Towards Scalable Data Management for Map-Reduce-based Data-Intensive Applications on Cloud and Hybrid Infrastructures

Gabriel Antoniu, Julien Bigot, Christophe Blanchet, Luc Bougé, François Briant, Franck Cappello, Alexandru Costan, Frédéric Desprez, Gilles Fedak, Sylvain Gault, et al.

Towards Scalable Data Management for Map-Reduce-based Data-Intensive Applications on Cloud and Hybrid Infrastructures

Gabriel Antoniu[a,b] gabriel.antoniu@inria.fr
Julien Bigot[c] julien.bigot@inria.fr
Christophe Blanchet[d] christophe.blanchet@ibcp.fr
Luc Bougé[e] luc.bouge@bretagne.ens-cachan.fr
François Briant[f] fbriant@fr.ibm.com
Franck Cappello[b] fci@lri.fr
Alexandru Costan[g] alexandru.costan@inria.fr
Frédéric Desprez[c] frederic.desprez@inria.fr
Gilles Fedak[c] gilles.fedak@inria.fr
Sylvain Gault[c] sylvain.gault@inria.fr
Kate Keahey[h] keahey@mcs.anl.gov
Bogdan Nicolae[g] bogdan.nicolae@inria.fr
Christian Pérez[c] christian.perez@inria.fr
Anthony Simonet[c] anthony.simonet@inria.fr
Frédéric Suter[i] frederic.suter@cc.in2p3.fr
Bing Tang[c] bing.tang@ens-lyon.fr
Raphael Terreux[d] r.terreux@ibcp.fr

Abstract: As Map-Reduce emerges as a leading programming paradigm for data-intensive computing, today's frameworks which support it still have substantial shortcomings that limit its potential scalability. In this paper we discuss several directions where there is room for such progress: they concern storage efficiency under massive data access concurrency, scheduling, volatility and fault-tolerance. We place our discussion in the perspective of the current

[a]Corresponding author
[b]INRIA Research Center, Rennes – Bretagne Atlantique, Rennes, France
[c]INRIA Research Center, Grenoble Rhône-Alpes, Lyon, France
[d]CNRS-Université Lyon1, Institut de Biologie et Chimie des Protéines, Lyon, France
[e]ENS Cachan – Antenne de Bretagne, Rennes, France
[f]IBM Products and Solutions Support Center, Montpellier, France
[g]Joint INRIA-UIUC Laboratory for Petascale Computing, Urbana-Champaign, USA
[h]Argonne National Laboratory, Argonne, USA
[i]CNRS, CC IN2P3, Lyon, France

evolution towards an increasing integration of large-scale distributed platforms (clouds, cloud federations, enterprise desktop grids, etc.). We propose an approach which aims to overcome the current limitations of existing Map-Reduce frameworks, in order to achieve scalable, concurrency-optimized, fault-tolerant Map-Reduce data processing on hybrid infrastructures. This approach will be evaluated with real-life bio-informatics applications on existing Nimbus-powered cloud testbeds interconnected with desktop grids.

Keywords: Map-Reduce, cloud computing, desktop grids, hybrid infrastructures, bio-informatics, task scheduling, fault tolerance

1 Introduction

The "data-intensive science" is emerging as a new paradigm for scientific discovery that reflects the increasing value of observational, experimental and computer-generated data in virtually all fields, from physics to the humanities and social sciences. As the volumes of generated data become increasingly higher, current solutions for data storage and processing show their limitations. In this context, Map-Reduce has arisen as a promising programming paradigm. Initially pushed by Google, it rapidly gained interest among a large number of cloud service providers, mainly for its potentially high performance scalability for data-intensive analysis.

At the core of Map-Reduce frameworks stays a key component with a huge impact on their performance: the storage layer. To enable scalable parallel data processing, this layer must meet a series of specific requirements. First, efficient *fine-grained access* to huge files is needed, since Map-Reduce applications deal with a very large number of small records of data. Second, a *high throughput* needs to be sustained when a large number of clients access the same files concurrently. Handling massively concurrent reads AND writes is here an important challenge. A popular optimization technique employed in today's Map-Reduce frameworks is to leverage locality by shipping the computation to nodes that store the input data; the goal is to minimize data transfers between nodes. For this reason, the storage layer must be able to provide the information about data location. This *data layout exposure* helps the framework to efficiently compute smart task schedules by placing tasks as close as possible to the data they need.

Although specialized file systems have been designed to address some of these critical needs (e.g. HDFS [22], GoogleFS [8]), they still have several major limitations. Most Map-Reduce applications process *a huge number of small files*, often in the order of KB: text documents, pictures, web pages, scientific datasets, etc. Each such object is typically stored by the application as a file in a distributed file system or as an object in a specialized storage service. With the number of files easily reaching the order of billions, a heavy burden lies on the underlying storage service, which must efficiently organize the directory structure such that it can efficiently lookup files. Therefore, one important challenge is to find scalable ways to organize data and to avoid complex namespaces that are slow to browse and

maintain. Another limitation regards the *metadata centralization*. In their original flavor, HDFS and GoogleFS use a single-node namespace server architecture, which acts as a single container of the file system metadata. Decentralization is a promising alternative, but it can be source of metadata consistency issues.

More importantly, the existing storage approaches also suffer from *limited throughput under heavy access concurrency*. HDFS does not support concurrent writes to the same file, and the data cannot be overwritten nor appended to. While HDFS is not optimized for small I/O operations, it uses client side buffering to improve the throughput. An important challenge is to define a consistency semantics and to implement a concurrency control mechanism for it to sustain a high throughput under heavy concurrency, while offering a clear set of semantic guarantees. Moreover, *dedicated fault tolerance mechanisms* which efficiently leverage the specifics of Map-Reduce software architectures are needed, as traditional solutions employed by current systems do not appear satisfactory enough.

