Active Data: Enabling Smart Data Life Cycle Management for Large Distributed Scientific Data Sets

Anthony Simonet

Inria, LIP – ENS Lyon, CNRS, University of Lyon

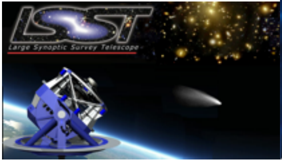
July 8th, 2015

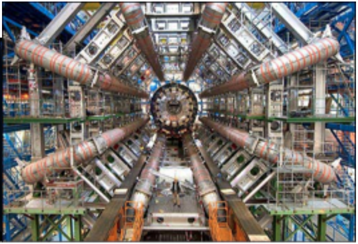
Outline

Outline

Big Data

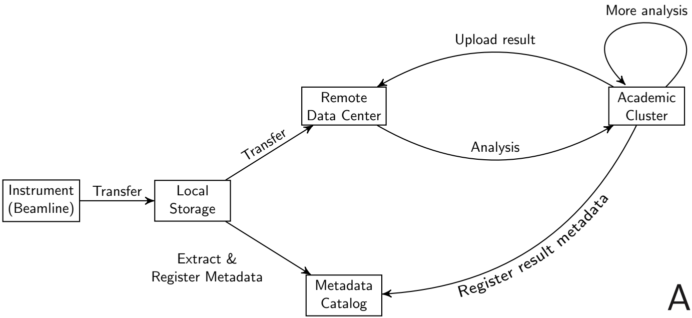
- Science and Industry have become data-intensive
  - Volume of data produced by science and industry grows exponentially
  - How to store this *deluge* of data?
  - How to extract knowledge and sense?
  - How to make data valuable?
- Some examples
  - CERN's Large Hadron Collider: 1.5PB/week
  - Large Synoptic Survey Telescope, Chile: 30 TB/night
  - Billion edge social network graphs
  - Searching and mining the Web



Cyber-Infrastructures for Data-Intensive Science

Infrastructures are globally distributed, heterogeneous and complex.
Example: the *Advanced Photon Source* experiment workflow.



Argonne NATIONAL LABORATORY

→ Assemblage of infrastructures that have very different characteristics (cost, administrative policy, local network and interconnection, performance).

Emergence of Data-Oriented Programming Models

Programmers need abstractions to exploit these complicated infrastructures. Programming models become implicitly parallel:

- *MapReduce*
- *AllPairs*
- *Pregel*
- *GraphLab*
- *Phœnix*
- *Ysmart*
- *Hive*
- *Spark*
- *Twister*
- *Pig*

Example: Evolution of the MapReduce Programming Model

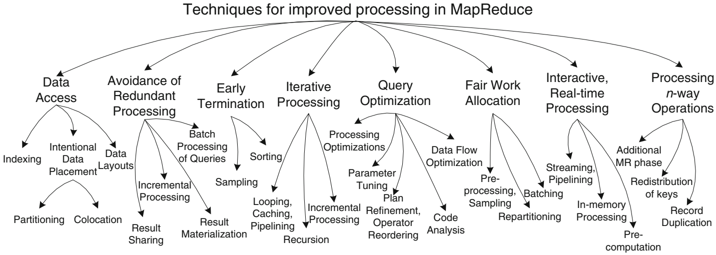


Figure: Taxonomy of MapReduce improvements for efficient query processing (source: Doulkeridis et al., 2014).

→ **People do not program data analysis without high-level abstractions anymore.**

Data Management Systems

Data Management System

A software system that performs one or more management operations on data as part of an application, such as storage, transfer, filtering, analysis.

Data management systems also provide high-level abstractions:

- High-level APIs to access heterogeneous resources
- Transparent data placement and replication
- Transparent fault tolerance
- Abstractions for mutating data
- New unstructured databases

Task-centric vs Data-centric

Workflow & dataflow systems are used to coordinate these systems.

