XAM and pNFS – A Case Study

Peter Cudhea, Staff Engineer

Sun Microsystems, Inc.

ABSTRACT

- Modern filesystems such as pNFS will increasingly include fixed content features, such as queryability, immutable objects, and retention management.
- Starting with a deep examination of the XAM SDK's Reference VIM, which uses simple files and directories to implement the full XAM object model, we delve into the similarities and differences in the fixed-content features of the emerging standards of XAM and pNFS.
- Participants will emerge with a deeper understanding of these standards and how they build on each other, as they do when pNFS becomes a substrate for XAM.

AGENDA

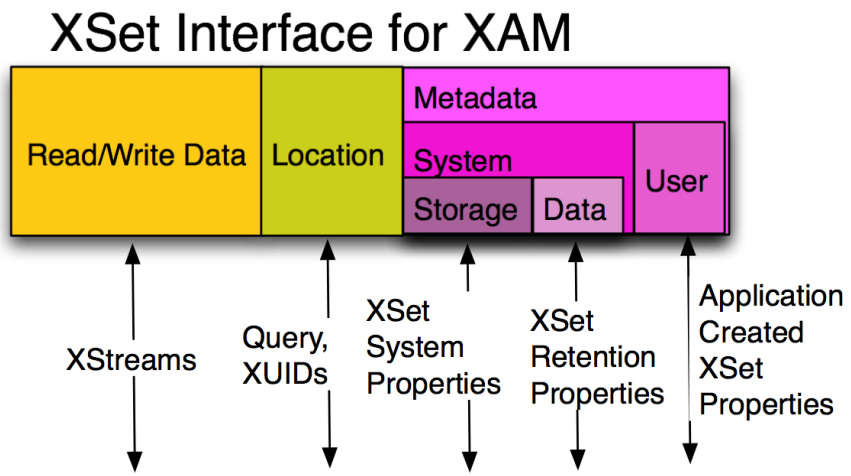
- Review of Fixed-Content Application Desiderata
- XAM Reference VIM
- Other ways to embed XAM in a filesystem
- Exploring pNFS as implementation substrate for XAM
- Fixed-Content Topics for pNFS
  - Immutable Objects
  - Query
  - Retention Management
- Conclusions – what can XAM and pNFS learn from each other?

•Fixed Content Aware Application

- Example Use Cases:
  - Digital Library – Corporate, Institutional, Worldwide
  - Global Scientific Workbook – Data Provenance
- **Immutable** Data with **Evolving** Usage Patterns
  - New query tools and languages
  - New "playback" methods and new applications
  - One-off applications and research questions
- **Rich** object model with **Smart** retrieval (Query)
- Ingest and Retrieve at application speeds (**fast**)
- Embed **Computation** into Storage
- Moving towards **Policy-Driven** Storage and Data management
- What is worth doing now that helps? (XAM, pNFS)

XAM API: A Data Storage Interface

- XAM is the first interface to standardize system metadata for retention of data
- XAM implements the basic capability to Read and Write Data (through XStreams)
- XAM has the ability to locate any XSet with a query or by supplying the XUID
- XAM allows Metadata to be added to the data and keeps both in an XSet object
- XAM uses and produces system metadata for each XSet
- For example Access and Commit times (Storage System Metadata)
- But it also uniquely specifies Data System Metadata for Retention Data Services
- XAM User metadata is un-interpretable by the system, but stored with the other data and is available for use in queries
- Given this we can see that XAM is a data storage interface that is used by both Storage and Data Services (functions)



What XAM Standardizes

- A data model with data streams and properties
  - (XAM Library Object, XSystem, XSet, XStream)
- Rules/Methods for Creating, Modifying, Finding, Using, and Deleting objects
- Rules for migrating objects between systems
- The idea of **binding** vs **non-binding** fields
  - XUID is tied to the values of the binding fields
  - non-binding fields can change in controlled ways
- Plug-in architecture for Vendors (VIM API)