Finally, an important challenge regards the target execution infrastructures. While the Map-Reduce programming model has become very visible in the cloud computing area, it is also subject to active research efforts on other kinds of large-scale infrastructures, such as desktop grids. We claim that it is worth investigating how such efforts (currently done in parallel) could converge, in a context where large-scale distributed platforms become more and more connected together.

The work presented in this position paper introduces an approach which aims to overcome the limitations mentioned above, in order to achieve scalable, concurrency-optimized, fault-tolerant Map-Reduce data processing on hybrid infrastructures relying on clouds and desktop grids. We are designing and implementing our approach through an original architecture for scalable data processing: it combines two approaches, BlobSeer [16] and BitDew [7], which have shown their benefits separately (on clouds and desktop grids respectively) into a unified system. The global goal is to improve the behavior of Map-Reduce-based applications on the target large-scale infrastructures. We will experiment the effectiveness of our approach on Nimbus clouds in connection with desktop grid testbeds, with real-life workloads generated by data mining bio-chemistry and bioinformatics applications.

This French-American collaborative research work is supported by the French National Research Agency in the framework of the MapReduce project (http://mapreduce.inria.fr, ANR-10-SEGI-001).

2 Motivations

2.1 An Application Case Study

In this work we consider a motivating scenario coming from the bio-chemistry area: it concerns protein structure analysis. Proteins are major components of life, involved in lots of biochemical reactions and vital mechanisms. Proteins also strongly participate to the molecular infrastructure of organisms (cell, organs, tissues). The three-dimensional (3D) structure of a protein is essential for its function and for its participation to the whole metabolism of a organism. However,

Target	Data size (models)	Time
Ara H1	23	3 h 18 m
Ara H1	3,541	73 h 10 m

Figure 1 Computation time in drug and food allergy experiments with SuMo

due to experimental limitations, few structures of proteins have been determined with respect to the number of known protein sequences. Indeed, there are roughly 60,000 structures referenced in a unique database, the Protein Data Bank (PDB), whereas millions of protein sequences are recorded in tens of other databases. To model the structure of a protein or to infer its function, an usual way is to compare its structural model to the whole set of known structures.

The SuMo (Surf the Molecules) [11] application permits coarse-grain comparisons of two protein structures. SuMo performs structural protein analysis by comparing the structure of the protein against a set of reference structures stored in a database. SuMo can for example make a comparison of a set of protein structures against the whole PDB, representing hundreds of gigabytes of data. The method is available online as a web portal opened to the scientific community [10].

The SuMo software was used to determine cross reaction allergy. An allergen database was created with the Modeome tool, which is an automatic homology modeler. From 6,000 initial allergen sequences, 3,541 models were computed and the database was converted in SuMo format. The experiment was done with this database and the "arachide Ara H1" allergen as the query component. The computations were ran on an intel Xeon 5160 3.00GHz 4-CPU 4GB RAM computer with a CentOS linux distribution. A first run was done with this allergen, pairwise compared to a set of 23 models. The computation time was 3 hours and 18 minutes, using full-time three CPU-cores. In the second run, the allergen was compared to a larger set, the whole 3,541-models database, and required 73 hours and 10 min (see the table in Figure 1). The results were compared to the literature, showing a great concordance with previous works.

The challenge now is to analyze larger datasets with the SuMo method, taking advantage of a Map-Reduce framework over a cloud computing hardware configuration or a hybrid testbed consisting of multiple clouds or combining clouds with (enterprise) desktop grid testbeds. The goal is to define the optimal configuration to split the data to fit the "Map" step. This first step consists in compiling the reference database and storing it according to the data placement rules of the used cloud. The second step, the "Reduce" one, consists in filtering and gathering the results, for example with a filter putting a threshold on the top 10 hits.

For pharmaceutical and biotech industries, such an implementation running over a cloud computing facility opens several new applications for drug design. Rather than searching for 3D similarities into structural-biology data, it could become possible to classify the entire structural space and to periodically update all derivative predictive models with new experimental data. This is a typical data-intensive application that can leverage the Map-Reduce model for a scalable execution on large-scale distributed platforms. Having a cloud- and dektop-grid-enabled version of SuMo will also be representative and help to define best

practices for the usage of other bioinformatics applications on the clouds and on hybrid infrastructures.

2.2 On the Usefulness of Hybrid Infrastructures

The goal of this work is to investigate ways to achieve scalable Map-Reduce processing on emerging large-scale distributed infrastructures. In the first place, we consider cloud infrastructures as made available today by IaaS (Infrastructure-as-a-Service) cloud service providers. As mentioned in Section 1, data storage facilities available on such clouds are still rudimentary and do not fit the needs of data-intensive applications with substantial requirements in terms of highly concurrent data sharing. Making progress in this direction is our first goal. Second, we also consider Desktop Grids as a type of large-scale infrastructure with specific characteristics in terms of volatility, reliability, connectivity, security, etc. In the general case, Desktop Grids rely on resources contributed by volunteers. Enterprise Desktop Grids are a particular case of Desktop Grids which leverage unused processing cycles and storage space available within an enterprise. Making such facilities available to external users is actually at the origin of the Cloud Computing concept! The emergence of cloud infrastructures has opened new perspectives to the development of Desktop Grids, as new types of usage may benefit from a hybrid, simultaneous use of these two types of infrastructures. We envision a typical scenario where an enterprise would not use dedicated, on-site hardware resources for a particular need for data-intensive analysis (e.g., to process commercial statistics): it would rather rely on free unused internal resources (using the Enterprise Desktop Grid model) and, in extension to them, would rent resources from the cloud when needed. Both architectures are suitable for massively parallel processing. This is a sample scenario that could benefit from the potential advantages of using such hybrid infrastructures. Finally, more complex combinations of platforms resulting from the use of multiple clouds in extension to an Enterprise Desktop Grid, for instance, can be imagined. The general architecture we are proposing can cope with such a general scenario.

