Persistent Inventory on distributed VR systems: A possible Solution

Boroondas Gupte (Second Life Resident)

September 9, 2007

Allthough this might look like a scientific document, it isn't. I just took my post[1] *on sldev-misc and thew some LaTeX at it.*

Contents

1 Assumptions

1.1 SIMs

Let's assume an online VR solution where different parts of the world(s) are being simulated by different servers (let's call them SIMs, for some reason ;-)). Whether these SIMs are connected side by side to populate a map or are only interconnected by portals (or not interconnected at all!) isn't relevant here. Nor is if they are on a common grid (whatever that might be, anyway). Just assume they're being run by different people of which users might trust some fully, some partly and some not at all.

1.2 Not-so-distributed data storage

What we want is that, from wherever a user logs into whichever SIM using the same account, he should have the same assets at hand. (Assets being what they are on SL, and perhaps even more) We can't put them on single SIMs, because they might not be persistent themselves, as well as not fully trusted. Also, those running a SIM probably wouldn't like to have to pay on bandwidth just to deliver content to others' SIMs.

We also can't put them on the users computer, because the user will want to have access to them when he logs in from other locations and because some of his assets might be required by other users while he isn't even logged in. (You see that I assume that every asset has a clear owner. Although I'll come to some permission management later on, this won't be anything like permissions in SL, as the owner will always have full permissions on his assets.)

1.2.1 No Peer-to-peer

We (well I, at least) also don't want to rely on the users seeding their assets to some (custom or already existing) P2P thingy, so they can later have access to them from other machines, as well as allowing others access while they're not online. Apart from just too much things that can go wrong here, the bandwidth needs for this might be too much,

[1] http://lists.daleglass.net/pipermail/sldev-misc-daleglass.net/2007-September/000004.html

if you want to run the VR-Simulation at the same time.

2 My Solution

2.1 Asset hosting

So let's put them on a separate asset server. Of course, this asset server could be one of a central authority, like it is in SL. But I think we shouldn't require that:

- Asset servers could run
  - on the machines of commercial Asset Hosters
  - on some SIMs
  - on the same machine the user's client runs on [2]
  - on the user's or one of his friends' own/dedicated server (if any)

  and different users might choose different solutions, fitting both their needs and what they're willing to pay.
- If there are several competitors for Asset Hosting, this might improve quality, as users will choose those with low (or no) inventory loss, low response time and high bandwidth, low prices and much per-account disk space.

Like with email providers, there might be people who get their asset account as part of a package with some different product, like from the hoster of a SIM or from their ISP, or even from their email provider maybe. Few(?) others would choose to run their own asset servers, like today some people run their own email servers.

2.2 Access to the user's asset repository

Now, how could all that work technically? The user would have full read and write access to their asset account via their client (after some authentication, of course). So if a user rezzes something on a SIM, the SIM could request the something from the client who, in turn requests it from the user's asset server, just to then pass it on to the SIM for public display to other users there.

As the users bandwidth might be more limited than the asset server one's, this is rather suboptimal. So let's give the SIM read access to the asset account. The client would tell the SIM which asset server to use for the user, either as part of the login data when joining the SIM, or when something's to be rezzed. On rezzing, the SIM would then request the required data from the user's asset server and display the item.

2.2.1 Limiting the access

Earlier on I mentioned not all of the SIMs might be equally trusted by the user. So the SIM should only be able to request data it needs to get, and no more. There would be three states for every asset:

- public
- active
- private

public These assets could be requested without authentication by anyone and anything, be it a SIM, another user or something else. This state would have to be set manually by the user via his client, which would then tell the asset server, which assets to put in this state. The assets would stay in this state until manually set non-public.

When writing a script referencing assets, the scripter would want to set the referenced assets public, so everyone running the script would have access to them.

active Non-public assets that are to be displayed inworld would be in the active state. This state could automatically be set by the client. Avatar data, including clothing and attachments would be active while the user is online and using them. When the user changes hit outfit, some assets would get private while others get active. Items rezzed inworld would stay active until being unrezzed. So they'd sometimes stay active even while the user's not online.

