# Enabling Computational Steering with an Asynchronous-Iterative Computation Framework

Alexandre di Costanzo[1], Chao Jin[1], Carlos A. Varela[2], and Rajkumar Buyya[1]
1. Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{adc, chaojin, raj}@csse.unimelb.edu.au

2. Department of Computer Science
Rensselaer Polytechnic Institute, NY, U.S.A
cvarela@cs.rpi.edu

## Abstract

*In this paper, we present a framework that enables scientists to* steer *computations executing over large-scale grid computing environments. By using computational steering, users can dynamically control their simulations or computations to reach expected results more efficiently. The framework supports* steerable *applications by introducing an asynchronous iterative MapReduce programming model that is deployed using Hadoop over a set of virtual machines executing on a multi-cluster grid. To tolerate the heterogeneity between different sites, results are collected asynchronously and users can dynamically interact with their computations to adjust the area of interest. According to users' dynamic interaction, the framework can redistribute the computational overload between the heterogeneous sites and explore the user's interest area by using more powerful sites when possible. With our framework, the bottleneck induced by synchronization between different sites is considerably avoided, and therefore the response to users' interaction is satisfied more efficiently. We illustrate and evaluate this framework with a scientific application that aims to fit models of the Milky Way galaxy structure to stars observed by the Sloan Digital Sky Survey.*

## 1 Introduction

Many scientific and engineering applications require high performance computing capabilities to complete their computations in a reasonable amount of time. Grids, like EGEE [11] and Grid'5000 [6], have been built to meet this challenge. Grids combine the power of large numbers of heterogeneous resources across geographically distributed sites. Resources are usually organised in clusters, which are managed by different administrative domains (labs, universities, *etc.*). Despite the computational power provided by grids, grids lack interactivity between users and their jobs. A typical use case of grids for an astronomer is: she first collects the data; then she submits the data to a grid to be processed; she waits until the processes return the results; next, she gathers the results and proceeds to their analysis; depending of this analysis she may need to refine some computations on some parts of the data; then she resubmits another computation to the grid and so on until she is satisfied with the results. These tasks are more complicated in a grid environment where she has to deal with challenges such as deployment, heterogeneity, fault-tolerance, communication, and scalability. Very often, significant computational resources and scientists' time are wasted as a result of the lack of interactivity.

This scientific process of submitting tasks and analyzing results can be accelerated by using *computational steering*. Steering consists of monitoring intermediate results and modifying running jobs. There are steering tools for grids, such as RealityGrid [5] that propose an API for instrumenting applications. However that approach requires users to modify their applications. Furthermore, consistently reaching intermediate points in the computation can be harder in heterogeneous environments. A solution to the heterogeneity problem is the use of virtual machine technologies. A virtual machine (VM) can be leased and used as a container for deploying applications [22]. Existing virtual machine-based resource management systems can manage a cluster of computers within a site allowing the creation of virtual workspaces [17]. They can bind resources to vir-

tual clusters or workspaces according to a user's varying demand of computational needs. These systems commonly provide an interface through which one can allocate VMs and configure them with the operating system and software of choice. Furthermore, more or less resources can be allocated on demand as the interest in the data changes.

Recently, the scientific community has started to use the MapReduce [15] programming model for treating large sets of data. MapReduce executes a *map* function, specified by the user, for processing the data and then merges the results using a *reduce* operation; both operations are run in parallel on a cluster. This paradigm is becoming more used for scientific applications, such as Blast [18] and data intensive analyses [12]. MapReduce has a lot of advantages for scientific applications because it is designed for data oriented applications and it provides a simple programming interface. However, MapReduce does not allow steering and is single-cluster oriented.

In this work, we propose a computational steering framework for grids that allows scientists to: dynamically refine their results, focus on the evaluation of data subsets, and re-distribute slow processes or renewed areas of interest on faster clusters. The implementation relies on an adaptation of Hadoop [13], an open-source MapReduce framework. Our framework dynamically sets several MapReduce clusters and asynchronously returns partial results to users. Users can completely control each partial result and remaining task of their computations.

