# Active Data: A Programming Model to Manage Data Life Cycle Across Heterogeneous Systems and Infrastructures

A. Simonet[1], G. Fedak[1], M. Ripeanu[2]

(1) {Anthony.Simonet, Gilles.Fedak}@inria.fr
INRIA, University of Lyon, France

(2) matei@ece.ubc.ca
University of British Columbia, Canada

The Ninth Workshop of the INRIA-Illinois Joint Laboratory on Petascale Computing
June 12-14, 2013, Lyon, France

# Outline of Topics

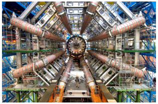

1. Introduction
   - Challenges of Life-Cycle Management
   - Active Data at a Glance
   - Related Works
2. The Model and Implementation
   - Life-Cycle Model
   - Execution Runtime
3. Experiments
   - Performance Evaluation
   - Use Cases
4. Conclusion

# Outline

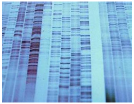

1. Introduction
   - Challenges of Life-Cycle Management
   - Active Data at a Glance
   - Related Works
2. The Model and Implementation
   - Life-Cycle Model
   - Execution Runtime
3. Experiments
   - Performance Evaluation
   - Use Cases
4. Conclusion

# Context: Data Deluge

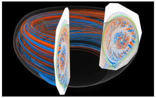
- Huge and growing volume of information originating from multiple sources.



Big Science



Instruments



Simulations


Internet


Open Data

- Impacts many scientific disciplines and industry branches
  - Large Scientific Instruments (LSST, LHC, OOOI), but not only (Sequencing machines)
  - Internet and Social Network (Google, Facebook, Twitter, etc.)
  - Open Data (Open Library, Governemental, Genomics)

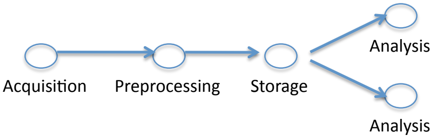
→ impacts the whole process of scientific discovery ($4^{th}$ paradigm of science)

# Big Data Challenges

- Big Data creates several challenges :
  - how to scale the infrastructure ?
    - end-to-end performance improvement, inter-system optimization.
  - how to improve productivity of data-intensive scientist ?
    - workflow, programming language, quality of data provenance.
  - how to enable collaborative data science ?
    - incentive for data publication, data-sets sharing, collaborative workflow.
- New models and software are needed to represent and manipulate large and distributed scientific data-sets.

# Focus on Data Life-Cycle

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




We're aiming at :

- A model to capture the essential life cycle stages and properties: creation, deletion, faults, replication, error checking . . .
- Allows legacy systems to expose their intrinsic data life cycle.
- Allow to reason about data sets handled by heterogeneous software and infrastructures.
- Simplify the programming of applications that implement data life cycle management.

# Active Data at a Glance

- a *life cycle model*, inspired by Petri Net, which allows data management systems to expose data life cycle through a well-formalized representation
- a *programming model* and a *runtime environment*, which allows to program applications by specifying for each step of the data life cycle, the code that will be executed.

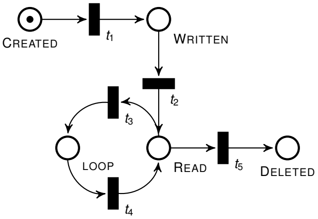



Figure: Representation of the *"Write-Once, Read-Many"* data life cycle.

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

# Active Data at a Glance

- a *life cycle model*, inspired by Petri Net, which allows data management systems to expose data life cycle through a well-formalized representation
- a *programming model* and a *runtime environment*, which allows to program applications by specifying for each step of the data life cycle, the code that will be executed.

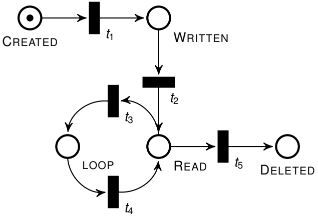



Figure: Representation of the *"Write-Once, Read-Many"* data life cycle.

```
TransitionHandler md5Handler = new ↘
    →TransitionHandler() {
  public void handler(LifeCycle lc, String ↘
      →transitionName, boolean isLocal) {
    MessageDigest md = ↘
        →MessageDigest.getInstance("MD5");
    String path = getPath(lc.getId());
    InputStream input = new ↘
        →FileInputStream(path);
    byte buffer[] = new byte[2048];

    int n = 0;
    while((n = input.read(buffer)) > 0)
      md.update(buffer, 0, n);

    byte[] digest = md.digest();
    BigInteger bigInt = new ↘
        →BigInteger(1,digest);
    String hash = bigInt.toString(16);
    while(hash.length() < 32 )
      hash = "0" + hash;

    OutputStream output = new ↘
        →FileOutputStream(path + ".md5");
    output.write(hash.getBytes());
    output.close();
  }
};
```

