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

Anthony Simonet, Gilles Fedak, Matei Ripeanu, Samer Al-Kiswany

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

Anthony Simonet[1], Gilles Fedak[1], Matei Ripeanu[2], Samer Al-Kiswany[2]
[1]INRIA/University of Lyon, France {anthony.simonet, gilles.fedak}@inria.fr
[2]Univeristy of British Columbia, Canada {matei, samera}@ece.ubc.ca

***Abstract*— Data-intensive science offers new opportunities for innovation and discoveries, provided that large datasets can be handled efficiently. Data management for data-intensive science applications is challenging; requiring support for complex data life cycles, coordination across multiple sites, fault tolerance, and scalability to support tens of sites and petabytes of data. In this paper, we argue that data management for data-intensive science applications requires a fundamentally different management approach than the current ad-hoc task centric approach. We propose Active Data, a fundamentally novel paradigm for data life cycle management. Active Data follows two principles: data-centric and event-driven. We report on the Active Data programming model and its preliminary implementation, and discuss the benefits and limitations of the approach on recognized challenging data-intensive science use-cases.**

Keywords—data management, distributed storage system

I. INTRODUCTION

Modern science is data-intensive. Large-scale simulations, new scientific instruments, and large-scale observatories all generate massive volumes of data that need to be transferred, preprocessed, mined for insights, and archived, often, by large geographically dispersed user communities. This trend is emerging in fields as diverse as bioinformatics, high-energy physics (e.g., the Large Hadron Collider experiment at CERN and the DØ experiment at Fermi Lab), satellite image processing (e.g., to detect earth movements), or sensor network based applications (e.g., in seismology, ocean science, or wild life monitoring).

We dub this sequence of complex data-oriented operations that support these science communities *the data life cycle*. 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. Data enter the system when they are acquired by an instrument, or created from some other data already present in the system; they leave it when they are physically erased, or when moved to storage outside of the system. Between these two points in time, data progress through a series of different stages (e.g., acquisition, cleanup, duplication, archival, transfer) that are either application initiated (e.g., transformation, aggregation, metadata extraction) or triggered by external events (e.g., failures that lead to data loss). In the remaining of the paper, we will use the terms *data life cycle* to denote the possible *states* of a data item (e.g. created, duplicated, deleted, backed-up), and *the sequence of state transitions* triggered by data operations on data (copy, transfer, duplicate, transform) or by external events (e.g., device failure).

Managing the data life cycle is complex for multiple reasons: First, as the volume of data grows and the science supported becomes more involved, the *data life cycle management* (DLM) systems deal with increasingly complex scenarios that coordinate the various operations performed on data which include acquisition, transfer, preprocessing, replication, caching, processing, archiving, disposal all interlaced in complex interactions. A second source of complexity is the need for coordination across storage and processing systems that have not been necessarily designed to work together, as well as the need to react in real-time to operational events (e.g., device failures, data corruption, space management). Third, the complexity of the life cycle is increased by the fact that data is often available in batches rather than as full datasets, by the fact that humans decisions are often involved in the life cycle, and by the need for efficient resource use (e.g., in many cases support for incremental processing is key for efficiency).

The current approach to data life cycle management is an ad-hoc task centric approach: each task in the life cycle is programmed independently, and tasks represent the main unit for scheduling and monitoring. This approach has the following fundamental issues: first, it makes it hard to program, maintain, debug, and verify the scripts that specify the life cycle, because of the lack of a formal system model that explicitly specifies the dependency between the tasks and the various stages data go through. Second, a task-oriented view complicates the coordination between different participating systems such as the instrument, the buffered pre-processing stages, and the tiered storage, due to the lack of central service or unified framework for coordination. Third, this complicates fault tolerance, as reaction to external events (e.g., failure that lead to loss of data) cannot be naturally modeled in this view. We believe scientific applications require a fundamentally different paradigm for data life cycle management, which has two distinguishing characteristics: *data-centric* (as opposed to task-centric) and *event-driven* (as opposed to task-completion triggered), data management approach.

Active Data, our implementation of this approach, formally defines the data life-cycle and presents a programming model to allow expressing the operations to be executed at each stage of the data life cycle. In Active Data, the programmer, specifies the set of data-related events (e.g., data item creation, replication, transfer completion, data loss, deletion) to be monitored per data item and programs the operations to be executed when these events happen. This programming model allows developing a broad range of data life cycle management (DLM) applications such as automated tiered storage, processing attached to any stage of the life cycle, coordination between data acquisition

mechanisms and remote storage, content delivery networks, deep storage archival, incremental data management, and so forth.