The rest of this paper is organised as follows. In Section 2, we provide background on computational steering, the MapReduce programming model, and the multi-sites grid environment. Then, we present our asynchronous-iterative computation framework with computational steering in Section 3. Next, Section 4 shows the considered experimental scenario and evaluates the framework. Related work is discussed in Section 5 and conclusions are presented in Section 6.

## 2 Background and Context

In this section, we first introduce Computational Steering, we next describe the MapReduce programming model, and we then present our previous work on InterGrid.

### 2.1 Computational Steering

Computational steering is a mechanism for scientific investigation, which allows users to guide the progress of their computations according to their specific interests [16]. For instance, steering can be used for searching special structures, patterns, trends, and relationships in a large scale of data by interacting with users; or it can help users to identify an area of interest quickly during an execution of a long time. In addition, when users find any incorrect problems, steering allows users to modify the incorrect section.

Steering has been investigated for many scientific applications, such as computational fluid dynamics [20], manufacture engineering, and evolutionary algorithms [3]. Moreover, most features of computational steering are investigated at the visualization level, which aims to present an interactive interface to users for specifying any guidance during the execution of scientific applications.

A *steerable* application must have a point in the control loop of the program where a steering task can be performed. Normally, a steering task consists of the following jobs: 1) the retrieval of current results; 2) the acceptance of the alteration of parameters; 3) continuation of the computation with the new parameters. Many scientific computations comply with these requirements. Especially, many computationally intensive problems can get performance benefits from steering.

### 2.2 MapReduce

MapReduce [15] has been influenced by the power and simplicity of the *map* and *reduce* operations in functional languages, such as Lisp. This model allows users to write map/reduce components with functional-style code. These components are then composed as a dataflow graph to explicitly specify their parallelism. Finally, the MapReduce runtime system schedules these components to distributed resources for execution while handling many tough non-functional problems transparently: parallelization, network communication, and fault tolerance.

A *map* function takes a key/value pair as input and produces a list of key/value pairs as output. The type of the output key and value can be different from the type of the input:

$$map :: (key_1, value_1) \Rightarrow list(key_2, value_2) \quad (1)$$

A *reduce* function takes a key and associated value list as input and generates a list of new values as output:

$$reduce :: (key_2, list(value_2)) \Rightarrow list(value_3) \quad (2)$$

In a large cluster environment or data center, the MapReduce model is supported by exploiting the massive data parallelism through two phases. In the first phase, all map operations can be executed independently from each other. In the second phase, each reduce operation may depend on the outputs generated by any number of map operations. All reduce operations can also be executed independently similarly to map operations.

This model can split a large problem space into small pieces and automatically parallelize the execution of small tasks on the smaller space. In addition, the MapReduce

relies on a master-worker architecture, which is adapted to distributed systems. However, the MapReduce is data-oriented and may not be suitable for grid environments.

### 2.3 Multi-Sites Environment

In previous work, we presented a system for resource sharing between Grids [10] inspired by the peering agreements established between Internet Service Providers (ISPs) in the Internet, through which ISPs agree to allow traffic into one another's networks. The InterGrid relies on InterGrid Gateways (IGGs) that mediate access to resources of participating Grids. The InterGrid also aims at tackling the heterogeneity of hardware and software within Grids. The use of virtualization technology can ease the deployment of applications spanning multiple Grids as it allows for resource control in a contained manner. In this way, resources allocated by one Grid to another are used to deploy VMs.

A Grid has pre-defined peering arrangements with other Grids, managed by IGGs and, through which they coordinate the use of resources of the InterGrid. An IGG is aware of the terms of the peering with other Grids; selects suitable Grids able to provide the required resources; and replies to requests from other IGGs. Request redirection policies determine which peering Grid is selected to process a request. An IGG is also able to allocate resources from a Cloud provider, such as Amazon EC2.

Although applications can have resource management mechanisms of their own, we consider a case where the resources allocated by an application are used for the creation of a Distributed Virtual Environment (DVE), which is a network of virtual machines that runs isolated from other DVEs. Therefore, the allocation and management of the acquired resources is performed on behalf of the application by a component termed DVE Manager. The DVE Manager currently supports bag-of-tasks applications. In this work we develop an extension of this basic DVE.

## 3 Asynchronous-Iterative Computing

In this section, we describe our computational steering platform, which allows users to take control of asynchronous-iterative computations executing on grids. In the next sections, we describe the principles, the role of each entity, and the steering controller. Finally, we present the architecture and its implementation.

