D^3-MapReduce: Towards MapReduce for Distributed and Dynamic Data Sets

Haiwu He
CSNET/CNIC
Chinese Academy of Sciences, China
haiwuhe@cstnet.cn

Anthony Simonet, Julio Anjos
José-Francisco Saray, Gilles Fedak
INRIA/Univ. Lyon, France
{first.last}@inria.fr

Bing Tang
Hunan University of Science and Technology
Xiangtan, China
btang@hnust.edu.cn

Lu Lu, Xuanhua Shi, Hai Jin
Huazhong University of Science and Technology
Wuhan, China
{llu,xshi}@hust.edu.cn

Mircea Moca, Gheorghe Cosmin Silaghi
Babeș Bolyai University
Cluj-Napoca, România
{first.last}@econ.ubbcluj.ro

Asma Ben Cheikh
Heithem Abbes
University of Tunis, Tunisia
{first.last}@latice.rnu.tn

***Abstract*—Since its introduction in 2004 by Google, MapReduce has become the programming model of choice for processing large data sets. Although MapReduce was originally developed for use by web enterprises in large data-centers, this technique has gained a lot of attention from the scientific community for its applicability in large parallel data analysis (including geographic, high energy physics, genomics, etc.).**

So far MapReduce has been mostly designed for batch processing of bulk data. The ambition of D^3-MapReduce is to extend the MapReduce programming model and propose efficient implementation of this model to: i) cope with distributed data sets, i.e. that span over multiple distributed infrastructures or stored on network of loosely connected devices; ii) cope with dynamic data sets, i.e. which dynamically change over time or can be either incomplete or partially available. In this paper, we draw the path towards this ambitious goal. Our approach leverages *Data Life Cycle* as a key concept to provide MapReduce for distributed and dynamic data sets on heterogeneous and distributed infrastructures. We first report on our attempts at implementing the MapReduce programming model for Hybrid Distributed Computing Infrastructures (Hybrid DCIs). We present the architecture of the prototype based on BitDew, a middleware for large scale data management, and Active Data, a programming model for data life cycle management. Second, we outline the challenges in term of methodology and present our approaches based on simulation and emulation on the Grid'5000 experimental testbed. We conduct performance evaluations and compare our prototype with Hadoop, the industry reference MapReduce implementation. We present our work in progress on dynamic data sets that has lead us to implement an incremental MapReduce framework. Finally, we discuss our achievements and outline the challenges that remain to be addressed before obtaining a complete D^3-MapReduce environment.

Keywords*—*Data management, MapReduce, Hybrid Computing Infrastructure, Incremental Processing.

I. Introduction

Increasingly the next scientific discoveries and the next industrial innovative breakthroughs will depend on the capacity to extract knowledge and sense from gigantic amount of information. Examples vary from processing data provided by scientific instruments such as the CERNs Large Hadron Collider (LHC); collecting data from large-scale sensor networks such as the OOI ocean underwater observatory; grabbing, indexing and nearly instantaneously mining and searching the Web; building and traversing the billion-edges social network graphs; anticipating market and customer trends through multiple channels of information. Collecting information from various sources, recognizing patterns and distilling insights constitute what is called the Big Data challenge.

Since its introduction in 2004 by Google [1], MapReduce has become the programming model of choice for processing large data sets. Two factors can explain the success of MapReduce. First, expression of the parallelism is very simple, almost hidden to the programmer. MapReduce borrows from functional programming, where a programmer can define both a Map task that maps a data set into another data set, and a Reduce task that combines intermediate outputs into a final result. The second factor of success is the wide availability of solid open-source solutions, as well as a mature technological ecosystem, which has been built around this paradigm: high level query languages [2], iterative computing [3], NoSQL databases, machine learning environments and many more. The consequence is that MapReduce is becoming a kind of universal substrate on top of which algorithms and programming environments are designed and developed.