Task Centric		Data Centric
▶ Task sequence		▶ Data-dependancy
▶ Implicit control flow		▶ Explicit control flow
▶ Monitor task completion	vs	▶ Monitor data production
▶ Coarse granularity		▶ Very fine granularity
▶ Hard to program, maintain, verify		▶ Direct link between data product & task
Swift, *DAGMan*, *Pegasus*		*Dryad*

→ Data intensive applications should be driven by data.

→ Infrastructure details must fade away, allowing programmers to focus on their analytics.

Data Provenance

Data provenance

The complete history of derivations and treatments throughout the life of data.

Recording and storing provenance:

- Helps preserving the quality of scientific data over time;
- Allows optimizations (recover vs regenerate);
- Is old research, but a new trend: scientists want to keep track of where their data sets come from.

- “The Open Provenance Model” (Moreau et. al, 2007)
- “Provenance-Aware Storage Systems” (Muniswamy-Reddy et. al, 2006)
- “The requirements of using provenance in e-science experiments” (Groth et. al, 2007)

Data Life Cycle: Definition

All of these management operations form the *Life Cycle* of data.

Definition 1

The *Data Life Cycle* is the course of operational stages through which data pass from the time when they enter a system to the time when they leave it.

- Creation/Acquisition
- Transfer
- Replication
- Disposal/Archiving

We need a rigorous approach for data management on heterogeneous distributed infrastructures.

Challenges with Data Life Cycle

More data is:

- more machines
- more disks
- more unexpected events

As the volume of data grows, managing the life cycle of distributed data requires more abstractions and the cooperation of more systems:

- Handling the complexity of infrastructures
- Handling the complexity of data management systems
- Being able to recover from unexpected situations
- Being able to exploit infrastructures at their best
- Allow cross-system optimizations

Challenges with Data Life Cycle: Related Works

Some attempts at addressing data life cycle management:

- "Addressing big data issues in scientific data infrastructure" (Demchenko et. al, 2013)
- "Storage and Data Life Cycle Management in Cloud Environments with FRIEDA" (Ramakrishnan et. al, 2015)

But:

- Until now, there has been no model for representing data life cycles formally in systems and across systems
- We need a model for specifying and programming data management applications

Objectives

This thesis aims at making distributed data life cycle management rigorous, easier and more efficient.

1. Offer a formal meta-model for representing the life cycle of distributed data in any system and across systems;
2. Define a model to provide a unified view of the same data in different systems and infrastructures;
3. Offer this unified view of the life cycle to users and programs with a programmable environment;
4. Propose a programming model that allows to develop data management applications by reacting to life cycle events, using the meta-model implementation;

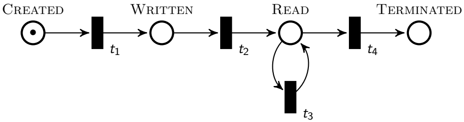
Outline

Methodology

- A formal approach:
  - Propose a model for representing the life cycle of data inside and across systems
  - Analyze data management systems, identify the features that must be modeled
  - Extensions of Petri Networks to construct a suitable meta-model
- An experimental approach:
  - Prototype implementation as a Java library (GPL)
  - Performance evaluation on Grid'5000
  - Evaluation of the programming model through usage scenarios and applications

Active Data principles

System programmers expose their system's internal data life cycle with a model based on Petri Nets.
A life cycle model is made of **Places** and **Transitions**

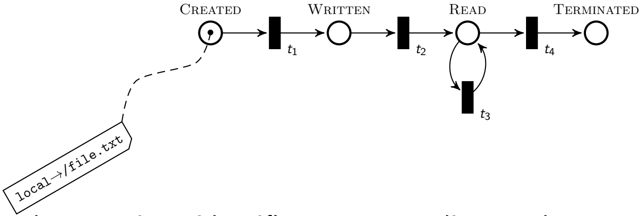


Each token has a unique identifier, corresponding to the actual data item's.