### 3.1 Principles

The aim of this framework is to provide tools that help users to interactively re-adjust the focus of their computations on some parts of the data. To achieve that goal, we propose *an asynchronous iterative map/reduce framework* that enables partial composable data analyses. The user can dynamically interact with her computation to refine some partitions of the data by doing more iterations, re-mapping partitions to clusters to obtain important results faster, *etc.* These operations allow the user to take advantage of grids for solving important problem instances, in multiple domains such as Astronomy.

The framework enables users to split data among clusters, evaluate partitions, iterate on these partitions, and dynamically re-organise or change the focus of the computation by receiving partial results asynchronously.

### 3.2 Entities

The proposed steering system is composed of four main components:

- *Controller*: is the entry point for steering. The controller allows the user to deploy and interact with her computation.
- *Collector*: controls a site, *i.e.*, a cluster. A collector is in charge of deploying the evaluators and iterators on each node, partitioning the data, and updating the controller asynchronously with new results.
- *Evaluator*: is a process that takes, as input, a partition of the data to evaluate; and returns as output, a result based on that partition. The evaluator is provided by the user.
- *Iterator*: is designed to locally process the result from an evaluator before the next iteration. For instance, the iterator can modify the local partition with the result that has been just calculated by the evaluator. The number of iterations on a partition is given by the user and can be dynamically updated. The iterator is also provided by the user.

Figure 1 shows the interactions between the different entities. The controller, on behalf of the user, splits the first set of data and submits the sub-sets to the collectors. Next, each controller splits the sub-set and sends a partition of work to the evaluators. The user provides the split function. Depending of the speed of the cluster, the user can set or dynamically modify the number of iterations on each partition. In parallel of the evaluation/iteration process, the controller collects partial results; the user via the controller can monitor and fetch these partial results. When the user starts collecting partial results, she can decide to reset the objective of the computation to other sub-parts of the data.

For instance, after a few iterations, she notices the partition $B$ looks important, thus she decides to stop the exploration of other partitions and re-split $B$ to focus the compu-

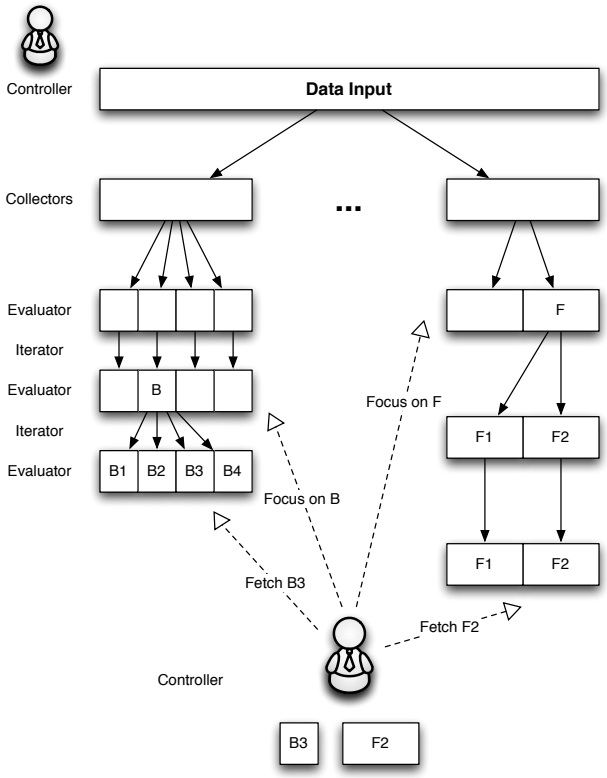


Figure 1: Architecture and data processing within the asynchronous-iterative computational framework.

tation on that partition. She can also re-map the data distribution from one cluster, *i.e.*, collector, to a faster one. In addition, the user can refine the computation of some partitions; for instance $F1$ and $F2$, the first iteration does not provide results with the expected quality, thus she decides to run one more iteration on those partitions to obtain better results.