However MapReduce has its own limitations. The model has been originally designed for "static" data sets, i.e. built on the principle that all the data items are available at the same time in a single infrastructure and do not change during the whole MapReduce execution. Nonetheless, there is a broad range of data sets for which these conditions are not verified, and where traditional MapReduce systems are ineffective. In this paper, we focus on data sets which are **dynamic**, i.e. that can grow or shrink in time, be partially updated, or produced as continuous stream, and **distributed**, i.e. data sets that are distributed over several heterogeneous systems, not always

Corresponding author: Haiwu He, Computer Network Information Center, Chinese Academy of Sciences, 4 Zhongguancun Nansijie, Haidian District, Beijing 100190, China

directly inter-connected. There exists now a large variety of Distributed Computing Infrastructures (DCIs) to execute large data-intensive applications, namely Grids, Clouds and Desktop Grids[4]. We believe that applications requiring an important volume of data input storage with frequent data reuse and limited volume of data output could take advantage not only of the vast processing power but also of the huge storage potential offered by Hybrid DCIs, i.e the assemblage of Clouds, Grids and Desktop Grids systems.

However, enabling MapReduce on Hybrid DCIs raises many research issues with respect to the state of the art in MapReduce runtime execution environment. Our objective is to propose an implementation of the MapReduce programming model that:

- allows to execute MapReduce computations on loosely connected computing infrastructures, including network of mobile devices.
- allows to process efficiently data sets which are partially available, either because of infrastructure failures, which dynamically grow or shrink, or which are incrementally updated.

In this paper we present the work being conducted to provide a complete runtime environment to execute MapReduce applications on Desktop Grids as well as Hybrid DCIs. Our prototype is based on the BitDew [5] middleware, developed by Inria, which is a programmable environment for automatic and transparent data management on computational Desktop Grids. BitDew relies on a specific set of *data attributes* to drive key data management operations, namely life cycle, distribution, placement, replication and fault-tolerance with a high level of abstraction. The BitDew runtime environment is a flexible distributed service architecture that integrates modular P2P components such as DHTs for a distributed data catalog, and collaborative transport protocols for data distribution, asynchronous and reliable multi-protocols transfers.

However, one of the biggest challenges is to enable infrastructure operability, i.e. being able to have several infrastructures and systems with different usage paradigms that collaborate to the processing and the management of the dataset. Active Data [6] is a programming model for Data Life Cycle Management (DLCM). Active Data allows to have a unified view of the data-sets when handled by heterogeneous software and infrastructures. We present our implementation of MapReduce with Active Data which also allows to process dynamic data-sets.

The rest of this paper is organized as follows; Section II reviews our background work on data life cycle management and MapReduce. We present the design choices, the requirements and the system architecture of our new prototype in section III. We evaluate our prototype on the Grid'5000 experimental platform in Section IV, and we also give preliminary results of our incremental MapReduce implementation. We conclude in section VI.

II. Background

In this section we review our background work in the field of MapReduce computing and data life cycle management.

A. Architecture of MapReduce over BitDew

Our MapReduce implementation [7] relies on BitDew to cope with the difficulty of using Desktop Grids. Using the data attributes, we set dependancies between map inputs, the mappers, intermediate results, the reducer and the final result. The BitDew runtime uses the dependancies to automatically create a data flow between the nodes. It places data on volunteer computers and deals with faults as they occur, replicating data items on different nodes when necessary.

In the remaining of this section we present important features and optimizations that have been designed specifically to address Internet Desktop Grid platforms.

Latency Hiding and Collective File Operation To hide the high latency caused by Internet network conditions, NATs, and firewalls, we implemented a multi-threaded worker that i) overlaps communications with computations, and ii) process concurrently multiple tasks and communications. MapReduce over BitDew implements efficiently a set of collective communications: 1) *Distribution*, used to distribute the initial file chunks, 2) *Shuffle*, used to distribute intermediate results between Mappers and Reducers and 3) *Combine*, used by the master node to collect the final result.

Fault Tolerance. To deal with nodes joining and leaving the computation during a Map task, we simply re-place the task data to another node. When a Reduce task fails, we distribute all the intermediate results to a different Reduce worker. File transfer failures tolerance is done internally by the BitDew runtime.

Scheduling. We follow the usual MapReduce paradigm of moving the computation where the data resides. MapReduce on BitDew uses a two-level scheduler. The first level is the BitDew scheduler that places data on nodes according to attributes. The second level is the master node; it detects *laggers* (nodes that slow down the whole computation taking too much time to perform a single task) and can in such case increase the data replication so an other node can take over the same task.

Distributed Result Checking. As Desktop Grids can always contain malicious volunteers, result certification is necessary. In this context, intermediate results might be too large to be sent back to the server to perform this certification. We presented in [8] a fully decentralized algorithm for result certification that relies on duplication of the Mapers and Reducers and majority voting to select the correct results.

