MEDIAFLUX VERSION: ALL

DOCUMENT VERSION: 1.0

STORAGE WORKFLOWS

OVERVIEW

Hierarchical Storage Systems (HSM) – typically tape-based – function optimally with fewer larger files, rather than many small files because the seek times are high when compared to disk, introducing potential performance issues when retrieving many files. Access to the HSM should be deferred to as late as possible in the data workflow.

This document describes how the following Arcitecta Mediaflux technologies can optimize storage workflows:

- File Compilation Profiles (FCP)
- Arcitecta Archives (AAR)
- Server-Side Archives
- Parallel I/O
- Storage
- Hierarchical Storage Management (HSM)
- POSIX file-system.

FILE-SYSTEM COMPILATION

Mediaflux includes a file system compiler, which looks for patterns with a file system. The patterns are defined within a File Compilation Profile (FCP)[1].

The file system compiler can:

- Traverse directories of files
- Group data into archives of varying levels of compression
- Assign logical MIME types to the data
- Extract metadata, including geo-spatial extents, from file paths, sidecar XML files and other contextual sources in the client, and before the server-side content analyzers, if any, extract information.

[1] For a detailed description of file compilation profiles and their syntax, refer to the *Technical Note – File Compilation Profile*.

The file system compiler can be invoked via the Desktop Integration (DTI) within a browser-based client, or from the Mediaflux command line interface `aterm`. The browser based DTI allows users to simply drag directories (or files) into the system, select an FCP, and then have the system automatically interpret and optimally package the contents of the input data. The command line interface `aterm` allows automated processes, driven by third-party applications and processes, to do the same.

For example, the military imaging format CIB normally has 1000-1500 files per image. An FCP can be configured to detect the occurrence of a CIB data set (by recognizing the A.TOC file), and automatically bundling into a single archive. This significantly reduces both network transmission overhead, and storage overhead, since only one (atomic) file is transmitted and stored instead of up to 1500 separate files/assets. The Arcitecta CIB content analyzer will look inside the archive to retrieve the geospatial extents and other metadata.

ARCITECTA ARCHIVE

The Arcitecta Archive (AAR) is an archive format developed by Arcitecta to coalesce large numbers of large files within an archive.

AAR files support the following:

- Can be up to 2^{63} bytes in length
- Can contain any number of files, up to an aggregate file size of 2^{63} bytes
- Each file can be up to 2^{63} bytes in size.

The latest version of AAR (version 3) is specifically optimized to deal with large amounts of data:

- Parallel (concurrent) compression/decompression
- The use of CPUs is asymmetric and controllable – the number of CPUs used for compression may be different to decompression, and may be specifically controlled by an application
- Tables of contents generated on the fly, enabling compression directly across a network boundary without any local storage overhead
- Table of contents can be extracted and stored locally
- Archives can be split and merged to create derivative archives without additional decompression/recompression steps.

AA3 (V3) scales approximately linearly in compression performance with the number of CPUs. It is approximately two times faster with two CPUs and four times faster with four CPUs etc. than standard compression tools, with the same levels of compression.

In addition, AAR (V3) provides support for error detection at partial file level, enabling recovery of parts of an archive (non-corrupt files, or non-corrupt segments of files) in the event that part of the archive has become corrupted by the storage system(s).

SERVER-SIDE ARCHIVES

The Mediaflux server has specific support for archives, including generation of various archive formats (e.g. ZIP, TAR.GZ, ISO Images). Whilst many of the following operations can be performed with other archive formats within the server, AAR has significant performance advantages:

- The table of contents can be requested and browsed without decompressing the archive
- The table of contents can be extracted and stored outside of the asset archive enabling access without recalling data from a HSM (or remote storage)
- Specific files can be extracted from within the archive (without decompressing) and transmitted to the client where they are decompressed therefore resulting in only one decompression cycle
- The integrity of a server-side archive can be checked using the service `asset.content.archive.check`
- Archives can be converted from one format to another using the service `asset.archive.convert`[2].

In addition, there is specific support for archives stored within a HSM – when retrieving files within the archive, the archive will tell the storage system to ensure the requisite byte ranges are retrieved. This partial recall of data allows selective retrieval of files within a very large archive without having to bring the entire file on-line.

PARALLEL I/O

Both the DTI (browser-based upload) and `aterm` utilise parallel I/O for transmitting the data to the server. This works for both direct file transmission, and/or streaming the output of on-the-fly archive generation.

Parallel I/O increases performance for wider area network transmission, but also improves performance for local area network transmission by ensuring the network is fully utilised.