### 3.3 The Steering Controller

The steering controller is designed as a set of command-line tools, hence users can combine and script these commands by using familiar tools such as UNIX-like terminals or Windows PowerShell. For the moment, we provide these commands as a set of low-level tools for composing applications, in the rest of this section we describe each of the commands.

In the code snippet below, we show how users can deploy and steer their computations with the proposed framework. In this example, the user requests for 50 nodes from a first site and 100 nodes from a second site, the nodes run each a VM containing Hadoop. That process is part of a previous work on InterGrid [10]. After InterGrid provides the resources and deploys the VMs, the user starts the framework with the command `dve_steering deploy`; when the framework is ready to serve computation, a job file is returned to the user, here `./job01.steering`, the file contains the information for interacting with the framework. This file is then used with all the other commands for identifying the computation. Next, the user submits the data to evaluate by using two commands: `submit` for deploying evaluators, iterators, and splitters; and `send` for submitting the data. Finally, she can check the status of the computation and fetch the results.

```
$ dve_sub -n 50 HADOOP_VM
Status = SCHEDULED
Job ID = 0001
$ dve_sub -n 100 HADOOP_VM
Status = SCHEDULED
Job ID = 0002
$ dve_steering deploy 0001 0002
Collector 01 deployed with 49 slaves
Collector 02 deployed with 99 slaves
Job saved in ./job01.steering
$ dve_steering -f ./job01.steering submit \
  /path/to_evaluator /path/to_splitter \
  /path/to_iterator
$ dve_steering -f ./job01.steering send \
  data_set_1 data-set_2 -iter 10
$dve_steering -f ./job01.steering  status
Partitions 1.[1,49] running
Partitions 2.[1,99] running
...
$dve_steering -f ./job01.steering fetch \
 1.35 2.[1,99] /target_folder
...
```

In addition to the commands presented in the previous example, the framework provides some others to stop the whole computation or a sub-part; and re-distribute the partitions between the clusters, *i.e.*, moving slower computations or promising data subsets to faster clusters. Finally, the set of commands is very flexible and allows the users to dynamically add or remove resources to/from their computations.

### 3.4 Architecture and Implementation

In this section we present how we use InterGrid to deploy the MapReduce model on multi-site grids. We also describe an extension of the DVE Manager for supporting computational data-oriented steering.

The main entry point of the architecture is the DVE Manager from the InterGrid platform. A DVE Manager interacts with the IGG by making requests for VMs and querying their status. The DVE Manager requests VMs from the gateway on behalf of the user application it represents. When the reservation starts, the DVE manager obtains the

list of requested VMs from the gateway. This list contains a tuple of public IP/private IP for each VM, which the DVE Manager uses to access the VMs. Then, the DVE Manager handles the deployment of the user's application. In the context of the steering application, the DVE is the main entry point for the user. The DVE Manager embeds the controller functionality and also provides a command-line interface. The command-line is implemented in Python.

As our framework relies on the MapReduce programming model, we use Hadoop [13], which is an open source implementation of MapReduce sponsored by Yahoo! and aims to be a general purpose distributed platform. With Hadoop, the problem space is stored as files contained in HDFS [13] and the optimum exploration problems are abstracted as analyzing files in HDFS by using map/reduce tasks.

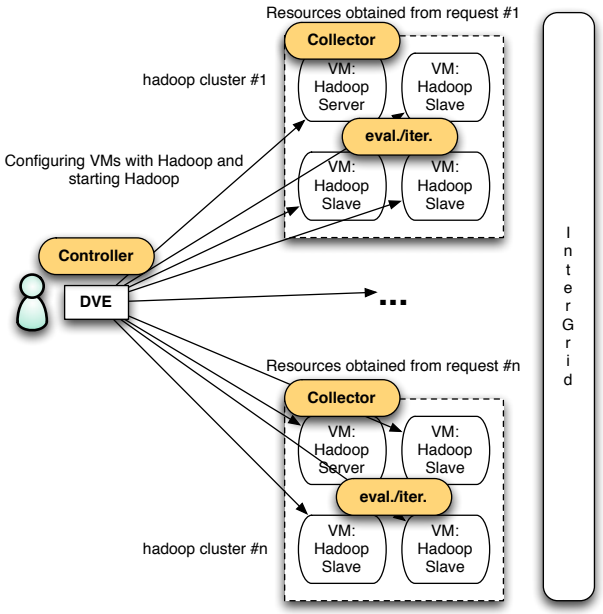


Figure 2: Architecture and implementation.