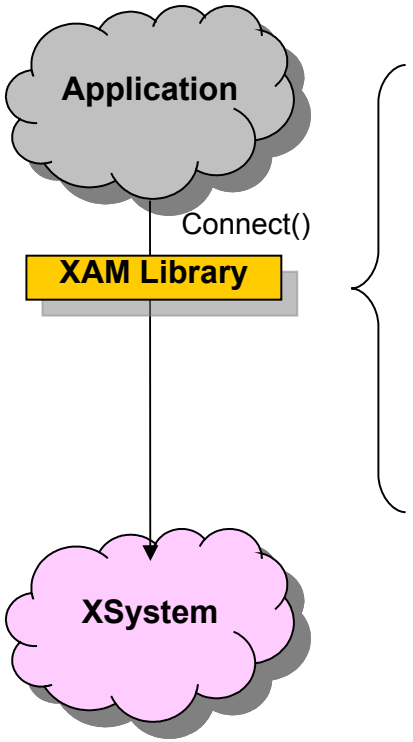
•Standardized Metadata for Policy

- Beginnings of Policy-driven data management
- "Storage System Metadata"
  - Retention, Hold, Deletion, Access Control, Storage
- With some support for reconciling them on import
- per-XSet manual, per-XSet by named Policy for each "slot", or per-XSet by overall management policy
- Vendor-defined services can always be requested in a vendor-defined manner

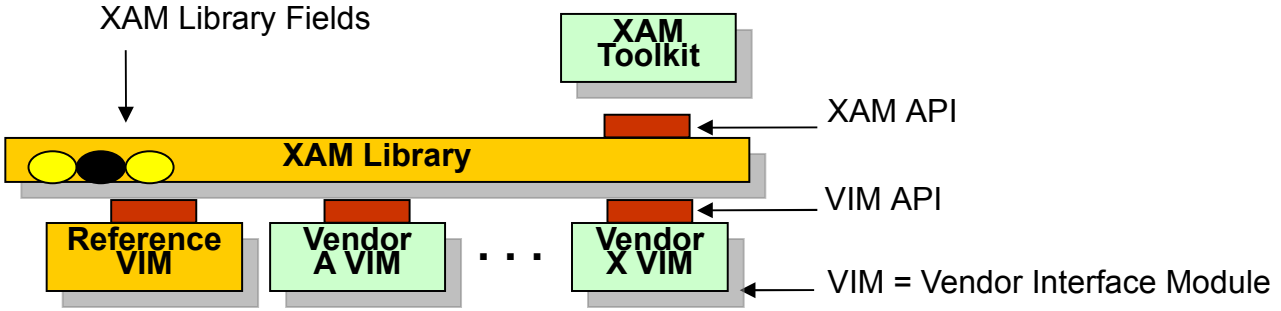
What XAM Does Not Standardize

- Implementation (a logical model, not physical)
  - Where is the storage actually located?
  - Am I talking to one system or several?
- Compare to POSIX API for files, directories, etc
- Preventing collisions (no locks)
- Exactly how to reconcile non-binding fields
  - Like having the **tip** of a revision control system, but without the revision history
  - Details of **merge/resolve** left non-standardized.
- Details of Access Control
  - no owners, groups, user identities, ACLs, etc
- Details of **how** the policies pick actual values
  - But once picked, the actual values stick around.

XAM SDK – Quick Review



The XAM SDK is dynamically linked by each application wishing to connect to and use XSystems.

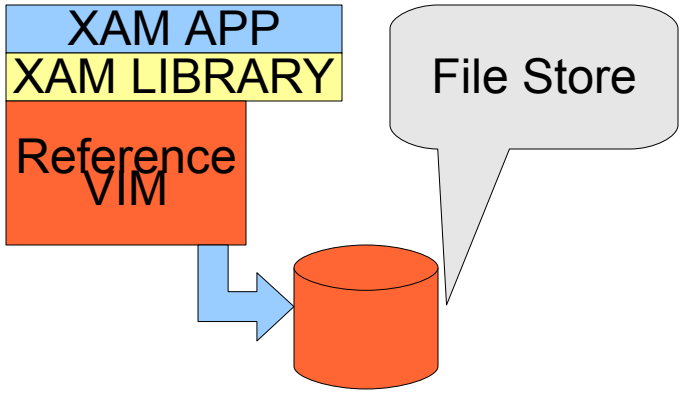


The XAM SDK includes several components:

- **XAM Library**, which implements the XAM API functions
- **Reference VIM**, which implements the reference behavior of an XSystem
- a framework which allows plug-able **Vendor VIMs**
- optional **XAM Toolkit** Libraries for convenience functions

Reference VIM

- Uses basic filesystem concepts
- Obvious construction, to explain and explore the standard
- When in doubt, choose simple
- A "Protocol VIM" for the filesystem?
  - Not really, because standard FS doesn't have Query



Export/Import Stream Format

- Entire XSet as a Stream
- XML document for field names, types, and for property values
- MIME segments for each XStream data value
- Not too useful as a **working** format
- Encodes "actual values" as assigned by Policy

