – we simply delete any version in the local version history at each node that is more than *x* days old. This will potentially cause 'false positive' conflicts to be detected when nodes which have been completely disconnected from the group for more than *x* days rejoin, but does not violate the 'no lost updates' policy. We anticipate that such problems will be rarely if ever seen in real-life scenarios, if the *x* value is set to a sufficiently large value such as 30 days.

Conflict resolution

It has been shown [31] after analysis of typical usage patterns that write sharing conflicts occur on a very seldom basis (about 1 conflict for 10000 file operations), hence justifying an optimistic approach. Kistler [32] also shows that concurrent file sharing in a distributed environment is rare: 99% of file updates are made by the same user that made the previous update, and the probability of different users modifying the same file in the same week is less than 0.4%. Huizinga [33] shows worse figures but still confirms Kistler's main points.

A number of previous projects have focused on the development of automatic resolvers [8], [34], which can automatically handle conflicting updates on particular file types. For instance, CVS is able to resolve update con-

flicts made to a given source code file provided that changes have not been made on the same lines of code within the file. More generally, it has been shown by Reiher *et al.* [35] that using automatic resolvers can reduce the number of unrecoverable conflicts by several orders of magnitude in a typical work environment.

In the Clique project, we did not focus on the automatic resolution of conflicting updates. Instead, when an update conflict between 2 machines is detected, the most recent file (according to a simple timestamp comparison) is stored in-place, while a new file is created in a designated Conflict directory containing the older file contents. A notification (pop-up dialog or e-mail warning message) is sent to the user who can proceed with a manual resolution of the problem.

Security

The Clique security model is currently very straightforward yet surprisingly useful in practice. Group member candidates use an out-of-band mechanism to obtain a password along with the group channel characteristics such as IP address and port for entry to a particular Clique group. All group communications are encrypted using this password. Users who have the password hence have full read/write permissions for all files in the Clique volume, although it is not possible to interpret any group message for a potential attacker who does not know the password.

Topological and routing issues

As the native communication channel used by Clique is based on IP multicast, it suffers from routability limitations. Multicast is disabled on most IP routers. Consequently, we have extended Clique's messaging protocol to allow communication between machines which are not members of the same IP subnet.

We have designed a specific extension of Clique which uses UDP as a transport layer. Although UDP is a one-to-one communications channel, it is routable and permits communication between machines which are separated from each other by routers and possibly firewalls. In Figure 3, we demonstrate the UDP link between machines *B* and *C* which makes it possible to extend the Clique community across both work subnets. Note that machine *B* runs two instances of the Clique agent concurrently. One of the agents communicates in multicast mode with the other peers in subnet 1, and the other communicates in direct point-to-point mode with machine *C* in subnet 2. The roaming laptop E intermittently connects to the work subnet 1 or to the home subnet. This *intermittent multiply connected* machine makes it possible for machines from different subnets which do not have any direct IP routing capability between them (such as machines A and G) to be members of the same Clique group.

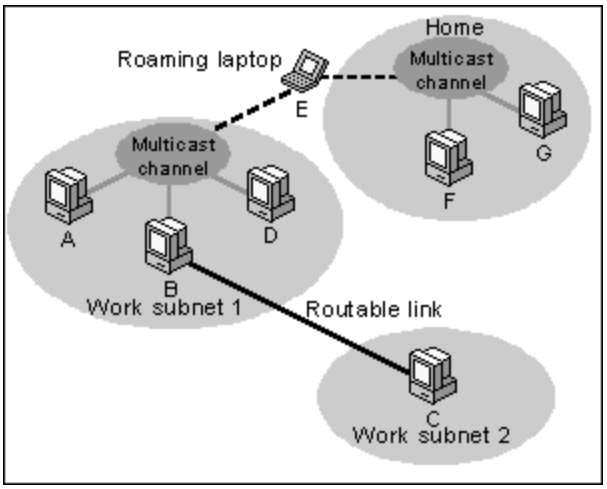

Figure 3: Topological capabilities

By extension, we can show that machines from *any* number of subnets can be added to the Clique group, provided that subnets linked by routable links or by intermittent multiply-connected machines form a connected graph.

Scalability

As described above, we are using carrier sensing and backoff algorithms at the channel protocol level. This, combined with the randomized periodic AdvertiseJournal message broadcasts by each node, ensures that the overall network bandwidth used remains constant regardless of the number of agents in the Clique group or the number of file versions in the volume. Note, however, that each individual peer has a proportionately reduced opportunity to communicate with the group and hence global convergence times are a function of the number of agents in the group.

4. Current Status and Future Work

We have developed a working prototype of Clique using Java. The abstraction offered by this language makes it possible to share files between various architectures such as PDAs, Windows or Linux PCs. There is more research work to be done in order to refine the optimal timer values for the channel protocol, which should offer good network load control, memory and CPU usage.