Figure 2 shows the architecture of the framework. The DVE interacts with the InterGrid layer to obtain and deploy VMs that are pre-configured with Hadoop. Next, the DVE configures on each site Hadoop and starts a collector on every site. The collector is the interface between the steering framework and Hadoop. The evaluator is the mapper operation, the reduce operation is not used in our case. The collector runs a wrapper, in Java, as a map task of Hadoop. This wrapper forks the evaluator and the iterator provided by the user. Both processes communicate by redirection of their standard input and output. The wrapper takes as parameter the number of iterations requested by the user. The collector is also in charge of splitting the partitions when requested by the user. The user provides the split command. The command takes as parameter the number of sub-partitions desired, reads on the standard input the data, and then return the sub-partitions as files.

To support computational steering from the architecture layer, the system needs to keep the intermediate status of computations and dynamically configure the load within the grid sites according to the interaction with users. We use a versioning system for helping the user to keep track of her computation. The versioning is based on the number of splits of a partition. After the first split each partition gets a major version and then it is incremented at each split by a dot and the sub-number of the partition.

## 4 Experiment and Evaluation

This section reports an experiment based on a scientific application in astronomy. This experiment aims to validate the framework and show its effectiveness.

### 4.1 A Typical Use Case with an Astronomer

Many scientific computations search for optimum values in a large multi-dimensional problem space, such as scientific model validation, engineering design, *etc.* These problems have to rely on an iterative style of optimisation through which the final optimum values are gradually explored. In particular, taking the results of prior steps as input, each step of iteration optimizes it and generates results for the next round of computation. During this series of computations, in comparison with totally automatic methods, computational steering after each iteration allows manual adjustment of search parameters or area of interest for the next round of optimization, which could accelerate convergence and prevent wasted computational cycles, resulting in more efficient overall optimization.

To validate the proposed framework we consider an *astronomical data analysis* framework. Using data from the Sloan Digital Sky Survey [2], we can validate models of galaxy structure and evolution by comparing their predictions to the actual observed data. This maximum likelihood estimation process can use genetic search, where population individuals are different parameter sets in the model, fitness is the quality of the model according to maximum likelihood, and newer generations represent models that more likely explain the observed data. Here, partial results are useful, we can generate new individuals even if all the previous generation of individuals has not been evaluated. Composability and incremental behavior can be done stochastically: with certain probability a new deserving individual is either mutated or reproduced. Such asynchronous genetic search eliminates the notion of generations and instead keeps a single incrementally updated population that

Table 1: Performance of the deployment and execution of MilkyWay application. All values are times in seconds.

| Sites | Nodes by Clusters | Configuration | Sending App. | Sending Data | Runtime |
|---|---|---|---|---|---|
| 1 site - 50 nodes: | - | 41.38 | 0.32 | 396.88 | 2227.99 |
| Orsay - 50 | 21 gdx, 29 netgdx | | | | |
| 1 site - 100 nodes: | - | 86.86 | 0.19 | 420.02 | 1059.16 |
| Orsay - 100 | 71 gdx, 29 netgdx | | | | |
| 1 site - 200 nodes: | - | 209.74 | 0.31 | 388.55 | 587.76 |
| Orsay - 200 | 171 gdx, 29 netgdx | | | | |
| 2 sites - 100 nodes: | - | 135.44 | 1.17 | 304.79 | 1275.64 |
| Orsay - 50 | 21 gdx, 29 netgdx | | | | |
| Rennes - 50 | 50 paraquad | | | | |
| 2 sites - 200 nodes: | - | 212.14 | 1.01 | 341.53 | 668.57 |
| Orsay -100 | 72 gdx, 28 netgdx | | | | |
| Rennes -100 | 63 paraquad, 33 paramount, 4 paradent | | | | |
| 2 sites - 300 nodes: | - | 308.91 | 1.16 | 308.44 | 611.55 |
| Orsay - 200 | 171 gdx, 29 netgdx | | | | |
| Rennes - 100 | 63 paraquad, 33 paramount, 4 paradent | | | | |

still converges to the most interesting models, even in the presence of failures and heterogeneity [9].