- Example exported XSet

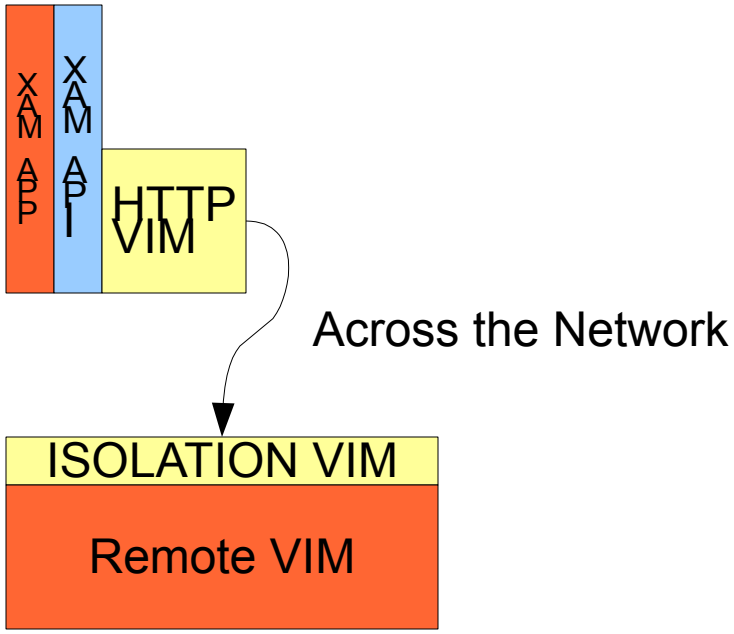
```
<xsets
xsi:schemaLocation="http://www.snia.org/2007/xam/export/XA
MCanonicalXSetDefinition.xsd">
<version>1.0.0</version>
<xset>
<properties>
<property name=".xset.management.policy"
type="application/vnd.snia.xam.string" binding="true"
readOnly="true" length="36">
<string>.org.snia.refvim.default.mgmt.policy</string>
</property>
<property name=".xset.xuid"
type="application/vnd.snia.xam.xuid" binding="true"
readOnly="true" length="21">
<xuid>AAA6AwAVvRAxMjIyMTc3NjMyNTUw</xuid>
</property>
<property name="testBoolean"
type="application/vnd.snia.xam.boolean" binding="false"
readOnly="false" length="1">
<boolean>true</boolean>
</property>
<property name="testDouble"
type="application/vnd.snia.xam.double" binding="true"
readOnly="false" length="8">
<double>1.4</double>
</property>
</properties>
<xstreams>
</xstreams>
</xset>
</xsets>
```

Reference VIM File Embedding

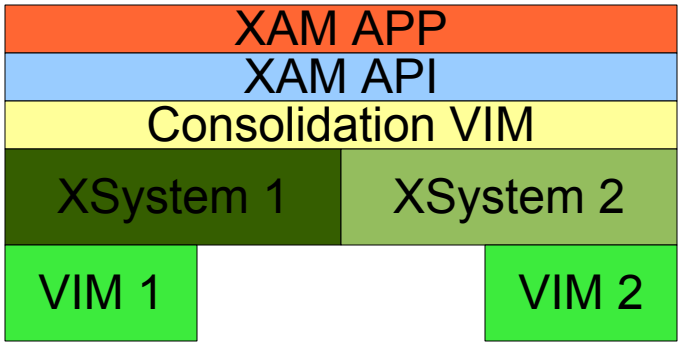
- One file for "Object plus Fields" in Export Format
  - Have to implement that format anyway
  - Simplifies file handling (one base file per object)
- One file for the data portion of each stream
  - MUCH simpler to have each stream separated out
- No sharing of streams yet! (but could and should)
- (picture of XSets with XStreams)

The HTTP VIM – a Stackable VIM

- The XAM Extension Cord



- The XAM Consolidation Pattern
  - (E Pluribus Unum)
- Specify:
  - On Store – where?
  - On retrieve – hint?
  - Periodic callback
- Result: One XSystem



XAM "working copy" embedding

- Consider a Revision Control System again
- Directories, Files, Attributes make a good representation for the "working copy"
- A second ingest/export format?
  - Point to a directory and say, **XAMify This!**
- Different Formats, One Goal
  - XSet as directory? or file?
  - Fields as attributes? or files within directory?
  - XStream data as files in separate "stream table"?
  - ==> All the options make sense, should be tried
- Proposal: XAM object as a File, Each field as an attr
  - Stream data in a separate "stream table"
  - Filehandle without directory is fine
  - XUID is not just the filehandle

Review of POSIX extended attrs

- Exposed as a hidden "directory" associated with file
  - Open one attribute as a file, or open the directory to look up contents
  - The "directory" could be virtual
- Some limitations are useful (one level, no cross links)
- Access to them is standard but not expectations
  - What is the maximum size of an attr?
  - What is the maximum number of attrs?
  - May not have the same performance or scalability as classic files (c.f. pNFS access only through MDS)
  - Java API for extended attrs not yet widely available
- => Use files, not attrs for XAM stream data
- => Don't use attrs at all for Reference VIM

Why pNFS?

