

MASSIVE-3 / HIVEK Consistency

Chris Greenhalgh, 24th August 1999.

Introduction

MASSIVE-3 provides flexible support for distributed data consistency in a CVE. This includes implementations of work from the University of Reading within the HIVE project on ownership transfer, advance communication and sufficient causality. It also integrates support for centralized updates, and tandem update/ownership requests (as in the CIAO system [2]). This technical note uses the `hiveClient` application to illustrate the capabilities of the system, and to explore the underlying issues.

After introducing the application, the aspects of consistency considered are:

- Basic consistency strategy.
- Centralized update support.
- Ownership transfer.
- Global time versus local time.
- Advance communication.
- Sufficient causality.

The application

`hiveClient` is a standard application implemented in the MASSIVE-3 system which allows the user to exercise many of the system's capabilities with regard to consistency. To run a `hiveClient` demo:

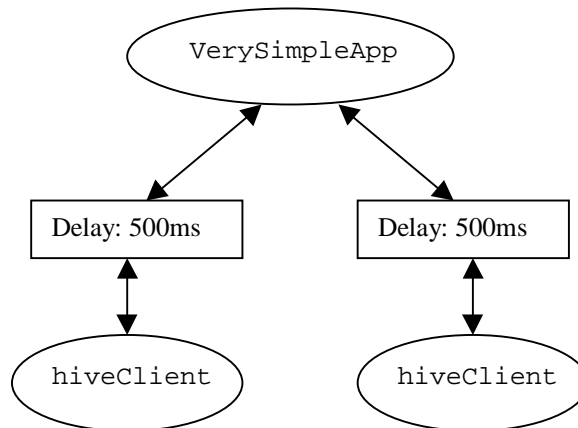
- Start a `Trader` process on every machine intended to take part.
- Start one instance of `VerySimpleApp` to act as the world server.
- Start two instances of `hiveClient`, specifying the Locale created by `VerySimpleApp` as the starting locale ("`<ip>/SimpleAppEnvironment`", where `<ip>` is the IP address of the machine running `VerySimpleApp`).

Because this work is oriented towards coping with network delays it is easiest (only possible?!) to see the effects when there is a significant network delay between the clients and the world server. For example, this can be introduced by a FreeBSD router (v.2.2.8 or 3.1 or above) which integrates `dummynet` version 2 [1]. This router should be between the client(s) and the server. For example, a delay of 500ms can be imposed on all transiting IP traffic using the following commands on the FreeBSD router:

```
> ipfw pipe 1 config delay 500ms
> ipfw add 100 pipe 1 ip from any to any in
```

The first line sets up an internal packet buffer (pipe) which imposes a 500ms delay. The second line uses the `ipfw` facilities (normally used to create packet filtering firewalls) to route all incoming IP traffic via that buffer.

Assuming that both clients are on one side of the delay and the server on the other then the logical network topology (assuming exclusive use of TCP without multicast) will be:



hiveClient's operation is controlled by the following keys (active only within the 3D graphical window):

- Left arrow – spin left at constant rate.
- Right arrow – spin right at constant rate.
- Up arrow – move forwards at constant rate.
- Down arrow – move backwards at constant rate.
- Space – stop all movement.
- 'r' – toggle time-relative flag for user trajectory updates (i.e. when on, start trajectory time is reset on receipt of update).
- 's' – cycle visual ownership 'show' mode, between 'normal' (displays all objects normally) and 'all' (use different colours to represent owned objects (yellow), requested objects (orange), unowned objects but unlocked objects (bright red) and unowned and locked objects (dark red)).
- 'm' – toggle use of mouse dwell to trigger ownership request (i.e. when on the client will request ownership of objects over which the mouse pauses);
- 'p' – toggle proximity-based ownership transfer (i.e. when on the client will request ownership of objects which are near to the user).
- Escape – give away ownership of any held objects to the world server.
- 'u' – cycle object 'update' mode, between 'normal' (request and obtain ownership first, then issue update), 'central' (do not request ownership – issue only update requests), and 'CIAO' (issue first update as a request in tandem with the request for ownership, as in the CIAO system [2]).
- 'f' – set of a virtual firework (subject to advance communication delay, below).
- 'a' – cycle advance communication time for triggering fireworks, between 0, 0.5s, 1s and 3s.

