Log Compression

David Barrett-Lennard
26 Feb 2008

Log compression before checkin to the repository

Log compression before checkin to the repository is extremely important. For example, during the normal editing of text most operations only insert or remove individual characters. For the purposes of interactive collaboration, such operations are assigned unique (s,t) values. Without log compression these single character operations would be very expensive to store in the repository.

Before performing a check-in to the repository it is desirable to perform the most aggressive possible log compression we can achieve. This form of log compression is lossy, and involves ignoring the (s,t) values of the individual operations in order to maximise compression by allowing adjacent half open intervals to *coalesce*. Without that it wouldn't be possible to convert single character operations into stringwise operations. It follows that we don't expect this form of merging to be compatible with IT,ET.

Furthermore, we want to only record the *net effect* of the overall operation on the working documents. Therefore sequences of moves should be compressed down into a single move. Redundant deletes should be removed, etc.

A helpful way to achieve this is to represent an operation as the sequence [D M C] (ie delete then move then create), because this representation is unable to delete characters that have been moved or created, and is unable to move characters that were created. A transformation from [C D M] to [D M C] is implicitly lossy.

Something to bear in mind is that the net effect on the effects document is quite different to the net effect on a working document. Given that redundant move operations cause insertions into the effects document we see that for maximum compression we need to break the assumption that log compression for the repository will provide an equivalent net effect on the effects documents. This is something to consider when a site performs a check-in because changes to its effect documents are recorded in the local PtoQ maps, so these will need to be adjusted to be compatible with the aggressive compression.

One possibility is that the compression algorithm is able to output appropriate changes to be applied to the relevant PtoQ maps. A simpler but probably less efficient approach is to undo all the work on the effects documents and redo according to the maximally compressed form.

Log Compression during an interactive collaboration

We aren't going to coalesce intervals during an interactive collaboration if we keep the (s,t) identity on the operations. Of course this information is needed in order for a site to work out what operations need to be sent to a peer, or to appropriately transpose adjacent operations in the history buffer in order to calculate a HB suffix to transform incoming operations.

However, we need to support latecomers to the collaboration. It is assumed that one of the sites is designated a *master*, and this site occasionally sends a message to its peers of a vector time up to which log compression is permitted.

Consider that this log compression doesn't completely ignore (s,t), and only ignores the t values. This form of merging is compatible with IT/ET (because only siteids affect these algorithms), and this would probably provide a reasonable level of per-user log compression. However, it doesn't seem to be worth while, because the t values are needed to make sense of the context of incoming operations, and for being able to send the appropriate operations to another site.

We would rather not expose the interactive collaboration to the quadratic complexity of ITing two large concurrent branches of operations that may arise due to an extended network outage during the collaboration. A simplistic solution is to detect a persistent network outage using a watch dog on all members of the session, and when connectivity is lost warn the user that continued work in the session could lead to very expensive merges.

A better approach is to support compression of operations in the HB is such a way that we don't upset the ability for sites to connect in new topologies, roll back to earlier states etc. This is not easily achieved. The basic idea is that compression can greatly reduce the time taken to bring latecomers up to date. Also, following a network outage, compression of operations can substantially reduce the time taken for IT dues to its quadratic complexity.

We need some way to designate sets of operations that need to be compressed. A pair of (causally valid) vector times v_1,v_2 provides a well defined set of operations $\chi(v_2) \setminus \chi(v_1)$.

Causally valid vector times
Definition. Vector time v is *causally valid* if

$$O_1 \in \chi(v),\ O_2 \notin \chi(v) \ \Rightarrow\ \neg(O_2 \rightarrow O_1)$$

or equivalently

$$O_2 \in \chi(v) \text{ and } O_1 \rightarrow O_2 \ \Rightarrow\ O_1 \in \chi(v)$$

This implies it is possible to divide a history buffer into prefix and suffix according to the vector time v, and it will never be the case that an operation in the suffix causally precedes an operation in the prefix.

More specifically it implies that it won't be necessary to ever transpose a pair of contextually serialised operations $[O_1, O_2]$ where $O_1 \rightarrow O_2$.