B. Active Data

Active Data is a system that aims to make distributed data management more efficient. It offers a formal and graphical model to represent the complete life cycle of distributed data, from creation to deletion in terms of *data state* and *state transitions*. This model is able to represent how individual systems process data. Active Data allows to compose various data life cycle models that represent movement from one system to another, offering a high-level and flat view of large applications. This is particularly important as MapReduce applications are often composed of several middleware that collaborate together to store, process and analyze the data (i.e. PIG, HDFS, Cassandra, etc.). Based on this application model,

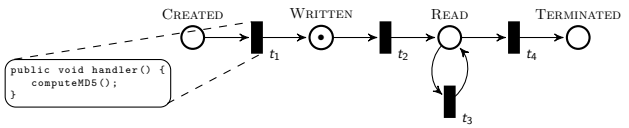


Fig. 1: Example of Active Data Life cycle. The life cycle presented is Write Once, Read Many. The handler code creates a signature of the file each time it is created.

Active Data offers an API and a transition-based programming model. Conceptually, data management systems report on their internal activity, and users specify actions to be conducted when specific data state transitions are reported, making it easy to implement decisions and optimizations.

Active Data provides a non-invasive method of improving coordination and obtaining feedback from loosely coupled systems. Specifically, we can use the Active Datas life cycle model to individually represent and expose the internal life cycle of the systems across the various infrastructures. Thus, this approach can provide interoperability both between the systems and the infrastructures.

As shown in Figure 1, the model of a system's life cycle is composed of places, represented by circles and transitions, represented by rectangles. Places represent all the possible states of data, and transitions represent all the legal changes of states. A set of directed arcs connect places to transitions, and transitions to places. Places may contain one or several tokens; each token represents a data replica, and the current place represents the current state of the replica. Several tokens on different places represent several replicas in different states at the same time. Each token holds an identifier that links the token to the actual piece of data (a URL or a path, for example). In addition, Active Data offers the ability to automatically run code to react to transitions being triggered. This code can be run anywhere (not necessarily where the event occurred) and can access the complete state of the life cycle.

III. MapReduce for Hybrid DCIs and Dynamic Datasets

In this section, we report on preliminary works to achieve MapReduce on hybrid DCI and MapReduce for dynamic datasets thanks to our Data Life Cycle Management approach.

A. Hybrid MapReduce

The implementation of MapReduce on top of BitDew is very flexible and allows the deployment of the processing and storage agents on many kinds of infrastructures:

- On Desktop PCs and mobile devices, the MapReduce/BitDew agents can directly fetch the data from Cloud storage servers, perform the Map and Reduce computation and upload the results to the stable storage.

On Cloud and Grid nodes, two approaches are feasible:

- The first one consists in deploying MapReduce/BitDew agents on the Grids and Clouds nodes following a Pilot Job approach. In this case, we can provision Cloud virtual instances that embed the MapReduce/BitDew agent, or we can schedule the MapReduce/BitDew using the Grid Resource Management software.
- The second approach consists in relying on Hadoop already installed on the Cloud or the Grid. In this case, we consider the usage of two different MapReduce middleware at the same time, MapReduce/BitDew and Hadoop. Thus, a first stage consists in splitting the dataset in two parts and transferring the data to the two middleware. Then both middleware process their respective data sets independently, which are then aggregated at the end of the computation.

Thanks to these advances, we can now execute MapReduce applications, which use any combination of Grids, Desktop Grids, and Clouds, even using legacy Hadoop installations. However, when considering a Hybrid DCI, it is necessary that a data scheduler splits the data-set and distribute the data according to the processing capabilities of each infrastructure. We are working now on defining the best strategies to distribute the data. The hybrid infrastructure enables the use of highly heterogeneous machines, with stable and volatile storage to avoid data loss. The extent to which a set of data- distribution strategies is applicable to a given scenario depends on how much bandwidth is available. Two independent DFS implementations are required to handle data distribution in two scenarios, namely low-bandwidth and high-bandwidth. The application profile is optimized for all file sizes in hybrid infrastructures, as the systems are independent and thus the different data size can be handled at the same time. The bandwidth and computational capacity of machines influence the initial assumptions for defining a straggler machine and because of this, each system must be treated in a different way.