The rest of this paper first motivates the Active Data approach through presenting a use case (section II), derives the Active Data life cycle management system requirements (section III), sketches the active data programing model (section IV), discusses the advantages and possible limitations of the proposed approach (section **Erreur ! Source du renvoi introuvable.**), and presents a preliminary evaluation of a system prototype (section V). We conclude with a survey of the related work (section VII) and a discussion of the future work (section VIII).

II. A Data Life cycle Management System Use Case

We review a key use case for data-management life cycle systems. Data life cycle in modern scientific applications involves a number of operations and complex interactions. The life cycle often includes most or all of the following: transfer from the data source (e.g., a science instrument generating the data or a simulation), metadata extraction, cleanup, preprocessing to extract the main features from each data item, compression of the raw data, data transfer to the collaborating sites, data archival, various forms of data transformation, fusing, aggregation, visualization, result archival, reaction to failures that led to data loss, and data disposal.

One example is, the Advanced Photon Source (APS) at Argonne National Lab [1]. In APS data is generated at a beamline facility; around 100TB of data per day. The APS center preprocesses the raw data to reduce its size. The preprocessed data is then transferred to collaborating sites for further processing and analysis. There data is first processed to extract and register metadata in a Globus dataset catalog [2]. Next the data is analyzed through various applications. The analysis results and provenance data are stored again in the same Globus dataset catalog and made available to external users. Finally, the data may be migrated to a persistent storage.

Currently, the APS data life cycle is managed through a set of independent scripts. Different stages in the life cycles are either triggered by the users, or through ad hoc interfaces (e.g. using ssh). This approach makes it hard to program, maintain, verify and debug the data management scripts, specially for complex data life cycles.

Besides reliably and scalably managing the data life cycle itself this use case highlights a number of additional challenges:

1. *Cross data-center coordination*: The data life cycle of modern applications often spans multiples research centers. Often one center generates the raw data, while multiple centers collaborate on processing and storing it.

   Unlike APS data life cycle, a number of scientific applications, especially in sensor networks, deal with data life cycles with complex coordination across multiple nodes or centers. For example, sensor network applications (e.g., in seismology, ocean science, or wide life monitoring) the application often needs only a subset of predetermined size of the current sensors' data. Avoiding processing and transfer of all sensor data can reduce the system overhead, save energy, and increase system efficiency. This requires, however, coordinated data throttling technique across multiple sites/nodes: that is the application needs at least *p* sensor samples in a time window. All other sensor data can be discarded without any processing.
2. *Extensibility*. It should be easy to extend the Active Data framework by defining new events, the event collection mechanism, and program the event handler operation. This is necessary to extend the framework to handle data events not known at framework implementation time, and to handle all possible failure events.
3. *Incremental processing*. A number of scientific (e.g. earth seismic monitoring and meteorology applications) and commercial applications run in epochs or process streaming data. Each epoch processes the raw data gathered since the last epoch, together with the last epoch results, to produce new results.

III. Data Life Cycle Management Systems: The Requirements

Through analyzing the target application domain, this section derives the requirements for the Active Data framework and programing model. In addition to addressing the challenges listed in section II, the framework should:

- *Make it simple to specify the data life cycle for various contexts and automate support for life cycle management (DLM) systems.* This involves: giving the programmer the ability to define events, and event handlers, and data life cycle progression, and providing the tools for runtime monitoring and verification.
- *Simplify the DLM modeling and reasoning.* A model is required to capture the essential life cycle stages and properties: creation, deletion, faults, replication and error checking. Further, the model should be able to model operations across decoupled systems, and be able to mix data life cycles of various systems.
- *Be easy to integrate with existing systems*, which requires an *extensible* framework that can be easily integrated with existing systems, detect or receive their events, and run operations.
- *Have scalable performances and minimum performance overhead over existing systems.*

IV. Active data The Programing Model

Our response to the above requirements is *Active Data*, a programming model and a runtime environment, which allows programing DLM applications by specifying for each state transition in the data life cycle, the code that will be executed. In this approach the programmer first models the data life cycle, then defines the events that trigger state transitions, and implements the events handlers that get executed at state transitions.