As usual, the right mouse button can be used to (attempt to) drag the object under the mouse. Double clicking the right mouse button will attempt to pick the object up to carry it; it may subsequently be put down by clicking the right mouse button on the (now highlighted) hand icon at the top-left of the graphical view. However the normal mouse-based navigation is disabled.

Basic consistency strategy

Ownership

MASSIVE-3 uses a conservative consistency strategy based on single transferable ownership for each data item. At any time at most one agent is the owner of a data item, and only this agent is allowed to update or delete the data item. Any agent may request ownership (via a reliable logical multicast request); the current owner may refuse the request, or may honor it by sending ownership to the requestor. An agent's response to requests for ownership are based on the lock status of the owned item as follows:

- an unlocked or "soft" locked item will be given away in response to any request;
- a "hard" or "control" locked item will never be given away in response to a request.

The significance of “control” locks is described in the following section (on support for centralized updates).

Basic causality and event sequencing

Every data item has a two-part state sequencer. The first part of this sequencer is incremented when:

- a reliable (“mandatory”) update made to the item;
- the ownership is transferred to another agent; or
- the item is deleted.

The second part is incremented only when an unreliable (“optional”) update is made to the item. The second part is also reset whenever the first part is incremented.

The system enforces that every mandatory update for a given item is enacted in sequencer order on every process which replicates that item. Optional updates are never applied out of order (instead they are discarded if they cannot be applied in order), and so there can be gaps in sequences of optional updates (but not mandatory updates).

When ownership is transferred the mandatory sequencer is updated and the new sequencer values are part of the ownership transfer event. This ensures that all updates prior to the ownership transfer will be enacted prior to the actual transfer of ownership taking place. Also, all updates subsequent to the transfer will be enacted after the transfer is observed by all observing processes. While an ownership transfer is in transit within the system the item, strictly speaking, has no current owner, since no agent is allowed to issue updates against it.

If an agent is lost from the system then the environment master agent will tidy up item ownership, either deleting or re-owning all items owned (or about to become owned) by the missing agent.

Centralized update support

As well as supporting transferable ownership, MASSIVE-3 also supports centralized updates as an alternative. This approach can be mixed dynamically with ownership transfer (subject to some constraints if updates on a single item are never to be lost).

Centralized updates are available in two forms, one for unlocked items, and one for control locked items. In both cases the agent wishing to update an item does not update it directly (which would require that it owned the item). Instead the agent creates a new UpdateRequest item as a child of the item to be changed. This UpdateRequest includes the desired new value of the parent item.

When an UpdateRequest is created as a child of an unlocked (or soft locked) item, the current owner observes this and generates a new update for the item using the supplied value. The net result is that one agent creates a request for update which travels to the current owner (and all other agents replicating the item); in response the current owner generates an actual update, which travels to all agents, including the requestor. Note that the owner issues updates in the order in which it observes UpdateRequests being created. This order defines a total order for updates to that item. This order is not necessarily the same as the order of absolute times at which requests were created (in particular for requests which originate from different agents). Also, each agent may observe UpdateRequests to occur in a different order (e.g. its own requests are observed immediately, while those of other agents are observed after a communication delay). Only the order observed by the current owner of the item is significant for determining the order of actual updates.

A control locked item will not be automatically updated by its owning agent. Instead, this is a hint to other agents that the item may be changed, but subject to additional constraints imposed by the current owner. For example, consider a virtual object whose creator wishes to constrain to always be placed on the ground plane. This could not be unlocked (soft locked) since another agent could then take ownership and make arbitrary changes to it (including moving it off the ground plane). Instead the owning agent can make the item control locked. Another agent which wishes to move it now creates an UpdateRequest item rather than directly manipulating it. The owner of a control locked item normally monitors the creation of UpdateRequest items; when one is created it can examine the requested new value and make its own update based on this (or ignore the request if it wants to). In this example, the owner would force the updated position to lie on the ground plane, and then generate an update accordingly.

