Fault Tolerance Use Cases

This page captures the challenges that the Fault Tolerance (FT) system will have to address. It includes analysis of FT-related use cases, different types of failures, and high level strategic options available.

Failure types

Repeatability

Failures can be:

- repeatable: failure always occurs in the same place for a given set of conditions. Example 1: application program fails when declination is set to 361. Example 2: database server fails if number of connections exceeds 255.
- occasional: failure occurs in the same place from time to time
- non-repeatable: failure can not be reproduced.

Location

A failure can occur on the

- client side (application program or hardware where application program is executed)
- server side (database/nfs/etc... daemon or hardware where such daemon runs)

A failure can occur in the

- software (application code fails, database daemon software fails)
- hardware (node where application code or database daemon runs fails)

Each of these can be repeatable, occasional or non-repeatable

Stateless vs stateful failures

A failure can be stateless or stateful. Stateful failures leave a "state" which needs to be roll-back (un-done). Example of a stateful failure: preserving provenance may involve capturing input from multiple disjoint sources, if one of these sources fails, the system might be left with partially written provenance information which should be removed.

One node vs distributed systems

Recovery from failures is difficult to implement for a distributed system: it requires coordination between all involved nodes. Complication comes from the fact that a failure can occur at any time, e.g. after some of participating nodes already committed there part. A common approach used to recover from faults in distributed systems is based on a two-phase commit protocol. Theory is well understood, but it is non-trivial to implement.

Scale of the failure

Scale of failure can significantly vary. On one end we have single isolated failures (1 disk on 1 node), and on the other we have failures taking down entire data center, for example massive unplanned power outages, or a natural disaster. Different scale of the same type of failure will likely result in different recovery procedures.

Slow execution

In some cases there is no failure, but the execution takes extremely long time. Example: it takes 1 sec on average to process data from 1 CCD of an image, but for certain images and certain CCDs it might take 300 seconds, or even indefinitely.

Example Use cases

1. One of the stages repeatably throws an exception for a given piece of data or a given value of a given parameter.

2. Occasional client software failure: one of the stages of one of a pipeline occasionally fails if a given parameter is set to a given value.

3. Occasional unknown client software failure: one of the stages of one of a pipeline occasionally fails in the same location, but the exact cause is unknown.

4. Non-repeatable client software failure: one of the stages of one of a pipeline failed once, cause unknown.

5. There is an unexpected power outage taking down (a) one cluster, (b) entire data center

6. Processing of an image involves processing 201 ccds in parallel: each ccd is processed by a different node. One of the client nodes fails.

7. Processing of an image involves processing 201 ccds in parallel: each ccd is processed by a different node. Stage x on node Y takes extremely long time to process its ccd (thousands of times longer than on average). It will eventually finish successfully if we let it continue. Repeatable. (do we kill such process and retry running it? Do we mark given ccd as failed? Do we mark entire image as failed?)

8. We must preserve provenance for all data. An image was processed by 201 nodes, but one node failed to provide its provenance. (Unclear what to do: do we remove data for a given ccd? Do we mark data for that ccd as "missing provenance"? Do we roll back data for entire image? Action item: understand granularity at which we recover/persist)

Strategic options available

1). Isolate nodes to minimize having to deal with distributed recovery:

- each node writes to local storage
- if writing to database is required, each node maintains its local database
- reading from shared resources is ok

- one central process coordinates sweeping all updates into a central resource (like main database catalog). It deals with recovery as needed.

Complexity in writing the coordinator.

2) Fully distributed system, 2 phase commit (?)

