CSBase: 10 years of experience with a framework for batch processing in Heterogeneous Computing Environments

Maria Julia de Lima[1], **Cristina Ururahy**[1],
Isabella Almeida[1], **Andre Clinio**[1], **Noemi Rodriguez**[2]

[1]Tecgraf Institute, PUC-Rio
Rio de Janeiro – RJ – Brazil

[2]Department of Informatics, PUC-Rio
Rio de Janeiro – RJ – Brazil

{mjulia,ururahy,isabella,clinio}@tecgraf.puc-rio.br

noemi@inf.puc-rio.br

Abstract. *For over ten years, the Tecgraf Institute at PUC-Rio has been developing and maintaining different systems for HPC batch submission at Petrobras, the Brazilian oil company. Most of these systems have been built using the CSBase framework that allows the reuse of common facilities in similar projects. CSBase offers a cooperative and integrated user environment to abstract the use of computational resources. Users can share data files and executable programs which are used to submit jobs over dedicated HPC resources. In this paper, we describe the CSBase framework, discussing the different alternatives it offers for integration and extensions, and how these alternatives have been successfully explored in the existing CSBase installations.*

Resumo. *Nos últimos dez anos, o Instituto Tecgraf da PUC-Rio vem desenvolvendo e mantendo diferentes sistemas para gerência e submissão de programas em ambientes HPC na Petrobras, empresa petrolífera brasileira. A maioria desses sistemas foi construída utilizando o framework CSBase, o que permitiu o reuso de funcionalidades em projetos similares. O CSBase oferece um ambiente de integração e cooperação para seus usuários fazerem uso transparente de recursos computacionais heterogênos e distribuídos. Em sistemas CSBase, usuários compartilham dados e programas para execução de tarefas em máquinas HPC dedicadas. Neste trabalho, descrevemos o framework CSBase e discutimos como suas facilidades de extensibilidade e integração tem sido utilizadas com sucesso em diferentes cenários.*

1. Introduction

For over ten years, the Tecgraf Institute at PUC-Rio has been developing and maintaining different systems for HPC batch submission at Petrobras, the Brazilian oil company. In 2001, the seismic analysis group at Petrobras detected the need for a system to control batch submission of tasks to a collection of clusters located within the company. In 2002 Tecgraf started working on a system that was to be called WebSintesi. This system offered facilities for resource management and remote execution in a distributed and heterogeneous computing environment (HCE). WebSintesi was built using a project-centric

data model: files are organized in project areas keeping the cooperation between users centralized over the same work space.

An year later, in 2003, another project required similar HCE facilities, but involving a different application domain. This motivated Tecgraf to build the CSBase framework [de Lima et al. 2006], so as to allow reuse of common facilities in similar projects. Each instance of CSBase has its own security model and sets of services and applications.

This refactoring of the HCE management system into a framework proved itself quite useful over the following years. Five instances of CSBase are now in use at Petrobras and a sixth one, CSGrid, is offered as a generic HCE middleware, freely available for download[1]. CSGrid is currently adopted as the back-end for SINAPAD (High Performance Processing National Center) in the construction of e-science portals for different domains[Gomes et al. 2014]. The other five instances in use at the Brazilian Oil Company encompass systems with around 750 users.

Over time, the integration of new applications to CSBase systems has motivated different types of interfaces for building these integration scenarios. The CSBase programming interfaces mainly differ with respect to the architecture layer on which applications depend. Some applications are integrated into the Virtual Desktop provided for user interaction with CSBase systems. Other applications use CSBase in an service-oriented model, through published interfaces in a language- and platform-independent way. The two current sets of programming interfaces are the result of experience in building these integration scenarios over the past years and have evolved to meet the different needs arising from new usages of CSBase framework.

In this paper, we describe the CSBase framework, discussing the different alternatives it offers for integration and reuse, and how these alternatives have been successfully explored in different CSBase instances. Section 2 describes the main features of the CSBase framework. Section 3 presents the CSBase architecture, with focus on its facilities for the construction of clients applications and integration with existing systems. Section 4 analyses different scenarios of CSBase usage, discussing in each case how the framework was specialized and integrated with new applications. Section 5 discusses some related work, and finally Section 6 contains our final remarks.

2. The CSBase Framework

The CSBase framework offers support for building collaborative submission systems that abstract the use of heterogeneous computing resources. The framework is targeted at batch submission of non-interactive programs previously uploaded and shared by the users. These programs — called *algorithms* in CSBase jargon — are composed of executable and configuration files and CSBase systems typically create the associated command to execute them dynamically, using parameters provided by the user. Executable programs, like all user data, are stored in server repositories and are under permission-related access restrictions. CSBase supports project-centric environments, allowing input and output data files to be organized in areas that are shared by all users affiliated to a project.

A typical CSBase system abstracts access to a number of different execution hosts,

[1] http://www.tecgraf.puc-rio.br/csgrid

possibly encompassing clusters as well as individual machines. Each execution host is associated with an SGA component that is responsible for the submission, control, and monitoring of its jobs. This component is also responsible for gathering information on the hosts' available resources. SGA is part of CSBase architecture and is presented in Section 3.4.

In the next subsections, we discuss some aspects of the CSBase framework in more detail.