Central updates can be observed by cycling update mode (‘u’ key) to central and manipulating an object. Note the universal delay in local feedback, since every update request is routed via the remote owner. Note also that observers typically see the same delay as the initiating process. Use ‘all’ show mode (‘s’ key) to check

ownership. If necessary disable mouse dwell and proximity ownership transfer (keys 'm' and 'p'), and give away ownership to the item being manipulated (ESCAPE key).

Ownership transfer

For the non-centralized update approach ownership transfer is a necessary pre-requisite to updating an item. As a partial aside, a form of semi-optimistic updates can be presented to the user via subjective display mechanisms available in MASSIVE-3, i.e. the user may see feedback on pending interactions which are not yet guaranteed. However all guaranteed updates are subject to implicit confirmation by holding ownership.

MASSIVE-3 supports two on-demand strategies for ownership transfer, and gives examples of two predictive strategies for ownership transfer. These are described below.

The normal (traditional) on-demand ownership transfer mode requests ownership of an item at the moment when the application attempts to perform an update and finds that it does not own the item. For example, if the user clicks on an item with the right mouse button then an ownership request is immediately generated. If the request is satisfied then the user client can then issue the update; otherwise the interaction fails. Subsequently, as long as that agent retains ownership of the item it can issue updates immediately with no additional delay (c.f. centralized updates). The ownership transfer requires one round-trip time from the requestor to the current owner and back. This can be seen by disabling mouse dwell and proximity-based requests (keys 'm' and 'p') and cycling the update mode to normal ('u'). Note: use the escape key to give away ownership in order to repeat the test.

The CIAO system [2] has introduced an alternative on-demand ownership transfer approach which combines the ownership request with a request for the first update to the item. When the current owner receives this combined request it can (in a centralized style) choose to issue the first update and then immediately transfer ownership to the requestor. This means that by the time the ownership has been received by the requestor its first update has already been distributed (rather than now being sent). This allows other processes to observe the first update more rapidly than with the traditional scheme. This can be observed by cycling the update mode to CIAO ('u') and disabling mouse dwell and proximity-based requests (keys 'm' and 'p').

A significant feature of Reading University's approach to consistency is to exploit advance or predictive ownership transfers to ameliorate the effects of network delay. In particular, the initial feedback delay on first interacting with a object in the on-demand schemes (above) is due to the round-trip for ownership transfer. In an application can reliably predict the need for ownership then it can request ownership in advance of its use. This general approach has to be realized in application-specific manners, appropriate to the tasks being performed and context in which they occur. MASSIVE-3 illustrates two predictive strategies, described below.

First, the user normally makes use of the mouse to drag and pick up objects, both of which require ownership of the object. The system monitors mouse movement within the 3D view, and when the mouse pauses for a pre-determined length of time (e.g. a few tens of milliseconds) then the client can request ownership of the item under the mouse, in case the user is about to manipulate it. This option can be enabled using key 'm' – mouse dwell request.

Second, the client can automatically request ownership of items which come close to the user. This is enabled with key 'p' – proximity request. The current example uses a relatively small volume in front of the user as a heuristic to estimate objects with which interaction is "likely".

The disadvantages of predictive ownership transfers are as follows:

- the manipulation may not actually take place, in which case the transfer represents wasted work and bandwidth;
- the item cannot be updated while ownership is in transit, and another agent which wants to update the item will have to wait, even though the requestor may not actually use the item;
- if the network delay is long compared to the degree of anticipation then it might have been better to use the CIAO-style of combined update and request, which requires that ownership not be transferred in advance.

Global time versus local time

Another technique advocated in Reading's work is the use of globally synchronized clocks to enhance consistency between behaviors (time-linked activities) on different machines. MASSIVE-3 supports simple

clock synchronization with each environment. The `hiveClient` application has two examples which allow comparison between using global time and local time:

- user movements are time-linked trajectories; and
- the user can set off “fireworks” which explode in the graphical view (a time-varying transformation) (the ‘f’ key).