Data life cycle modeling. Our modeling approach is inspired by the Petri Nets model [3]. This is a widely used approach for modeling complex systems with concurrency and resource sharing. A number of tools exist for graphical visualization of Petri Nets or simulating, analyzing and

verifying its properties [4]. Petri Nets can depict intuitively data life cycles (Figure 1 presents an example): *Places*, represented by circles are the states of the life cycle; *Transitions*, represented by rectangles are the operations that happen on data items; *Tokens*, represented by • in Places, are data items in a particular state of the life cycle.

Further, to express complex data life cycles our modeling approach extends Peri Nets with three features: *(i)* handling of replicated data. Each replica is represented by a single Petri Net token. Several tokens on different places represent the individual states of several data replicas distributed on different nodes; *(ii)* life cycle termination rules, which allows detecting errors and illegitimate operations, and *(iii)* an approach to present a unified view of the life cycle even though it involves several heterogeneous systems. Due to space limitation we do not detail our data life cycle modeling approach and refer the reader to our technical report [5].

Implementation. In a nutshell, Active Data works as follows: data management systems expose their intrinsic data life cycle according to a well-specified formalism. We consider each creation, modification or deletion of a data item by the data management system as a progression of the data item through its life cycle. We call *transition* the move of a data item from one state to another state.

Informally, the programming model proposed by Active Data could be called a *transition-based* programing model. A programmer provides a routine or code that we call *transition handler* which is executed whenever a transition is triggered. The paradigm used by Active Data to propagate transitions is based on Publish/Subscribe [6]. Data management systems publish transitions to a centralized service called *Active Data Service*.

A. Example

To illustrate a data life cycle model, Figure 1 shows a part of a data life cycle for a file. In this part, the file is transferred from Globus online (a set of online services for supporting access to science data), to an iRODS (data management system [7]) system (this use case is detailed in section V). A data item starts its life in the Created place in Globus online, and then it is transferred to iRODS. The transfer may either be TRANSFER_SUCCEDED or TRANSFER_FAILED, and eventually the file may be DELETED from Globus online. The black boxes represent events that trigger changes to the data state. For instance, the transition t_1 corresponds to the action of transferring the data, t_2 for transfer failure, t_3, t_4, and t_5 represent the delete operation, and t_{10} is the composition transition which allows to link file from Globus online to iRods .

For each of these events (t_1 to t_9) the programmer may implement a handler, and register the handler to the event ID number. The handler is a Java object that implements the TransitionHandler interface. The handler() method is invoked by the system when the transition is triggered. In addition to the event information, the handler receives information about the data (status, life cycle, file path …etc), the name of the transition that was triggered (as one handler may serve multiple events), and a flag indicating whether the transition was triggered on the same node, to enable optimizations to increase access locality.

This relatively simple example (we implement and demonstrate this example in section V.A) highlights the following properties of our approach: first, the proposed modeling approach is intuitive, making it easy to reason about the data life cycle, verify and debug it, second, it is simple to handle failures as a state transition in the model (e.g. TRANSFER_FAILED state in Figure 1), third, it is easy to coordinate among collaborating sites through defining which events should trigger event handlers in the other system (this example uses two systems: Globus online and iRODS).

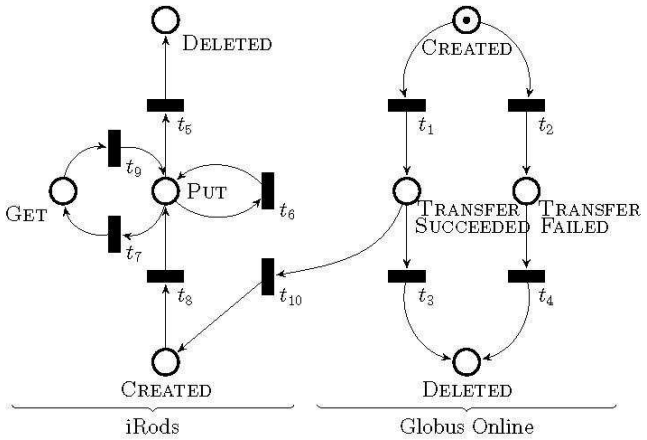

Figure 1. Data life cycle model for data acquisition, transfer and processing involving two systems: Globus online and iRODS.