3 Our Proposal: Global Architecture Overview

Our first focus is the data storage and management architecture, which aims to enable highly-scalable Map-Reduce-based data processing on various physical platforms: clouds, desktop grids or hybrid infrastructures built by combining these two. To build this architecture, we choose to rely on two building blocks which have already shown their benefits, as shown in Figure 2. One of them is BlobSeer [16], a distributed data management system that combines distributed metadata management with multi-versioning techniques to enable fine-grain access to massive, distributed data under heavy concurrency. Preliminary studies [20] with BlobSeer as a storage substrate in Map-Reduce frameworks for Hadoop-like file systems have already demonstrated substantial gains for many access patterns which exhibit concurrency: concurrent reads to the same file, concurrent writes to the same file, concurrent reads (or writes) to different files. Consequently, BlobSeer appears as an appropriate approach that outperfoms Hadoop's HDFS

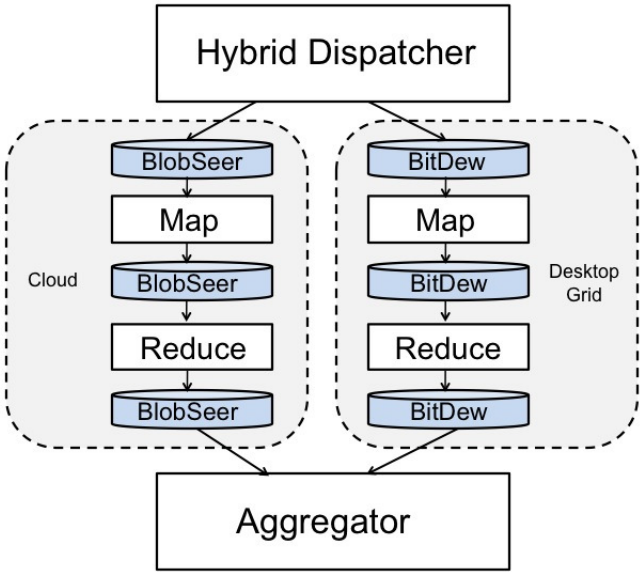

Figure 2 The global architecture of the hybrid approach.

file systems for Map-Reduce processing on clouds. BitDew [7], a data-sharing platform for desktop grids, is the complementing building block. Within BitDew, we design a new network of storage peers. As usual, it handles content delivery, but also, more originally, the main collective data operations which exist in Map-Reduce, namely the Scatter (distribution of file chunks), Shuffle (redistribution of intermediate outputs) and Combine (assemblage of the final result) phases. These two data management systems will be used in conjunction to experiment with hybrid infrastructures.

Second, we aim to introduce new scheduling techniques for large executions of Map-Reduce instances, able to scale over heterogeneous platforms (see Section 5). From the low level details of the scheduling of instance creations to the scheduling of map and reduce tasks over various platforms, we integrate scheduling heuristics at several levels of the software suite.

Finally, we explore techniques to improve the execution of Map-Reduce applications on large-scale infrastructures with respect to fault tolerance and security (detailed in Section 6). As each of these aspects alone is a considerable challenge, our goal is not to address them in an extended way: we will rather focus on a few specific aspects related to the support of the Map-Reduce paradigm on clouds and on desktop grids. Given the dynamic nature of such platforms and the long runtime and resource utilization of Map-Reduce applications, an efficient checkpoint-restart mechanism becomes paramount in this context. We propose a solution to this challenge that aims at minimizing the storage space and performance overhead of checkpoint-restart.

4 Focus: Data Storage Architecture

4.1 Concurrency-optimized Storage and Processing on Clouds: BlobSeer

In this section we introduce BlobSeer [16], a concurrency-optimized data-management system for data-intensive distributed applications. BlobSeer can help to overcome some of the limitations presented in Section 1 and to provide new perspectives in the context of cloud data storage. To meet the specific requirements of Map-Reduce based data-intensive applications for large-scale infrastructures, the BlobSeer system was designed to comply with the following principles:

Massive unstructured data.

In BlobSeer, data is organized as a set of large, unstructured sequences of bytes, denoted BLOBs (Binary Large Objects), uniquely identified. This ensures *data-location transparency*, allowing users to access data only by knowing a simple identifier, instead of being aware of the location of a specific piece of data. Furthermore, compacting many KB-sized files generated by distributed applications into huge files enhances scalability by reducing the management overhead of many filenames and associated namespace hierarchies. However, to provide a useful tool for processing such data, BlobSeer also provides an interface for *fine-grained access* to the stored data sequences, enabling concurrent processes to retrieve the needed data blocks without needing to sequentially search through the whole BLOB.

Data striping.

Each BLOB is split into equally-sized *chunks* which are distributed across multiple storage nodes. The size of each BLOB is specified by the user, so that it can be fine-tuned according to the needs of the applications. BlobSeer is able to achieve high aggregate transfer rates due to the balanced chunk distribution among storage nodes, especially when considering many simultaneous clients that require access to non-overlapping chunks of the same BLOB.

Distributed metadata management.

