Fault tolerance can be enforced in many different ways. Different strategies have different characteristics, including characteristics such as different response time, different complexity, required hardware or cost. We expect there won't be a single strategy that will cost-effectively meet fault tolerance needs of the entire LSST system. Instead, we expect to support several different types of strategies, and apply the most appropriate strategy to each part of the system. As an example, providing fault tolerance for real time alert pipeline which has to process a visit in under a minute requires rapid reaction thus a more expensive solution is justified, however a different approach will likely be preferred for a system producing co-added images, likely running for weeks.

Design Strategies

This section describe fault tolerant design strategies that are expected to be applied for different components of the LSST DMS

Redundancy with automatic failover / full replication

Redundancy means providing multiple identical instances of the same system and switching to one of the remaining instances in case of a failure; the switching is usually automated. An example of a fully redundant system: two identical database servers running on separate hardware and kept in sync, one of them is elected as â€œmasterâ€ and serves all the requests. In case of a failure, the other server starts serving the requests.

Replications means providing identical instances of the same system or subsystem, directing tasks or requests to all of them in parallel. An example of a fully redundant system: two identical database servers running on separate hardware and kept in sync, load is distributed evenly across both of them. In case of a failure, the other server serves all the requests.

On time delivery with no delivery of failed data

LSST images will be processed in parallel, likely by different processing nodes, and failures of individual nodes will be unavoidable. This strategy involves dropping the pieces of data that failed, and deliver on time data processed by nodes that succeeded. An example: data for each amplifier is processed by different core of a 200 16-core node cluster, in case of a failure of one node, data corresponding to 16 amplifiers of processed image will be lost, the remaining data will be delivered on time.

On time delivery with spare capacity used to deliver failed data

This strategy is a slight variation of the previous one. It involves processing of failed parts on the spare nodes instead of dropping the pieces of data that failed. Depending where in the processing pipeline the failure occurred, it is likely the re-processed data will not be delivered on time. It might still be better to deliver data later than not to deliver it at all. For example, we way miss the 30 sec alert deadline, but may be able to deliver alerts coming from re-processed data 1 minute after the deadline.

Degrading data quality

This strategy involves falling back to most-recent working version. The fall back may involve falling back to older version of data, or older version of software. An example demonstrating the former: before starting nightly processing we take a full snapshot of the database. If during the night the live copy of the database gets corrupted and becomes unusable, we fall back to the most-recent working version, which may not contain the most recent set of updates, but is guaranteed to work. An example demonstrating the latter: if processing

of given data consistently fails inside version x of algorithm A, we fall back to version x-1.

Design Practices

There are many design practices that application developers need to be aware of when choosing algorithms and implementing them. Using these practices can dramatically improve complexity and cost of building a robust and fault tolerant system.

Avoid overwrites. It is much easier to provide fault tolerance in system where data is appended. To recover from failure in system where data is overwritten requires making a snapshot of data before each update.

Segregate mutable and immutable data. Imagine a 1 petabyte database catalog, where only a thousand individual rows are updated during any given night. Keeping these 1000 rows separate from the remaining billions of rows would open many powerful ways to optimize and speed up recovery from a failure during an update.

No I/O by application stages. LSST DMS pipelines and stages should not make direct disk accesses. This includes operations like operating on files or issuing database queries. Funneling disk I/O through middleware will simplify capturing faults, retrying and recovering from hardware failures, as well as properly capturing provenance.

[not finished...]