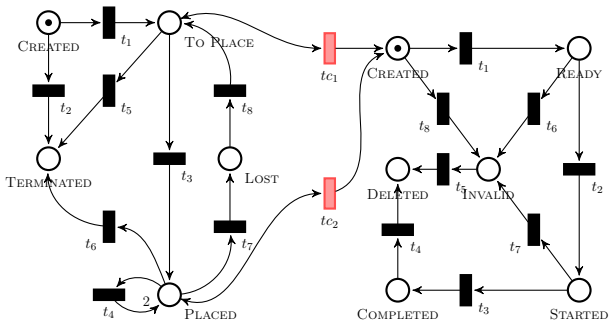
B. Global MapReduce Dataflow Surveillance

However to implement efficient Hybrid MapReduce, it is necessary that the infrastructures and the systems report the progress of the MapReduce computation, for instance to enable redistribution of the data if some bottleneck appears.

To observe the life cycle of the data involved in a MapReduce/BitDew and Hadoop computation, we use Active Data.

Programmers supply code to a special service in BitDew called "Active Data", specifying at what step of the data life cycle the code must be executed. At the same time, the BitDew and Hadoop runtime must inform the Active Data service when it uses or alters data. Any data item has a corresponding life cycle model that documents everything that can be done with it.

Figure 2 presents the BitDew life cycle. The paradigm used by Active Data to propagate transitions is based on Publish/Subscribe. In our implementation, every node in BitDew can be publisher and subscriber at the same time. BitDew workers as well as other BitDew services publish transitions to the centralized service. Workers and other BitDew services pull transition information from the Active Data Service as well. When a client of Active Data (whether a BitDew worker



(a) Bitdew Scheduler (b) Bitdew File Transfer

Fig. 2: BitDew Data Life Cycle

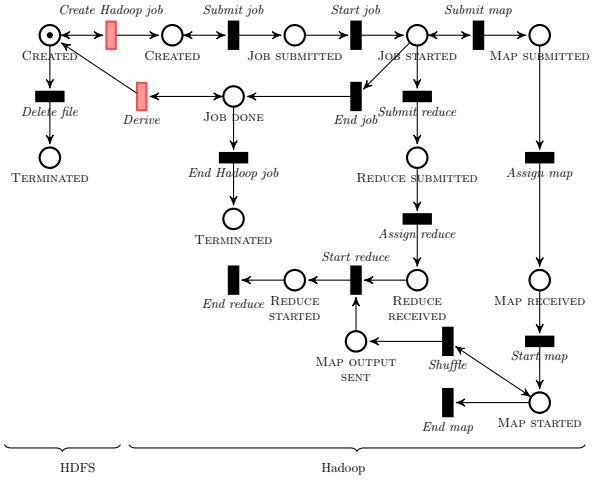


Fig. 3: Hadoop and HDFS Data Life Cycle

or a BitDew service) wants some code to be run when some data transitions happen, it provides an object called *Transition Handler* that implements a special interface.

The life cycle model Hadoop represents the life cycle of a file used as input for a Hadoop MapReduce job. Because the input and output files of a Hadoop job are stored in HDFS, the life cycle model of Hadoop is connected to the life cycle model of HDFS. The life cycle model for Hadoop and HDFS presented on Figure 3 is composed of two separate models; the one on the left represents the life cycle of a file stored in HDFS with a coarse granularity (only one transition is included) because not enough meaningful information can be extracted from the logs yet; the one on the right represents the life cycle of a file during a Hadoop job. The Hadoop model features transitions for job submission and termination, and map and reduce tasks submission, distribution, launch and termination (Submit map, Assign map, Start map, End map etc.). It also represents data transfers during the shuffle phase (transition Shuffle). A second composition transition called Derive represents the production of one or several output files in HDFS.

After making Hadoop, HDFS and MapReduce/BitDew "Active Data compliant", we are now able to have several MapReduce applications that share a unified data life cycle and that are able to cooperate on a unique MapReduce application execution. As an example of optimization, which is now possible, we now consider the issues of incremental MapReduce.

C. Incremental MapReduce with Active Data

One of the strongest limitations of MapReduce is its inefficiency to handle mutating data; when a MapReduce job is run several times and only a subset of its input data set has changed between two job executions, all map and reduce tasks must be run again. Making MapReduce incremental i.e. re-run map and reduce tasks only for the data input chunks that have changed, necessitates to modify the complex data flow of MapReduce. However, if the MapReduce framework becomes *aware* of the life cycle of the data involved, it can dynamically adapt the computation to data modification.