Other than the public state, the active state would be maintained by the asset server not only

[2] this would require to either always be connected to the internet or to not to have any assets you want others to grant access to while you're not

per asset *but also per SIM*. So the SIM would have to authenticate to the asset server to request those assets. Apart from SIMs, some assets could also be active to other users (and their clients), for example while you transfer something to someone else.

private Everything not public or active would be private. Only the user's client would have access to them. This would include about everything that isn't currently in use on a SIM or by another user and that the owner didn't decide to make public. Script sources will always be private unless set public or active by the user. Script binaries, which will be required to be active or public to run, would be different assets.

3 Protecting intelectual property—or not

3.1 Licenses and other meta data

Whoever has access to an asset *can* do whatever they wish with the copy the retrieved. This doesn't mean that they also *may* do so. So let's add some meta data to all assets. Apart from the assets name, date of creation, creator and owner, which would only have informational purpose, there could be a machine readable usage license for the asset. It could say (when translated to human readable form) something like "This texture might only be referenced by scripts owned by user XYZ" or "This texture might only be referenced by scripts created by user XYZ". Off course one could also think of "This script source is governed by GPL v2" or "This prim is public domain, all rites reversed" and the like.

SIM and client implementations could then help the users and those running SIMs to keep to such licenses, but would not technically enforce it. (DRM doesn't work for free software, does it?)

3.2 How to protect your content

Some simple (and probably obvious) rules follow for users who'd like to protect their content:

- Don't set anything public if you don't trust *everyone* to comply with your license on it.
- Don't wear or rez items on SIMs where you don't trust the SIM owner and (to some extend) the other users on the SIM to comply with your license on those items.
- Don't transfer anything to anyone who you don't trust to comply with your license.

Except of course, you don't mind them (or some of them) to disregard your license.

3.3 Legal issues

Of course, if someone disregards your license *and* thereby brakes applicable copyright law (which might not always be the case), you could take legal action against them, like you can when someone grabs the pictures from your website and uses them on their own without permission.

3.3.1 Some more meta data

Technically, owners of assets, as well as everyone else who can get hold of them, *can* change their meta data including the license before passing them on. If they also *may* do so (now from the legal point of view), depends on the license, of course. Creators could digitally sign their works, including the meta data, so users can be sure the license attached is unaltered. Digital watermarks might help to prove who is the original creator of something.

However this would mostly help the ones using the item, as they'll be able to prove the licensor put the asset under a specific license if he insists not to have done, later. Nothing would hinder anyone in hold of the asset to remove or replace the license data, together with the original signature.

4 Save saving

Now, how to save something inworld back to the inventory, either by its owner or by anyone else if the license allows (or is disregarded `;-)`)? In the setup where only the owner's client has write access to the account's asset repository, the client would have to request a copy of those parts of the inworld asset it hasn't cached locally (e.g. from viewing them). Then the client could upload them to the asset server.

Considering that writes (saving changed assets) will be much less frequent than reads, we could as well stay with that. But we can do better. It should be save to assume that the client's upstream bandwidth is the bottleneck: people running SIMs will want to make sure they have enough bandwidth for doing so, as well will those running asset servers. The client might have an asymmetric internet connection with much higher downstream data rates than upstream. This is the case e.g. on ADSL connections as well as most satellite ones.

4.1 Temporal writing

So let's allow the SIM to do temporal writes on the asset repository that are just being canceled if not approved within some given time.

Then, whenever the client request saving something, the SIM would send the required data to both, the client and the asset server. Which data is required might be different for the client and the asset server, depending on what data they already have. The sent data and the data they already have would than be combined and hashed by both, the client and the asset server. The client would request the asset server to either deny or allow the write, identified by the hash. If the hashes match, the asset server would make the write permanent (in case of 'allow') or withdraw it (in case of 'deny'). If they don't match, the temporal write would stay temporal until it times out, and the client would be informed about that by the asset server.

Of course the client (and its user) can't rely on an arbitrary SIM to allow him to save even his own things, but as there won't be anything like no-copy assets, this shouldn't be a too big problem.