

List of Basic Use Cases for the SPP

Use case 1, initial au distribution: Given a schema instance describing h hosts and n au's, generate LOCKSS "requests" (config file sections) , such that altogether, (a) for each au, there are at least k hosts that harvest it, (b) for each host, the sum of the max size of all au's harvested is less than the storage commitment offered by this host.

"Flow of control: schema instance is updated, the schema server processes the instance and sends requests to the hosts, hosts accept, hosts start harvesting from original publishers"

"Note: the "requests" will be implemented using the new LOCKSS "batch au update" request capability. This will provide an API for a content provider to request that LOCKSS nodes add AU's to their harvesting configurations. We will need to test this for our purposes, and follow up with an RFE to the LOCKSS team specifying if we need additional functionality. (E.g., does this feature allow the node administrator to configure the node to automatically accept all requests from particular requestors?)"

Use case 2, adding additional au's: Given a schema instance describing h hosts and $n+j$ au's, *and* an existing assignment of n au's to hosts (as would be a result of use case 1), generate additional "requests" for the additional au's, *keeping the n assignments fixed*, and satisfying 1a,b for the additional au's

Use case 3, adding additional hosts to 1: similar to 2, but easier, since by construction, replication conditions are already satisfied

"Note: Use case 3 does not require any requests to be sent. It probably involves only an audit check (see UC8) to verify the commitments of the new hosts."

Use case 4, replacing hosts: given a schema instance describing h hosts and n au's, an existing assignment of n au's to hosts, a set h_d of hosts being deleted and h_a of hosts being added, such that $\#h_a \geq \#h_d$ for each h_di , $\text{sum}(\text{resources}(h_di)) < \text{sum}(\text{resources}(h_ai))$; generate a additional "requests" (config file sections) for hosts *all* hosts,

- keeping the au assignments fixed for hosts in $h-h_d^*$, such that 1a,b

are satisfied.

"Flow of control: new hosts are added, schema server is updated, the schema server processes the instance and sends requests to the hosts, hosts accept, hosts start content transfers, old hosts are removed"

"Note: This assumes that each host inherits previous harvesting commitment. If not, there are two related issues to contend with. First, reharvesting from the publisher is not sufficient -- the nodes to be replaces may have previous versions of content that require retention. Second, if a replacement nodes needs to take responsibility for previous versions of content it did not harvest,

it needs a special mechanism to obtain that content -- since it doesn't have the credentials to request a restore from the LOCKSS peers. The special mechanism could be use of SSL certs and trusted hosts. Also see "Flexibility" below and "Locks Trust Model Constraints" below on content restoration"

Use case 5, au exceeds max growth: like 2, but au grows above its max promised space. Must indicate a new max space, determine whether (a) at least k hosts already harvesting au have sufficient resources for new size; (b) if not a, then can additional hosts in the network be designated to harvest au, and some existing hosts stop harvesting *while retaining older files*; (c) if not (b) minimum #/size hosts that would have to be added

"Flow of control: schema instance is updated, the schema server processes the instance and sends requests to the hosts, hosts accept, hosts start harvesting from original publishers"

Use case 6, recovery: Given an au identifier and a date, deliver a copy of that au, as of that date

"Note: Different possible ways to restore content, see LOCKS trust model constraints below. "

Use case 7, host failure: host h_i fails and is replaced with a new host of equal or greater capacity but no content. No reallocation necessary, update schema and audit.

"Note: We are assuming that the credentials and configuration of h_i is saved in some way outside of the system and is restored onto the new node. Standard LOCKSS mechanisms would take care of the content restoration of the previously harvested AU's, but the schema server would want to update its records of commitments, and perform an audit to make sure restore was successful. Alternately, we might require that either (a) the schema server be able to reissue a complete set of "requests" to the restored node or (b) the configuration of each server be exposed for harvesting by other nodes in the network. "

Use case 8, auditing: Given an au identifier, report: (a) any lack of compliance between the replication policy described by the schema and the actual state of the network (b) more detailed information such as (i) which hosts hold that au; (ii) historical/most recent (iii) min/max date of modification of au as indicated by original source; (iv) harvesting dates; (v) replication dates; (vi) external verification dates; (vii) historical/most recent au sizes (viii) historical/most recent au file manifests

For example, suppose the schema states that there should be 4 copies of all AU's updated daily, a successful audit will discover 4 copies of each AU in the schema, and check that each has been updated no longer than 24 hours ago, and report that the network complies with the schema.