An additional step forward would be to improve our file monitor. This component currently performs a file examination periodically. We would like to replace it by a kernel-level component that would react to file system operations such as file creation, modification, move or rename, and directly notify the local Clique agent. This would reduce the CPU load and would also be more reactive to user changes, as these would be immediately detected rather than periodically. Such a low-level piece of

code would however break the platform independency requirement.

We also envisage a number of additional features that would greatly enhance the application functionality from the user perspective. A quota system will be implemented to eliminate the possibility of overflowing the maximum local storage capacity. A manual 'refresh' option could also be incorporated so that a user can guarantee that his local volume contains the most recently modified files, available on other currently connected peers.

In case of update conflict, our current resolver is only able to keep copies of the conflicting versions of a file and report the conflict to the user. A useful extension of Clique would be to port these automatic resolver platforms described earlier [8], [37] to our architecture in order to limit the number of conflict instances that require user intervention.

Our security model could be extended to provide file ownership semantics as well as *access control lists* over files, the goal being to relax the *'all nodes own all files'* criterion and replace it with mechanisms for setting read/write rights for files. This will allow the system to prevent inadvertent deletions and introduce a level of privacy control. Repudiation of group members is a security issue that we did not handle in the current version of Clique. Golding [35] shows that decentralized group management in a scalable system is difficult, so we may have to loosen our scalability constraints in order to provide a higher level of security.

5. Conclusion

We described our original work on Clique, a peer-to-peer replicated file system. Essentially, Clique links together nodes within a group and facilitates file and directory sharing across group members. It offers a number of interesting benefits at the user level. These include automatic and transparent operation, and adaptation to real-world conditions such as varying network capabilities and support for disconnections through epidemic replication and asynchronous messaging. The project also satisfies a number of important technical constraints such as global convergence of the system and a no lost update policy. The Clique distributed algorithm is based on a one-to-all communication paradigm, taking advantage of the IP multicast transport layer.

Our paper describes the prior art in this field, and how we designed the Clique system, its agent architecture and its distributed algorithms. The protocols we developed are session-less, providing good scalability in terms of the number of peers in a group, while preserving the network usage, processing and memory overhead necessary for the system to operate. An initial prototype implementation has been developed which demonstrates that the approach we have taken is a practical one.

6. References

[1] R. Thomas, "A majority consensus approach to concurrency control for multiple copy databases", ACM Trans. On Database Systems (TODS), 1979.

[2] D. Agrawal, A. El Abbadi, and R.C. Steinke, "Epidemic Algorithms in Replicated Databases", Proceedings of PODS '97, pp. 161-172

[3] N. Vidot, M. Cart, J. Ferrié, M. Suleiman, "Copies convergence in a distributed real-time environment", Proceedings of the ACM Conference on Computer Supported Cooperative Work, 2000.

[4] L. Kawell, S. Beckhardt, T. Halvorsen, R.R. Ozzie, I. Greif, "Replicated Document Management in a Group Communication System", Proceedings of the Second Conference on Computer Supported Cooperative Work, 1988.

[5] "RFC1094: NFS: Network File System Protocol Specification", http://www.faqs.org/rfcs/rfc1094.html

[6] "RFC1813: NFS Version 3 Protocol Specification". http://www.faqs.org/rfcs/rfc1813.html

[7] J. Kistler, M. Satyanarayanan et al. "Coda: A Highly-Available File System for a Distributed Workstation Environment". IEEE Transactions on Computers 39(4); 447-459, April 1990.

[8] David H. Ratner. "Roam: A Scalable Replication System for Mobile and Distributed Computing", PhD thesis, UC Los Angeles, 1998. UCLA-CSD-970044.

[9] D.S. Parker, G. Popek, G. Rudisin, A. Stoughton, B. Walker, E. Walton, J. Chow, D. Edwards, S. Kiser, C. Kline, "Detection of mutual inconsistency in distributed systems", IEEE Transactions on Software Engineering, 9(3):240-246, 1983.

[10] I. Clarke, O. Sandberg, B. Wiley, T.W. Hong, Freenet: A Distributed Anonymous Information Storage and Retrieval System in Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, Springer: New York, 2001.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications", Proceedings of ACM SIGCOMM'01, San Diego, 2001.

[12] A. Rowstron, P. Druschel, "Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems", Proceedings of IFIP/ACM Middleware 2001, Heidelberg, Germany, 2001.

[13] B. Zhao, J. Kubiatowicz, A. Joseph, "Tapestry: An infrastructure for fault-resilient wide-area location and routing", Technical Report UCB//CSD-01-1141, U.C. Berkeley, 2001.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, "A Scalable Content-Addressable Network", Proceedings of ACM SIGCOMM'01, San Diego, 2001.