In this work, we adapt the genetic algorithm application to our computational steering framework. The MilkyWay application developed for BOINC [4], has a master-worker architecture. We use the work-unit as the evaluator and we have adapted some of the genetic algorithms of the server part as the splitter and iterator. There are many more applications that can benefit from our framework, such as global warming models.

## 4.2 Experiments

This experiment uses the French experimental Grid platform, Grid'5000 [6], as the test bed. Grid'5000 is a scientific instrument for the study of large scale parallel and distributed systems. It aims at providing a highly reconfigurable, controlable and monitorable experimental platform to its users. Now it is composed of nine sites geographically distributed in France, providing a total of 14 clusters.

Our experiment used up to 300 machines located in 2 different sites in France. During the experiment, only one controller was deployed and each site was assigned one collector. Each site may provide machines from different clusters and therefore the configuration of machines has big differences. In particular, the model of machines located in Orsay is IBM eServer and most of them are equipped with one AMD Operon 2GHz CPU and 2GB memory, while the network connection is using PCI-X Gigabit Ethernet. The machines in Rennes consist of HP, Sun and Dell Servers and the size of memory on each machine varies between 2GB and 32GB, while the network consists of Gigabit Ethernet, Myri-10Gb and InfiniBand 10Gb. The network connecting different sites is 10Gb/s optical fibre. Our experiment run over virtual machines. Therefore, although the heterogeneity of physical machines is big, we were using the same operating system, Debian GNU/Linux 5.0.

With the experiment, we aim to show two features of our system: 1) the simplicity and relative small overhead of deploying scientific applications in a large scale distributed environment with computational steering; 2) the performance scalability of our framework with the Milkyway application.

Table 1 summarizes the performance results of our experiment under a varied number of machines. The overhead of conducting the Milkway computation by using Hadoop over clusters of VMs mainly consists of 4 parts. These are *Configuration*, *Sending App.*, *Sending Data* and *Runtime* respectively in Table 1. *Configuration* represents the time required to set up each cluster with Hadoop and it includes editing the configuration file and starting HDFS and Hadoop daemons on each machine. *Sending App.* includes the time to send the files required by the Milkyway applications to each site, i.e., the master node in each Hadoop cluster. In our case, the size of all the MilkyWay files is about 1.7MB, including collector, evaluator and iterator. *Sending Data* is the time used to send input data to each cluster. In this case, it means to put files into HDFS. During the experiment, each Milkyway execution took 600 files as input, the total size of which was 1,860 MB. *Runtime* presents the time to execute the Milkyway application over the varied number of resources. The time required to boot VMs is not covered in our work, since it is a one-time-only cost.

In Table 1, the time consumed by *Configuration* increases as more resources are added into the computation. The reason is that Hadoop was configured and started on each cluster sequentially. Therefore, more clusters and more resources increase the overhead required by *Configuration*. We will optimize and parallelize the *Configuration* phase in the future. Fortunately, it just incurs a small percentage of the entire execution of Milkyway. This comparatively small overhead allows a convenient access to our

framework.

The time required to send Milkyway files to each sites is considerably small and it can even be ignored in comparison with the execution time. The reason is the total size of Milkyway files is very small, just 1.7MB. However, in contrast, the time to send 600 input files with the size of 1,860 MB is comparatively larger, between 300 and more than 400 seconds.

The final column in Table 1 presents the performance scalability of Milkyway over our framework. The entire execution time of Milkyway decreases as more resources are added into the computation. Due to the heterogeneity between machines from different sites and clusters, the performance does not increase linearly. However it is still consistent with our expectations, except the performance result over 300 machines from 2 sites. It is because currently Hadoop does not tolerate big heterogeneity very well [13] and the communication between more clusters incurs additional overhead.