In BlobSeer, *metadata* denotes the information needed to map the location of each BLOB chunks on the storage nodes. Each chunk is uniquely identified in the system by its *BLOB identifier*, *offset* within the BLOB and *size*. Such information is stored on specifically designed nodes and is employed by the users to discover the location of each chunk that has to be retrieved. Distributing metadata also has an additional advantage, namely it can eliminate *single points of failure* when the metadata are replicated across multiple servers.

High throughput under heavy concurrency.

This requirement is addressed in an original way through *versioning-based* concurrency control. In BlobSser, data is never overwritten. Instead, each new WRITE performed on a specific BLOB results in a new version. Each BLOB

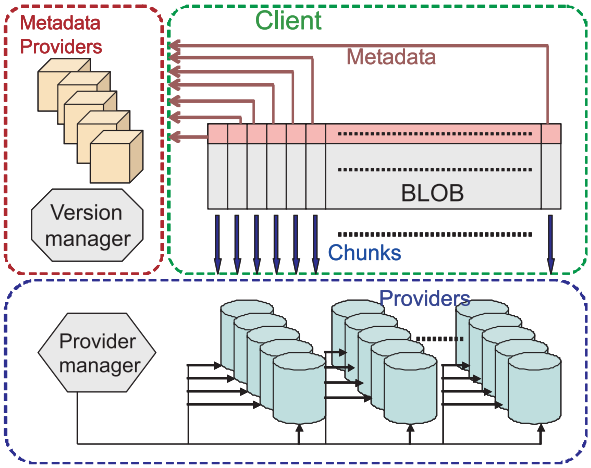

Figure 3 The architecture of the BlobSeer system.

version stores only the differential update with respect to the previous versions, but exposes the whole BLOB obtained as a result of the WRITE operation. This approach enables an efficient implementation of concurrent updates of the same BLOB, by considering all data and metadata immutable. Furthermore, a versioning-based design allows for a complete decoupling between concurrent READ and WRITE operations performed on the same BLOB.

The architecture of BlobSeer is based on a set of distributed entities illustrated in Figure 3.

Data providers host data chunks. Each *data provider* is implemented as a high-performance key-value store, which supports fast upload or download of data chunks. This in-memory cache is backed by a persistency layer built on top of BerkleyDB, an efficient embedded database.

The provider manager is responsible for assigning *data providers* to the WRITE requests issued by users, using a configurable strategy. The default strategy is to select the provider that stores the smallest number of chunks and the smallest number of pending WRITE requests. More complex strategies can be implemented if the system has access to more detailed information about the state of the *data providers* and the user requirements.

Metadata providers keep track of the chunks distribution across *data providers*. For each BLOB, the metadata is organized as a *distributed segment tree* where each node corresponds to a version and to a chunk range within that version. Each leaf covers just one chunk, recording the information about the data provider where the page is physically stored. The metadata trees are stored on the *metadata providers*, which are processes organized as a Distributed Hash Table.

The version manager ensures the serialization of the concurrent WRITE requests and the assignment of version numbers for each new WRITE operation. Its goal is to create the illusion of instant version generation, so

that this step does not become a bottleneck when a large number of clients concurrently update a specific BLOB.

The client library makes available an access interface to applications. The *client* supports the following operations: CREATE BLOBs, READ, WRITE or APPEND contiguous ranges of bytes.

A typical setting of the BlobSeer system involves the deployment of a few hundreds data providers, storing BLOBs of the order of the terrabytes. The typical size for a chunk within a BLOB can be smaller than 1 MB, whence the challenge of dealing with hundreds of thousands of chunks belonging to just one BLOB. BlobSeer provides efficient support for heavily-concurrent accesses to the stored data, reaching a throughput of 6.7 GB/s aggregated bandwidth for a configuration with 60 metadata providers, 90 data providers and 360 concurrent writers, as shown in [15].

4.2 BSFS: BlobSeer File System for Map-Reduce

We integrated BlobSeer with Hadoop, by building a BlobSeer-based file system (BSFS), that can successfully be used as storage backend for Map-Reduce applications executed with Hadoop [20]. BSFS can be used as both a stand-alone DFS and as a replacement for HDFS in the Hadoop framework. BSFS takes advantage of BlobSeer's features to provide high I/O throughput and to sustain it under highly-concurrent access to data. To this end, we implemented an API on top of BlobSeer, with several functionalities:

Managing the file system namespace : The BSFS layer keeps the file system metadata and directory structure, i.e., the information regarding file properties and the hierarchy of files and directories.

Mapping files to BLOBs : BlobSeer offers a flat storage space, in which data is kept in BLOBs uniquely identified in the system by a key. In order to use BlobSeer as a regular DFS, BSFS maps each file to a BLOB, by associating the file name to the BLOB id.

Implementing the Hadoop API : By implementing the file system interface through which Hadoop accesses its storage backend, BSFS can serve as storage for Map-Reduce applications running on top of Hadoop. Moreover, BSFS can act as a stand-alone DFS that can be accessed through an HDFS-like file system interface.

Communication between clients and BSFS is kept to a minimum, only for operations concerning the file system metadata; data accesses are performed through a direct communication between the client and the BlobSeer storage nodes. BSFS also implements *client-side buffering* to reduce overhead. This consists in prefetching a whole chunk when a read of 4 KB is issued, and in collecting the small records issued as final output until the data collected reaches at least the size of a chunk (64 MB by default). Hence, the actual reads and writes from/to the underlying storage layer are performed on data large enough to compensate for network traffic overhead.