To make the MapReduce implementation incremental, we add a "dirty" flag to the input data chunks. When a chunk is flagged as dirty, the mapper that previously mapped the chunk executes again the map task on the new chunk content and sends the updated intermediate results to the reducers. Otherwise, the mapper returns the intermediate data previously memoized. Reducers proceed as usual to compute again the final result. To update the chunk's dirty flag, we need the master and the mappers to react to transitions in the life cycle of the chunks. More precisely, nodes listen to two transitions triggered by BitDew, thanks to Active Data:

- When a transfer is completed, the master node checks whether it is local and whether it modifies an input chunk. Such case happens when the master puts all the data chunks in the storage system before launching the job. If both conditions are true, the transition handler flags the chunk as dirty.
- When a transfer starts, mappers check whether it is distant and if one of their input chunks is modified. In this case, the transition handler on the mapper flags the chunk as dirty.

IV. Experimental Evaluation

In this Section, we report on preliminary results. Because the evaluation involves several DCIs and complex middleware stack, it poses a considerable challenge in term of methodology. We developed a dedicated simulator for hybrid MapReduce environment based on the SimGrid framework. BigHybrid [9] simulates two MapReduce middleware: Hadoop over BlobSeer and MapReduce/BitDew on two different computing environments Clouds and Desktop Grids. Furthermore, this simulator, which is validated on Grid5000 is used to develop and prototype our ideas for data placement strategies.

A. Evaluation Against an Emulated Desktop Grid Environment

We have performed all of our experiments in the GdX and NetGdX clusters which are part of the Grid'5000 infrastructure. The two clusters are composed of 356 IBM eServer nodes featuring one 2-core 2.0GHz AMD Opteron CPU and 2GB RAM, running Debian with kernel 2.6.18, and interconnected by Gigabit Ethernet network. All results described in this paper were obtained using Hadoop version 0.21.0, while the

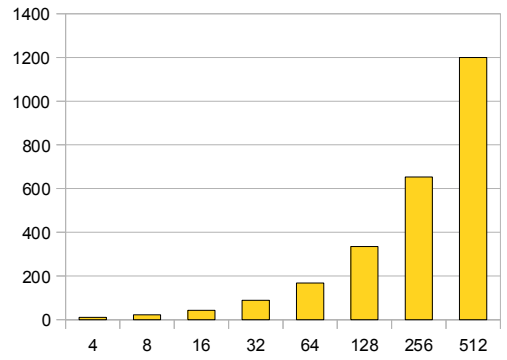


Fig. 4: Scalability evaluation on the WordCount application: the y axis presents the throughput in MB/s and the x axis the number of nodes varying from 1 to 512.

data was stored with 2 replicas per each block in Hadoop Distributed File System. We perform our experiments by repeatedly executing the wordcount benchmark, with 50GB dataset generated by Hadoop RandomTextWriter application, and set chunk size to 64MB, map slot number to 2, reduce slot number to 1 and total reduce number to 10.

We emulate a desktop grid environment using several artifacts to modify the machine and network characteristics of the platform. The goal of these modifications is to study several aspects such as scalability, fault tolerance, host churn, network connectivity, error computing and sabotage, CPU heterogeneity, stragglers and network failure tolerance.

The rest of this section gives an overview of the how we emulated each aspect of a Desktop Grid platform and some key results we obtained.

Scalability The first experiment evaluates the scalability of our implementation when the number of nodes increases. Each node has a different 5GB file to process, splitted into 50 local chunks. Thus, when the number of nodes doubles, the size of the whole document counted doubles too. For 512 nodes[1], the benchmark processes 2.5TB of data and executes 50,000 Map and Reduce tasks. Figure 4 presents the throughput of the WordCount benchmark in MB/s versus the number of worker nodes. This result shows the scalability of our approach and illustrates the potential of using Desktop Grid resources to process a vast amount of data.