## 5 Related Work

Computational steering is an important feature required by many scientific applications. Petra Wenisch *et al.* [20] work on a project which supports computational steering on a large scale of computational fluid dynamics (CFD) applications over super-computers. This work is specialized for MPI-based (Message Passing Interface) applications. D-Grid [19] provides a run-time environment that monitors the execution status of High Energy Physics jobs and supports a steering system, called RMOST (Result Monitoring and Online Steering Tool). With RMOST, users can connect to the running job, optimize its parameters, control its correctness and investigate unexpected results online. Shenfield *et al.* [3] works on a set of APIs and runtime environments for engineering design problems implemented by using evolutionary algorithms. Different from the above computational steering frameworks designed for special applications, RealityGrid [23] [5] aims to provide a set of APIs and run-time environment for more general steerable applications, especially legacy scientific applications. However, users have to change their applications in order to use RealityGrid.

In comparison, our work aims to provide a transparent framework for iterative applications and therefore no significant changes are required to use the computational features provided. Furthermore, our asynchronous-iterative framework tries to provide a cooperative run-time environment by harnessing the computing capabilities from multiple sites in grids. By using Hadoop, an open source implementation of MapReduce, the implementation difficulties, such as fault tolerance and parallelism, are significantly decreased. MapReduce [15] has been used by Google for web search applications, which selects the results best suited to the users' requirements within billions of web pages. Many scientific problems also can be taken as searching best results from various problem spaces with a large scale. For example, bioinformatic scientists have to seek clues from DNA sequences, while the Milkyway application is exploring all the possibilities to discover the best suitable galaxy model. With the aim of exploring the high level similarities between scientific and web search applications, DISC [21] starts to work on suitable programming models for data-intensive computational problems by using MapReduce. MRPSO [1] utilizes the MapReduce model to parallelize a compute-intensive application, Particle Swarm Optimization, while MRPGA extends the MapReduce model to support general genetic algorithms [8]. Our work tries to benefit from the advantages of MapReduce to build a framework for computational steering over Grid environments.

Hadoop On Demand (HOD) [14] is a system for provisioning virtual Hadoop clusters over a large physical cluster. It uses the Torque resource manager to do node allocation. On the allocated nodes, it can start Hadoop MapReduce and HDFS daemons. It automatically generates the appropriate configuration files for the Hadoop daemons and client. HOD also has the capability to distribute Hadoop to the nodes in the virtual cluster that it allocates. In short, HOD makes it easy for administrators and users to quickly set up and use Hadoop. It is also a very useful tool for Hadoop developers and testers who need to share a physical cluster for testing their own Hadoop versions. However, HOD works only with Torque. In comparison, our framework has no special requirements to automatically deploy Hadoop. We only require SSH (Secure Shell), which is now within the standard release of most Unix-based systems.

Grid'BnB [7] is a parallel branch and bound framework for grids. Branch and Bound (B&B) algorithms find optimal solutions of search problems and NP-hard optimisation problems. Grid'BnB is a Java framework that helps programmers to distribute problems over grids by hiding distribution issues. It is built over a master-worker approach and provides a transparent communication system among tasks. This work also introduces a mechanism to localize computational nodes on the deployed grid. This mechanism is used in Grid'BnB to reduce inter-cluster communications.

## 6 Conclusions

In this work, we proposed an asynchronous-iterative computation framework which enables users to perform computational steering. The architecture of the framework is based on previous work, InterGrid, that enables multi-sites grid use and the MapReduce programming model. Computational steering is enabled by a set of command-line tools that allows users to interact with their computations. We demonstrate how users can adapt a representative

master-worker application to our framework.

The experiment was conducted using Grid'5000 by running the Milkyway application from our framework over up to 300 physical machines in 2 sites. It proves that our framework is able to provide a considerably small overhead and scalable performance for applications following the master-worker model. Therefore, it is a good choice for steering scientific computations in Grid environments.

In future work, we would like to provide fault-tolerance, which is not fully considered. However, users can indirectly detect failures by seeing no advance in sub-computations' status and can decide to move the failed data partition to another cluster. We also would like to provide more tools for helping users to analyze their computations, such as visual-based computational steering. Finally, we want to investigate more real-world applications that can benefit from our computational steering framework.

## 7 Acknowledgements

This work is partially supported by research grants from the Australian Research Council (ARC) and Australian Department of Innovation, Industry, Science and Research (DIISR). It is also partially supported by U.S.A. NSF CAREER CNS Grant No: 0448407. Some experiments were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER, several universities, and other funding bodies (see https://www.grid5000.fr).