2.1. Algorithm Management

CSBase supports a work model in which users repeatedly work on a set of programs, checking and comparing results for different input data sets. The algorithm repository is thus a crucial part of the system, and is maintained in a separate area from that of projects. The algorithm repository maintains information about available versions, execution requirements, executable files necessary for different platforms.

CSBase also maintains a configuration file for each algorithm in the repository. A configuration file is written in an XML-based notation and describes the data to be used as input in the execution of the algorithm as well as the output files it will generate. This information is used when CSBase builds — dynamically — the command that is to be submitted for remote execution. When the data is contained in files, names and paths always refer to files in the project to which a specific submission is associated.

CSBase supports different user roles. Some users are responsible for the development of programs while others just need to execute them without knowing which executable and configuration files constitute these programs. The algorithm repository is managed by specific users who have permissions for writing the repository, creating and modifying program versions, and uploading executable files and configuration files that define in/out parameters for program execution. Whenever a new program is installed, it is dynamically integrated to the environment and is made available to those users with permission to execute it; no rebuilds or restarts are required.

2.2. Project-centric Environment

CSBase's usage model is based on the *project* abstraction. Each project represents a data area, organized as a hierarchical structure containing input and output data files. Users can typically create their own projects and configurations. Every project is owned by the user who created it, which has total control over access rights over all files in the project. CSBase allows users to share their projects with other users, defining collaboration contexts. All data files needed as input to program submission should be previously available in an associated project area. Submitted programs also write output files in project area.

This project-centric approach aims at supporting data-intensive operations in which data is moved as little as possible. User data is a critical resource and should be organized and shared between users who work in a collaborative way. Also, different executions of a single program can be associated with a common project, allowing different simulations to run in parallel on the same project input files, and the different results to be organized in a single area.

Users are organized in groups and permissions on different projects can be assigned at the level either of individual users or of groups.

2.3. Submission, Monitoring and Control of Jobs

CSBase allows users to submit a job to the execution hosts in the current installation. In CSBase, a job is always an execution request for a pre-installed algorithm over which the user must have the appropriate permissions. Users can specify different resource requirements for the execution of the job, such as amount of memory, CPU availability, etc.

When the user submits a job, it is placed in a single priority queue. Jobs are not necessarily run in the order in which they are submitted. They are scheduled according to their requirements and to the resource information gathered from the execution hosts. To select the execution host, CSBase also takes into account the platforms for which there are executable versions of the requested algorithm in the CSBase repository.

After selecting the execution host, CSBase builds the execution command itself, using the algorithm configuration file. This command is a `ksh` script containing all the steps to set up the appropriate execution environment. To build this script, CSBase may need some platform-specific information, such as the syntax for file paths.

Before submitting the command for execution on the selected SGA, CSBase orchestrates the file staging required, i.e., data and executable files. If the CSBase server and execution host are on the same network, the repositories files can be shared through NFS (network file system). If the execution host does not have access to the server's repositories, CSBase uses a proprietary mechanism, CSFS [Santos and Cerqueira 2006], to transfer files. CSBase also provides flexibility to extend the server with new data transfer mechanisms.

Finally, the execution script is transmitted to the selected SGA. The SGA then becomes responsible for dispatching and monitoring its execution, using the platform-specific code in the `sgasubmit` library discussed in Section 3.4. For instance, in the case of the PBS-compliant SGA, this platform-specific code will invoke the PBS library to enqueue the script for execution.

3. The CSBase Architecture

The main components of the CSBase architecture are illustrated in Figure 1. There are other architecture configuration alternatives, for instance, with a central CSBase server maintaining a global administration data repository and multiple secondary servers, geographically distributed. At current installation environments, however, a CSBase server typically runs on a single machine and is responsible for controlling all the resources available to its clients. These resources include the data repositories and the execution hosts.

The CSBase Server is organized as a set of services, mostly written in Java, that implement the functionality described in the previous section. The *Algorithm Management Service* maintains the repository of uploaded programs. The *Project Service* manages projects and their associated data. A *Monitoring Service* maintains state information about remote nodes and executing jobs. The *Submission Service* takes care of remote submission of jobs, maintaining a job queue that it continually visits to create the next submission request.

In most scenarios, these services are accessed through RMI, from a web-launched

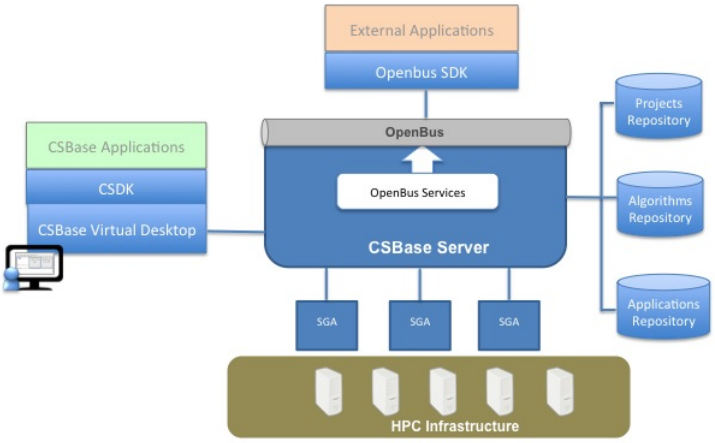

Figure 1. CSBase Architecture Components

Java client, called the *CSBase Virtual Desktop*. The web-launched Virtual Desktop is an integral part of the CSBase distribution and, in its basic form, offers facilities for managing the algorithm repository, uploading and downloading data files, submitting execution requests, and monitoring machines and executions. These facilities are subject to user permissions. The CSBase Virtual Desktop provides an administration interface for creating those permissions and assigning them to individual users or user groups.