Active Data principles

System programmers expose their system's internal data life cycle with a model based on Petri Nets.

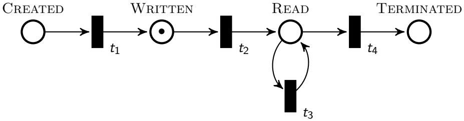
A life cycle model is made of **Places** and **Transitions**



Each token has a unique identifier, corresponding to the actual data item's.

Active Data principles

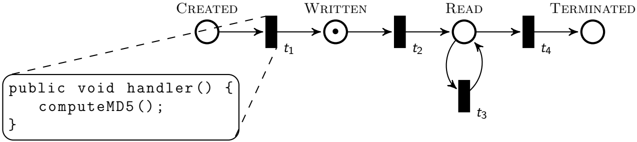
System programmers expose their system's internal data life cycle with a model based on Petri Nets.
A life cycle model is made of **Places** and **Transitions**



A transition is fired whenever a data state changes.

Active Data principles

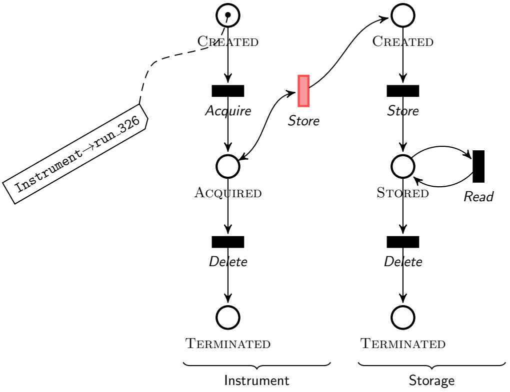
System programmers expose their system's internal data life cycle with a model based on Petri Nets.
A life cycle model is made of **Places** and **Transitions**



Code may be plugged by clients to transitions.
It is executed whenever the transition is fired.

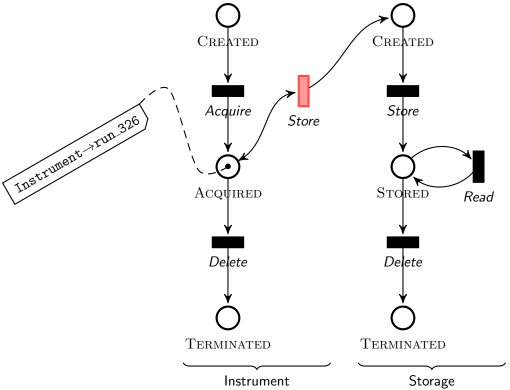
Composition

Offers a unified view of data in different systems...



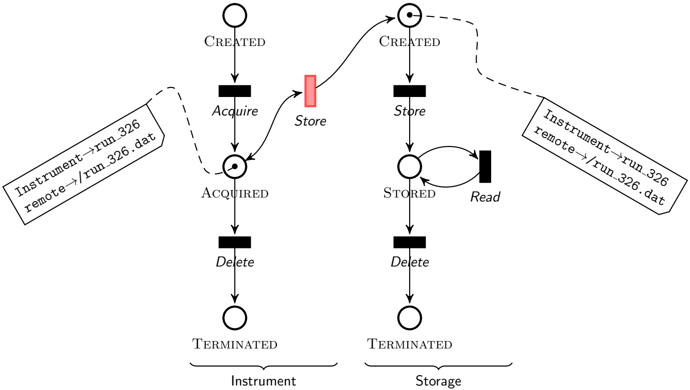
Composition

Offers a unified view of data in different systems. . .



Composition

. . . and keeps track of identifiers.



Life Cycle Meta Model

Petri Networks are a natural fit for representing life cycle models.

