Active Data
A Data-Centric Approach to Data Life-Cycle Management

Anthony Simonet\textsuperscript{1}  Gilles Fedak\textsuperscript{1}
Matei Ripeanu\textsuperscript{2}  Samer Al-Kiswany\textsuperscript{2}

\textsuperscript{1}Inria, ENS Lyon, University of Lyon  \textsuperscript{2}University of British Columbia

November 18th, 2013
Outline

Introduction
  Data Life Cycle Management
  Use-case
  Requirements

Active Data
  Active Data: principles & features
  Exemple: Globus Online and iRODS

Discussion
  Advantages
  Limitations

Conclusion
  Related works
  Conclusion
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
Data Life Cycle

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

Definition

The 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.
Complicated scenarios

- Execution of workflow
- Complex interactions between software
- Need to quickly react to operational events

Ad-hoc task-centric approaches

- Hard to program, maintain and debug
- No formal specification
- Complicates interactions between systems
Data Life Cycle Use-case

Example: the Advanced Photon Source at Argonne National Lab

- 100TB of raw data per day
- Raw data are preprocessed and registered in a Globus dataset catalog
- Data are analyzed by various applications
- Results are stored in the dataset catalog and shared
**Use-case**

**Task Centric**
- Independent scripts
- Hard to program, maintain, verify
- Coarse granularity

**Data Centric**
- Express data-dependencies
- Cross data-center coordination
- User-level fault-tolerance
- Incremental processing
Requirements

Challenges: a perfect system should . . .

▶ Simply represent the life cycle of data distributed across different data centers and systems
▶ Simplify DLM modeling and reasoning
▶ Hide the complexity resulting from using different infrastructures and systems
▶ Be easy to integrate with existing systems
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.
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.
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.
Active Data features

The Active Data programming model and runtime environment:

- Allows to react to life cycle progression
- Exposes transparently distributed data sets
- Can be integrated with existing systems
- Has scalable performance and minimum overhead over existing systems
Implementation

- Prototype implemented in Java (≈ 2,800 LOC)
- Client/Service communication is Publish/Subscribe
- 2 types of subscription:
  - Every transitions for a given data item
  - Every data items for a given transition
Several ways to publish transitions

- Instrument the code
- Read the logs
- Rely on an existing notification system

The service orders transitions by time of arrival
Implementation

- Clients run transition handler code locally
- Transition handlers are executed
  - Serially
  - In a blocking way
  - In the order transitions were published
Performance evaluation: Throughput

Figure: Average number of transitions per second handled by the Active Data Service

Clients publish 10,000 transitions in a row without pausing.
Performance evaluation: Throughput

Figure: Average number of transitions per second handled by the Active Data Service

The prototype scales up to 30,000 transitions per seconds.
**Exemple: Data Provenance**

**Definition**

The complete history of data life cycle derivations and operations.

- Assess the quality of data
- Keep track of the origin of data over time
- Specialized Provenance Aware Storage Systems
Exemple: Data Provenance

Definition

The complete history of data life cycle derivations and operations.

- Assess the quality of data
- Keep track of the origin of data over time
- Specialized Provenance Aware Storage Systems
  \[\rightarrow\text{What about heterogeneous systems?}\]
**Definition**

The complete history of data life cycle derivations and operations.

- Assess the quality of data
- Keep track of the origin of data over time
- Specialized Provenance Aware Storage Systems

$\rightarrow$ What about heterogeneous systems?

Example with **Globus Online** and **iRODS**

- File transfer service
- Data store and metadata catalog
Data events coming from Globus Online and iRODS

Exemple: Globus Online and iRODS
Exemple: Globus Online and iRODS

Data events coming from Globus Online and iRODS

```java
public void handler() {
    iput(...);
}
```
Data events coming from Globus Online and iRODS

Exemple: Globus Online and iRODS

Id: {GO: 7b9e02c4-925d-11e2, iRODS: 10032}
Data events coming from Globus Online and iRODS

```java
public void handler() {
    annotate();
}
```
Exemple: Globus Online and iRODS

$ imeta ls -d test/out_test_4628
AVUs defined for dataObj test/out_test_4628:
attribute: GO_FAULTS
value: 0
---
attribute: GO_COMPLETION_TIME
value: 2013-03-21 19:28:41Z
---
attribute: GO_REQUEST_TIME
value: 2013-03-21 19:28:17Z
---
attribute: GO_TASK_ID
value: 7b9e02c4-925d-11e2-97ce-123139404f2e
---
attribute: GO_SOURCE
value: go#ep1/~test
---
attribute: GO_DESTINATION
value: asimonet#fraise/~out_test_4628
Advantages

- Simple and graphical way to program DLM operations
- Allows to formally verify some properties of data life cycles
- Easy coordination between systems
- Easy to scale
- Easy to debug
- Easy fault tolerance
- Fine-grain interaction with data life cycle
Limitations

- Complexity to reason in terms of life cycle events
- Lack of standard
Related works

Data-centric parallel processing

- Programing models:
  - MapReduce and higher level abstractions: PigLatin, Twister
  - Incremental systems: MapReduce-Online, Percolator, Chimera, Nephele
  - Other models with implicit parallelism: Swift, Dryad, Allpairs

- Storage systems
  - BitDew
  - MosaStore
  - Provenance Aware Storage Systems
  - Active Storage
Active Data is . . .

▶ Data-centric & Event-driven
▶ System-level data integration

What’s next?

▶ Advanced representation of operations that consume and produce data: represent data derivation
▶ Data collection abilities
▶ Distributed implementation of the Publish/Subscribe layer
Thank you!

Questions?

Inria booth #2116