Soon after the web-launched Virtual Desktop was first deployed, more than 10 years ago, users started requesting extensions, such as specific visualization facilities. Originally, support for the construction of these extensions — or new desktop *applications* — was very limited: the CSBase server kept a registry with available applications, but all application classes had to be distributed along with the Virtual Desktop code. This tight coupling limited the flexibility of the system, allowing new desktop applications to be added only when new releases were launched. The CSBase team then decided to loosen this coupling between desktop applications and the CSBase server, and developed the CSDK toolkit [Clinio and Almeida 2015], which provides well-defined interfaces for the construction of domain-specific applications.

Alongside this evolution of the support for the development of domain-specific CSBase applications, a different class of client applications appeared. Many existing systems, not coded in Java, needed to be integrated with a CSBase system in order to handle user data and submit jobs in its environment. A CORBA API wrapper was built around some of the CSBase services to facilitate this integration. The first use of these wrappers was for the integration of V3O2 and WebSintesi. V3O2 is C++ system developed for Petrobras for the visualization and interpretation of seismic data. The V3O2 system was adapted to programmatically consume the CSBase service that gives access to user's project data using the first CORBA CSBase service. Later, new CSBase CORBA services were developed over the core funcionalities. Today, the CSBase CORBA services are being used for several systems from different domains, such as geomechanics and supply chains. The deployment of these services is facilitated by the use of the *OpenBus*[2] CORBA bus service.

[2]http://www.tecgraf.puc-rio.br/OpenBus

SGA components integrate the CSBase Server with the heterogeneous computing environment available for batch submission. The SGA component was initially developed as a daemon for single execution hosts running different Unix flavors or Windows. It is written in Lua[Ierusalimschy 2013] and C++. Today, the SGA has its own API to allow developers to integrate third-party Resource Management Systems to the CSBase environment. Currently, there are SGA gateways implementations for the Sun Grid Engine [Gentzsch 2001], TORQUE/PBS [Staples 2006], OurGrid [Brasileiro et al. 2007] and SLURM [Jette et al. 2002].

3.1. Virtual Desktop

Since its beginnings, CSBase is deployed as a built-in client that runs at the user's machine and abstracts the complexities of the computational environment. The Virtual Desktop allows several users located on different sites to cooperate through a project workspace. Depending on the user's permissions, the Virtual Desktop presents the front-end with different operations and there are also specific customization each user can apply based on its own preferences. By supporting a set of controlled predefined operations the system guarantees access control and isolation of the workspaces and makes the system easier to use.

The Virtual Desktop consists of a Java Web Start application that is automatically downloaded and started when the system's URL is accessed. When started, it connects to the CSBase server and authenticates the user. The communication between the Virtual Desktop and the CSBase server is implemented through the Java RMI protocol.

Figure 2 shows a typical Virtual Desktop. On the left is a hierarchical tree representing an open project, similar to a remote file system. On the right, the applications installed at the CSBase application repository are shown and can be launched. At the bottom, there is a text area used to communicate with other users and to notify system events, such as to indicate the completion of job submission.

The Virtual Desktop allows users to request the execution of a selected algorithm on a execution host. Based on the algorithm configuration, CSBase builds the user interface through which the algorithm's parameters are provided, with no need for recompilation. Figure 3 shows the algorithm execution interface dynamically built when the user select an algorithm available. When the execution of an algorithm ends, CSBase notifies the user who requested it. Algorithm monitoring facilities also allow users to monitor an algorithm execution and, if needed, to interrupt it. Users can also view the execution history of all jobs they have submitted. For each job, related information is displayed such as run times, resources (CPU and memory) consumed, log and input parameters provided to the execution.

Another utility allow users to monitor the execution hosts giving an overview of active machines and their properties, such as CPU and memory available. Based on the presented information, the user can select an adequate machine for the execution of an algorithm. This selection can also be delegated to the system, which will base its decision on the monitoring information gathered from the execution hosts.

There are also facilities to the users who have permission to manage the algorithm repository. These users typically need to install and test a new algorithm version, before