The ‘r’ (relative time) flag toggles whether these are linked to global or local time, as follows:

- In local (“relative”) time, each time-based behaviour is started from the beginning when it reaches an agent. Typically (with zero delay jitter) each observing agent will see the same complete trajectory, uniformly delayed by the time which it took to reach that agent. Thus the originating agent’s view will always be ahead of the observing agent’s view.
- In global (“absolute”) time, each time-based behaviour can be thought of as continuing to evolve even while the message is in transit. Thus, the user’s movement or the firework’s explosion will already be part-way through when the event is received. So the observer will not see the beginning of each non-deterministic behaviour, but when it does learn about the behaviour it will then see it at exactly the same stage of evolution as every other agent which knows about the behaviour (including the originator).

In this basic form, the global time version is typically subjectively more disconcerting than the local time version. This is especially true for user movement with (relatively) long network delays, because the user will also overshoot the end of each trajectory, before the correction arrives.

Use of global time typically needs to be enhanced by combining it with one (or both) of two additional techniques:

- Advance communication (described in the next section); or
- Delay-based individual perception filtering (sometimes referred to as “virtual relativity”).

These have both been expounded at the University of Reading, although the latter is outside the scope of the HIVE project. A Perception filter maintains an underlying notion of global time, but presents to each user a systematically delayed view of the world, where each object’s presentation is delayed according to the underlying communication delay to its owner. This gives an effect very like relative time, but within a unifying temporal model which can support additional reasoning about events and interpolation between owners.

Advance communication.

So far we have considered non-deterministic events which (ideally) cause immediate changes in the virtual world. For example, the user presses a key or a mouse button and expects an immediate change. If, however, we can introduce a systematic delay between non-deterministic input and its effects in the world then we can make use of advance communication, as supported by MASSIVE-3.

Technically, MASSIVE-3 allows every event in an environment to be marked with a minimum enactment (delivery) time; the event will then be delayed within the receiving system (according to the globally synchronized clock) before being delivered. This delivery constraint is in addition to the causal constraints (using item sequencers) described elsewhere in this report. Thus an agent can determine that some event must take place at some time in the future. It can issue the event with this minimum delivery time, the event is then propagated through the system but has no effect until the specified time is reached. At that global time, every process which has received the event will deliver it simultaneously. So, if an event is delayed by a time greater than the time it takes to reach every process then all processes will observe the event and its results simultaneously, in the same global time.

Reading have prototyped advance communication schemes which apply to user navigation and interaction with objects; these require the user to commit to actions in advance of their effects. We have included one example of advance communication in `hiveClient`, which is a kind of virtual fuse for the fireworks already described. In particular, a range of advance communication times can be selected (key ‘a’, cycling between 0, 0.5s, 1s and 3s). In this case, pressing ‘f’ – the non-deterministic input – generates an event with a minimum delivery time an amount in the future as specified. If this time is greater than the communication delay then every process will see the firework synchronized in global time. If the advance time is less than the delay then the effect of global/local time divergence will be reduced but not removed, i.e. the firework will be later on the observer (local time), or the start of the firework will be missed by the observer (global time).

Sufficient causality

The final consistency idea from Reading which is implemented in MASSIVE-3 (but not readily visible in this application) is sufficient causality. Each event (e.g. item creation, update, deletion) in MASSIVE-3 implicitly identifies the full set of events which must precede it; it does this by specifying required values for item sequencers. By default, an event specifies the sequencer values only for the item to which it applies; this enforces the order of operations on a single item, but allows the order of operations on different items to be different as viewed by different observers. In addition, the application initiating an event can specify an arbitrary number of other object sequencer values which it must also follow. In this way, the ordering constraints of the application are made explicit to the communication infrastructure, and can be enforced. However, the infrastructure need not enforce any additional (unspecified) constraints, and reorder events accordingly (e.g. to take account of advance communications which are awaiting delivery).

References

- [1] Luigi Rizzo, http://www.iet.unipi.it/~luigi/ip_dumynet/ (tested 24th August 1999).
- [2] Un-Jae Sung, Jae-Heon Yang and Kwang-Yun Wong, "Concurrency Control in CIAO", Proceedings of IEEE Virtual Reality, March 13-17, 1999, Houston, Texas, IEEE Computer Society.