High Performance Tape Streaming in Tapr

Klaus Birkelund Jensen, <birkelund@nbi.dk>
eScience, Niels Bohr Institute

Communicating Process Architectures 2016
Niels Bohr Institute, Copenhagen, Denmark

August 23, 2016

Outline

1. **Introduction to the domain**
   An introduction to high-throughput imaging and related challenges.
2. **Tape in the 21st Century**
   Description of the LTO standard and LTFS.
3. **Tapr, an automated tape library management system**
   Design and some implementation details of Tapr.

Industrial Computed Tomography

Basically the same as getting a CT scan at the hospital, but much higher energies and the subject is rotated instead of rotating the radiation source.

Some Numbers For Perspective

Raw tomography data (2D) are typically **2560** $\times$ **2560** pixels (i.e. TOMCAT[1] at the Swiss Light Source, Paul Scherrer Institut)

Stored using 8-12 bit/channel (**no compression!**): 19-28 MB. The number of projections varies for each experiment (360 to thousands).

[1] TOmographic Microscopy and Coherent rAdiology experimenTs

Some Numbers For Perspective

Raw tomography data (2D) are typically **2560** $\times$ **2560** pixels (i.e. TOMCAT[1] at the Swiss Light Source, Paul Scherrer Institut)

Stored using 8-12 bit/channel (**no compression!**): 19-28 MB. The number of projections varies for each experiment (360 to thousands).

Size of reconstructed volumes explodes (**single precision!**).

Number of voxels	**Size**
$1K^3$	4 GB
$2K^3$	32 GB
$4K^3$	256 GB
$8K^3$ (not yet used)	2048 GB

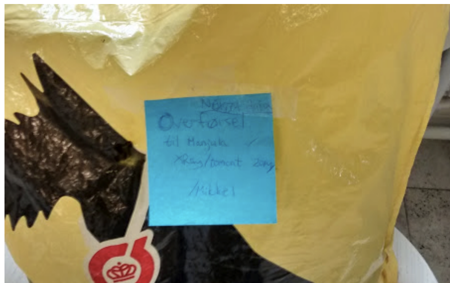
[1] TOmographic Microscopy and Coherent rAdiology experimenTs

JBOD? Just a Bag Of Disks?

Our physicist colleagues can easily generate double digit terabytes (raw 2D tomography data) doing a 24-hour timeslot at a beamline.

JBOD? Just a Bag Of Disks?

Our physicist colleagues can easily generate double digit terabytes (raw 2D tomography data) doing a 24-hour timeslot at a beamline.



Shopping bag from discount grocery store with five 1 TB USB drives labeled *"Transfer to Manjula, X-Ray/TOMCAT 2560, from Mikkel"*.

Magnetic Tape Storage

- **high capacity**, **cost-effective** and **low power** long-term storage.
- The 7th generation of the Linear Tape Open (LTO) standard (December 2015) specifies bandwidth of **300 MB/s** and native storage capacity of **6 TB** (uncompressed).
- From the 6th generation, LTO expect data to compress at a ratio of **2.5:1**.
- Designed for offline storage of **15 to 30 years**. Continuous use causes wear and tear and lowers the lifespan.

Growth in Capacity

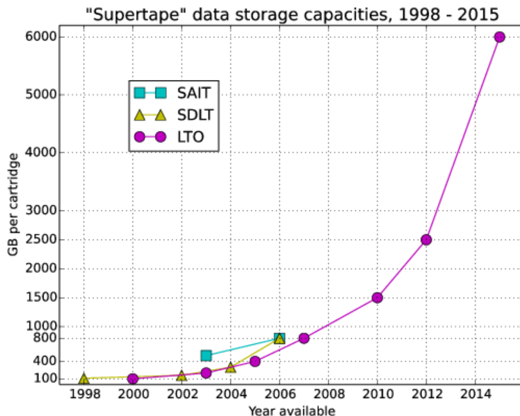


Image credit: Wikipedia

High-performance Tape Streaming with LTFS

Tapes have always, and still are, accessed through low-level SCSI.

Until 2010, no standard format existed. **tar** used in UNIX and derivatives. Proprietary systems (IBM Tivoli Storage Manager, Veritas NetBackup) uses their own format. **No interoperability**.