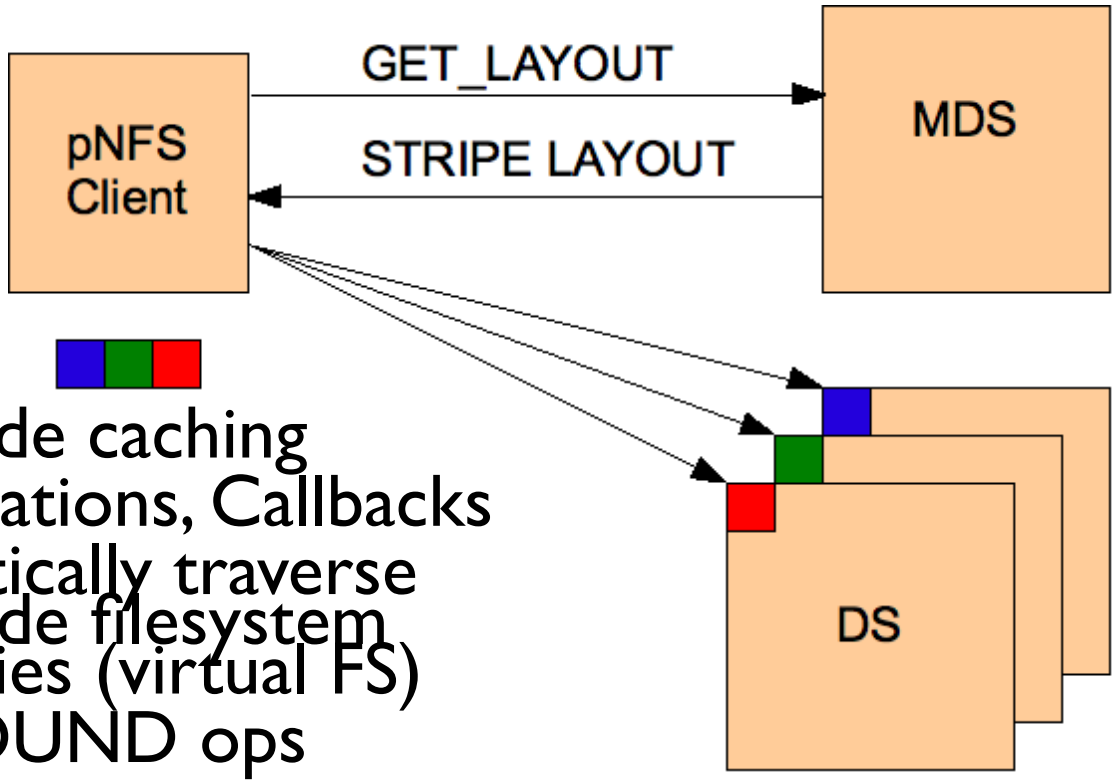
- Need to build in query to make FS-XAM work, but how?

- Need to share the data store ==> network FS
- Too many round trips ==> Add caching and buffering
- Many streaming apps want fast data access
- pNFS servers already want to support fixed content applications. They want to learn from XAM.

so...

- Explore using pNFS as an **implementation substrate** for XAM
- Explore building XAM-like **fixed-content features** into pNFS

pNFS in Action



- Client-side caching
  - Delegations, Callbacks
- Automatically traverse server-side filesystem boundaries (virtual FS)
- COMPOUND ops
- Data Location Independence
- Data Streaming

Review of NFS 4.1 and pNFS

- Standardize the network protocol, not the API
  - Files, Directories, Filesystems, extended attributes
  - Standardized attributes (req'd or recommended)
  - Authentication, Owners, Groups, ACLs
  - Exchanging user and group ids across domains
- pNFS layouts speed up ingest and retrieve of streams

- Delegations and Reliable Callbacks support client-side (readonly) caching – useful for retrieve
- COMPOUND RPC calls string a series of related operations together with clean semantics ==> Could create an entire File+Attributes in one Round Trip
- IF we can combine the XAM client-side cache with the NFS 4.1 client-side cache.