[15] J. Liebeherr, M.Nahas, W. Si, "Application-layer Multicasting with Delaunay Triangulation Overlays", University of

Virginia Department of Computer Science, Technical Report CS-2001-26, 2001.

[16] F. Dabek, M. F. Kaashoek et al, “Wide-area co-operative storage with CFS”, Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), 2001.

[17] A. Rowstron, P. Druschel, “Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility”, Proceedings of ACM Symposium on Operating Systems Principles (SOSP'01), 2001.

[18] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, “OceanStore: An Architecture for Global-Scale Persistent Storage”, Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), 2000.

[19] B. Kantor, P. Rapsey, “RFC977: Network News Transfer Protocol”, http://info.internet.isi.edu/in-notes/-rfc/files/rfc977.txt

[20] A.J. Demers, D.H. Greene, C. Hauser, W. Irish, J. Larson, "Epidemic algorithms for replicated database maintenance", 6th Symposium on Principls of Distributed Computing (PODC), Vancouver, Canada, 1987.

[21] M.J. Lin, K. Marzullo, "Directional gossip: Gossip in a wide-area network:, 3rd European Dependable Computing Conference, Prague, Czech, pp. 364-379 1987.

[22] Y. Saito, B.N. Bershad, H.M. Levy, "Manageability, availability and performance in Porcupine: a highly scalable, cluster-based mail service", In 17th Symposium on Operating System Principles (SOSP), Kiawah Island, SC, USA, 1999.

[23] J. Gray, P. Helland, P. O'Neil, D. Shasha, "Dangers of replication and a solution", In International Conference on Management of Data, Montréal, Canada, 1996.

[24] Y. Saito, M. Shapiro. “Replication: Optimistic Approaches”. HP Labs Technical Report HPL-2002-33, 2002.

[25] L. Kawell, et al. "Replicated Document Management in a Group Communication System", in Second IEEE Conference on Computer-Supported Cooperative Work. Portland, Oregon, 1988.

[26] J. Howard, “Reconcile User’s Guide”, Technical Report TR99-14, Mitsubishi Electric Research Laboratory, 1999.

[27] W.K. Edwards, E.D. Mynatt, K. Petersen, M.J. Spreitzer, D.B. Terry, M.M. Theimer, " Designing and Implementing Asynchronous Collaborative Applications with Bayou", In Proc. of ACM Symp. on User Interface Software & Technology, pages 119-128, 1997.

[28] A.-M. Kermarrec, A. Rowstron, M. Shapiro, P. Druschel, "The IceCube approach to the reconciliation of diverging replicas", 20th Symposium on Principles of Distributed Computing (PODC), Newport, RI, USA, 2001.

[29] T.W. Page Jr., R.G. Guy, J.S. Heidemann, D.H. Ratner, P.L. Reiher, A. Goel, G.H. Kuenning, G.J. Popek, "Perspectives on Optimistically Replicated, Peer-to-Peer Filing", Software, Practice and Experience, 28(2), pp. 155-180, 1998.

[30] N.V. Stenning, “A data transfer protocol”, Computer Networks, 1(2):99-110, 1976.

[31] A. Wang; P.L. Reiher; R. Bagrodia, "A simulation evaluation of optimistic replicated filing in mobile environments," Proceedings of International Performance, Computing and Communications Scottsdale, AZ, USA, 1999, pp.43-51.

[32] J.J. Kistler, M. Satyanarayanan, "Disconnected operation in the Coda file system", ACM Transactions on Computer Systems, 10(1), 3-25, 1992.

[33] D.M. Huizinga, K.A. Heflinger, " Experience with Connected and Disconnected Operation of Portable Notebook Computers in Distributed Systems" In Proceedings of The Workshop on Mobile Computing Systems and Applications, 1994.

[34] P. Cederqvist et al, “Version management with CVS”, http://www.cvshome.org/docs/manual, 2001.

[35] Reiher, P., Heidemann, J., Ratner, D., Skinner, G., and Popek, G. Resolving File Conflicts in the Ficus File System. In Proc. the Summer USENIX Conference, pp. 183--195, 1994.

[36] R.A. Golding, "Weak-consistency group communication and membership", Ph. D. thesis, University of California Santa Cruz Technical Report no. USCS-CRL-92-52, 1992.

[37] Reiher, P., Heidemann, J., Ratner, D., Skinner, G., and Popek, G. Resolving File Conflicts in the Ficus File System. In Proc. the Summer USENIX Conference, pp. 183--195, 1994.

[38] A. Muthitacharoen, B. Chen, D. Mazières, “A Low-bandwidth Network File System”, In Proceedings of the. 18th Symposium on Operating Systems Principles, Banff, Canada, 2001.