Intersection and union of vector times
Definition: For vector times v_1,v_2, $v = v_1 \uparrow v_2$ denotes the vector time satisfying

$$\forall s \in S,\ v(s) = \max(v_1(s), v_2(s))$$

Definition: For vector times v_1,v_2, $v = v_1 \downarrow v_2$ denotes the vector time satisfying

$$\forall s \in S,\ v(s) = \min(v_1(s), v_2(s))$$

Definition: For set of vector times V, $v = \downarrow(V)$ denotes the vector time satisfying

$$\forall s \in S,\ v(s) = \min \{ v'(s) \mid v' \in V \}$$

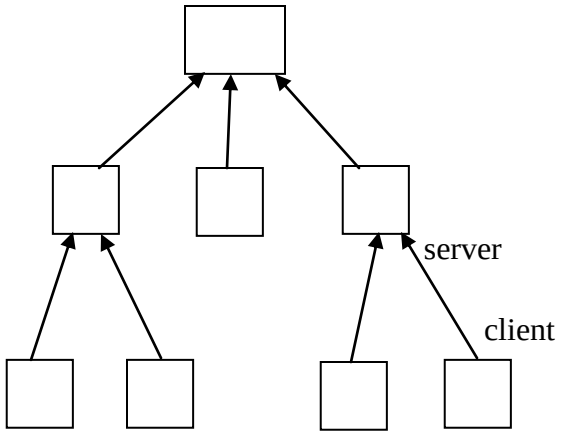
Definition: For set of vector times V, $v = \uparrow(V)$ denotes the vector time satisfying

$$\forall s \in S,\ v(s) = \max \{ v'(s) \mid v' \in V \}$$

Claim: v_1,v_2 are causally valid $\Rightarrow$ $v_1 \downarrow v_2$ and $v_1 \uparrow v_2$ are causally valid
Claim: $\forall v' \in V$, v' is causally valid $\Rightarrow$ $\downarrow(V)$ and $\uparrow(V)$ are causally valid

Definition of a group
Between a given pair of machines that have connected we can distinguish the client and server. Therefore the network topology is a directed graph.

It is assumed that each client connects to exactly one server machine and there are no cycles. This gives us a tree structure.



For machine m, let parent(m) denote the parent machine of m (if any), children(m) denotes the immediate children of m, and group(m) denotes the set of machines in the sub-tree rooted in m.

Lower bound of what's present in a group
Let $v_h(m)$ denote the current history buffer vector time of machine m.

Let each machine calculate a vector time $v_{lb}(m)$ that represents a lower bound on the set of operations that have been stored across all sites in group(m). Conceptually it is calculated as follows

$$v_{lb}(m) = v_h(m) \downarrow \ (\downarrow \{ v_{lb}(m') \mid m' \in \text{children}(m) \})$$

Each machine m occasionally calculates and sends $v_{lb}(m)$ to parent(m), with the caveat that it assumes the operations in the HB according to $v_h(m)$ have been made durable - ie the relevant transaction has been flushed.

The actual calculation of $v_{lb}(m)$ may use older (and "smaller") values of $v_{lb}(m')$. Nevertheless it still serves as a lower bound on what operations are currently stored in history buffers.

The cached values of $v_{lb}(m')$ are initialised to the base vector time for the collaboration (ie w.r.t. the branch on the repository).

Claim: $op(s,t) \in \chi(v_{lb}(m)) \Rightarrow$ op(s,t) is recorded on all machines in group(m)

Upper bound on operations that existed before session began
On machine m let $v_{orig}(m)$ be a causally valid vector time associated with the set of operations that were present before the session began.

Consider that $v_{ub}(m)$ is calculated as follows

$$v_{ub}(m) = v_{orig}(m) \uparrow \ (\uparrow \{ v_{ub}(m') \mid m' \in \text{children}(m) \})$$

op(s,t) present on a machine before session began
$\Rightarrow$ exists m' in group(m) st $op(s,t) \in \chi(v_{orig}(m'))$
$\Rightarrow$ $op(s,t) \in \chi(v_{ub}(m))$

By contrapositive:

$op(s,t) \notin \chi(v_{ub}(m)) \Rightarrow$ op(s,t) not present on any machine in the group when session began

Defining a set of operations in the history buffer that can be compressed
Consider that site m is a master for a group of computers group(m).

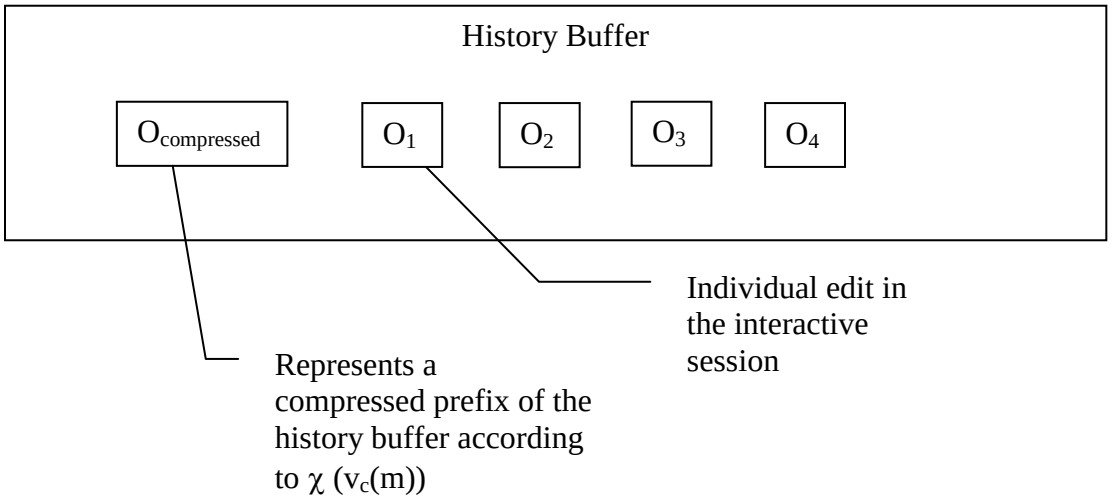
Site m can estimate that it is appropriate to compress operations within its subtree in