Fault Tolerance Use Cases Categorized by Major Type and Specific Cause

Facility faults

- Natural disaster (fire, earthquake, flood, tornado, etc.)
- Act of war (aircraft collision, explosion, sabotage, etc.)
- Security broken
    - ♦ Computer firewall breach (hacker, virus, etc.)
    - ♦ Unauthorized computer-room access
- Staff problems (negligence, retention/turnaround, labor strike, slow down, sick out, specialist availability, etc.)
- System resets (checksum mismatches correlate with this)
- Air-conditioning malfunction
- Electrical fuse blown

Hardware faults

- CPU failure
- Network failure
- Disk problems
    - ♦ Catastrophic failure (disk media, disk controller, etc.)
    - ♦ Corrupted data block
        - ◊ Latent sector errors caught by disk controller S/W
        - ◊ Silent corruption (docushare 5842; checksum mismatches, parity inconsistencies, and identy discrepancies)
- Hardware upgrade not compatible with software (portability issues, backward uncompatibility, etc.)

Resource faults

- Power failure (black out, brown out, etc.)
- Disk space
- Disk performance degradation (can occur for disks > 90% full)
- Disk threshing
- Disk/network speed mismatch (bandwidth, maximum number of reads/writes per second, etc.)
- Database (bandwidth limitations caused by resource over-allocation; availability of connections; performance degradation caused by large tables filling up, insufficient disk-space allocation, usage statistics not updated, progressive index computation slowdown, mistuning, etc.; etc.)
- Network bandwidth limitation (sustained or peak specifications exceeded)
- Memory segment fault (stack size exceeded, insufficient heap allocation, misassignment of

large-memory process to small-memory machine, etc.)
- OS limits exceeded (queue length for file locking, number of open files per process, etc.)
- Bottleneck migration (e.g., increase in processor throughput hammers database harder)

Software faults

- Software inadequacies/limitations/bugs flushed out by data-dependent processing
- Missing or bad input data
    - ♦ Bad images (missing, noisy data, or instrument-artifact-contaminated pixels; not enough good sources for sufficient asterometric and/or photometric calibration; etc.)
    - ♦ Missing/unavailable database data (e.g., PM and operations activities not syncronized)
    - ♦ Unavailability of calibration images (missing observations, calibration-pipeline error, etc.)
        - ◊ Use lower quality fallback calibration data (affects SDQA)
        - ◊ Missing fallback calibration data
    - ♦ Unavailability of configuration data files (e.g., version-control mix-ups)
- Environment misconfiguration or loss (binary executable or third-party software not in path, dynamic library not found, etc.)
- Pipeline failures (can't open file, file not found, can't connect to database, database record not found, division by zero, etc.)
- OS, library, database software,or third-party-software upgrade problem (incompatibility with other software, discovery of bugs and/or hard-wired settings)
- Cron job, client, or daemon inadvertently stopped

Common Database Faults

- 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
- Can't execute query (missing grant to database role)
- Stored procedure execution fails (missing or bad inputs, or missing grant to database role)
- Queries take too long (database tuning or statistics need updating)
- Table locking
- Server down
- Inserting record with primary key violation or missing foreign key
- Not enough disk space allocated for large table (inefficient in the best case)
- Transaction logging out of disk space
- Bugs in upgraded versions of database software

Nightly Pipeline Step-through

Template for analysis:

Name/Summary?

Description

as needed

Type

Repeatability:	*repeatable*	*occasional*	*non-repeatable*
Location:	*software*	*hardware*; *client*	*server*	*N/A*
Manifestation:	*catchable exception*	*process death*	*degraded performance*	*degraded quality/incorrect results*	*unknown*
Scale of Failure:	*slice, pipeline, node, cluster, system, database*
State to clean up/roll back?	*N/A*

Expected Method for Detection

exception caught at application level | *exception caught at framework level* | *faults/errors detected at monitor level* | *problem detected in SDQA* | *explicit test confirming success of operation* | *unknown* | *N/A (designed out)*

Possible Reactions/Corrections/Solutions?

describe: retry, alert human, mark bad data and move on, drop data, ...

