Posted in: Development (/spaces/127/sandpiper/forums/5366/development)

How to tell if a Slice has changed?

Unsubscribe from this discussion
Subscribe to RSS (../../../../../../spaces-cf/forums/rss-space-posts.ashx?spaceID=127&topicID=5395&forumID=5366&key=rrer%2B1xDo%2BlxpC7jBRbM5w%3D%3D)

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
12/15/2019

In the current design, a sync happens at the `slice` level. The process is for a subscriber to first request a list of object-ids for a slice, then determine the ones they don't have, and finally, request the missing ones.

The current thinking is that a subscriber (client) will periodically "poll" the publisher (server) for changes. This polling could be an expensive process if it means asking for all object-ids every time.

One way to know if something has changed is by using a hash function (https://en.wikipedia.org/wiki/Cryptographic_hash_function) (a mathematical algorithm that changes data of arbitrary size to a fixed, usually much smaller, size). Since a change in underlying data creates a new hash value, we can store a hash-value with the slice to represent all the data contained in that slice. This provides a simple method to see if a slice needs syncing. Excellent!

It turns out sha-1 (https://en.wikipedia.org/wiki/SHA-1) (which generates a 20-byte string) is a very good hash function for our purposes.

But what is the best way to calculate this hash-value when it is made up of lots of separate chunks of data?

Since we already have a unique-id assigned to each data-object, there's no need to hash the actual payload-data (which is probably much larger than the id). So it seems like we could simply concatenate all the object-ids and pass that string to the hash-function to calculate a master hash

value (stored with the slice).

This also means that every time we add or delete a data-object, we need to perform that master hash calculation again. This may, or may not, be a performance problem. We will wait and see.

Note: As an added precaution for the sync, it might be a good idea to store an `object-count` for the slice as well. So "polling" would simply be a request to the server for a list of (`slice-id` , `slice-name` , `slice-hash` , `object-count`) for each of my slices.

hash-function (/spaces/127/sandpiper/searchresults?keyword=%22hash-function%22&searchtags=1)
polling (/spaces/127/sandpiper/searchresults?keyword=%22polling%22&searchtags=1)
sync (/spaces/127/sandpiper/searchresults?keyword=%22sync%22&searchtags=1)

Like

Reply (/forums/post?tid=5395&ReplyPostID=5396&SpaceID=127)

Luke Smith (https://autocare.communifire.com/people/autopartsource)
12/18/2019

I think the hash input should simply be all of the slice's UUID strings concatenated together with no delimiters. The hash could be used by a subscriber in two different ways. 1) The subscriber could keep track of slice hashes from one polling check-in to the next. This would not require much subscriber-side database interaction or processing. 2) The subscriber could hash the slice's object UUIDs from its local database and compare the result to the server's reported slice hash. This second approach would be more resource intensive but would guarantee synchronicity.

We should also avoid locking in a specific hash algorithm since it may fall out of favor and be removed from software libraries. Practically any hash algorithm will work for our fingerprinting purposes - it does not have to be "cryptographically secure". SHA1 would be a good one to recommend. It will be around for a while even though the crypto kids have shunned it. I suggest we add a "hash-method" element to the slice meta-data. It would indicate how to assemble the input data and which algorithm to use.

Doug, you touched on a fundamental concept that is worth re-enumerating here. The obvious question a neutral observer will ask is this: "Why don't you skip the convoluted object UUID thing and simply base the whole sync-change-detection system on hashes of the content itself?"

This was actually my initial thought on how we should approach change detection. Here are the "gotchyas". In our loose affiliation of players in this ecosystem, there's no agreeing on what exactly constitutes an "application" or "item", etc. We would have to define a one-size-fits-all approach to hashing specific elements of the content. Hashing every *significant* element of an

ACES App, PIES Item, etc would become a constantly moving target for the standards to keep up with. Adding a new element to ACES or PIES (or worse, a third-party standard) would mean that someone (ACA) would have to define how the new element gets sequences into the hash input. It would also mean that all the implementations in the wild would have to be re-tooled (sender and receiver) to hash properly in locked-step. Another issue is that element significance is in the eye of the beholder. We may have different takes on what changes to elements in an App or Item warrant a re-sync. Does a spelling correction in an item's description warrant a fingerprint change and therefore re-syncs? How about a capitalization correction?

By using the UUID, the authoring PIM gets to make that choice. If the PIM changes something on the source record that it considers substantive, it also changes the UUID.

Like

Reply (/forums/post?tid=5395&ReplyPostID=5399&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
12/18/2019

You've brought up several points. Here are my thoughts:

1. Input to the Slice hash function. I think we're in agreement if you're saying the hash function should calculate the `slice-hash` based on a concatenation of all related object-ids. But why are you saying they shouldn't be delimited? I'd usually put a plus sign or vertical bar between them (more for debugging than anything else).

On the client side, I've also been considering whether or not the client should recalculate the slice-hash each time, or simply store the slice-hash from each sync and then use that saved value for sync comparison. This latter choice does feel a lot like a time-based sync. I think maybe we should code it initially using a full calculation approach and then see if we have a performance problem.

2. Add a "hash-method" element to the slice meta-data. In a completely closed system (where we control all access to the data), this shouldn't be necessary. The question is whether or not Sandpiper is a black box. I think a well-defined (and complete) API should allow it to be closed. This means that even though an external PIM is responsible for authoring data, the API handles all access to the database.

Since we use API versioning, we could change the hash function between versions if we felt that was necessary. If we make the hash function configurable, why stop there? Why not also make the payload encoding scheme configurable? I think it just adds unnecessary complications (unless we need to support multiple schemes from the start.