Fault Tolerance and Host Churn When considering a large distributed system consisting of volatile nodes, the independent arrival and departure of thousands or even millions of peer machines leads to host churn and massive faults. We periodically kill the MapReduce worker process on one node and launch it on a new node to emulate the host churn effect. To increase the survival probability of Hadoop job completion, we increase the HDFS chunk replica factor to 3, and set the DataNode heartbeat timeout value to 20 seconds. Because the BitDew MapReduce runtime does not waste the work completed by failing workers, host churn causes very small effects on the job completion time. On the other hand, as shown in table I, for host churn intervals of 5, 10 and 25 seconds, Hadoop jobs could only progress up to 80% of the map phase before failing.

[1]GdX has 356 double core nodes, so to measure the performance on 521 nodes we run two workers per node on 256 nodes.

Churn Interval (sec.)	5	10	25	30	50
Hadoop job makespan (sec.)	failed	failed	failed	2357	1752
BitDew-MR joa makespan (sec.)	457	398	366	361	357

TABLE I: Performance evaluation of host churn scenario

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

TABLE III: Incremental MapReduce: time to update the result compared with the fraction of the dataset modified.

Network Connectivity We set custom firewall and NAT rules on all the worker nodes to turn down some network links and observe how MapReduce jobs perform. In this test, Hadoop cannot even launch a job, because it needs inter-communication between worker nodes, and BitDew mapreduce is almost independent of such network conditions.

To work in a loosely connected environment, the runtime system must be resilient to temporary network failures. We set the worker heartbeat timeout value to 20 seconds for both Hadoop and MapReduce/BitDew, and inject a 25-second offline window period to 25 worker nodes at different job progress points of the map phase. In this test, the Hadoop JobTracker simply marks all temporary disconnected nodes as "dead" when they are still running tasks. The tasks performed by these nodes are blindly removed from the successful task list and must be re-executed, significantly prolonging the job makespan, as Table II shows. Meanwhile, the MapReduce/BitDew clearly allows workers to go temporarily offline without any performance penalty.

B. Evaluation of the incremental MapReduce/BitDew

To evaluate the performance of incremental MapReduce, we compare the time to process the full data set compared with the time to update the result after modifying a part of the dataset. The experiment is configured as follows; the benchmark is the WordCount application running with 10 mappers and 5 reducers, the data set is 3.2 GB split in 200 chuncks. Table III presents the time to update the result with respect to the original computation time when a varying fraction of the dataset is modified. As expected, the less the dataset is modified, the less time it takes to update the result: it takes 27% of the original computation time to update the result when 20% of the data chunks are modified. However, there is an overhead due to the fact that the shuffle and the reduce phase are fully executed in our implementation. In addition, the modified chunks are not evenly distributed amongst the nodes, which provokes a load imbalance. Further optimizations would possibly decrease the overhead but would require significant modification of the MapReduce runtime. However, thanks to Active Data, we demonstrate that we can reach significant speedup with a patch that impacts less than 2% of the MapReduce runtime source code.

Job progress of the crash points		12.5%	25%	37.5%	50%	62.5%	75%	87.5%	100%
Hadoop	Re-executed map tasks	50	100	150	200	250	300	350	400
	Job makespan (sec.)	425	468	479	512	536	572	589	601
BitDew-MR	Re-executed map tasks	0	0	0	0	0	0	0	0
	Job makespan (sec.)	246	249	243	239	254	257	274	256

TABLE II: Performance evaluation of the network fault tolerance scenario

V. Related Works

Of course, many systems that allow massive processing of distributed and dynamic data sets have been designed. To cite a few works that give a taste of research results in this direction: incremental processing of data sets dynamically updated [10] [11], data streams parallel processing [12] [13], widely distributed data sets [14]. Our approach contrasts from the previous work in the sense that we aim at adapting the MapReduce programming model to these emerging constraints on data sets. This approach would allow people to continue to "think" MapReduce, while avoiding the constraint of processing static data sets.

There have been several attempts at providing MapReduce runtime execution environments for Desktop Grids [15] and the Internet [16], [7]. In previous works, we investigated MapReduce on Hybrid DCIs [17], [18], [9], combining Desktop Grid and Cloud infrastructures. In this work, we extend those results by considering dynamic datasets thanks to Active Data. In addition, Active Data can be used to provide additional important features, such as monitoring and optimizing the execution of MapReduce applications on Hybrid DCIs.

VI. Conclusion

Several DCIs offer vast amount of computing resources, which can be efficiently used for running Big Data applications. However, as data generated from scientific instruments are continuously increasing, many efforts on utilizing Hybrid DCIs for data-intensive applications are being pursued.