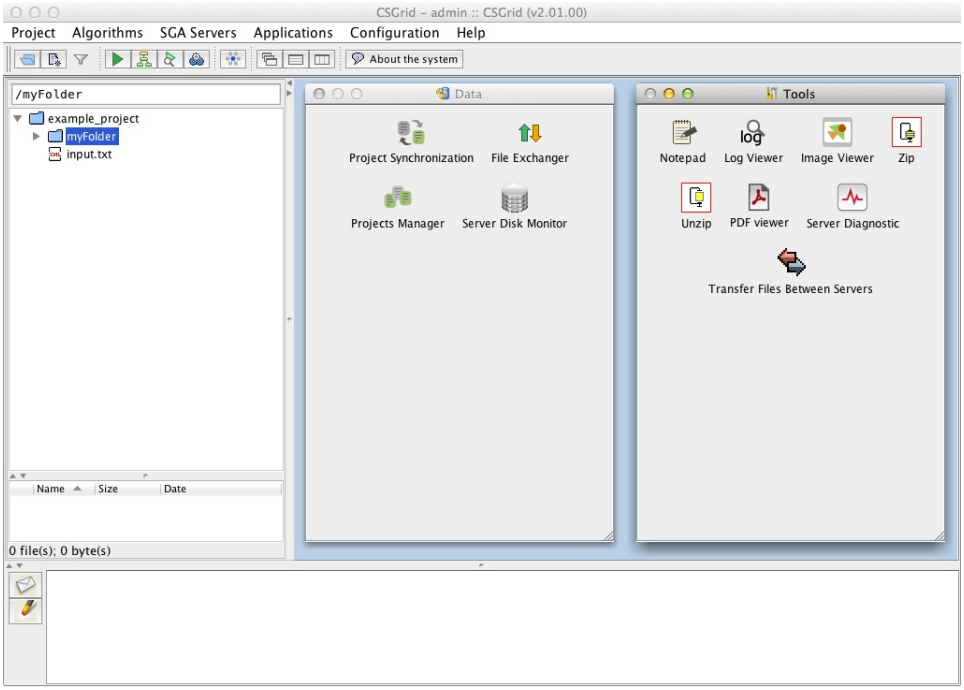

Figure 2. The CSBase Virtual Desktop

making it available to other users. When a user installs a new algorithm, it is automatically integrated to the environment.

The Virtual Desktop has evolved over the last years to be enriched with new CSBase functionalities. One of them is the better support for developing the applications deployed to the environment, subject of next section about the CSDK.

3.2. CSDK

CSDK (CSBase Development Kit) is a development environment for building extensions (desktop applications) to the CSBase Virtual Desktop. This is a Java-based environment and the interface to CSBase services is through remote method invocations. The Virtual Desktop is used by most CSBase systems and many of them customize its appearance or

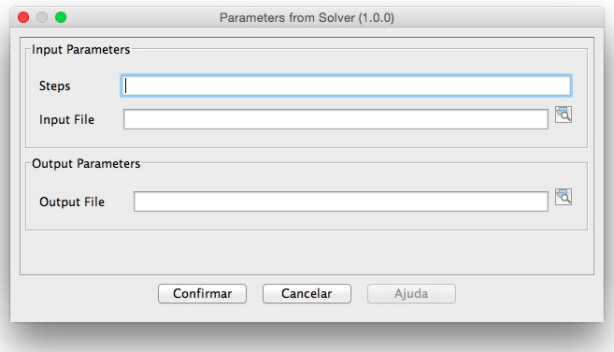

Figure 3. Example of the algorithm execution utility

extend its behaviour with new domain-specific applications developed with CSDK. The toolkit also offers a *runner* environment, which simulates the final CSBase environment in a local context, facilitating tests.

Besides programming facilities, the development kit allows the system administrator to update all installed applications independently from each other and from release schedules. It is only necessary to install the new application (a JAR package) in the CSBase *application repository*. When the desktop is launched, all applications are available and the user can select any of them by clicking on the appropriate icon. When the user clicks on an application icon, the desktop downloads the corresponding JAR package to local cache. A dynamic class loader written specifically for CSDK applications then loads the package, guaranteeing isolation between the new application's classes and those in the original Virtual Desktop.

Two sets of interfaces are defined in CSDK: an *application interface*, which the application developer must implement, and an environment interface, which encapsulates access to all available CSBase resources. The application interface includes methods such as `onApplicationStart` and `onMessageReceived`, allowing the developer to program the application's response to events generated by the execution environment. The environment interface is a single entry point through which the application can retrieve a set of *context* interfaces for access to CSBase services. CSDK contains a core set of contexts, including interfaces for tasks such as file handling and program execution. Other context interfaces may be added by specific environments, allowing applications to explore extensions.

The code below illustrates the use of CSDK. Class `CommandExecutionExample` implements a simple application interface. Its constructor receives a reference to an environment interface and stores it in a local variable. In this case, method `onApplicationStart` retrieves this reference and uses it to submit an execution request.

```
public class CommandExecutionExample implements IApplication {
  private final ICSDKEnvironment csdkInterface;
  public CommandExecutionExample(final ICSDKEnvironment csdkInterface) {
    this.csdkInterface = csdkInterface;
  }
  public void onApplicationStart() throws ApplicationException {
    IApplicationContext appContext =
       csdkInterface.getContext(ICommandContext.class);
    Map<String, String> parameterValues = new HashMap<String, String>();
    parameterValues.put("STEPS", "5");
    parameterValues.put("INPUT_FILE", "input.txt");
    parameterValues.put("OUTPUT_FILE", "output.txt");
    String algorithmName = "Solver";
    String algorithmVersion = "1.0.0";
    IAlgorithmTemplate info =
      new AlgorithmTemplate(algorithmName,
                            algorithmVersion,
                            parameterValues);
    String commandDescription = "Testing the Solver execution";
    commandContext.executeAlgorithm(info, commandDescription,
      null, this.mainFrame, new ICommandObserver {
        public void onCommandEnd(final ICommandInfo cmdInfo) {
          SwingUtilities.invokeLater(new Runnable() {
          public void run() {
            String message = "The job has exited with status " +
                             cmdInfo.getFinalizationType();
            JOptionPane.showMessageDialog(mainFrame, message);
          }
```

```
        });
      }
    );
    }
  }
}
```

The environment abstraction is the basis for the construction of the CSDK simulator. When applications are connected to the *runner* environment, all context interfaces implement access to local resources, maintaining the same interface as the remote CSBase services.