The **Linear Tape File System** (LTFS) introduced in 2010 with the LTO-5 standard changed this.

- **Standardized** storage format, append-only.
- Accessed as a **POSIX file system**, mounted through FUSE.
- Supports files, directories.

The Linear Tape File System

Tape is **partitioned** in two:

1. A relatively small (36 GB) **index** partition.
2. A large **data** partition.

The index itself is append-only making the tape implicitly versioned.

The tape is **self-describing**. If your system can read LTFS, it can read any tape formatted with it.

Quick listing of contents, only index is read at mount time (and index is located in the beginning of the tape).

The index is held **in-memory** and written at unmount.

Physical Characteristics of LTO

An LTO tape is not a dual-spindle *cassette*. Specification of an LTO-6 tape cartridge:

- Single spindle
- Total length of **846** meters
- Four data **bands**
- 34 **wraps** of about 18 GB (uncompressed) for each **band**
- 16 **tracks** for each **wrap**
- Track writing uses *shingled magnetic recording* (the drive has 16 data read/write heads)
- Drive must make **136** *passes* over the tape to fully write it. Each pass requires a direction reversal.

Unintuitive Behaviour of LTFS

LTFS unmount duration

Seconds

130
120
110
100
90
80
70
60
50
40

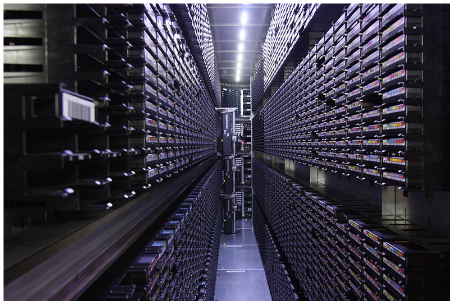
1 2 4 8 16 32

GB written to tape

The Automated Tape Library

An automated tape library consists of

- A large number of **storage slots**
- One or more **robot changers**
- Multiple **tape drives**



Tapr

Open-source automated tape library management software (ATMS) developed at the Niels Bohr Institute for two distinct purposes:

1. To replace high-cost proprietary software for long-term archival of research data in the *Electronic Research Data Archive* (ERDA).
2. High-throughput recording of imaging data from synchrotron installations.

Tapr works with **streams** of data. (raw industrial image data, output from climate models, data archives).

Tapr

- **Stream multiplexing**
  Tapr constantly multiplexes streams onto drives to ensure full bandwidth utilization of the drive/media.
- **Stream filtering**
  Custom functions working on streams can be applied as data enters or leaves the volume (current use case: climate/oceanography).
- **Transparent stream fail-over**
  Streams are automatically moved to other drives when a volume is full or if an error is encounted.
- **Built-in disaster recovery**
  In the event of database failure, the index partitions of all volumes can be quickly scanned to rebuild the inventory database.

Tapr Components

- **Tape interaction**
  A package for interacting with the automated tape library through the *MTX: Media Changer Tools*.
- **An HTTP/2 API**
  The main interface is a REST HTTP/2 API with administrative as well as user commands to store, retrieve and list archives.
- **Inventory**
  A relational database (currently SQLite) that tracks status of volumes.
- **Chunk management**
  A fast key/value store (BoltDB) supporting fast prefix scans of archive chunks.
- **LTFS**
  A Go package for managing LTFS-formatted volumes.

Design

I/O clients

API

create process

Automated library

storage slots

drive A

changer

drive B

Tapr

server/maintenance processes

inventory process

drive, writer, changer processes

relational database

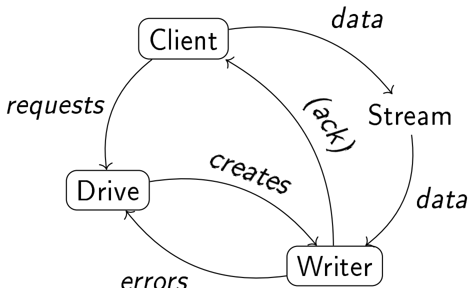
SCSI

SCSI

SCSI

Tapr

Tapr is highly concurrent and uses a *communicating process architecture*.



Implemented in the **Go Programming Language**.