V. EXPERIENCE WITH AN EARLY PROTOTYPE

We have implemented an Active Data DLM prototype. The system prototype is composed of two parts: *(i)* the execution runtime system, which manages data life cycles, publishes transitions, triggers handler code executions, and guarantees that the execution is correct with respect to the model, and *(ii)* the programming interface (API), which allows data management system to publish transitions and programmers to develop applications by developing and registering their transition handlers.

The paradigm used by Active Data to propagate transitions is based on Publish/Subscribe [6]. Thus, nodes publish transitions to a centralized service, and pull events from the same service. To ensure that life cycles remain in a coherent state, the service stores all the life cycles, which are updated when transitions are notified. For correctness handlers are executed serially, in the order in which the transitions were published. This means the service maintains a total order on transitions. Moreover, handlers are executed in a blocking fashion. To avoid blocking the execution of all the subsequent handlers, handlers are expected to return shortly. Any long running task should be performed in a separate thread.

Active Data provides the programmer with two kinds of transition subscriptions to register their handler code: *(i)* subscribe to a specific transition for any data items, or *(ii)* subscribe to a specific data item and be notified for any life cycle transitions. To avoid getting overloaded by transition notification, the service provides a method to

return a copy of a life cycle. This way, nodes can occasionally get the state of a life cycle without the need to get all the transitions.

One of the most interesting features of the model implementation is that the runtime can enforce the model by dynamically checking transitions when they are published. Exceptions are thrown whenever a node violates a the model, for example: a node tries to *(i)* publish a transition that is not in the model; *(ii)* publish a transition that is not enabled, and *(iii)* publish a transition for a life cycle that has terminated.

The system prototype supports a wide range of mechanisms to collect data related events: for instance, it can leverage the notification API used by collaborating systems, if any, Linux inotify for file system monitoring, or even receive notification emails from the collaborating systems.

To demonstrate the active data programing model, we build four systems with different data life cycle properties: *(i)* a caching system for Amazon S3 service, *(ii)* a complex data life cycle for distributed coordination and throttling in sensor networks, *(iii)* an incremental MapReduce framework, and *(iv)* a data provenance solution for data life cycles spanning multiple systems. These use cases demonstrate the Active Data framework ability to simplify the development of complex data life cycles, support for incremental computations, across systems coordination, and overall system efficiency. Due to the space limitations, next, we present the last use case; we refer the reader to our technical report [5] for the other use cases, and for the synthetic benchmark evaluation.

A. Example Senario: Across Systems Data Provenance

Data provenance constitutes the complete history of data life cycle derivations and operations, and is essential to preserve the quality of each scientific data asset over time. We demonstrate our solution by reconstructing provenance history from heterogeneous systems. To our knowledge, no framework allows to construct provenance when multiple loosely coupled data management systems and infrastructures cooperate.

We consider a scenario (the scenario's Petri network is presented in Figure 1) where file transfers are launched from a remote Globus Online endpoint to a local temporary storage every few seconds. In this scenario Active Data receives events regarding data evolving in two independent systems: Globus Online [2], a service for handling file transfers, and iRODS [7] a system for storing the data and providing a metadata catalog.

In our scenario, for any data file in iRODS, we want to record file transfer provenance: the transfer endpoints, start and completion times, and transfer failures, if any.

Active Data is the glue that enables both Globus Online and iRODS to see the part of data life cycles that is outside their scope. We use iRODS's user-defined metadata to record provenance information along with data files.

When a Globus Online file transfer starts, a transfer task is created and the returned Task Id becomes the token identifier.

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. The reception of a ``success'' email notification causes transition t_9 to be triggered, and a handler to store the file in iRODS. iRODS returns a DATA_ID that is added to the token; it now contains both identifiers.

A second transition handler is attached to iRODS's creation transition: it is executed when any iRODS data is created. This handler requests the life cycle from the Active Data Service to see if it contains a Globus Online identifier. In such case, it queries Globus Online to get file transfer information.

To demonstrate our solution, file transfers are launched from a remote Globus Online endpoint to a local temporary storage every few seconds. We observe that when a transfer ends, the transfer metadata appears immediately in the iRODS data catalog with the correct Globus Online Task Id and meta-data information (endpoint, completion date, request time). The following listing shows the metadata set for one of these iRODS files after the transfer is done.

```
$ 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 -123139404 f2e
----
attribute: GO_SOURCE
value: go#ep1 /~/ test
----
attribute: GO_DESTINATION
value: asimonet#fraise /~/ out_test_4628
```

Active Data ability to provide a unified view of data-sets over heterogeneous systems significantly simplifies the challenge of global provenance reconstruction.