Immutable Data

- Moving towards desired fixed-content capabilities...
- Immutable Data can be achieved in pNFS Server with no client-visible changes ==> the immutable layout
- Wishlist:
  - Preserve metadata as reliably as the data
  - Compute a Content Hash (eventually: end-to-end)
  - Survive any 2 node failures ==> R/S Encoding M+N
  - Deduplication
  - "Versioned" files – any file can become fixed content, after which old versions stick around
- Simple conceptual model
  - Once you "check in" a revision of a file, it gets its own XUID as a link in /xuid/xxx. Modifying the file creates a new XUID and links the file to the new one.

Immutable Data - Refinements

- The distinction between binding and non-binding metadata could be useful.
- Per-Object Cloning and Copy-on-write
  - We want both reference counting plus cloning. Currently you have to choose
  - Request for ZFS: per-file cloning
  - Moves towards having a versioned filesystem
- Two kinds of delete
  - "Normal" delete just removes the current name of an object. Immutable objects stay around.
  - A special "destroy" action is needed to destroy the immutable objects.
- More generally, an impedance mismatch between mutable files and immutable objects.

Query and Job

- Expect to see huge waves of innovation here, so build a plug-in model for different JOB types
- Embed the JOB request in a standard API operation:
  - XAM – submit a job by creating an XSet with XStream input. Read results in an XStream output.
  - pNFS – submit a job in a special directory lookup. Read results in a file (or directory)
  - Ability to share Job results with other processes
- Query Engine could be embedded in pNFS server or could run separately.
- Standard management: submit, display, manage, cancel

Exposing Query Engine Status?

- Query index is very useful for fast query
  - Exposing the "Query Schema" is under discussion
- But query index can easily be out of date
  - Database failure and recovery
  - Change of database schema
  - Bias to allow ingest even if query not available
  - Expecting a 100% accurate query index doesn't scale.
- Query results are often useful even when the query index is incomplete
- XSet property after query for how complete it was?
- XSystem property for how up to date the index is?

Layout-Aware Computation?

- Server-side computation to implement deep searching
- ...and just about anything else (cf Google and Hadoop)
- Map/Reduce Computation
- Goal: Use same mechanisms to implement plugins such as computing the hash values and de-dup scores
- Code runs in a distributed server context and has access to the internal layout details of each object.

Retention Management

- THE most requested policy for fixed content
- Hence standardized by both XAM and pNFS
- pNFS retention modeled on early XAM
  - XAM: base, event, and named retentions. Includes auto-deletion and shred mode
  - pNFS: base, event only
- What gets retained?
  - XAM: XSet is undeletable but non-binding fields can be modified
  - pNFS: File is unrenamable, undeletable, unmodifiable. (Cannot even add links!). Standard does not specify which, if any, attributes can be modified.
- ==>A XAM-friendly pNFS server could implement XAM base and event retention directly
- ==>pNFS might want to track XAM in this area.

Administrative Hold

- Both: Place the file/XSet in a readonly state
- XAM: An unlimited number of named holds
- pNFS: 64 holds in a bitmap
- => Could embed XAM holds directly in pNFS holds, with the limitation of no more than 64 holds per XSet
- Under discussion: JOB chaining (e.g. apply a Hold to all the results of the following query)

Fixed Content App - Reprise

- Having retention, immutability, and query implemented right down with the data is a big win.
- Query languages are never perfect – our app may eventually want to roll our own kind of query using embedded computation!
- Identity is a huge deal for 100-year data
  - Identity of objects (citation address)
  - Identity of users, roles, etc -- who is that masked man?
- Both XAM and pNFS are useful steps along the way

Conclusions

- You can implement XAM over pNFS and you get some good leverage:
  - client-side caching and buffering
  - data streaming and high-speed interconnects
  - integrated with standard network and storage mgmt (e.g. backup)
  - most useful if the pNFS client is also XAM-aware
- pNFS can also learn from XAM
  - useful capabilities (immutable objects, query, retention). Some already standardized in part.
  - completely hidden from the client in many cases
  - Fit well with proposed list of "beyond NFS 4.1"
- Useful to have an MDS for state management
- Issues of Horizontal Scale recede for a while.
- But Global Namespace issues are here to stay!

Where to go from here?

- XAM Resources:
  - XAM Developers Group (on Google Groups):
    - http://groups.google.com/group/xam-developers-group
  - SNIA XAM Home
    - http://www.snia.org/xam

- pNFS Resources:
  - pNFS – Current Internet Draft
    - http://tools.ietf.org/html/draft-ietf-nfsv4-minorversion1-26