Definition 2

A Petri Network is 5-tuple $PN = (P, T, F, W, M_0)$ where:

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places represented by circles;
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions represented by rectangles;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of oriented arcs between places and transitions and between transitions and places;
- Places in a Petri Net may contain tokens represented by $\bullet$;
- $W : F \rightarrow \mathbb{N}^+$ is a weight function which indicates how many tokens every transition consumes and how many tokens it produces;
- $M_0 : P \rightarrow \mathbb{N}$ is a function that indicates the initial marking of places.

Life Cycle Meta Model

Life cycle models are Petri Networks with additional elements. . .

Definition 3

A data life cycle model is a 6-tuple $LC = (P, T \cup T', F \cup F', G, W, M_0)$ which represent respectively a set of places, transitions, arcs, inhibitor arcs, a weight function and an initial marking.

. . . for supporting data life cycle features:

- Identification
- Replication
- Composition
- Termination

Active Data API

First, Active Data needs to know the life cycle model of applications. Users construct an object-oriented representation of the LCM:

```
// Instantiate a Life Cycle Model
LifeCycleModel model = new LifeCycleModel("storage");

// Add places, transitions and arcs
Place created = model.getStartPlace();
Place written = model.addPlace("Written");
Place terminated = model.getEndPlace();
...
Transition write = model.addTransition("Write");
Transition delete = model.addTransition("Delete");
...
model.addArc(created, write);
model.addArc(write, written);
```

Then, systems must notify Active Data when they created new data:

```
// Publish the new life cycle
ActiveDataClient client = ActiveDataClient.getInstance();
LifeCycle lc = client.createAndPublishLifeCycle(model, "12345");
```

After that, Active Data will maintain the state of this data item and be able to receive transition publications.

Subscribing to transitions

Clients can react to DLC progress by *subscribing* code called *transition handler*. Active Data offers two ways of subscribing transition handlers:

- Subscribing to a transition for any data item

```
// Subscribe to the transition
TransitionHandler myHandler = new TransitionHandler() {
  public void handler(Transition transition, bool isLocal, Token[] inTokens, Token[] outTokens) {
    System.out.println("Reacting to transition " + transition.getName());
  }
}
client.subscribeTo(write, myHandler);
```

- Subscribing to any transition for a specific data item

```
// Subscribe to all transitions of the life cycle
TransitionHandler myHandler = new TransitionHandler() {
  public void handler(Transition transition, bool isLocal, Token[] inTokens, Token[] outTokens) {
    System.out.println("Reacting to transition " + transition.getName() + " for life cycle " +
      inTokens[0].getUid());
  }
};
client.subscribeTo(lc, myHandler);
```

Publishing & Querying

Data management systems must notify Active Data of operations they perform on Data. This is called *publishing a transition* and allows Active Data to update the state of the life cycle:

```
// Publish a transition
client.publishTransition(write, lc);
```

From a partial view (local identifier in a single system), Active Data allows to examine the global state of a data item (every token on every place).

```
// Query the complete state of another life cycle
LifeCycle lc = client.getLifeCycle("storage", "12345");
```

Clients can now *look* beyond their scope.

System Design

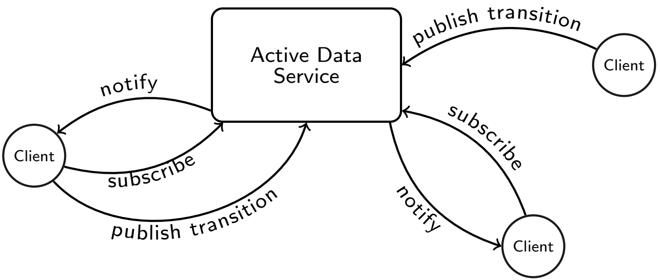


Figure: Architecture of Active Data: clients (data management systems and users) communicate with a centralized service in a Publish/Subscribe fashion.