The benefits of using BSFS as storage layer for Map-Reduce applications were validated through large-scale experiments; these tests compare BSFS and HDFS in different scenarios. They show a substantial gain in execution time with respect to HDFS (in the order of 35%). Moreover, BSFS has additional functionalities that HDFS does not support: concurrent appends, concurrent writes at random offsets and versioning.

4.3 Data-intensive on Desktop Grids: BitDew

This section presents a Map-Reduce implementation targeted at Desktop Grids. Desktop Grids harvest computing and storage power of volunteer idle desktop computers. Very large platforms like Boinc [1] proved that volunteer peers can provide huge storage potential.

The challenge we address here is to broaden the scope of applications that can benefit from Desktop Grids. The typical application class for this platform are Bag-of-Tasks applications composed of independent tasks with small input and output data. We prove that granted we overcome several challenges, Desktop Grids are able to run Map-Reduce computations. The first challenge comes from the difficulty of performing the collective file operations necessary during the shuffle phase when intermediate results are exchanged between the Map and Reduce phases. The difficulty comes from the volatility of hosts, unstable network conditions, the presence of NATs and firewalls that make collective communications difficult to achieve. The second challenge comes from the high volume of intermediate results that makes it impractical to send them all back to the server. Result certification necessary to guarantee that the results have not been falsified by malicious volunteers [21] must be performed in a decentralized fashion, when it is usually centralized on existing Desktop Grid platforms. The third challenge comes from the dependencies between Reduce and Map tasks that can slowdown the whole process when hosts are too volatile and some perform much slower than the others.

For Map-Reduce to perform efficiently on Desktop Grids, we propose several solutions relying on latency hiding, distributed result checking through replication of data and tasks and barrier-free reduction.

4.4 Architecture of Map-Reduce over BitDew

In this section we present key BitDew features and optimizations needed to run Map-Reduce computations on Internet Desktop Grids.

BitDew [7] is a middleware developed at INRIA; its primary goal is to provide high-level APIs to manage large data-sets on hybrid distributed infrastructures composed of Desktop Grids, Clouds and Grids. BitDew takes care of fault tolerance, data scheduling and supports several data transfer protocols transparently. These features involve decisions that are driven by data attributes provided by the programmer.

Our Map-Reduce implementation [24] relies on BitDew to cope with the difficulty of using Desktop Grids. Using the data attributes described above, we set dependencies between map inputs, the mappers, intermediate results, the

reducer and the final result. The BitDew runtime will use the dependencies to automatically create a data flow between the nodes. It will place data on volunteer computers and deal 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. To hide the high latency caused by Internet network conditions, NATs, and firewalls, we overlap communications with computation. BitDew offers asynchronous APIs to several synchronous transfer protocols such as HTTP and FTP. Workers are multi-threaded so that a maximum number of Map and Reduce threads can be configured and incoming tasks are enqueued as soon as they are received.

Collective File Operation. Map-Reduce requires a set of collective communications inherited from parallel programming. These operations are 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. To implement such collective operations in BitDew, we designed DataCollection, used to manipulate a collection of data as a single object, and DataChunk that are individual items of a DataCollection.

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

Barrier-Free Computation. Host churn necessitates that the barriers present in traditional Map-Reduce implementations be removed. To avoid barriers we duplicate input data and intermediate results. Reducer nodes examine their queue and keep only the last version of intermediate files when duplicates exist, so the first version of an intermediate result is reduced. Additionally, Reduce tasks can work on segments of a key interval. Thus, unlike traditional Map-Reduce implementations, Reduce nodes start working as soon as intermediate results are available. The reducer knows when its task is finished because the number of data chunks and the number of reducers are known in advance. Replication and early reduction allows to eliminate the barrier between Map and Reduce tasks.

Scheduling. We follow the usual Map-Reduce paradigm of moving the computation where the data resides. Map-Reduce 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 that another node can take over the same task.

Distributed Result Checking. As Desktop Grid can always contain several 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, so it has to be decentralized. Here we use the majority voting certification algorithm [14]. Input files, Map tasks and thus intermediate results are replicated and sent to reducers. Once a reducer has received at least n of p intermediate results, it compares them and keeps the one that appears the most. Reduce tasks are replicated as well and the master node takes care of checking the Reduce outputs.

5 Scheduling Issues

The Map-Reduce paradigm has been designed in order to simplify the task of the application programmer using a large set of nodes to perform data intensive computations. The frameworks implementing the Map-Reduce paradigm have to schedule the Map and Reduce tasks. There are several places in a Map-Reduce framework where optimized schedulers are needed. This includes choosing the amount of data each node has to process, how to rebalance the data in order to make each node get the data it has to process and choosing the order of the transfers between the Map and Reduce tasks.

5.1 Previous Work

The main goal of the scheduling phase can be to minimize the makespan. Taking as much parameters into account as possible makes the schedule closer to the global optimal solution. The data processed by the Map function is made block by block. Thus, computing the amount of data to be processed by every node is an integer variables problem. However, as described by the Divisible Load Theory [3], this kind of problem can be solved as a real variables problem and then rounded to integers. This allows to compute efficiently a schedule from what is known about the platform and about the transfer that will happen just after the end of the Map computation.

Berlinska and Drozdowski [2] took a similar approach where they use the Divisible Load Theory to compute the amount of data processed by each node taking into account the time needed to transfer data between the Map and Reduce tasks.