VI. Discussion

A. Advantages

The Active Data approach brings the following advantages compared to the current ad-hoc task-centric model.

- *Programmability*: Active Data exposes the life cycle in a simple and graphical way, which makes it easy for the programmer to identify the events that trigger Active Data operations and program those operations.
- *Verification*: Modeling the data life cycle enables verifying its correctness and completeness of the model. For instance, several tools (e.g. CPNTools [4]) have been developed to automate the verification of Petri Net properties such as, deadlock free networks, boundedness or liveness, which can be extended to support data life cycle verification.

- *Ease of debugging*: Due to its event based structure, Active Data is easy to debug as the model clearly separates each stages from the interaction among stages, making it easy to identify bugs and fix them.
- *Ease of fault tolerance modeling and implementation.* Extending an Active Data specification to include for fault tolerance is straightforward as faults and repair operations can be modeled in the data life cycle (by identifying the faults as events that trigger repair operations).
- *Scalability*: The event oriented view of Active Data enables scaling, as it enables distributed implementations, with no central component responsible for the data operations.
- *Easy of scheduling and optimization*: Scheduling is implicit in Active Data: the site/node that holds the data executes the triggered operation, this scheduling policy increases access locality.
- *Ability to decouple management and operations*. Active Data clients are extremely light and can be deployed on low profile devices, while the heavier services, which implement life-cycle management, can be deployed on a scalable and reliable cloud infrastructure.
- *Fine grain interaction with data-life cycle*. In contrast with task-centric programming, where each action typically takes place at the start or at the end of the task using pre- or post-processing actions, Active Data allows to interact at every incremental step during the processing or during special events. For instance, an Active Data -enabled file transfer service would make it possible to trigger actions during a file transfer as well as when the transfer stalls.

B. Limitations

- *Complexity*: In general, without proper tooling, it is harder to reason about complex interactions in event driven designs. Complexity can be handled by building modeling and verification tools that can model, verify, and help in implementing and debugging complex data life cycles.
- *Lack of standard*: in an ideal world, many systems would provide an Active Data interface. However, this is not the case yet, and it depends on the admins at each site to provide an interface to the data management system. The process can be made simpler by following a clear methodology to instrument the legacy code or to gather the information that would allow reconstructing the data life-cycle. To mitigate this hurdle, Active Data support a wide range of mechanisms to collect data related events (section V)

VII. Related Work

To the best of our knowledge there is no system that manages data life cycle for scientific applications. However, the following efforts are relevant to this paper.

Active Message. Active Data borrows from Active Message; the idea of executing user-provided code when a message is received [8]. Following the same direction, Active Disks [9] allows execution of application code directly at storage devices to offload the CPU and increase access locality.

Programing Models. A number of programing models have been proposed for large scale data processing, few gained wide adoption. MapReduce [10] to model the application into two main operations: map and reduce, Dryad [11] for dataflow parallel computing, Allpairs [12] to perform massive pair-wise comparisons in large data sets, Swift [13] to script and automate the manipulation of large parallel scientific dataflow, or as evolution of the paradigm, e.g., PigLatin [14] to provide high level query interface on top of MapReduce, Twister [15], a framework for iterative MapReduce computations. Few systems support incremental computation, including Percolator [16], Nephele [17], MapReduce-Online [18] and Chimera [19]. While these efforts provide a programing model for large scale data analysis, they target data analysis in one system, not managing the data life cycle involving multiple systems.

Storage Systems. Few storage systems enable file level optimizations: e.g. optimizing the file placement, replication level or caching to better serve the file access pattern. BitDew [20] allows the user to specify data behavior such as fault-tolerance, replication, file transfer protocol or affinity placement. MosaStore [21] is a file system optimized for the main collective file pattern operations (gather/scatter, reduce, broadcast) that can be found in workflows. Active Data can be integrated with the BitDew or MosaStore to enable per file storage system optimizations.

VIII. Summary and Future Work

This work proposes the *data life cycle* as a new paradigm to model the data needs of existing data-centric science. We believe scientific applications require a fundamentally novel paradigm for data life cycle management which has two key characteristics: is *data-centric* (as opposed to task-centric) and *event-driven* (as opposed to task-completion triggered). We present the Active Data programing model and framework that implements this proposed paradigm.