# Related Works

- Data-centric parallel programming languages (MapReduce, Dryad, Allpairs, Twister, PigLatin . . . )
- Runtime execution environments for dynamic data : incremental processing (Percolator), parallel stream processing (Nephele, MapReduce Online), workflow (Chimera)
- Event based processing (Mace, libasync, Incontext)
- Data Provenance addresses the issue of representation of data-set derivation (PASS, Open Provenance Model)
- Data Management Software (BitDew, Chirp, MosaStore, Globus Online, DCache, iRods and many more)

# Outline


1. Introduction
   - Challenges of Life-Cycle Management
   - Active Data at a Glance
   - Related Works
2. The Model and Implementation
   - Life-Cycle Model
   - Execution Runtime
3. Experiments
   - Performance Evaluation
   - Use Cases
4. Conclusion

# Life-Cycle Model: the Basics

- A Petri Net is a 5-tuple $PN = (P, T, F, W, M_0)$ where:
  - $P = \{\text{CREATED}, \text{DELETED}, p_1, p_2, \ldots, p_m\}$ is a finite set of places;
  - $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions;
  - Data items are represented by tokens $\bullet$.
  - $W : F \rightarrow \mathbb{N}^+$ is a weight function which indicates how many tokens every transition requires and how many token it produces.
- **Data Identification**, we tag each token $\delta$ with a triplet $(id, i, p)$ where $id$ is a unique data identifier, identical for all the replica of a single data item, $i$ is the replica number and $p \in P$ is a place.
- **Data Creation** a data item is created as a token in the CREATED place labelled with the triplet $(id, 1, \text{CREATED})$.
- **Data Replication**




Figure: Data instance creation.

# Life-Cycle Model: the Basics

- A Petri Net is a 5-tuple $PN = (P, T, F, W, M_0)$ where:
  - $P = \{\text{CREATED}, \text{DELETED}, p_1, p_2, \ldots, p_m\}$ is a finite set of places;
  - $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions;
  - Data items are represented by tokens $\bullet$.
  - $W : F \to \mathbb{N}^+$ is a weight function which indicates how many tokens every transition requires and how many token it produces.
- **Data Identification**, we tag each token $\delta$ with a triplet $(id, i, p)$ where *id* is a unique data identifier, identical for all the replica of a single data item, *i* is the replica number and $p \in P$ is a place.
- **Data Creation** a data item is created as a token in the CREATED place labelled with the triplet $(id, 1, \text{CREATED})$.
- **Data Replication**

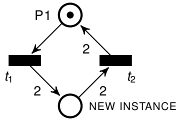



Figure: Data instance creation.

# Composition of Life-Cycle Model

- **Life Cycle Composition** To compose two independent life cycles $A$ and $B$, we define the *start places*, $Sa_A \subseteq P_A$, and *stop places*, $So_B \subseteq P_B$.
  - when one token reaches $Sa_A$, it creates a new token in the place CREATED of $B$. New tokens are labeled with the quadruplet $(id_A, id_B, r_B, q_B)$.
  - When one token reaches $So_B$, it creates a new token in the place DELETED of $A$.
- **Life Cycle termination** We say that a data item *terminates* its life cycle, when all of its tokens reach the DELETED place.
- **Composed Life Cycles Termination** We say that a data item *terminates* a composed life cycle, represented by $A$ and $B$, their start sets $Sa_A$ and $Sa_B$, and stop sets $So_A$ and $So_B$, when it terminates $A$ and terminates $B$.