That leaves your recommendation for a configuration that indicates "how to assemble the input data" (for the hash function). What different approaches do you see here? Is this something that might require supporting more than one method from the start?

3. Using a unique object-id instead of an object-hash. I agree with this concept and now even more after reading your justifications.

hash-algorithm (/spaces/127/sandpiper/searchresults?keyword=%22hash-algorithm%22&searchtags=1)
hash-function (/spaces/127/sandpiper/searchresults?keyword=%22hash-function%22&searchtags=1)
hash-method (/spaces/127/sandpiper/searchresults?keyword=%22hash-method%22&searchtags=1)
sync-process (/spaces/127/sandpiper/searchresults?keyword=%22sync-process%22&searchtags=1)

Like

Reply (/forums/post?tid=5395&ReplyPostID=5401&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
12/18/2019

Delimiters would be a reasonable compromise for debugging. I was just trying to conserve bandwidth. Commas would be the best choice.

You make a strong point about complexity. Let's go with SHA1 until further notice.

On the subject of blackboxedness: Hopefully, PIMs will quickly start to incorporate native Sandpiper features. They will implement Sandpiper from the ground-up. The PIM will be the gatekeeper between all local databases and other Sandpiper nodes. The only guaranteed closed system is the reference implementation that we build.

My comments about "assembling input data" was was purely about the path we are not taking - the hash-the-content-directly approach.

Like

Reply (/forums/post?tid=5395&ReplyPostID=5402&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
1/15/2020

> **On 12/18/2019 1:43 PM, Luke Smith said:**
>
> Hopefully, PIMs will quickly start to incorporate native Sandpiper features. They will implement Sandpiper from the ground-up. The PIM will be the gatekeeper between all local databases and other Sandpiper nodes.

I really doubt this will happen anytime soon. Why would they duplicate the sync process if that's handled well in an open-source way? There's a lot of complexity there.

I'll be happy just to see PIM's take over the role our CLI tools provide to create grains in the primary pools.

Like

Reply (/forums/post?tid=5395&ReplyPostID=5540&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
1/15/2020

I guess you are thinking more about the our specific open-source implementation, and I'm thinking more broadly about other implementations. In my nirvana scenario, there would only be the two PIMs and no middle-ware. The two PIMs (made by different solutions providers) would quietly whisper Sandpiper to each other day and night without ceasing.

Like

Reply (/forums/post?tid=5395&ReplyPostID=5541&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/11/2020

The final design includes both grain ids and slice metadata in the hash calculation:

https://autocare.communifire.com/spaces/127/sandpiper/forums/development/5986/hash-calculation-now-includes-slice-metadata (https://autocare.communifire.com/spaces/127/sandpiper/forums/development/5986/hash-calculation-now-includes-slice-metadata)

Like

Reply (/forums/post?tid=5395&ReplyPostID=6099&SpaceID=127) Answer

Adrian Parker (https://autocare.communifire.com/people/aparker)
8/12/2020

Did you consider what would happen if you concatenated and hashed the following two records:

Column 1	Column 2
1	11
11	1

... You'd get the same hash for both records. This is an oversimplified example though, I don't know if the equivalent can happen in your data (it almost certainly can).

And if you do use a delimiter, you must encode it in your data. The HTML encoding method I find is the simplest way to do this.

Like

Reply (/forums/post?tid=5395&ReplyPostID=6103&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)

8/12/2020

I do normally delimit values before a hash for the reason you describe, but in this case, the values are UUIDs (for the first part) and so would not need delimiting (all the same length).

The second group of values to hash are key/value pairs, and since the keys are unique (and from a pre-determined list), I didn't think it would be necessary to delimit them either.

Here's the logic I used to create the value to be hashed (`b` is a buffer used to build the value):

```
var hash = func(meta sandpiper.MetaArray, ids []uuid.UUID) string {
 if len(meta) == 0 && len(ids) == 0 {
  return ""
 }
 for _, u := range ids {
  b.Write(u[:])
 }
 for _, m := range meta {
  b.WriteString(m.Key)
  b.WriteString(m.Value)
 }
 return fmt.Sprintf("%x", sha1.Sum(b.Bytes()))
}
```

Like

Reply (/forums/post?tid=5395&ReplyPostID=6105&SpaceID=127) Answer

Adrian Parker (https://autocare.communifire.com/people/aparker)
8/12/2020

This looks like Go?

Like

Reply (/forums/post?tid=5395&ReplyPostID=6106&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/12/2020

Yep, golang was chosen for the "reference" implementation:

https://github.com/sandpiper-framework/sandpiper (https://github.com/sandpiper-framework/sandpiper)

or, more specifically, here (https://github.com/sandpiper-framework/sandpiper/blob/b394e74050983109469912b9c7e3167d77d22530/pkg/api/slice/platform/pgsql/slice.go#L330).

Like

Reply (/forums/post?tid=5395&ReplyPostID=6109&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/12/2020

Here is more on our decision to use Go:

https://autocare.communifire.com/spaces/127/sandpiper/forums/development/5368/technology-stack (https://autocare.communifire.com/spaces/127/sandpiper/forums/development/5368/technology-stack)

Like

Reply (/forums/post?tid=5395&ReplyPostID=6110&SpaceID=127) Answer

Copyright © 2021 Axero Solutions LLC.
Auto Care Association powered by Communifire ™ Version 8.0.7789.8960

© 2021 - AUTO CARE ASSOCIATION (http://autocare.org) | LEGAL & PRIVACY STATEMENT (https://www.autocare.org/privacy-statement/)