Future work will focus on several aspects. The model can be extended on the following directions: advanced representation of computations that would investigate consumption and production of data items; representation of collections of data items that would allow collective operations on data sets. Concerning the implementation of Active Data, we plan to investigate rollback mechanisms for fault-tolerant execution of applications and evaluate distributed implementations of the publish/subscribe substrate. Finally, several application prototypes are being developed using Active Data: a distributed and cooperative content delivery network to distribute virtual appliance images embedding large HEP applications to Internet Desktop Grid resources [22] and a distributed network of checkpoint image server featuring server selection using network distance [23].

IX. REFERENCES

[1] Y. Wang, F. D. Carlo, D. C. Mancini, I. McNulty, et al., *A High-Throughput X-ray Microtomography System at the Advanced Photon Source*. Review of Scientific Instruments, 2001. **72**(4): p. 2062-2068.

[2] *Globus Online*. 2013 [cited; https://www.globusonline.org/.

[3] T. Murata, *Petri nets: Properties, analysis and applications.* Proceedings of the IEEE, 1989.

[4] *Colored Petri nets tools (CPN Tools)*. 2012 [cited; http://cpntools.org/.

[5] A. Simonet, G. Fedak, and M. Ripeanu, *Active Data: A Programming Model for Managing Big Data Life Cycle*. INRIA Technical Report N° RR-8062, 2012.

[6] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, *The many faces of publish/subscribe.* ACM Computing Surveys, 2003. **35**: p. 114-131.

[7] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, et al., *iRODS Primer: integrated Rule-Oriented Data System*. Synthesis Lectures on Information Concepts, Retrieval, and Services. 2010 Morgan and Claypool Publishers.

[8] T. v. Eicken, D. Culler, S. Goldstein, and K. Schauser, *Active Messages: A Mechanism for Integrated Communication and Computation*, in *Proceedings of the 19th International Symposium on Computer Architecture*. 1992. p. 256--266.

[9] A. Acharya, M. Uysal, and J. Saltz. *Active disks: programming model, algorithms and evaluation*. in *International conference on Architectural support for programming languages and operating systems (ASPLOS)*. 1998. New York, NY, USA.

[10] J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2004.

[11] M. Isard, M. Budiu, Y. Yu, A. Birrell, et al., *Dryad: Distributed Data-parallel Programs from Sequential Building Blocks.* SIGOPS European Conferenceon Computer Systems (EuroSys), 2007.

[12] J. Bulosan, D. Thain, and P. J. Flynn. *All-pairs: An abstraction for data-intensive cloud computing*. in *International Symposium on Parallel and Distributed Processing*. 2008.

[13] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, et al., *Swift: A language for distributed parallel scripting.* Parallel Computing, 2011.

[14] J. Dean and S. Ghemawatta. *Pig latin: a not-so-foreign language for data processing*. in *SIGMOD international conference on Management of data*. 2008.

[15] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, et al. *Twister: a runtime for iterative mapreduce*. in *International Symposium on High Performance Distributed Computing (HPDC)*. 2010.

[16] D. Peng and F. Dabek. *Large-scale incremental processing using distributed transactions and notifications*. in *USENIX conference on Operating systems design and implementation (OSDI)*. 2010.

[17] O. Kao, B. Lohrmann, and D. Warneke. *Massively-parallel stream processing under QoS constraints with nephele*. in *Symposium on High PErformance Parallel and Distributed Computing (HPDC)*. 2012.

[18] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, et al. (2010) *Mapreduce online*. USENIX conference on Networked systems design and implementation (NSDI),

[19] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. *Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation*. in *14th Conference on Scientific and Statistical Database Management*. 2002.

[20] G. Fedak, H. He, and F. Cappello. *BitDew: a programmable environment for large-scale data management and distribution*. in *International Conference on High Performance Networking and Computing (Supercomputing)*. 2008.

[21] S. Al-Kiswany, A. Gharaibeh, and M. Ripeanu. *The Case for Versatile Storage System*. in *Workshop on Hot Topics in Storage and File Systems (HotStorage)*. 2009.

[22] H. He, G. Fedak, P. Kacsuk, Z. Farkas, et al. *Extending the EGEE Grid with XtremWeb-HEP Desktop Grids*. in *Workshop on Desktop Grids and Volunteer Computing Systems*. 2010.

[23] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. *Topologically-Aware Overlay Construction and Server Selection*. in *INFOCOM'02*. 2002. New York.