# Composition of Life-Cycle Model

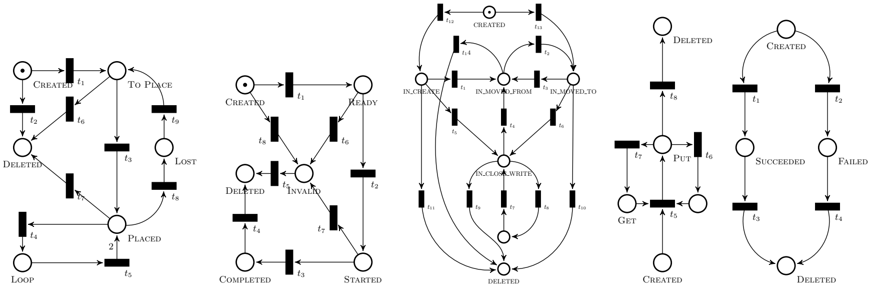
- **Life Cycle Composition** To compose two independent life cycles $A$ and $B$, we define the *start places*, $Sa_A \subseteq P_A$, and *stop places*, $So_B \subseteq P_B$.
  - when one token reaches $Sa_A$, it creates a new token in the place CREATED of $B$. New tokens are labeled with the quadruplet $(id_A, id_B, r_B, q_B)$.
  - When one token reaches $So_B$, it creates a new token in the place DELETED of $A$.
- **Life Cycle termination** We say that a data item *terminates* its life cycle, when all of its tokens reach the DELETED place.
- **Composed Life Cycles Termination** We say that a data item *terminates* a composed life cycle, represented by $A$ and $B$, their start sets $Sa_A$ and $Sa_B$, and stop sets $So_A$ and $So_B$, when it terminates $A$ and terminates $B$.

# Execution Runtime

Active Data is composed of two parts:

- the *execution runtime*
  - manages data life cycles, publishes transitions, triggers handler code executions, and guarantees execution correctness
  - distributed system based on Publish/Subscribe
- the *programming interface* (API)
  - allows data management systems to publish transitions
  - allows programmers to develop applications by registering their transition handlers
  - two kinds of transition subscription :
    - subscribe to a specific transition for any data items
    - subscribe to a specific data item and being notified for any life cycle transitions.

# Integration with Data Management Systems




(a) Bitdew Scheduler (b) Bitdew File Transfer (c) inotify (d) iRODS (e) Globus Online

- BitDew (INRIA), programmable environment for data management.
- inotify Linux kernel subsystem: notification system for file creation, modification, write, movement and deletion.
- iRODS (DICE, Univ. North Carolina), rule-oriented data management system.
- Globus Online (ANL) offers fast, simple and reliable service to transfer large volumes of data.

- Data scheduling and replication
- Fault tolerance
- Composition of File Transfer and Data Scheduler

# Outline


1. Introduction
   - Challenges of Life-Cycle Management
   - Active Data at a Glance
   - Related Works
2. The Model and Implementation
   - Life-Cycle Model
   - Execution Runtime
3. Experiments
   - Performance Evaluation
   - Use Cases
4. Conclusion

# Performance Evaluation

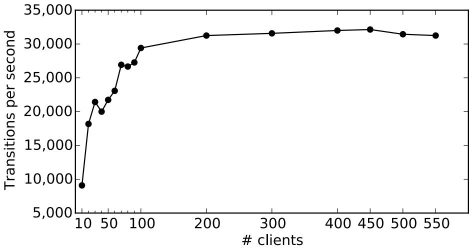



Figure: Average number of transitions per second handled by the Active Data Service

| | | med | $90^{th}$ centile | std dev |
|---|---|---|---|---|
| Latency | 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<br>38.04 *s* | with AD<br>40.6 *s* (4.6%) |

Table: Latency in milliseconds for life cycle creation and transition publication and overhead measured using BitDew file transfers (1K files) with and without Active Data.

# Amazon S3 Cache

Scenario: Caching Amazon S3

- evaluates the ability to rapidly prototype a data management application using Active Data and BitDew.
- *Write-through* cache policy expressed using life-cycle transitions.
- evaluated using a pseudo master-worker application (10 nodes, 200MB input)