## References

[1] A. W. McNabb, C. K. Monson, and K. D. Seppi. Parallel PSO Using MapReduce. In *Proc. of the Congress on Evolutionary Computation*, 2007.

[2] J. Adelman-McCarthy, M. Agüeros, S. Allam, C. Prieto, K. Anderson, S. Anderson, J. Annis, N. Bahcall, C. Bailer-Jones, I. Baldry, et al. The sixth data release of the sloan digital sky survey. *Urbana*, 51:61801.

[3] M. A. Alex Shenfield, Peter J. Fleming. Computational steering of a multi-objective evolutionary algorithm for engineering design. *Engineering Applications of Artificial Intelligence*, 20-8:1047–1057, 2007.

[4] D. P. Anderson. Boinc: A system for public-resource computing and storage. *Fifth IEEE/ACM International Workshop on Grid Computing*, 0:4–10, November 2004.

[5] J. Brooke, P. Coveney, J. Harting, S. Jha, S. Pickles, R. Pinning, and A. Porter. Computational steering in RealityGrid. In *Proc. of the UK e-Science All Hands Meeting, September*, volume 2, pages 885–889. Citeseer, 2003.

[6] F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, and O. Richard. Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In *Grid'2005 Workshop*, Seattle, USA, November 13-14 2005. IEEE/ACM.

[7] D. Caromel, A. di Costanzo, L. Baduel, and S. Matsuoka. Grid'bnb : A parallel branch and bound framework for grids. In S. Aluru, M. Parashar, R. Badrinath, and V. K. Prasanna, editors, *HiPC*, volume 4873 of *Lecture Notes in Computer Science*, pages 566–579. Springer, 2007.

[8] Chao Jin, Christian Vecchiola, Rajkumar Buyya. MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms. In *Proc. of 4th IEEE International Conference on e-Science*, 2008.

[9] T. Desell, B. Szymanski, and C. A. Varela. An asynchronous hybrid genetic-simplex search for modeling the milky way galaxy using volunteer computing. In *Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 921–928, Atlanta, Georgia, July 2008.

[10] A. di Costanzo, M. D. de Assunção, and R. Buyya. Building a virtualized distributed computing infrastructure by harnessing grid and cloud technologies. *Internet Computing, IEEE*, Sept-Oct 2009. To appear.

[11] EGEE. http://public.eu-egee.org/, 2004.

[12] J. Ekanayake, S. Pallickara, and G. Fox. Map-Reduce for Scientific Applications. In *Proc. of the 4th IEEE International Conference on e-Science.*, 2008.

[13] Hadoop. http://hadoop.apache.org.

[14] Hadoop on demand. http://hadoop.apache.org/core/docs/current/hod_user_guide.html.

[15] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of the 6th Symposium on Operating System Design and Implementation*, 2004.

[16] R. v. L. Jurriaan D. Mulder, Jarke J. van Wijk. A survey of computational steering environments. *Future Generation Computer Systems*, 15-1:119–129, 1999.

[17] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the Grids. *Scientific Programming*, 13(4):265–275, 2006.

[18] A. Matsunaga, M. Tsugawa, and J. Fortes. Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *Proceedings of the Fourth IEEE International Conference on eScience*, pages 222–229, USA, 2008.

[19] R. Müller-Pfefferkorn et al. User-centric monitoring and steering of the execution of large job sets. In *Proc. of German e-Science Conference*. Citeseer, 2007.

[20] O. W. Petra Wenisch and E. Rank. Harnessing high-performance computers for computational steering. *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science*, 3666/2005:536–543, 2005.

[21] R. E. Bryant. Data-Intensive Supercomputing: The Case for DISC. In *CMU-CS-07-128, Technical Report, Carnegie Mellon University*, 2007.

[22] L. Ramakrishnan, D. Irwin, L. Grit, A. Yumerefendi, A. Iamnitchi, and J. Chase. Toward a doctrine of containment: Grid hosting with adaptive resource control. In *2006 ACM/IEEE Conference on Supercomputing*, page 101.

[23] Shantenu Jha, Stephen Pickles, Andrew Porter. A Computational Steering API for Scientific Grid Applications: Design, Implementation and Lessons. In *Proc. of Grid Application Programming Interfaces Workshop*, 2004.