In their model, it is considered that the Map and Reduce operations are performed by distinct nodes. This is a simplification that eliminates the need for a special case where a node would transfer the data to itself. Every node starts sequentially. That means that a given node cannot start before the previous node has finished its sequential startup. Then, it is assumed that every node has access to the data without contention, so that the time needed for the Map operations to finish is directly proportional to the amount of data to process. The same happens for the transfers between the Map and Reduce tasks. Moreover, it is considered that every mapper node will have the same amount of data to send to every reducer node. And finally, there is a one-port constraint that means that every node can send or receive data to / from only one node at the same time and the number of parallel transfer is limited.

Given their transfer scheduling algorithm, they have to use a linear program to compute the data size for each mapper node to process. This is quite a heavy computation that does not scale very well.

5.2 Communications Optimization

In our work, we use the same model as Berlinska and Drozdowski and the same constraints. Under certain circumstances, the result given by the linear program shows some remarkable regularities. Especially, the computations end just when the first transfer of the previous node ends. This allows to simplify the linear program which becomes a linear system. This linear system can be solved in $O(m)$ time, with m being the number of mapper nodes, which makes it scale much better than a linear program.

Another observation about the results from Berlinska and Drozdowski is that they use a static transfer scheduler which leads to a suboptimal bandwidth usage. By using a scheduler that starts the transfers as soon as possible, still enforcing the same constraints on the order in which they may happen, we can reduce the transfer duration by around 15 - 20%.

We also tried to release some constraints on the order of the transfers and allow the transfers with higher priority to preempt other transfers. This always leads to an improvement of the transfer time, but the improvement is then negligible.

In the near future we plan to implement this in the global framework trying to take into account the data management by BlobSeer or BitDew. We also plan to implement a feedback from the compute nodes to the master node in order to adjust the schedule with respect to some measured parameters like the processing speed or some other hard-to-guess parameters of the model.

6 Handling Fault Tolerance

The Map-Reduce paradigm was designed specifically to scale, being able to take advantage of the computational resources in large datacenters. However, due to the large amount of resources needed to run Map-Reduce applications and the fact that they run on commodity hardware, failures are rather the norm than the exception [25]. Thus, *fault tolerance* becomes a critical issue that needs to be addressed.

Due to the similarities between Map-Reduce and the *master-worker* paradigm, production-ready Map-Reduce frameworks handle fault tolerance by simply rescheduling the failed tasks for another execution until they succeed. This scheme has an important advantage: it is easy to design and implement. However, its performance is highly dependent on the complexity of the tasks: if they take a long time to finish and involve a lot of resources, then a simple rescheduling wastes a lot of computational resources. As an example, consider the case of scientific applications that can be solved using Map-Reduce but whose *mappers* cannot be expressed as simple, short computations. In this case, the mappers are rather complex and need to be expressed as mini-instances of tightly-coupled computations. Taken to the extreme, these mini-instances might become complex enough to even need to be distributed themselves. In this context, simple

rescheduling is not enough and needs to be complemented with more advanced approaches than can deal with such complex mappers.

We propose the use of *Checkpoint-Restart (CR)* [6] to address this issue: fault tolerance for complex mappers is achieved by saving recovery information periodically during failure-free execution and restarting from that information in case of failures, in order to minimize the wasted computational time and resources. Although extensively studied in the context of High Performance Computing (HPC) applications, CR faces an important challenge in our context: it needs to adapt to the datacenter infrastructure typically used for Map-Reduce applications, i.e. build out of commodity hardware. With the growing popularity of Infrastructrue-as-a-Service (IaaS) clouds, this infrastructure also includes a virtualization layer that isolates Map-Reduce workloads in their own virtual environment. We focus our efforts in this direction.

To this end, we rely on *BlobCR* (BlobSeer-based Checkpoint-Restart) [18], a checkpoint-restart framework specifically optimized for tightly-coupled scientific computations that were written using a message passing system (e.g. *MPI* [9]) and need to be ported to IaaS clouds.

Unlike traditional HPC infrastructures, IaaS clouds typically feature local disk storage that does not generate operational costs (i.e. it can be used "for free" by the user). Leveraging local storage is a common scenario for Map-Reduce applications: it has the potential to both improve I/O performance and cut on operational costs compared to the typical solution of using a parallel file system. However, local disk storage is not guaranteed to survive beyond the life-time of a VM, which can at any point be terminated due to failures. Thus, one challenge we address is to extend CR for the scenario where data is stored on the local disk rather than a parallel file system. Doing so enables us to support an important CR feature that is not available on traditional HPC infrastructures: the ability to roll back file system changes.

To achieve this, BlobCR introduces a dedicated checkpoint repository that is able to take incremental snapshots of the whole disk attached to the virtual machine (VM) instances. More specifically, during normal execution, all writes of the application processes are performed on the virtual disk. When a checkpoint needs to be taken, the state of the processes are saved into files on the virtual disk and then a snapshot of the virtual disk is taken. This works either at application-level, where the process state is managed by the application itself, or at process-level, where the process state is captured transparently at the guest operating system level using tools such as *BLCR* [5]. Obviously, a mechanism must exist that enables the application to request snapshots of its virtual disk, which can be taken only by the physical machine. To this end, BlobCR exposes a special remote API inside the VM machine instances. Synchronization of the snapshots to reflect a globally consistent checkpoint is the responsibility of the checkpointing protocol. If a VM fails, BlobCR re-deploys a new instance of it and resumes the computation inside the VMs from the most recent successful checkpoint by restoring the corresponding virtual disk snapshots and relaunching the application.

To facilitate the resume process, we expose VM disk snapshots as first-class objects. All snapshots are incremental in order to minimize performance and resource consumption overhead during execution. The snapshots are stored in a striped fashion on the local disks, with each stripe replicated for resilience. Several

optimizations such as lazy instantiation [17] and adaptive prefetching [19] are used to accelerate the instantiation of new VMs. A more detailed description of how these all techniques are leveraged by BlobCR is available in [18].