Execution Model

Active Data's execution model

- is asynchronous, based on Publish/Subscribe
  - any client can be publisher and subscriber
  - facilitates deployment on uncooperative infrastructures
  - the service maintains a queue of transitions for each subscriber
  - Active Data orders handler execution by publication time
- allows to run transition handlers anywhere
- does not guarantee if or when transition handlers will be executed
- allows transition handlers to publish transitions

Token Tags & Guarded Execution

Systems can generate a very large number of notifications.

- Active Data allows *Tags* to be attached to tokens
- Then clients can subscribe their code to be executed only for tokens having certain tags (*Guarded Execution*)

Tags can be any string:

- File type, e.g. *"JPG"*, *"BINARY"*.
- Data collections
- Remote information, e.g. *"Test A passed"*.

Tags can be attached:

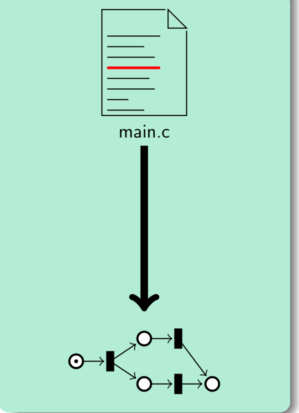
- On the server side, with no client intervention (*Taggers*)
- On the client side

Filtering is performed by the server.

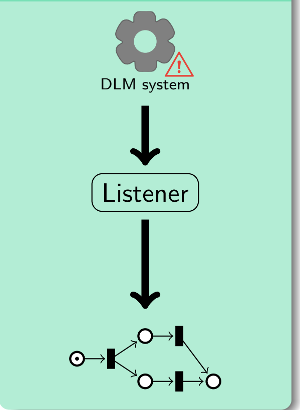
System Integration

Multiple intrusive and non-intrusive methods for making systems "Active Data" enabled:

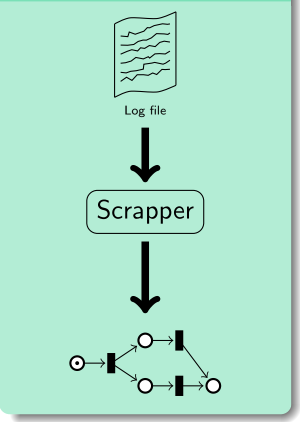
Instrumentation



Notification systems



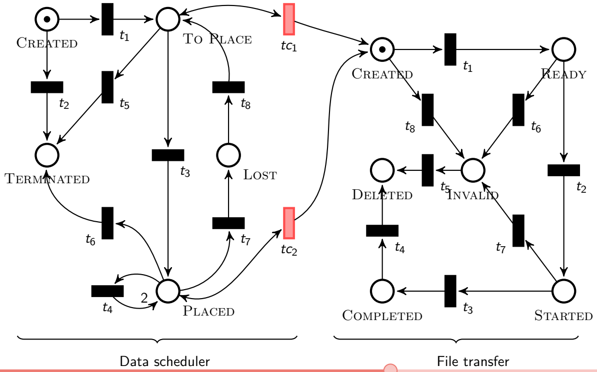
Log processing



System Integration

Five data management systems are already *Active Data*-enabled.

- **BitDew**
- inotify
- iRODS
- Globus Online
- Hadoop & HDFS



Outline

Context

Active Data

Evaluation

Data Surveillance

Conclusion

Micro benchmarks: Experimental set up

All experiments have been performed on two clusters of the Grid'5000 experimental testbed[1].

Cluster (Site)	Griffon (Nancy)	Suno (Sophia)
Nodes	92	45
CPUs	$2 \times$ 4-core Intel Xeon L5420 @ 2.5Ghz	$2 \times$ 4-core Intel Xeon E5520 @ 2.26GHz
Memory	16GB	32GB
Storage	320GB hard drive	$2 \times$ 300GB hard drives
Network	Gigabit Ethernet	
Operating system	Debian Linux 3.2	

[1] `http://www.grid5000.fr`

Micro benchmarks: Transition publication throughput

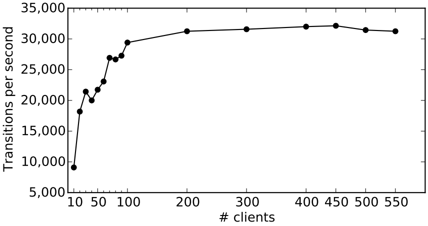


Figure: Average number of transitions handled by the Active Data Service per second with a varying number of clients. Each client publishes 10,000 transitions without pausing between iterations.

