Russ Laher's Notes from the LSST Fault Tolerance Workshop

Day 1: July 15, 2008

FT = fault tolerance

There is no interface document for ingesting raw images.

- scheduled vs. received raw image
- checksum (MD5)
- uncorrected errors in TCP communications are common
- multiple copies of raw data
- backup copy of raw data off-site
- procedure for buying backup storage to put on the mountain
- N *days backup storage on mountain
- higher reliability hardware for raw *data storage
- plan for replacing disk hardware and migrating data
- scrubbing every year or six months (full blown and statistical), track that this has been done for specific images; track error rate; dedicated scrubbing vs. incorporating the scrubbing into the pipeline processing.

Define Scope Infrastructure vs. Middleware vs. Applications

Interface should be requirements for reliability. Off the shelf vs. highly reliable hardware. Requirements specified frequency, duration, extent -- not just percentage availability.

Should shoot for high reliabilty for maximizing science return and cost savings.

Need requirements for

- Servers
- Pipeline machines
- Disk Storage
- Network

Scheduled maintenance downtime. Won't be taking data 100% of the nights. Issue of when to do the scheduled maintenance.

Unscheduled downtime. Strive for low levels of this.

Algorithmic faults.

Worst-case catch-all for unclassified faults.

User-contributed software. Run on separate machines and sandbox with firewall isolation. Can't impact operations processing. Another option is VM with virtual walls.

Need system to mark data that is found bad later after it has been distributed to the public.

Guiding principle for fault tolerance:

1. Reducing human intervention for common problems -- automated
2. Tracebility
3. Real-time processing vs. reprocessing
4. Address/fix problems

Detecting class of faults

1. Resource failures fail or succeed when pipeline run is repeated (network load).

Database transactions.

Issues surrounding faults:

- Location
- Repeatabilty
- State

Failure can be in terms of getting done, but not getting done with required time period.

Failures can be

1. Loud, noisy, or run-away
2. Silent
3. Corrupting

Should we look at correlation of failures?

Fault of FT system!!!

What is the role of SDQA results in overall fault tolerance and what is the impact on middleware design?

High-level requirements related to FT. How should we interpret these requirements?

- SDR

- Very little, except for 60-s alerts and OTT1.

- FDR
    - Storage: 0% data loss (raw data, metadata); 98% availability
    - Communications: 0.1% alert-publication failure; 98% availability.
    - p. 21 TBD has FT implications
    - p. 40 - There are pipeline requirements relevant to FT.
    - Lots of requirements about data release - p. 14
    - Software licenses have to be kept (p. 19 is an impossible requirement).
    - p. 59-61 - Reliability requirements
    - Can't finish four hours after night's observations end, because the nightly pipelines have to be

executed on all images first.
- Scheduling of observations has to be fedback to, for example, calibration-pipeline execution and production of calibration images.

K.-T. proposed two documents outcome from this workshop:

1. Overall - Hardware and SDQA components
2. Middleware specific

What do we mean by FT?

What are the criteria for failures that we want to address?

Criteria such as something that causes data products to not be available to public.

Strategies for meeting goals Plans Recommendations

FT methodology (different philosophical approaches)

- master driven system
- peer to peer, independent fault checkers

Instead of one local disk per multi-core CPU (box), have a SAN clustered to, say, three CPUs.

Intrinsic failure rate for image processing (or portions of image).

Hardware redundancy to reprocess an image segment (amplifier).

Reprocessing since last checkpoint.

At one granularity is it practical to drop/lose a portion of the processed image (amplifier or smaller).

Triply redundant processing done in a Monte Carlo fashion (or just one amplifier, the same amplifier).

Rendevous of data (FWHM of PSF overlapping adjacent CCD, ghost images,...)

Understand the consequences of failure

Day 2: July 16, 2008

Sample Pipeline Exception -- check that data is accessible to pipeline

- Possibilities
  - template images must be at base and cached on disk
  - database data must be cached
  - calibration images must be available
  - policy files must be available

- Detection strategy
  - Test run pipeline prior to commencement of processing

- ♦ Check for file existence and retrieve from alternate location, if necessary
- ♦ Check whether database query ran successfully

OCS says they are pointing one place, but then is really pointing somewhere else

Possible fault unique to LSST, which image data are not stored as FITS files, a mismatch between image and image metadata.