However, using Hybrid DCIs to perform MapReduce applications brings the challenge of interoperability between the infrastructures and the systems. We think that focusing on the data life cycle is a promising approach to address the issue of interoperability in Big Data applications. Accordingly, in this paper, we presented several preliminary results, which goes in the direction of providing interoperability with Active Data in the context of MapReduce Computing. We presented results obtained by simulation and using an experimental framework based on the Grid'5000 platform. We presented Active Data version of MapReduce/BitDew and Hadoop, and show that this novel programming model is able to make our implementation incremental. Overall, the DLCM approach allows to address more dynamic data-sets in more complex infrastructures composed of heterogeneous MapReduce systems.

Acknowledgements

This work is partially supported by International Science & Technology Cooperation Program of China under grant No. 2015DFE12860, and NSFC under grant No. 61370104, by the French National Research Agency (MapReduce ANR-10-SEGI-001) and by the Chinese Academy of Sciences President's International Fellowship Initiative (PIFI) 2015 Grant No. 2015VTB064.

References

[1] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI'04: Proceedings of the 6th Symposium on Operating Systems Design and Implementation, USA, USENIX Association (2004) 137–149

[2] Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, ACM (2008) 1099–1110

[3] Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.H., Qiu, J., Fox, G.: Twister: a runtime for iterative mapreduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. HPDC '10, New York, NY, USA, ACM (2010)

[4] Crin, C., Fedak, G., eds.: Desktop Grid Computing. Chapman & All/CRC Press (May 2012)

[5] Fedak, G., He, H., Cappello, F.: BitDew: A Programmable Environment for Large-Scale Data Management and Distribution. In: Proceedings of the ACM/IEEE SuperComputing Conference (SC'08), Austin, USA (November 2008) 1–12

[6] Simonet, A., Fedak, G., Ripeanu, M.: Active Data: A Programming Model to Manage Data Life Cycle Across Heterogeneous Systems and Infrastructures. Future Generation in Computer Systems (2015)

[7] Tang, B., Moca, M., Chevalier, S., He, H., Fedak, G.: Towards mapreduce for desktop grid computing. In: P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on, IEEE (2010) 193–200

[8] Moca, M., Silaghi, G., Fedak, G.: Distributed results checking for mapreduce in volunteer computing. In: Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, IEEE (2011) 1847–1854

[9] Anjos, J.C.S., Fedak, G., Geyer, C.F.R.: BIGhybrid: A Simulator for MapReduce Applications in Hybrid Distributed Infrastructures Validated with the Grid5000 Experimental Platform. Concurrency and Computation: Practice and Experience (2015)

[10] Bhatotia, P., Wieder, A., Rodrigues, R., Acar, U.A., Pasquin, R.: Incoop: Mapreduce for incremental computations. In: Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM (2011) 7

[11] Peng, D., Dabek, F.: Large-scale incremental processing using distributed transactions and notifications. In: Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation. (2010)

[12] Lohrmann, B., Warneke, D., Kao, O.: Massively-parallel stream processing under qos constraints with nephele. In: Proceedings of High-Performance Parallel and Distributed Computing, ACM (2012) 271–282

[13] Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Elmeleegy, K., Sears, R.: Mapreduce online. In: Proceedings of the 7th USENIX conference on Networked systems design and implementation. (2010) 21–21

[14] Corbett, J.C., et al.: Spanner: Googles globally-distributed database. in Proceedings of OSDI (2012) 1

[15] Lin, H., Ma, X., Archuleta, J., Feng, W.c., Gardner, M., Zhang, Z.: Moon: Mapreduce on opportunistic environments. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ACM (2010) 95–106

[16] Marozzo, F., Talia, D., Trunfio, P.: P2p-mapreduce: Parallel data processing in dynamic cloud environments. Journal of Computer and System Sciences **78**(5) (2012) 1382–1402

[17] Tang, B., He, H., Fedak, G.: HybridMR: A New Approach for Hybrid MapReduce Combining Desktop Grid and Cloud Infrastructures. Concurrency Practice and Experience (2015)

[18] Antoniu, G., all: Scalable Data Management for MapReduce-Based Data-Intensive Applications: a View for Cloud and Hybrid Infrastructures. International Journal on Cloud Computing **2**(2-3) (January 2013)