Micro benchmarks: Response Time & Overhead

		med	90th centile	std dev
Response time	Local	0.77 *ms*	0.81 *ms*	18.68 *ms*
	Eth.	1.25 *ms*	1.45 *ms*	12.97 *ms*

Overhead	Eth.	w/o AD 38.04 *s*	with AD 40.6 *s* (4.6%)

Table: Response time in milliseconds for life cycle creation and publication, transition publication and overhead measured using BitDew file transfers with and without Active Data.

Micro benchmarks

We run the Hadoop TeraSort benchmark with a 1TB data set, 280 mappers and 70 reducers.

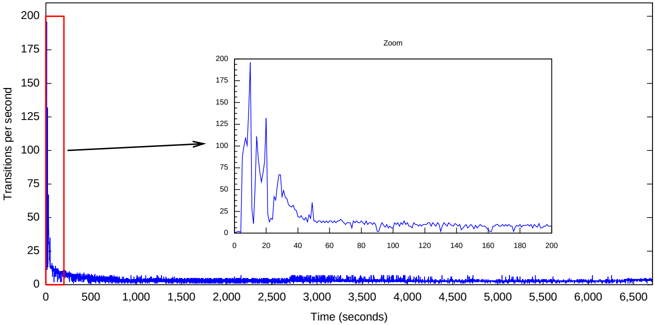


Figure: Number of transitions published each second during the Hadoop Terasort execution.

Maximum: 196 transitions per second.

Evaluation: Usage Scenarios

We evaluate the expressivity of the programming model with 4 usage scenarios:

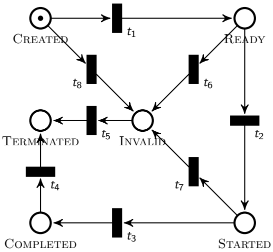
- Storage cache (writing distributed applications based on the data life cycle)
- Collaborative sensor network (managing data sets distributed across systems or infrastructures)
- Data provenance (using a unified life cycle for recording provenance across systems)
- **Incremental MapReduce** (optimizing an existing system for coping with dynamic data)

The use-cases also demonstrate how Active Data can improve existing systems by extending their scope to the whole application.

Use-Case: Incremental MapReduce

Once a system is *Active Data-enabled*, it can cope with dynamic data by subscribing to modification transitions.
Can we make BitDew MapReduce incremental by just changing a few lines of code?

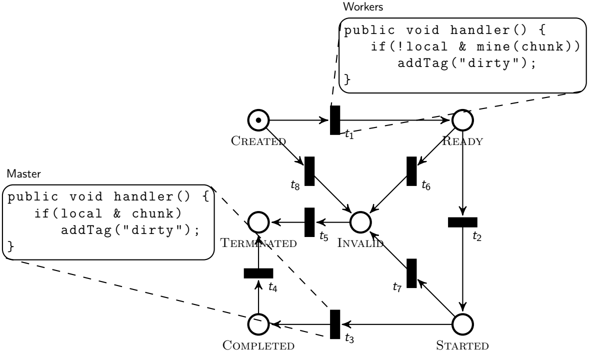
- Workers can *observe* all modifications of their input chunks
- When the job is re-executed, they can process only the modified chunks



Use-Case: Incremental MapReduce

Once a system is *Active Data-enabled*, it can cope with dynamic data by subscribing to modification transitions.
Can we make BitDew MapReduce incremental by just changing a few lines of code?