Compared to state-of-art (that uses full VM snapshots to provide fully transparent checkpointing at the virtualization layer directly), we demonstrate large performance gains and much lower resource utilization [18] both using synthetic benchmarks and real life HPC applications. Encouraged by these results, we plan to integrate BlobCR into Hadoop, an open-source Map-Reduce framework and study its benefits in the context of complex mappers that need to spawn mini-instances of tightly-coupled scientific computations. We expect to obtain better overall performance and lower resource utilization compared to plain rescheduling.

7 Putting Everything Together

7.1 Overview of the Approach

One major challenge to overcome in order to execute Map-Reduce applications on hybrid infrastructures is to connect and configure all the elements into a coherent and efficient framework. Our goal is not to re-implement a full new Map-Reduce framework but to leverage existing building blocks as much as possible. Software component models [13, 23] provide concepts with respect to software assembly architecture and code re-use. Our approach is to use a state-of-the art component model to describe the global structure and the elements to configure. Then, scheduling and decision algorithms are used to derive a concrete structure of the runtime elements. Let us first introduce HLCM, the component model we have chosen to leverage.

7.2 Introducing HLCM

High Level Component Model (HLCM) [4] is a software component model that supports concepts of hierarchy, genericity and connectors—and in particular the novel concepts of *open connection*. It aims at providing a framework to transform a description of an application in a model independently of the resources to a deployable assembly.

HLCM is independent of the definition of primitive components and connections: specializations of HLCM have to define them. For example HLCM/Gluon++ is a specialization of HLCM where primitive components and connections are defined in Gluon++, a thin layer turning Charm++ objects [12] into components. HLCM/Gluon++ has currently only one primitive connector: Charm++ RMI Use/Provide.

Another example is HLCM/L2C. *Low Level Component* (L2C) is a component model where primitive components are C++ classes with three kinds of primitive connectors: C++ Use/Provide, Corba Use/Provide and MPI communicator. HLCM/LLCMj is a last example of specialization for Java based components.

An HLCM specialization generates a representation on an application in a format that depends on the specialization. This generation makes use of specific algorithms and resource models to control which non-functional concerns have to

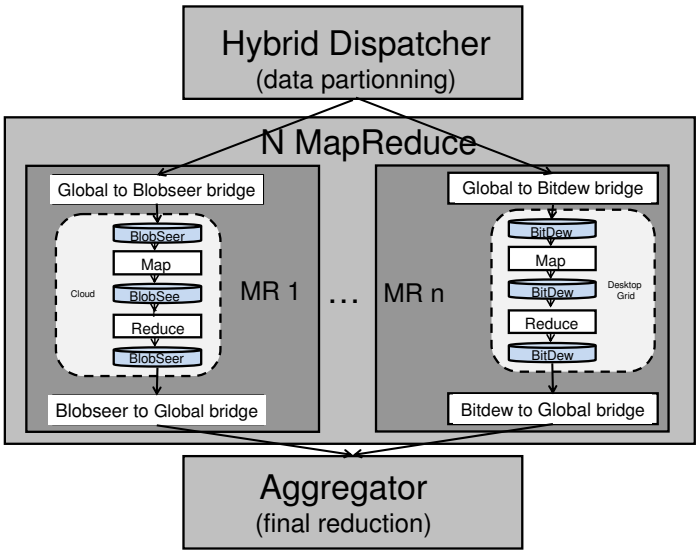

Figure 4 High level description of an hybrid Map-Reduce application. The value of N as well the actual implementation of Map-Reduce component type are determined by scheduling and decision algorithms.

be optimized (completion time of the application, computation cost, etc). The actual deployment of the generated application is not handled by HLCM.

7.3 Applying HLCM to Map-Reduce processing on hybrid infrastructures

To implement the vision described in Figure 2, HLCM offers very interesting features. First, it does not impose any primitive component model. Hence, any native code – in Java, C++, or any other language – can be supported by HLCM as long as a specialization is defined. Similarly, any native communication model can be integrated with the same approach, also without introducing any overhead. Therefore, HLCM appears as a very interesting model to manage a hybrid Map-Reduce platform.

Second, HLCM enables to describe an abstract architecture with many choices left to the transformation algorithm. The proposed approach is to connect the Hybrid Dispatcher component to an arbitrary number of components of type Map-Reduce, as shown in Figure 4. This number and the actual implementation of these Map-Reduce components are left to the transformation algorithm. Based on performance metrics and optimization goals, it will select the actual implementations as well as a partitioning of the data.

The most difficult part of wrapping an existing Map-Reduce runtime into a Map-Reduce component is to define bridges to convert data formats, if needed. For example, BlobSeer and BitDew need specialized components to copy data to/from their space – similarly as Hadoop works with data in HDFS.

8 Conclusions

The Map-Reduce programming model is currently subject to many research efforts, however state-of-the-art solutions are still far from meeting many requirements of data-intensive computing applications. This paper discusses several major limitations in current Map-Reduce frameworks and proposes an approach which aims to overcome these limitations. We focus on an original setting where target execution infrastructures possibly rely on several cloud service providers in extension to desktop grid platforms. We propose a fault-tolerant data management architecture whose goal is to enable highly-scalable Map-Reduce data processing on such hybrid infrastructures built by combining clouds and desktop grids. We complement this solution with new scheduling techniques for large executions of Map-Reduce instances, able to scale over heterogeneous platforms. All building blocks of this architecture are connected together by means of a software component model and are exposed to clients as a coherent and efficient Map-Reduce framework. Current work is focusing on demonstrating the effectiveness of the proposed integrated architecture on real platforms relying on Nimbus clouds and on Grid'5000-based desktop grids with real-life workloads generated by data mining bio-chemistry and bioinformatics applications.