"Note: Auditing will be based on the "cache manager" created by the LOCKSS team, and currently in pre-release state. The cache-manager currently gathers information from LOCKSS nodes, stores this in a SQL database, and generates reports. We will need to set up a cache server, and generate additional reports for our specific auditing needs. Also, currently this requires privileged access to the LOCKSS nodes. The LOCKSS team is planning to extend the nodes to support read-only access to allow the cache manager to function without full privileges.

We should follow up with the team on timeline for this, and on obtaining a cache manager instance"

Design and Implementation Constraints

LOCKS Trust Model Constraints

A basic assumption inherent in the use case is that we should be consistent with the LOCKSS trust model. This has at least the following implications:

- Since LOCKSS nodes do not trust single node to provide correct

content: Each node must be able to harvest the au's it is responsible for **directly** from the source declared in the schema -- **not** simply access it from another LOCKSS node. (Although note that trusted hosts authenticated through SSL are permitted to obtain content from other hosts even if they have not had it previously. Still this is intended to be used for restores, not initial harvesting.)

- Since LOCKSS nodes never permit deletion of content
 - content is never deleted from a node in the SSP,
 - content is never deleted from a node in the SSP, unless that node is

removed entirely (in which case it would be replaced and automatically restore its previous content)

- Since a centralized authority that was able to make arbitrary changes

to config files would clearly violate the trust model, adding harvesting assignments is done by "request" -- with approval by the local node administrator and not through a "super user" with privileges on all nodes.

- This will use the new "batch request" functionality provided by the

LOCKSS team

- This request functionality should be configurable to automatically

approve requests to add au's from the schema server, but not to delete or discontinue harvesting. This is necessary to ensure that no single authority can cause the system to violate replication commitments already given to au's

- LOCKSS requires verification that you content was previously held

before a peer will deliver that content back. This implies that content transfer for use cases 4, 6, 7 need to be verified. However, a new release of LOCKSS will allow hosts authenticated by a "trusted" ssl certificate to obtain content without proof of previously holding it.

- For UC7 restoring the original configuration and credentials should

be all that is needed for standard LOCKSS mechanisms to restore the content

- For UC4 standard LOCKSS mechanisms are sufficient if there is a 1-1

replacement of nodes with each node inheriting the AU's of its predecessor.

- For UC6 standard LOCKSS mechanisms are sufficient to restore the

content to the owner if the owner's node harvests their own content. Or if the request is made from a host with a trusted SSL certificate.

Implementation Flexibility and Robustness

- The schema and use cases above intentionally set minimum replication

requirements but not maximum. That is, more hosts could replicate the AU than specified in the schema. This is to allow for some performance tuning flexibility and to allow for nodes with historical copies of AU's not to be in technical violation. In most cases, it is assumed that the SPP will maintain the minimum number of copies unless there is a specific reason to do more.

- For these use cases, we would also need to consider off-the-path

scenarios of the use case, where a request is either refused or not acknowledged. There are at least two ways of handling this:

- Use a two-phase commit, where requests are refused, accepted, and

then reconfirmed (if all requests are accepted), or cancelled, and a new set reissued. This is robust, but complex to implement.

- Assume that a refusal indicates a major error -- misconfiguration,

bug in code, violation of assumption, and that this should cause a human to be summoned to investigate and resolve outside the system. This is less robust, but makes the system easy to implement -- possibly as something as a few scripts and PGP mail. This is our initial assumption -- i.e. that requests will normally be accepted, and any refusals must be debugged by hand.

- We assume that resource changes are monotonically increasing
 - if nodes are replaced, they have more resources.
 - nodes are added, but the size of the network never shrinks

- Auditing and version control
 - The schema instance itself should be placed under version control,

but this is not necessarily a part of the SPP software itself

- The audit log should be preserved
- audits should be run regularly
- The result will be a record of the preservation policy (as embodied

in the schema instance) along with audit logs that demonstrate whether network complied to policy at regular intervals. This is not necessarily enough to debug out-of-compliance situations completely, but does provide proof of compliance.