3.3. OpenBus Services

OpenBus [Maia and Roenick 2015] is a CORBA-based middleware used to integrate multi-platform and multi-language systems based on a service-oriented architecture. It provides a service directory, access control mechanisms and peer-to-peer communication. External applications can use the OpenBus development kit (OpenBus SDK) to authenticate, publish, discover and access services components, based on policies configured by the OpenBus administrator. The CSBase server publishes its services as OpenBus components, allowing CSBase clients to be written in a number of programming languages and operating systems.

OpenBus is built over a component model that provides dynamic features and enforces a compositional approach. Components exhibit interfaces with *facets* and *receptacles*, ans systems are assembled by connecting facets to receptacles. Facets describe point-to-point operations that are provided by a component to be invoked by other components. Receptacles indicate a dependency of a component on an operation provided by another component. In the OpenBus component model, facets and receptacles are defined as named CORBA IDL interfaces.

The CORBA CSBase Services were developed based on the OpenBus component model. Each CSBase system defines the facets and receptacles of its components that are to be published to the OpenBus middleware used by its clients. This enables a single CSBase system to support different versions of the same component facets. Moreover, new facets can be easily added to the CSBase system, extending existing components.

The OpenBus middleware provides governance over the component facets and receptacles of CSBase systems, allowing the adoption of deployment policies that are independent of the CSBase systems themselves. For authentication purpose, OpenBus also ensures that all CSBase service requests come from trusted authenticated users.

Currently, the CSBase server provides the following components through the OpenBus middleware:

- OpenDreams (Open Distributed Resource and Algorithms Management Service): offers a set of operations for the submission, monitoring and control of jobs on remote execution nodes, and for monitoring the execution nodes themselves. It also provides access to algorithm configurations, abstracting parameters and requirements. The OpenDreams component is based on OGF's DRMAA 1.0 specification [Forum 2008].
- Hierarchical Data Service: offers a set of operations for accessing and manipulating files in the user's project area. Each project is organized as a hierarchical

structure that stores users' files. It provides mechanisms to efficiently read and write unstructured data from and to project files. Also, structured views can be added to represent specific type of data in such a flexible way.

The following Java example illustrates the use of OpenDreams API to request execution of version 1.0 of the algorithm called *Solver*, previously installed on the target CSBase system. The call to `findOpenDreams` encapsulates the establishment of a connection to OpenBus. The algorithm configurator requires one integer value and one file as inputs, and creates as its output one single file. The file specified as input parameter should already be available in the *example_project* project that was used to establish a session with the OpenDreams service. The Hierarchical Data Service can be used to read and write the files to the project and all requests are authenticated using the OpenBus middleware.

```
IOpenDreams opendreams = findOpenDreams();
Session session = null;
session = opendreams.getSession("example_project"));
session.init("");
OpenDreamsJobTemplate jt = (OpenDreamsJobTemplate)
  session.createJobTemplate();
jt.args = new String[] { "-name", "Solver", "-version", "1.0.0" };
jt.jobDescription = "Testing the Solver execution";
jt.jobParameters = new String[][] {
   { "STEPS", "5" },
   { "INPUT_FILE", "input.txt" },
   { "OUTPUT_FILE", "output.txt" }
};
jt.email = new String[] { "user@tecgraf.puc-rio.br"};
String jobName = session.runJob(jt);
OpenDreamsJobInfo jobInfo = (OpenDreamsJobInfo) session
           ._wait(jobName, Session.TIMEOUT_WAIT_FOREVER);
System.out.println("The job has exited with status " +
  jobInfo.exitStatus);
session.exit();
```

3.4. SGA

An *SGA* (originally, an acronym for *Algorithm Management System* in Portuguese) running on a host or cluster-manager node provides facilities for remote submission and exchange of state information. Upon activation, an SGA registers itself with the CSBase Server.

The SGA component communicates with the CSBase server through CORBA, with method calls in both directions. The component invokes the server when registering and in other interactions, like when updating information on resource usage. The server invokes the SGA for job submission and control and for gathering information about the execution environment.

The SGA is flexible enough to be instantiated in different scenarios. An SGA can be deployed as a daemon in host machines to provide an execution interface to each individual host, but it can also be set as a gateway to third-party Resource Management Systems. In the former case, each host machine becomes a server for the execution of algorithms. In the latter case, the SGA acts as an intermediary, translating execution and monitoring requests made by the CSBase server to the specific system interface.

The main functionalities provided by SGA are:

- Resource Configuration Information: Provides mechanisms for querying the configuration of available computational resources for the execution of algorithms, identifying the attributes and requirements of the platform.
- Specific Platform Information: Collects and provides information about the platform-specific environment. This information is consolidated and sent to the CSBase server, together with a standard set of attributes.
- Commands Submission: Provides mechanisms for the execution, monitoring and control of commands.
- Sandbox to Algorithms: Provides mechanisms for the configuration of an optional sandbox area used for the input and output data in algorithm execution.

To facilitate the construction and adaptation of SGAs to new platforms, the component is organized in libraries. One of these is the `sgasubmit` Lua library, which encapsulates platform-specific submission code, and is the only part that needs to be modified when implementing an SGA for a new platform or queue-management system. The use of Lua makes it easy to provide a single interface regardless of implementing the library with command-line tools or with other libraries for job submission.