- Workers can *observe* all modifications of their input chunks
- When the job is re-executed, they can process only the modified chunks



Usage Scenarios: Incremental MapReduce

1. We measure the whole execution once
2. We modify a fraction of the input chunks and measure the time to re-run the job

Word count benchmark:

- 10 mappers
- 5 reducers
- 3.2 GB input file
- 200 16MB-chunks files

Fraction modified	20%	40%	60%	80%
Update time	27%	49%	71%	94%

Table: Incremental MapReduce: time to update the result compared with the fraction of the data set modified.

Significant speedup with less than 2% of the code changed thanks to Active Data.

Outline

Context

Active Data

Evaluation

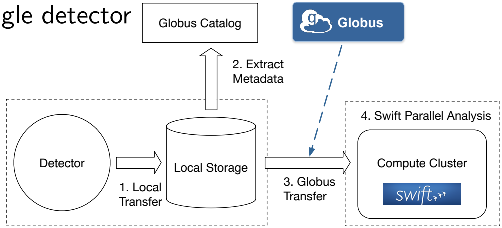
Data Surveillance

Conclusion

The Advanced Photon Source

Data-intensive distributed application involving multiple software

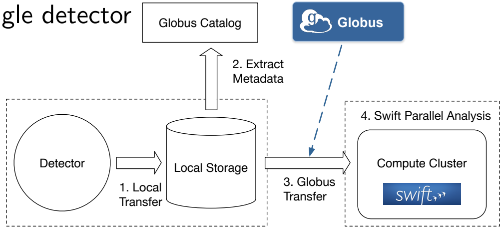
- 3 to 5TB of data per week on a single detector
- 3 tools involved:
  - Globus Transfers
  - Globus Catalog
  - Swift
- Tasks are launched manually



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



What is inefficient in this workflow?

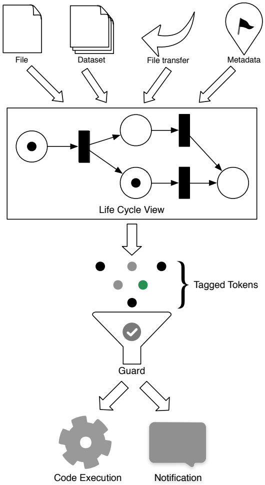
- Many error-prone tasks are performed manually
- Users can't monitor the whole process at once
- Small failures are difficult to detect
- A system alone can't recover from failures caused outside its scope

Goals

We want to use Active Data to achieve the following goals:

- End-to-end progress monitoring
- Automation
- Error discovery & recovery
- Sharing & notifications

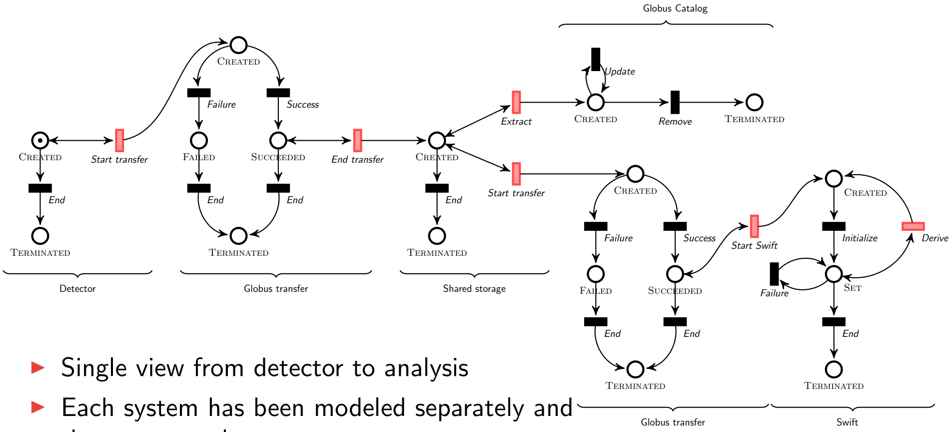
Data Surveillance Framework



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

APS Experiment Life Cycle Model



- Single view from detector to analysis
- Each system has been modeled separately and then composed