9 Acknowledgements

This work was supported by the Agence Nationale de la Recherche under Contract ANR-10-SEGI-001.

References

[1] David Anderson and Gilles Fedak. The Computational and Storage Potential of Volunteer Computing. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 73–80, Singapore, May 2006.

[2] J. Berlinska and M. Drozdowski. Scheduling divisible MapReduce computations. *Journal of Parallel and Distributed Computing*, 71(3):450–459, March 2010.

[3] Veeravalli Bharadwaj. *Scheduling divisible loads in parallel and distributed systems*. Wiley-IEEE Computer Society Pr, Los Alamitos-Washington-Brussels-Tokyo, 1996.

[4] Julien Bigot and Christian Pérez. *High Performance Scientific Computing with special emphasis on Current Capabilities and Future Perspectives*, volume 20 of *Advances in Parallel Computing*, chapter On High Performance Composition Operators in Component Models, pages 182–201. IOS Press, 2011.

[5] J. Duell, P. Hargrove, and E. Roman. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. Technical Report LBNL-54941, Future Technologies Group, 2002.

[6] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34:375–408, September 2002.

[7] Gilles Fedak, Haiwu He, and Franck Cappello. BitDew: A Data Management and Distribution Service with Multi-Protocol File Transfer and Metadata Abstraction. *Journal of Network and Computer Applications*, 32:961–975, 2009.

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. *SIGOPS - Operating Systems Review*, 37(5):29–43, 2003.

[9] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI (2nd ed.): portable parallel programming with the message-passing interface*. MIT Press, Cambridge, MA, USA, 1999.

[10] M. Jambon, O. Andrieu, C. Combet, G. Deleage, F. Delfaud, and C. Geourjon. The sumo server: 3d search for protein functional sites. *Bioinformatics*, 21:3929–30, 2005.

[11] M. Jambon, A. Imberty, G. Deleage, and C. Geourjon. A new bioinformatic approach to detect common 3d sites in protein structures. *Proteins*, 52:137–45, 2003.

[12] Laxmikant V. Kale, Eric Bohm, Celso L. Mendes, Terry Wilmarth, and Gengbin Zheng. Programming Petascale Applications with Charm++ and AMPI. In D. Bader, editor, *Petascale Computing: Algorithms and Applications*, pages 421–441. Chapman & Hall / CRC Press, 2008.

[13] M. D. McIlroy. Mass-produced software components. *Proc. NATO Conf. on Software Engineering, Garmisch, Germany*, 1968.

[14] Mircea Moca, Gheorghe Cosmin Silaghi, and Gilles Fedak. Distributed results checking for mapreduce on volunteer computing. In *Proceedings of IPDPS'2011, 4th Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid 2010)*, Anchorage, Alaska, May 2011.

[15] Bogdan Nicolae, Gabriel Antoniu, and Luc Bougé. Enabling High Data Throughput in Desktop Grids Through Decentralized Data and Metadata Management: The BlobSeer Approach. In *Euro-Par '09: Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, volume 5704 of *LNCS*, pages 404–416, Delft, Netherlands, 2009. TU Delft.

[16] Bogdan Nicolae, Gabriel Antoniu, Luc Bougé, Diana Moise, and Alexandra Carpen-Amarie. BlobSeer: Next Generation Data Management for Large Scale Infrastructures. *Journal of Parallel and Distributed Computing*, 71(2):168–184, February 2011.

[17] Bogdan Nicolae, John Bresnahan, Kate Keahey, and Gabriel Antoniu. Going back and forth: Efficient multideployment and multisnapshotting on clouds. In *HPDC '11: 20th International ACM Symposium on High-Performance Parallel and Distributed Computing*, pages 147–158, San José, USA, 2011.

[18] Bogdan Nicolae and Franck Cappello. Blobcr: Efficient checkpoint-restart for hpc applications on iaas clouds using virtual disk image snapshots. In *SC '11: 24th International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 34:1–34:12, Seattle, USA, 2011.

[19] Bogdan Nicolae, Franck Cappello, and Gabriel Antoniu. Optimizing multi-deployment on clouds by means of self-adaptive prefetching. In *Euro-Par '11: 17th International Euro-Par Conference on Parallel Processing*, pages 503–513, Bordeaux, France, 2011.

[20] Bogdan Nicolae, Diana Moise, Gabriel Antoniu, Luc Bougé, and Matthieu Dorier. BlobSeer: Bringing High Throughput under Heavy Concurrency to Hadoop Map-Reduce Applications. In *24th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2010)*, Atlanta, United States, February 2010. IEEE and ACM. A preliminary version of this paper has been published as INRIA Research Report RR-7140.

[21] Luis F. G. Sarmenta. Sabotage-Tolerance Mechanisms for Volunteer Computing Systems. *Future Generation Computer Systems*, 18(4):561–572, 2002.

[22] K. Shvachko, H. Huang, S. Radia, and R. Chansler. The Hadoop distributed file system. In *26th IEEE (MSST2010) Symposium on Massive Storage Systems and Technologies*, May 2010.

[23] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[24] Bing Tang, Mircea Moca, Stéphane Chevalier, Haiwu He, and Gilles Fedak. Towards MapReduce for Desktop Grid Computing. In *Fifth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC'10)*, pages 193–200, Fukuoka, Japan, November 2010. IEEE.

[25] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204, Indianapolis, USA, 2010.