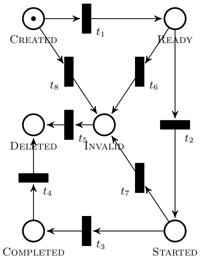


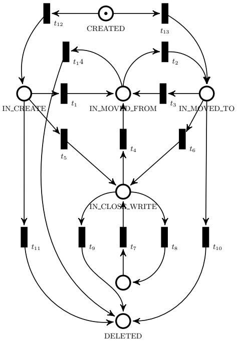

Figure: BitDew File Transfer

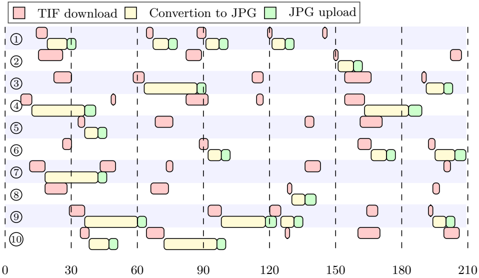
| | w cache | w/o cache | Difference |
|---|---|---|---|
| In | 2350 Mb | 2350 Mb | 0 Mb |
| Out | 0.15 Mb | 1976.17 Mb | 1976.02 Mb |
| #Put | 13 | 13 | 0 |
| #Get | 0 | 20 | 20 |
| Dollars | 0.3 | 0.53 | 0.23 |

- $t_1$ If *cache hit*, the handler serves the file from the cache; if *cache miss*, the handler gets the file from Amazon S3
- $t_4$ the handler transfers the data item to Amazon S3 + cache eviction policy.

# Distributed Sensor Networks

- evaluates the ability to develop distributed application based on data life-cycle.
- sensors: images acquisition, pre-processing, archiving to centralized remote storage
- use Active Data to implement *distributed data throttling*




- $t_{12}$: we check if the transition is local or remote: if it is remote we increment the local counter $p$.
- $t_5$: if the transition is local, we upload the file only if $p < n$.

# Incremental MapReduce

**Scenario:** Incremental MapReduce

- we investigate if an existing system can be optimized by leveraging on Active Data's ability to cope with dynamic data.
- Making MapReduce incremental : re-run map and reduce tasks only for the data input chunks that have changed
- Evaluated using the word count application (10 mappers, 5 reducers, 3.2 GB split in 200 chunks)

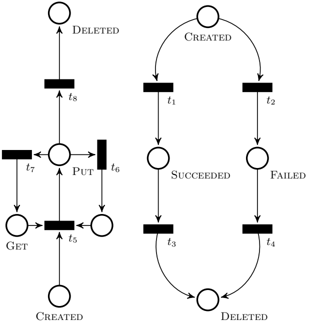
- Implemented using BitDew-MapReduce
- Active Data notifies that a file transfer is modifying an input chunk. The Mapper Transition handler flags the chunk as *dirty*.
- the mapper executes again the map task on dirty chunk and sends the updated intermediate results to the reducers.
- Reducers proceed as usual to compute again the final result.

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

# Data Provenance

Scenario: Data Provenance

- take advantage Active Data's unified view of data-sets over *heterogeneous systems*.
- file transfers service (Globus Online) + metadata catalog (iRODS).
- query iRods and obtain file transfer information : endpoints, completion date, request time




(d) iRODS (e) Globus Online

- To compose the two life cycles, the place SUCCEEDED from Globus Online is a start place which creates a token in the iRODS life cycle model.

- $t_1$ a handler to store the file in iRODS. The token now contains Globus Online and iRods identifiers .
- $t_5$ the handler queries Globus Online to get file transfer information and publish as iRods meta-data.

# Outline


1. Introduction
   - Challenges of Life-Cycle Management
   - Active Data at a Glance
   - Related Works
2. The Model and Implementation
   - Life-Cycle Model
   - Execution Runtime
3. Experiments
   - Performance Evaluation
   - Use Cases
4. Conclusion

# Conclusion

## Active Data, a programming model for supporting data life cycle management applications

- formal definition of the data life cycle allows unified view of data across heterogeneous systems and infrastructures
- runtime execution environment + API to publish transitions and execute transition handlers,
- low overhead ($< 4\%$) and high transition throughput ($> 30K/sec$)
- ability to plug into legacy systems (data scheduler, file system, file transfer service, rule-based management system)
- Evaluated with several complex use cases.

## Future Works/Collaborations

- Model: advanced representation of computations; collective operations on data sets.
- Runtime: rollback mechanisms for fault-tolerant execution; distributed implementations of the publish/subscribe substrate.
- Application leveraging Active Data: a MapReduce runtime for dynamic data, incremental and asynchronous workflow (Swift/Mosastore)

Thank you !