Using Active Data to Provide Smart Data Surveillance to E-Science Users

Anthony Simonet[1] Gilles Fedak[1]
Kyle Chard[2] Ian Foster[3]

[1]Inria [2]University of Chicago
[3]Argonne National Laboratory

March 4th, 2015

The problem

Offer real time end-to-end monitoring & easy data management paradigm to data-intensive application users.

↓

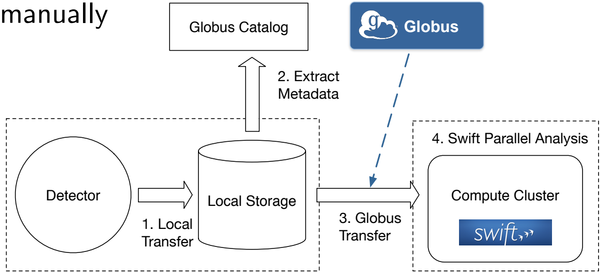
"Data surveillance framework"

One example: the **Advanced Photo Source** (APS)

The Advanced Photon Source

Data-intensive distributed application involving multiple software

- 3 to 5TB of data per week on a single detector
- 3 tools involved:
  - Globus Transfers
  - Globus Catalog
  - Swift
- Tasks are launched manually



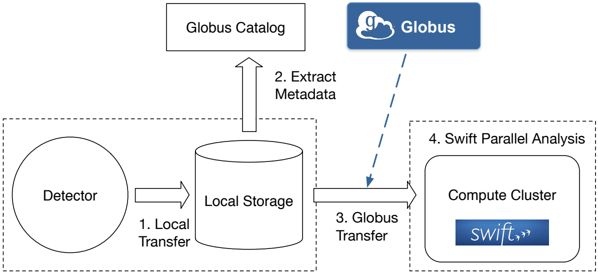
Problems with APS

What is inefficient in this workflow?

- Many error-prone tasks are performed manually
- Users can't monitor the whole process at once
- Small failures are difficult to detect
- A system alone can't recover from failures caused outside its scope

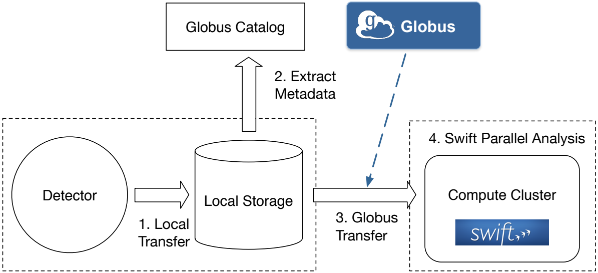
Goals

Goals: Progress Monitoring



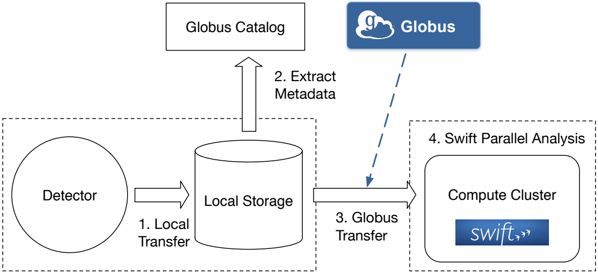
- Get a high-level view of the whole workflow progress
- Generate reports
- Merge several related events in different systems in a single meaningful notification
- Identify steps that take longer to run than usual

Goals: Automation



- Automate most human interventions
- Launch transfers
- Create datasets and extract metadata
- Run Swift scripts

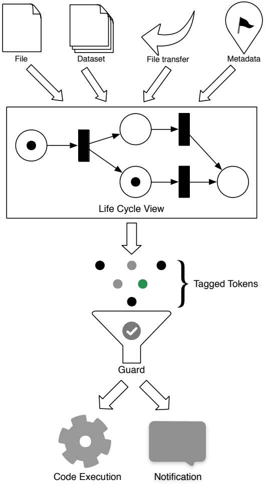
Goals: Error Discovery and Recovery



- Provide each system with a complete view of the whole workflow
- Automatically recover from unexpected events
- Reduce the need for low-level human interventions

System Design

Data Surveillance Framework

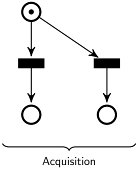


Framework features:

- Single namespace for all the files, datasets and metadata manipulated by the workflow
- High-level *life cycle-centered* view of data
- Runtime data tagging system
- Custom user reaction to data progress
  - Custom code execution
  - Custom notifications
- Powerful filters based on data tags

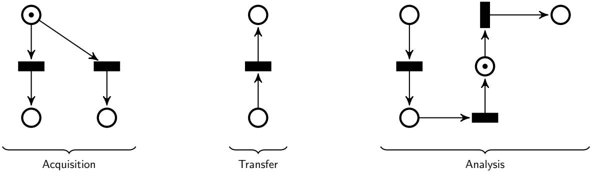
Active Data

- Developed at Inria
- Active Data wants to know everything happening to data
    1. Construct a model of the data life cycle in each system
    2. Connect them in a single end-to-end data life cycle model
    3. Give the model as input to Active Data
    4. Have every system report data operations to Active Data
- In return Active Data tells you everything, at runtime
    1. Declare what events interest you
    2. Provide code to run in reaction
    3. Get notified
- Centralized Publish/Subscribe