$\chi(v_{lb}(m)) \setminus \chi(v_{ub}(m))$.

Note that $A \setminus B = A \setminus (A \cap B)$. Therefore we can compress the operations in the HB bounded by $v_{ub}(m) \downarrow v_{lb}(m)$, and $v_{lb}(m)$.

Justification:
Suppose $op(s,t) \in \chi(v_{lb}(m)) \setminus \chi(v_{ub}(m))$
$op(s,t) \in \chi(v_{lb}(m))$ so op(s,t) is recorded on all machines in group(m).
$op(s,t) \notin \chi(v_{ub}(m))$ so op(s,t) was not present when the session began
So op(s,t) is an operation generated by a machine in the group since the session began that has been propagated to all machines in the group.

Compression of the prefix of a history buffer
Each site supports the ability to compress a prefix of its history buffer. This prefix is associated with a *Compression Vector Time* (CVT). Let $v_c(m)$ denote the current CVT on machine m.

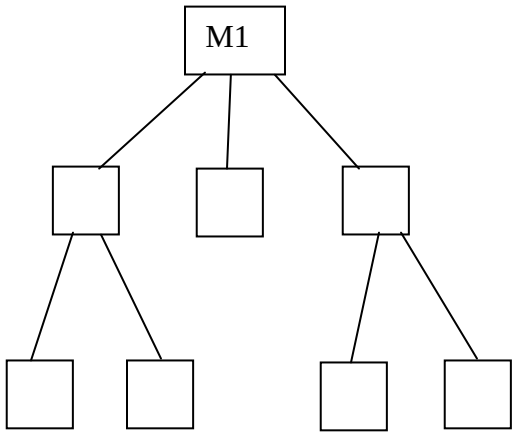


A site m compresses all operations in $\chi(v_c(m))$ in its history buffer into a *single operation* that only needs to describe the net affect on all effects documents, without needing to record (s,t) values of the individual edits. Note therefore that the log compression is lossy and incompatible with subsequent IT/ET against operations representing individual edits in the interactive session.

Note that $v_c(m)$ monotone increases on a given machine according to atomic transactions on its local persistent store.

Master sites in a collaboration

Consider a tree structure of sites involved in an interactive session. The root of the tree is designated a *master*, and this site occasionally sends a CVT to its immediate children specifying a vector time up to which log compression is permitted.



M1 will only generate a CVT v_c by setting $v_c = v_{lb}(M1)$. Note therefore that v_c is causally valid and represents a lower bound on all the operations in group(M1).

It is expected that each site forwards this message onto its peers, so that all sites agree on the sequence of CVTs received from M1. Note that when a site receives a CVT v_c it should find that all the operations in $\chi\ (v_c)$ are already present in its local history buffer. When a site receives v_c it is permitted to compress all operations in $\chi\ (v_c)$ in its history buffer into a single operation.

Consider that network connectivity is lost. An extreme case is that all machines in the group become disconnected. Suppose that they continue to perform some work, and then reconnect - perhaps in an entirely new topology.

When two machines first connect each sends their $v_h(m)$ and $v_c(m)$ values to the peer.

Consider that machines m and m' connect. Machine m receives the current values of $v_h(m')$ and $v_c(m')$ from m'. Machine m should find that $v_c(m) \leq v_h(m')$ because $v_c(m)$ should be a lower bound on the content of all history buffers in the group.