Cancellation Problem

In a communicating process architecture, the following pattern is often seen:

```
1  // create a communication channel
2  ch := make(chan *drive)
3
4  // send a "request" to all drives
5  for _, drv := range drives {
6    // start a light-weight concurrent process
7    go func(drv *drive) {
8      // block until available
9      drv.acquire()
10
11     // send the drive on the result channel
12     ch <- drv
13   }(drv)
14 }
15
16 // receive allocated drive
17 drv := <-ch
```

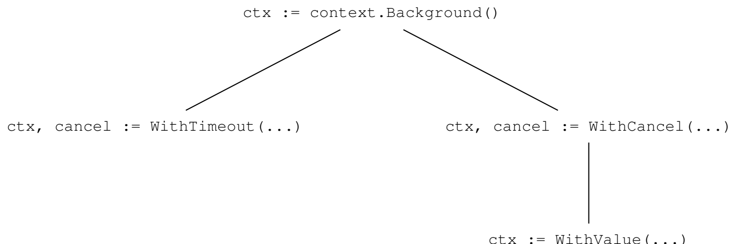
A send operation (`channel <- value`) requires an available receiver to proceed. If the value is not read from the channel (`v := <-channel`) the process *leaks* and never terminates.

Deep Cancellation using Contexts

Go includes a package to solve this, the `net/context` package.

A `context` is a special type that carries **deadlines**, **timeouts** and **cancellation** signals across function calls and between processes.

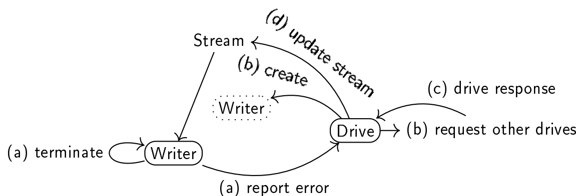
Contexts can be **derived** from other contexts to form a tree:



Deep Cancellation using Contexts

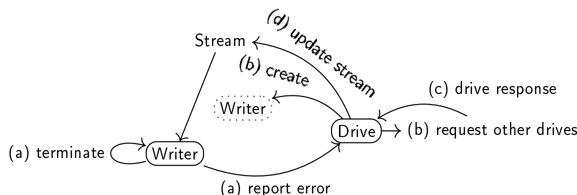
```
1  func reserve(ctx context.Context) (d *drive) {
2    derived, cancel := context.WithCancel(ctx)
3    ch := make(chan *drive)
4
5    for _, drv := range drives {
6      go func(drv *drive) {
7        if err := drv.acquire(derived); err != nil {
8          return
9        }
10
11       select {
12       case <-derived.Done():
13         drv.release()
14         return
15       case ch <- drv:
16       }
17     }(drv)
18   }
19
20   select {
21   case <-ctx.Done:
22     return
23   case d = <-ch:
24     cancel()
25   }
26
27   return d
28 }
```

Stream Hand-off



a Writer experiences an error, reports it to the drive process and terminates.

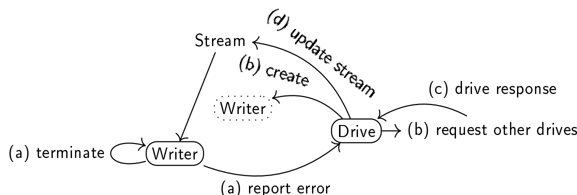
Stream Hand-off



a Writer experiences an error, reports it to the drive process and terminates.

b The drive process concurrently requests access to other drives and starts a new writer process.

Stream Hand-off

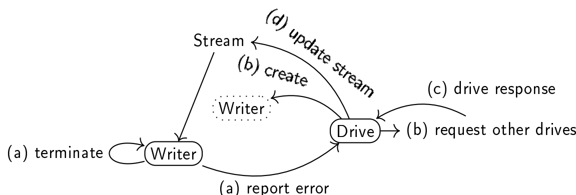


a Writer experiences an error, reports it to the drive process and terminates.

b The drive process concurrently requests access to other drives and starts a new writer process.

c If another drives responds that it is willing to take over the stream the drive cancels requests to other drives.

Stream Hand-off



a Writer experiences an error, reports it to the drive process and terminates.

b The drive process concurrently requests access to other drives and starts a new writer process.

c If another drives responds that it is willing to take over the stream the drive cancels requests to other drives.

d The stream structure is updated and new chunks will go to either another drive or to the new volume in the current drive.

