Here are some initial thoughts on reworking the Survey Tool backend.

Contents

The problem

It is clear that the survey tool needs major performance and reliability improvements. For example, just checking now:

- ~2 users, 226pg/uptime: 1:55:00/load:33%
- Takes about 1.5 minutes to open a new locale (Croatian).
- Takes about 2.5 minutes to open the vetting view.
- Takes about 0.4 minutes to open a zoomed item view.
    - Now, these are first times; subsequent zoomed views seem to be quite fast.

It also places quite a load on the Unicode server, and doesn't scale well to lots of users. And we need to reboot very often. So here are some thoughts on a possible re-architecture.

Data

Here is roughly the data and structure we need. But this is from the outside: Steven is really the one who would know all the guts.

pathId → path,
path → pathId
// the path→pathId using StringId is algorithmic and immutable. Both can be stored in memory to optimize.
// we could use PrettyPath, but the downside is that we have to update that every time we change the DTD

voter → organization, voterLevel (eg VETTER), authorizedLocales
// relatively constant data.

pathId → valueInfo+
// ordered by voteCount then UCA (so first is winning, second is 'next best')

valueInfo = value, isInherited, coverageLevel, voteCount, voter*, errorStatus*, example?

// that is, a value like "Sontag", whether the value is inherited, what the coverage level is (computed algorithmically), what the voteCount is (computed from the voters: computed and cached), the errorStatus (computed and cached), and the example text (computed and cached). Maybe add dependentPaths (see below).*

errorStatus = error/warningID, message

value → pathId*
// used for computing display collisions

staleLocales → locale*
// used for updating the cache

Operation

1. When a user votes for a value, the valuesInfos are re-sorted if necessary.
2. When a user adds a new value, all the values, including errorStatus and examples, are computed. The valueInfo is added to pathId→valueInf+, and the pathId is added to value→pathId*. (If a value is deleted, then the corresponding entries are removed.)
3. In either of these cases, if the first (winning) value changes, then the locale and its children are added to the staleLocales queue.

StaleLocales Queue

A (logical) "LinkedHashSet" of locales (but any child locale is automatically repositioned after its parents).

1. A separate process walks through the queue, processing and removing locales (see Issues).
2. It walks through all of the pathIDs for a locale and recomputes the errorStatus and example.
3. If ever the locale or its parents are added to the queue while it is processing that locale, it restarts.

Issues

Issue: We could precompute the dependencies on paths, which are static. Then if the winning value for path x changes, then we know just those paths that may need to change the error status and/or examples, and don't have to walk them all. Note that changes to English or root

may need error checking to be redone everywhere.

Issue: We could make root and English completely frozen, since the TC is responsible for all changes to them. So the items would need to be updated with a manual data update. *Added 4005*

Issue: Host on http://code.google.com/appengine/docs/whatisgoogleappengine.html ? Would allow for better scalability, robustness, etc. Take load off of Unicode server. *We're investigating this at Google.*

Issue: the Voter map changes occasionally. For new users, we don't have to do anything. The only real change is if the voterStatus changes. In that case, we need to revisit all of the authorizedLocales. Because it is very infrequent, it is probably ok not try to optimize (eg not keep a back map of voter→pathId*).

Issue: With multiple machines (or app engine) we could shard the processing; divide up the locales by base language, and divy them out to different machines. (Clumps would have to be slightly larger where we have sibling aliases.)