Pipeline Configuration/Preparation? Fails to complete

Description

missing data, missing software, missing configuration/policy files

Type

Repeatability:	repeatable
Location:	*software*	*hardware*; *client*	*server*	*N/A*
Manifestation:	*catchable exception*
Scale of Failure:	*pipeline, database*
State to clean up/roll back?	possibly

Expected Method for Detection

Test pipeline run prior to real start.

Possible Reactions/Corrections/Solutions?

Quiet period/no manual change period prior to processing and after last test. All for on-the-fly retrieval of missing data

OCS instruction error

Description

OCS tells us to prep for the wrong part of sky

Type

Repeatability:	*repeatable*
Location:	*software*
Manifestation:	without a check, may produce garbage data;
Scale of Failure:	*slice, pipeline, node, cluster, system, database*

State to clean up/roll back?	*N/A*

Expected Method for Detection

check to make sure metadata of raw data matches expectation; WCS may fail, DQA may detect garbage

Possible Reactions/Corrections/Solutions?

do check; if disagrees with expectation, either delay results while needed data is retrieved or drop the frame and issue fault message.

Disks Full

Description

as needed

Type

Repeatability:	repeatable
Location:	hardware
Manifestation:	catchable exception
Scale of Failure:	node, cluster, system, database
State to clean up/roll back?	possibly

Expected Method for Detection

If test run is done, this should be detected. If disks fill during execution, exceptions may be thrown or processes may die.

Possible Reactions/Corrections/Solutions?

Database Catalog (e.g. Object) is corrupted

Description

as needed

Type

Repeatability:	repeatable
Location:	database
Manifestation:	catchable exception, degraded quality
Scale of Failure:	pipeline, database
State to clean up/roll back?	yes

Expected Method for Detection

Possible Reactions/Corrections/Solutions?

Have a day-old copy of catalog that can be swapped in.

Debilitating Processing Node Failure

Description

This might include machine going down, key processes dying. (silent)

Type

Repeatability:	*repeatable*	*occasional*	*non-repeatable*
Location:	hardware
Manifestation:	process death, process or node hangs
Scale of Failure:	node, cluster, system
State to clean up/roll back?	*N/A*

Expected Method for Detection

MPI commands will fail; a "watch-dog" mechanism may detect failure;
Note: independent and dependent (log messages) heartbeats are both useful.

Possible Reactions/Corrections/Solutions?

drop processing on that node, or reschedule that processing, backing up to last check-point (pipeline pass or stage). Need to be able shoot node in the head: take it out of the pipeline, network, ...; power it down.

Disk Failure

Description

disk read/write errors prevent proper processing

Type

Repeatability:	repeatable or occasional or non-repeatable
Location:	hardware
Manifestation:	catchable exception, process death, degraded performance	degraded quality/incorrect results
Scale of Failure:	*slice, node, database*
State to clean up/roll back?	*N/A*

Expected Method for Detection

catch exceptions, degraded performance might be detected by noticing consistent slower performance, may cause total failure of process or node (see node failure). On-the-fly checksum check done while reading data into memory can also catch corruptions.

Possible Reactions/Corrections/Solutions?

if monitoring system concludes disk failures are degrading performance, then on next visit, redeploy processes on different node (marking hardware as bad); if causes processing failure, see node failure.

Name/Summary?

Description

as needed

Type

Repeatability:	*repeatable*	*occasional*	*non-repeatable*
Location:	*software*	*hardware*; *client*	*server*	*N/A*

Manifestation:	*catchable exception* | *process death* | *degraded performance* | *degraded quality/incorrect results* | *unknown*
Scale of Failure:	*slice, pipeline, node, cluster, system, database*
State to clean up/roll back?	*N/A*

Expected Method for Detection

exception caught at application level \| *exception caught at framework level* \| *faults/errors detected at monitor level* \| *problem detected in SDQA* \| *explicit test confirming success of operation* \| *unknown* \| *N/A (designed out)*

Possible Reactions/Corrections/Solutions?

describe: retry, alert human, mark bad data and move on, drop data, ...