4. Scenarios

4.1. Integration of Seismic Applications

WebSintesi is a computational environment for interpretative processing, where users can have access to all aspects of seismic processing and visualization. It was the first of the batch submission systems Tecgraf developed for Petrobras. WebSintesi was initially developed from scratch with a monolithic approach, but later refactored for the construction of CSBase. After this refactoring, WebSintesi became one of the instances of the framework, evolving along with the updates to CSBase.

When Petrobras approached Tecgraf for the construction of WebSintesi, it presented requirements that laid out the basis for the CSBase model. Seismic data treatment involves huge amounts of data and intensive computing. The need to avoid moving data around and duplicating it led to CSBase's project-centric model, in which data files remains in a central repository, available to groups that are cooperating in their analysis. The requirement for intensive computing led to the construction of a remote submission system, with a submission daemon (the SGA) running on each remote execution host. Data security concerns led to the early incorporation of permission restrictions.

Because interpretative processing requires data to be integrated in several ways, WebSintesi users were the first to demand specific graphical applications as extensions to the basic desktop, for instance involving well log files, grid manipulation, and traces indexation. The original approach for developing these applications was too tight coupled to CSBase server, as reported in Section 3. The development of the CSDK kit greatly facilitated this process. CSDK also favoured third-party development of WebSintesi applications, and by now we have teams entirely from Petrobras developing domain-specific applications. Nowadays, WebSintesi hosts thirteen CSDK applications; each of them related to specific issues (viewers and analyzers for several types of information).

Each WebSintesi CSDK application explores CSDK contexts in different ways. As an example, a SEGY file editor, that supports a specific file format for storing geophysical data for seismic processing, explores two alternative file-handling contexts. The

local context, provides access to the local file system, while the *project* context, provides access to the CSBase server project area.

In other CSDK application, there was a demand to develop a graphical interface that uses the job submission context. Using this context programmers can code all algorithm parametrization and job control with a well defined API. During the development phase, programmers are coding all graphical interface and job submission stubs in offline mode. At a final phase, the application can be tested inside WebSintesi production servers with the sensitive algorithms, in online mode.

There are yet many CSBase applications tightly coupled to CSBase server that would benefit from this new architecture, allowing independent releases and testing facilities. As a result of the positive experience with CSDK, the CSBase and WebSintesi teams intend to migrate some native CSBase features to the this new architecture, using the concepts of contexts and applications. This migration aims to move combined service and native application parts of CSBase to CSDK. One important example is in the area of usage logs. A new standard context, providing access to usage data, is being defined, allowing different CSDK applications to analyze and visualize data. The native CSBase graphical component that provides visualization of this data could be adapted as a CSDK application using this context.

WebSintesi has been running at Petrobras for eleven years, and currently is in use by around a hundred seismic analysts. The current infrastructure consists of one main production server and other five minor installations for specific purposes. The main server, in Petrobras headquarters, has around 170 programs in its algorithm repository and has access to approximately ten clusters (with a variable number of 20 to 300 nodes).

4.2. Federation of HPC Resources

CSGrid is a freely distributed open source CSBase system and can be used for both academic and commercial purposes. It can therefore be seen as a showcase of the general functionalities of CSBase framework. CSGrid offers a collaborative and extensible environment to abstract the use of distributed computational and storage resources for High Performance Computing (HPC), providing functionalities that can be used both directly, by end users, or through programming interfaces. A typical CSGrid installation in Petrobras offers end users an integrated environment for running geomechanical simulations related to oil exploration and production, such as marine riser and reservoir engineering analysis.

Another important CSGrid installation is at SINAPAD[Gomes et al. 2014], the Brazilian National HPC Network. In partnership with LNCC (National Scientific Computing Laboratory), CSGrid has been used in the SINAPAD platform as the underlying middleware that abstracts the access to its geographically distributed, highly heterogeneous, computational environment, allowing execution of applications over dedicated HPC resources, as well as over desktops using an opportunistic computation model. One of SINAPAD's offerings is the *mc2* platform (*Minha Cloud Cientifica — My Scientific Cloud*), which provides a RAD (Rapid Application Development) toolset[Gomes et al. 2014] to lower the barrier for scientific application developers to launch new, personalized science gateways. These tools use the CSGrid middleware services for job and data management over an underlying federation of HPC resources.

Moreover, mc2 uses the extensible mechanisms of CSGrid to add new functionalities to the environment, such as data provenance tracking and restricted anonymous access.

As part of the mc2 architecture, the PortEngin tool creates instances of science gateways that offer Web interfaces for researchers to upload and download input and output data, parameterizing the job template of a specific algorithm and submitting and monitoring jobs. The PortEngin tool is developed as a client to the CSBase OpenDreams and Hierarchical Data services published in the OpenBus service bus. Figure 4 provides an overview of the mc2 relied on CSGrid as the underlying service oriented middleware architecture.