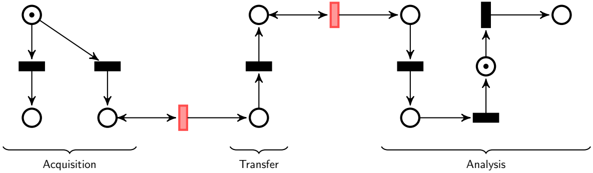
Active Data

- Developed at Inria
- Active Data wants to know everything happening to data
    1. Construct a model of the data life cycle in each system
    2. Connect them in a single end-to-end data life cycle model
    3. Give the model as input to Active Data
    4. Have every system report data operations to Active Data
- In return Active Data tells you everything, at runtime
    1. Declare what events interest you
    2. Provide code to run in reaction
    3. Get notified
- Centralized Publish/Subscribe

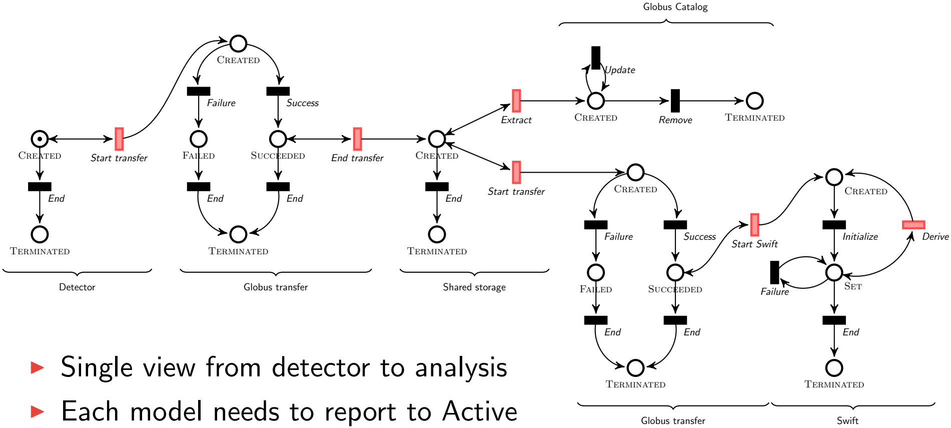


Active Data

- Developed at Inria
- Active Data wants to know everything happening to data
  1. Construct a model of the data life cycle in each system
  2. Connect them in a single end-to-end data life cycle model
  3. Give the model as input to Active Data
  4. Have every system report data operations to Active Data
- In return Active Data tells you everything, at runtime
  1. Declare what events interest you
  2. Provide code to run in reaction
  3. Get notified
- Centralized Publish/Subscribe



APS Experiment Life Cycle Model



- Single view from detector to analysis
- Each model needs to report to Active Data

Results

Error Detection & Recovery

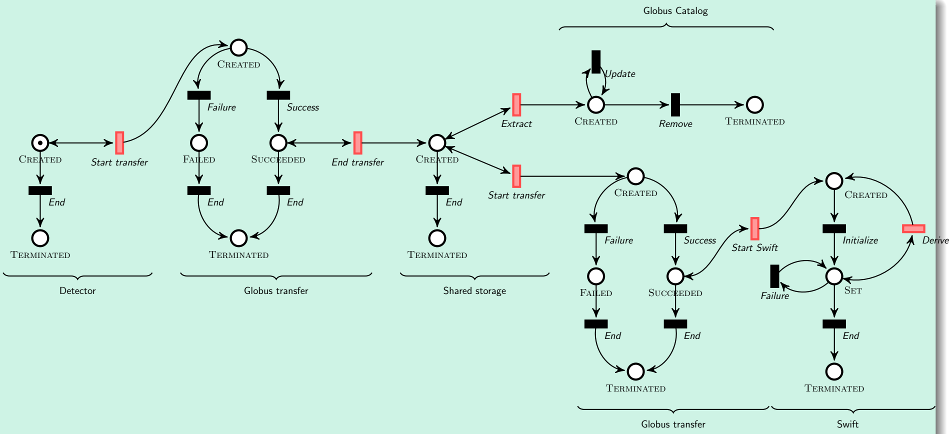
Example scenario

Recover from system-wide errors: faulty acquired files are detected only after Swift fails to process them.

In this situation, the user manually:

- Drops the whole dataset
- Removes any associated file and metadata
- Re-acquire the dataset using the same parameters

Error Detection & Recovery



Error Detection & Recovery

User code

```
TransitionHandler handler = new TransitionHandler() {
   public void handler(Transition t, boolean isLocal, Token[] inTokens, ↘
        →Token[] outTokens) {
      // Get the dataset identifier
      LifeCycle lc = ad.getLifeCycle(inTokens[0]);
      datasetId = lc.getTokens("Shared storage.Created")[0].getUid();

      // Remove the dataset annotations from the catalog
      String url = "https://catalog.globus.org/dataset/" + datasetId;
      Runtime r = Runtime.getRuntime();
      Process p = r.exec("catalog_client.py remove " + url);
      p.waitFor();

      // Locally, remove the datasets
      String path = "~/aps/" + datasetId;
      FileUtils.deleteDirectory(new File(path));

      // Publish the "Detector.End"
      Token root = lc.getTokens("Detector.Created")[0];
      ad.publishTransition("Detector.End", lc);

      // Notify the user
      sendEmail("user@server.com", "APS - Corrupted dataset " + datasetId);
   }
};

ad.subscribeTo("Swift.Failure", handler);
```

Conclusion

- We proposed an implicit monitoring system for data intensive applications
- We provided users with intuitive features, making data management and analysis simpler
  - Progress monitoring
  - Automation
  - Error detection & recovery
- We did not change or alter any of the existing tools

Perspectives:

- Dynamically constructed life cycle models
- Integrate more systems
- Applications to provenance

Thank you!

Questions?