Streams and Chunks

Streams is the primary abstraction. A data stream is chopped up into **chunks**.

When a stream is finalized (closed) it is converted to an **archive** of chunks identified by

- The archive id (currently just a byte string)
- A *volume global* chunk id
- An id that signifies the chunk location in the archive
- Example: `<global_id>-<archive_name>.cnk<chunk_id>`

Chunks are stored in the chunk database with key `<archive_name>-<chunk_id>` and value of the volume serial the chunk is located on. The key format allows fast prefix scans.

Interleaving of Streams

Tapr uses *stream interleaving* or *multiplexing* to maximize utilization of the drives.

Each drive process keeps track of bandwidth utilization using exponentially weighted moving averages of 3, 10 and 30 second periods.

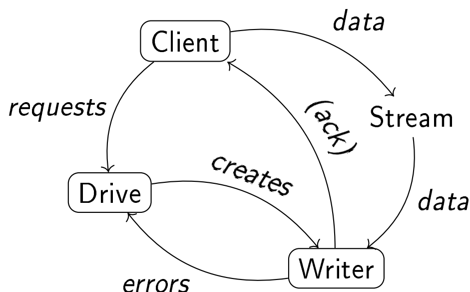
Other approaches

- disk staging (e.g. IBM Spectrum Protect) with regular flushing to tape. Allows the server to better co-locate data from different clients.
- combined disk staging and multiplexing (Veritas NetBackup). Both benefits at the expense of an high-end disk array.

Interleaving of Streams

A `writer` process serializes writing of *chunks* to the tape drive. **Only one I/O operation at a time**.

`client` processes write to **stream structures** that assembles chunks and sends them to the `writer` process.



Interleaving of Streams

Write throughput

MB/s

1400
1200
1000
800
600
400
200
0

1 2 4 8 16 32 64

Number of concurrent streams of 1 GB

parallel, chunk ack
parallel, no ack
single, chunk ack
single, no ack

Interleaving of Streams

Write throughput, different chunk sizes

MB/s

1300
1200
1100
1000
900
800
700
600
500
400

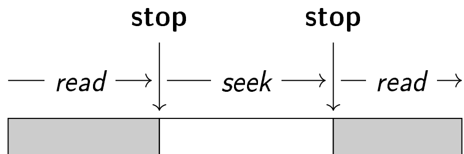
1 2 4 8 16 32 64

Number of concurrent streams of 1 GB

4 MB
32 MB
64 MB
128 MB

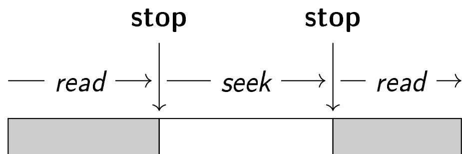
Read strategies

Naïve strategy: **selective**

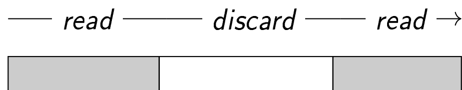


Read strategies

Naïve strategy: **selective**



Less naïve: **contiguous**



Read strategies

Read throughput, 64 MB chunks

MB/s

160
140
120
100
80
60
40
20
0

1 2 4 8

Number of concurrent write streams of 1 GB

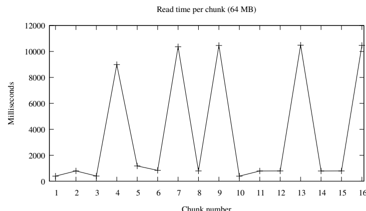
selective read
contiguous read

Retrieval Prediction

If using the contiguous read strategy, Tapr can predict the retrieval time for a given archive.

The *volume-global* chunk id is essential for this as it allows Tapr to precisely predict how much data it needs to read before the archive has been retrieved.

If selective read is used, it is a bigger challenge. Seek times varies by drive and media.



Future Work

- **Coupling of data transformations and streams**
  Stream transformations are stored on volume and applied when reading. Appling transformations to ingressed data directly.
- **Adaptive chunk size**
  Chunk sizes are automatically changed to fit the concurrency level at the volume.
- **A scheduling plugin for SLURM**
  Allow SLURM to get an estimate on when data can be made available on disk for a job to run.

Thank You

Questions?