The parallel I/O system can be configured to specify the number of concurrent packet transmissions, along with the garbage collection time for failed transmissions.

During upload to the server, data can be transmitted and arrive out of sequence – it is properly ordered within Mediaflux before passing to the service to receive/process the data.

STORAGE

Mediaflux can store asset content/data in the database (atypical), a file system (typical) and a Hierarchical Storage Management system (HSM), or any combination of these types of systems. The data store utilized is associated with an asset namespace.

The data store can be changed over time and existing assets (and asset versions) may either be left in the original storage system, or migrated to the new storage. This migration can occur en masse or selectively using an asset query.

For example, consider the following scenario:

- One or more projects are stored in a namespace that utilises a file system
- During the course of the project, or some time later, the storage architects decide they would like to move the video content onto a different storage system that is specifically optimised for video
- That migration should be done incrementally on a per-project basis.

[2] Note that the typical process is to convert “lesser” archives (such as ZIP) to AAR, not the reverse since there are often size and performance limitations of these other formats.

The following operations would be performed:

- The new data store created using the service `asset.store.create`
- The new store associated with the project's namespace using the service `asset.namespace.store.set`
- Use an asset query to select all video assets, and then pipe the results into the service `asset.content.move.to.namespace.store`

e.g.

```
asset.query :where "type>=video" :action pipe \
  :service -name asset.content.move.to.namespace.store
```

The integrity of data can be checked at any time using the service `asset.content.validate`. This service will retrieve the data, recompute check-sums and compare against the stored check-sums. This is a costly operation because the data has to be retrieved and read into memory for computation of the check-sum. However, it can be scheduled to run incrementally and periodically by selecting the assets using an asset query and piping into the service `asset.content.validate`. Checking data validity is important in long-term archives, since "bit rot" can still occur in systems with error detection and correction.

HIERARCHICAL STORAGE MANAGEMENT

Mediaflux has specific support for SGI Data Migration Facility (DMF). A DMF based data store can be created by specifying a file-system type of "dmf-file-system" when using the service `asset.store.create`.

Mediaflux has the following capabilities for DMF:

- It can report the content status (using the service `asset.content.status`)
- Content can be explicitly migrated off- or on-line (using the service `asset.content.migrate`)

These services can be used in queries to construct metadata specific operations on the HSM.

For example, the following query might be used to bring all data for a project on-line:

```
asset.query :where "namespace>='my project'" :action pipe \
  :service -name asset.content.migrate < :destination online >
```

When migrating a file on-line, the entire file may be migrated or specific byte ranges may be migrated. The default is the entire file.

POSIX FILE SYSTEM

Mediaflux has a new POSIX File System interface which allows the server to be mounted as a POSIX compliant file system. The support is available for servers running Linux and Mac OS/X based systems.

The POSIX compliant file-system interface allows direct access to namespaces and assets as though they are directories and files within a file system. The "shape" of the file system can be determined per user by linking to asset views. That is, for one user, the file system may only present files ("assets") that have been labelled as PUBLISHED.

PUTTING IT ALL TOGETHER

The following diagram illustrates an ingest workflow:

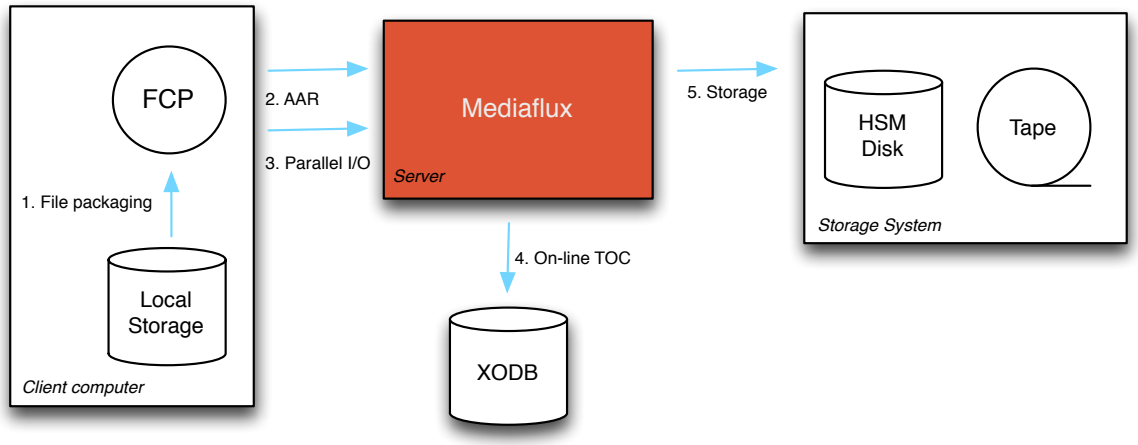

Figure 1 - Ingest Workflow

The process of ingesting is as follows:

1. The file system compiler bundles files into archives. The client application is one of a browser-based client (in which case the user simply drags and drops a directory onto the system), or via aterm[3]
2. The file system compiler packages data into one or more archives at some compression level (as defined by the FCP)
3. The file system compiler transmits the data it has generated to the server using parallel I/O
4. The server optionally extracts the table of contents from the archive and stores it in the database for later browsing
5. The data is committed to storage – note, the data is actually being cached on the HSM disk as it arrives from the client to ensure no extra data copies occur.

[3] Custom client applications may be built using the DTI.

The following diagram illustrates an egest workflow:

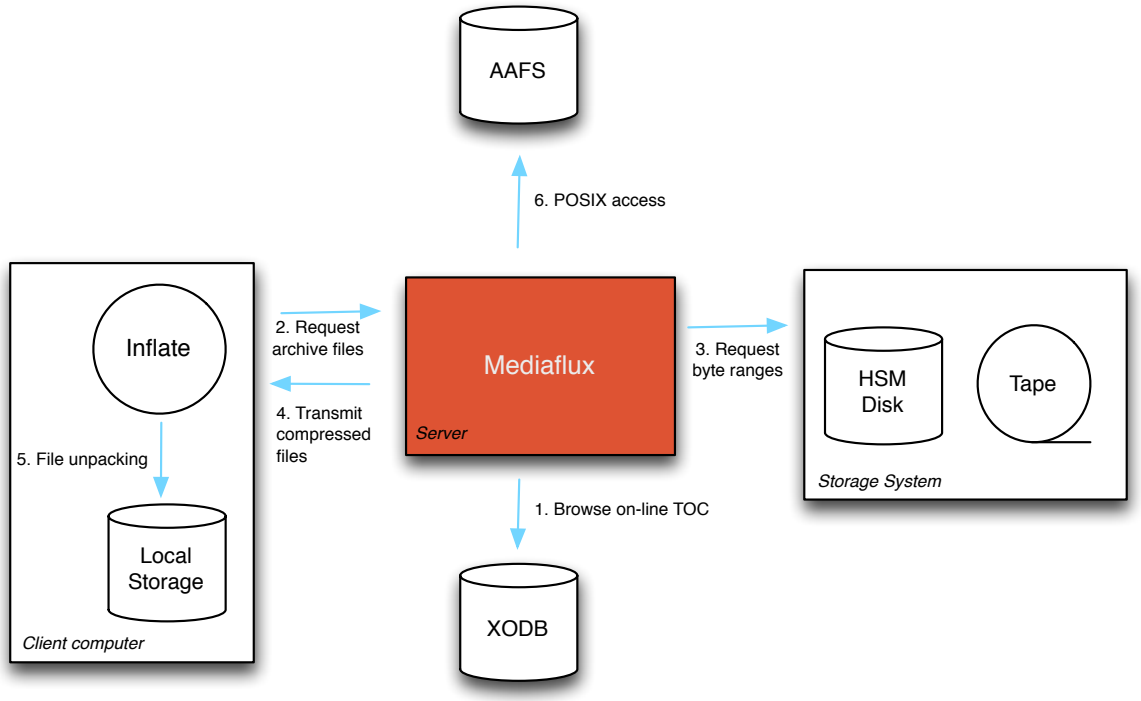

Figure 2 - Egest Workflow

The process of egesting is as follows:

1. User (application) searches for assets using some query. Finds archive and asks to look within the content of the archive. Mediaflux will use the on-line table of contents (without retrieving the data from the HSM)
2. The user requests specific files from within the archive
3. The server requests the requisite byte ranges for those files from the storage system - HSM in this case
4. The files are extracted (without decompressing) from the archive and transmitted to the client
5. The DTI (or aterm) re-inflates the archive content as it arrives at the network boundary of the client computer
6. Alternatively, user Mediaflux is mounted to local or NFS file system and files accessed directly.

Arcitecta Pty Ltd
ABN: 83 081 599 608 ACN: 081 599 608
Postal Address: Suite 5/26-36 High Street Northcote Victoria Australia 3070
Telephone: + 61 3 8683 8523
Fax: + 61 3 9005 2876
Website: www.arcitecta.com
Email: info@arcitecta.com