Results: Error Detection & Recovery

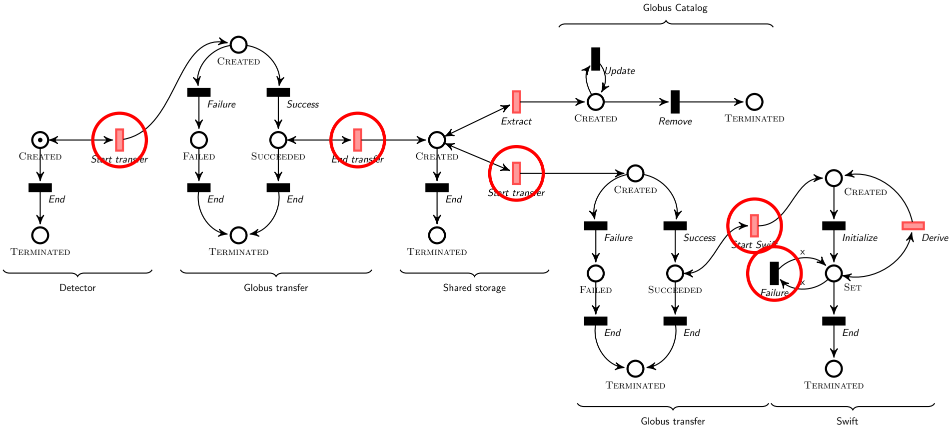
Example scenario

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

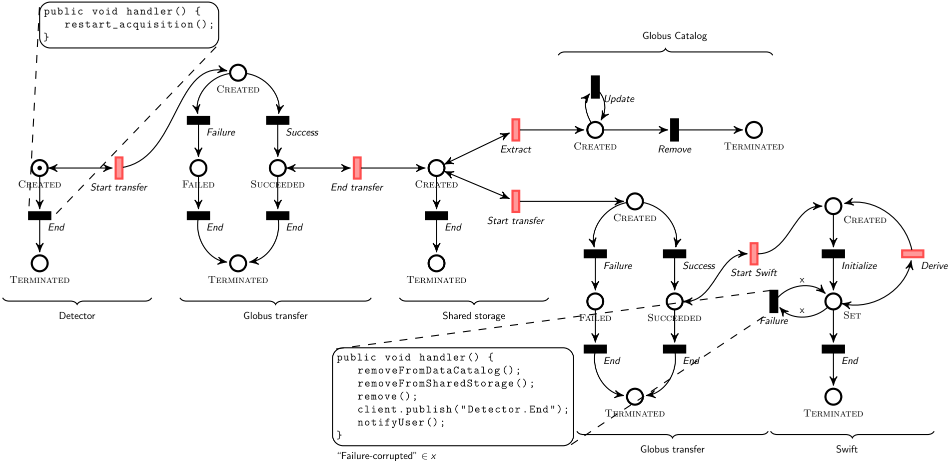
In this situation, the user manually:

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

Results: Error Detection & Recovery



Results: Error Detection & Recovery



Outline

Conclusion

This thesis tackles the problem of managing large-scale data sets on hybrid infrastructures, with a formal and an experimental approach:

- We studied the characteristics of applications and devised the first meta-model to represent them
- We proposed *Active Data*, implementation of the model that brings an end-to-end view of applications to programs and users
- We proposed a programming model for managing distributed data sets
- We evaluated the programming model with micro-benchmarks and usage scenarios
- We confronted Active Data to a real-life application in collaboration with ANL

Perspectives

There is always more to be done:

- Provenance recording
- Data traceability
- Cross-system optimizations
- Cloud service deployment
  - Asma Ben Cheikh (University of Tunis)
- Verification

Thank you!

Questions?