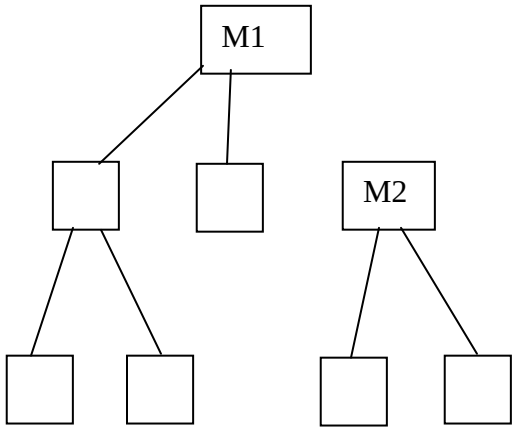
It should be found that if $v_c(m) \neq v_c(m')$ then either $v_c(m) < v_c(m')$ or else $v_c(m') < v_c(m)$. If $v_c(m) < v_c(m')$ then machine m should compress its history buffer up to vc(m'). This may require transposing operations in its history buffer in order to separate it into the appropriate prefix and suffix, so that the prefix can be compressed. Note therefore that the machines agree to compress up to the same vector time $vc = v_c(m) \uparrow v_c(m')$.

Only after a site has completed this step is it permissible for it to send any operations from its history buffer. This means at most one of the two sites may need to do some work before it can begin sending operations.

Consider that machine m' sends machine m an operation O with execution context v = ec(O). ie O is executed in the context of having executed all operations in χ (v). It can be assumed that $v_c(m) \leq v$. Therefore it is possible for machine m to use transpose to divide its HB into prefix and suffix according to vector time v (ie despite the fact that it has compressed all operations in χ ($v_c(m)$)), in order to calculate a suffix which can be IT'd with O in order to transform it so as to apply it and append it to the end of the HB.

When a client connects to a server they first exchange vector times to establish what operations they have and haven't got. If we assume there is no cycle in the connection graph we can assume there is no hidden pathway for sending operations that aren't present on the other side of the link between the two machines.

Consider that for each link, the client side uses a watch dog to check that the connection is working. When a link becomes broken the client chooses to become a master. For example



We assume that M1 doesn't know or care about the appearance of M2. Therefore M1 will continue generating vector times to specify the position up to which log compression is allowed.

Now the message system is asynchronous. M2 can tell what surely wasn't sent to the M1 group. However it can't tell whether all operations that appeared to be sent to the M1 group were actually processed. Eg they may have been lost in transit. We don't have a distributed transaction concept so it will not be possible to determine this until the link is reestablished.

When M2 "begins" there are three important vector times to be defined

1. The last compression vector time vc received from M1
2. A vector time vb providing an upper bound what may have been sent to the M1 group

M2 should forward all last compression vector times (that were received from M1) into other sites within its group. All sites will agree to compress the log up to vc.

M2 should then send a special message to specify vb to members of its group. Basically this says : don't try to compress operations in χ (vb)\ χ (vc). vb will serve an the base vector time for subsequent log compression.

No holding areas!

We want to support creation of new text, movement of existing text and deletion of text. Since IT/ET is quite complicated, it would appear useful to reduce complexity by reducing everything down to moves by the introduction of additional buffers called *holding areas* to allow for the *net effect* of creation and deletion as far as the working documents are concerned. ie a move from a holding area looks like a creation, whereas a move into a holding area looks like a deletion.

In other words, a holding area is a document with an OID and persists that is used to allow for creation of destruction of matter as far as the "real" documents are concerned.

This simplifies IT/ET because there is only one type of operation (ie "move"). However the use of holding areas raises the question of how merging will be achieved such that intervals are coalesced where possible. Somehow the identity of the holding areas must be regarded as not significant, providing opportunities to allow for proper log compression.

Unfortunately the use of holding areas makes log compression rather difficult. For example, consider that three sites concurrently perform the following operations.

O_1 : Delete character : move from S_1 to holding area H_1
O_2 : Delete character : move from S_1 to holding area H_2
O_3 : Move character : move from S_1 to document S_2

Suppose the siteids are such that the dominance relation satisfies $O_1 < O_2 < O_3$. ie O_3 dominates O_1,O_2. Let site 1 store [O_1 O_2' O_3']. Then O_2' will move the character from H_1 to H_2, and O_3' will move the character from H_2 to S_2. Note therefore that O_3' looks like a pure create operation even though it logically represents a move.

This leads to a rather nasty observation: Given a log [O_1, ... , O_n] we will have operations that refer to holding areas by their identity and we can't assume any of the following

1. a move into a holding area doesn't necessarily represents a real delete, because it may be dominated by a move that brings the deleted characters back to life.
2. A move from a holding area into a document doesn't necessarily represent a real create.

In fact, we can't really assume much at all unless we process the entire log. This makes it difficult to see how we could implement some form of incremental log compression algorithm.

References

[1]	David Barrett-Lennard. Operational transform - Merging Operations. Sep 2005
[2]	David Barrett-Lennard. Vector Time. Feb 2008