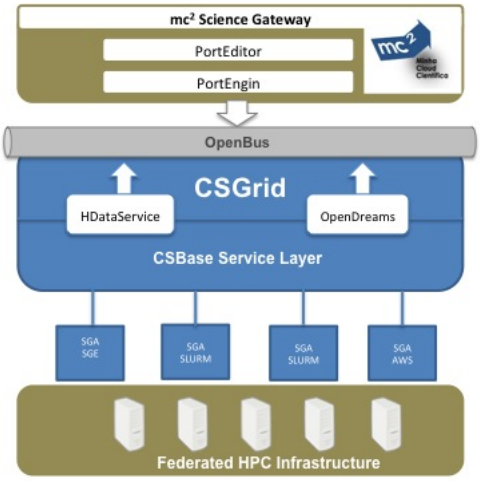

Figure 4. mc2 Platform Architecture

SINAPAD's main CSGrid installation currently has around 60 registered users working on a set of 50 algorithms. Computational resources include SGI, Sun, Altix, and Bull clusters. The SGI and Sun clusters run the SGE queuing system, while the Altix clusters use Torque PBS and the Bull machines run SLURM. CSBase offers SGA variants that interface with each of these queuing systems.

4.3. Distributed Workflow for Running Scientific Simulators

BR-SiOP is an instance of CSBase framework whose main purpose is to support the engineers' decisions, helping them in the process of optimizing oil and gas production. In its first version, BR-SiOP was a single application developed for another instance of CSBase. The results were so promising that a new instance, specially for oil and gas optimization, was created.

Several applications were developed for BR-SiOP, mostly for data-input, remote execution of the optimizer (installed as an algorithm), and for data-output, allowing comparison with real data.

Among the optimizers, one of the most commonly used is Pipe-It[Petrostreamz 2010], a software application that supports the integration of models and optimizes petroleum assets. Pipe-It works as an workflow where the developer user can graphically describe the dependencies among the models and connect the optimization

steps. In the first scenario, all optimization steps were executed in one single machine, as illustrated in Figure 5.

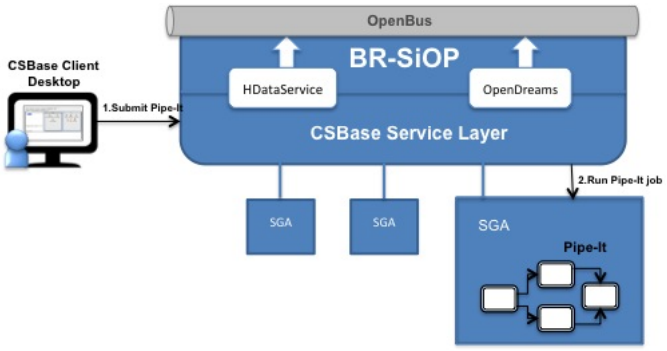

Figure 5. Optimization workflow in one single execution host.

Later, we decided to create a *launcher* program. This allows us to take advantage of the distributed computing resources even though the CSBase server has no knowledge about Pipe-It workflows. As illustrated in Figure 6, the launcher connects to the OpenDreams and Hierarchical Data services, via OpenBus, and requests the execution of an algorithm, transferring files to/from the server as needed. We then rebuilt the workflow system so that, for each optimization step, this launcher is used to start the specific program in BR-SiOP. In this scenario, the user submits the workflow manager, which runs as an algorithm in some remote machine, but this manager calls BR-SiOP back to execute each step of the optimization as an independent algorithm. Each step thus runs on a different machine, making the optimization much faster.

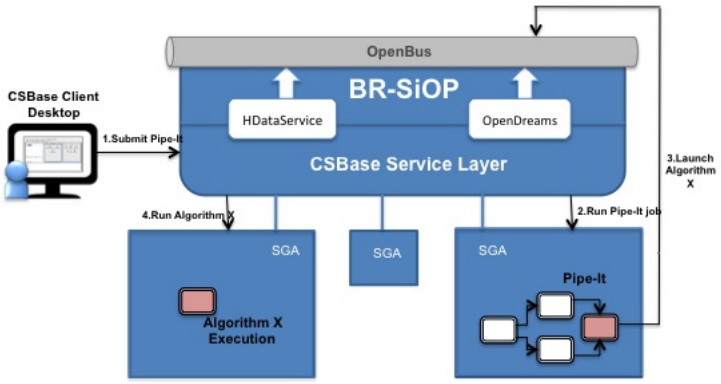

Figure 6. Distributed optimization workflow, via OpenBus.

A third scenario we are currently working on is making the launcher start an algorithm not installed on BR-SiOP, but on MARLIM, another CSBase instance running at Petrobras. MARLIM's algorithms are used as part of a few workflows in BR-SiOP. Originally, these algorithms were duplicated in both systems, but in this new scenario, such duplication won't be necessary anymore, since the execution will take place in MARLIM's environment and only the output files will be transferred back to BR-SiOP's project.

Even though the user can use all these systems in a completely integrated way, during the development of the optimization workflow it is useful to log into each system,

MARLIM and BR-SiOP, and test optimization steps individually before putting all of them together. Originally, the user would log in MARLIM, run an optimization, download output files locally, log in BR-SiOP, upload MARLIM's output files, and then use them as input for another optimization. Yet another CSDK application was developed to allow file transfer between CSBase servers, allowing the user to have a seamless experience when using the two environments.

5. Related Work

Parts of the CSBase architecture are somewhat similar to that of other submission systems such as Condor [Thain et al. 2005]. The SGA software that controls individual execution hosts has a role similar to that of the daemon that Condor runs on each computing resource. Because in Condor the emphasis is on the use of distributed computing resources, it has more sophisticated techniques in that area, including the matchmaking facilities for choosing execution nodes and support for checkpointing and migration. On the other hand, Condor does not contemplate data or program repositories, and is not concerned with the construction of different submission systems with domain-specific facilities.