Mountain catalog storage strategies to maximize utility of available disk storage and give some fault tolerance:

- store small portion of catalog of bright sources
- store two copies of either summer or winter sky

Use case for SDQA:

- Image metadata is missing, garbage, or inconsistent with image data
- WCS may fail (may have limitations or need bootstrapping)

Common practices:

- Watchdogs deployed on separate machines
- Redundancy (hardware, database server, database replication)

Processor failure

Disk failure

- Can't open/close file
- RAID monitoring and continuous scrubbing (block-level checksuming)
- Query/monitor ECC bad-block activity increases (limited value)
- Silent corruption detect by pipeline-external checksum verification
- Multi-level checksum verification of file data and memory data

Database problems

- record(s) missing
- more than one record unexpectedly returned
- too many database connections
- can't connect to database (permission problem, server down)
- can't set database role
- cant execute query (role missing grant)
- table locking
- queries take too long (database tuning or statistics need updating)
- server down
- inserting record with primary key validation
- not enough disk space allocated for large table (inefficiency)
- transaction logging out of disk space

Corruption of communication between nodes

General FT

Testing and comparison RBT Verification/testing (watchdogs) Duplicating things in space and time (eliminate single points failure) Mechanisms for detection failure Detection vs. response mechanisms Redundanct execution of processes Limit overwrites Separate mutable vs. non-mutable data

Prevention of failure

Response to failure

Reconfigure system on the fly

CMSD cluster technology, separate from hardware, for communication, with replicatable master server (Anthony)

Double the capacity without checkpointing, or a few additional 10% with checkpointing, is needed to meet 60-s alert requirement (zero failures). Redo affected CCD, not just amplifier. Need extra boxes for small number of failures a minute late.

High-speed SAN

Action item:

Spreadsheet the nightly-pipeline data volume and rate through a core. Need to size the requirement throughput to meet the 30 s. There will be addition 30 s budgeted for source association, alert generation and transfer down the mountain.

2 x 11.5 GB / 30 s = 767 MB /s (internal memory bandwidth is not an issue) | reading AND writing

---

Define classes of failures, redundancy, hot spare, check-pointing, impact on system.

What specifically needs to be monitored

Maintenance throughout mission

> disk defragmentation disk replacement add transaction-log space add file-storage space database tuning database data verification database indexing

Engineering automated maintenance

Human monitoring component

Enumerate specifically every fault that needs to be handled

Requirements document (or section in planning document)

Use Cases document (or section in planning document)

Number of personnel needed for LSST operations

Four major areas of LSST fault tolerance:

1. Middleware
2. Database
3. Hardware
4. Facility

SDQA FT is out of band (not defined to generate "exceptions" in the sense of this workshop, but, rather, "alerts").

Application software exceptions cannot be handled automatically -- there must be human intervention to fix the problem. If something can be done automatically to fix the problem, the fix will be algorithmic and should be handled within the application layer (either in C++ code or Python script).

Applications developers must follow robust code. We have to deal with software exceptions from the applications layer. Code checker software. CCB policing. Coding guidelines. Regression testing.

Specific application exceptions can be subclassed from the middleware base class for catch-all application exceptions.

Detecting and validating dependencies of pipelines on specific.

Some middleware exception handling relates to I/O:

File systems Sockets Memory allocations Database access

Middleware API for getting calibration files is needed.

Store subversion revision numbers of third-party software tar balls.

Application software does no I/O. Its input data are only read from the clipboard, and its output data are only written to the clipboard. Clipboard just holds pointers to objects.

Variance on processing time for data-dependent data reduction.

Three products from this workshop:

1. Near term summary
2. PDR presentation
3. Operations plan (beyond PDF)

Design plan, but not development/implementation plan

Use DC3 to evaluate the feasibility of check-pointing?

Need to cost out clusters with SANs (Storage Area Network, a high speed, special-purpose network that connects to storage devices).

Estimate how often a box will fail -- use industry data.

Hardware includes rack power supplies, and can include rack-isolated cooling systems, switch, line card, disk storage, box (multi-core CPU, CPU cache, RAM, local disk)