The COMPs [Tejedor and Badia 2008] system is organized as a set of components that can be compared to CSBase services, and programs in different languages can act as clients to these components by using well-defined interfaces. However, in COMPs the purpose is to invoke components directly from the final application program. Annotations in the user code identify parts of the program that can be run in parallel, and COMPs manages dependencies and schedules tasks over distributed computing resources.

Unicore [Erwin 2002] is, to our knowledge, the system most similar in nature to CSBase. The development of Unicore had as its goal allowing the German supercomputing centers to offer their heterogeneous resources in an intuitive way. This goal itself has much in common with the use of CSGrid at SINAPAD. Unicore is organized as a set of components that can be installed in several configurations, allowing users to deploy new submission portals. These different instances of Unicore seem to differ less in goals and organization than is the case with CSBase instances, but the system has been in use for over ten years as an open-source project and seems to have a solid community. The client seems to be more tightly knit with the server than in the case of CSBase, but it supports extensions and legacy code can be linked to it through a wrapper. The execution model appears to be slightly different from that of CSBase in that the Unicore user explicitly chooses the HPC execution sites he/she has access to.

6. Final Remarks

In this paper we have discussed the CSBase framework, which has now been in use for over 10 years. Tecgraf has developed and deployed several instances of this framework for Petrobras, all of them continuously updated and extended.

The initial work that was done to refactor WebSintesi and transform it into a framework has repaid itself several times over the years, providing the Tecgraf Institute development team with a valuable tool for easily creating job submission systems. The decision to provide a well-defined extension interface, through the CSDK interface and library, was also very successful. As we reported in Section 4, CSDK has been extensively used

to tailor the CSBase desktop client for domain-specific use, and the concept of alternative contexts works very well for developing and testing new applications. At the other hand, as we have presented in the same section, the decision for making available CSBase services as OpenBus components expanded even more the range of possibilities for new scenarios of CSBase usage.

As the downside to having so many working systems based on a single framework, it has become relatively hard to introduce major changes in the architecture of CSBase. The encapsulation of services in interfaces allows the CSBase team to reprogram internals with no compatibility worries, but any proposal that has impact on clients and/or on the execution hosts raises issues regarding the ensuing programming and deployment costs.

The current architecture of CSBase was developed for the scenario where HPC clusters were the main infrastructure used for job submission. While HPC dedicated resources are the main target hosts used by CSBase systems, cloud computing is gaining interest across different areas of computational use, including the scientific computing community. The integration of CSBase with cloud platforms can bring new features regarding the elasticity of resources and the flexibility to describe the capabilities of resources and the job requirements. We are currently developing a prototype integration of CSGrid with the Azure Cloud Platform.

Also regarding cloud computing arena, the CSBase development team is, at the moment, participating in the EUBrazilCC project [EUBrazilCC 2015]. The *EU-Brazil Cloud infrastructure Connecting Federated Resources for Scientific Advancement* (EUBrazilCC) is a Research and Development project that aims at providing a user-centric test bench enabling European and Brazilian research communities to test the deployment and execution of scientific applications on a federated intercontinental e-infrastructure. This infrastructure includes private cloud resources, federated by fogbow [Barros et al. 2015] and HPC resources, federated by CSGrid. To comply with EUBrazilCC's requirements, CSGrid is being adapted to interact with fogbow, allowing the execution of jobs with resources dynamically instantiated by cloud providers. This is particular useful when applications are required to run partially on HPC clusters and partially on virtual machines.

For the purpose of testing the EUBrazilCC e-infrastructure, CSGrid will support scientific use cases addressing two complementary problems in cardiovascular modeling. Heterogeneous supercomputing and virtualized infrastructures are being integrated with the orchestration of a heart simulator from Spain and a vascular simulator from Brazil. CSGrid will be adopted as part of the architecture to support the use case on the Brazilian side, providing the underlying middleware to run the Anatomically-Detailed Arterial Network (ADAN) model, developed at the Laboratório Nacional de Computação Científica (LNCC).

As part of a new e-Science research project started at the beginning of this year, the CSBase team is working on development of a REST API for CSBase. This is an important step towards supporting the integration of CSBase systems with web-based applications. The adoption of open and standards-based API will contribute for creating new opportunities and enabling more applications to take advantage of the CSBase systems.

There are several other directions in which we would like to work on CSBase. After these years, we have a great deal of data at our disposal. All instances of CSBase log

submissions along with their configuration. This is interesting from the point of view of data provenance. The logged information could be exposed to the user, allowing queries about execution histories and data generation. This history could also be explored as input to decision making, for instance when scheduling algorithm execution. Accounting and auditing are other areas of interest, as is the study of a more sophisticated system for matchmaking between jobs and resources.

Finally, over the past 10 years of development, CSBase has been accumulating a large amount of software process experience through many real-client project software engineering practices. Moreover, the challenge of maintaining and evolving a system that is in intensive use in projects with specific demands and restraints are non-trivial tasks. An understanding of how all these software engineering aspects have contributed to CSBase's success could be further investigate from system planning perspective.

Acknowledgement

We would like to thank Arndt von Staa (Department of Informatics, PUC-Rio) and Carlos Cassino (Tecgraf Institute, PUC-Rio) for helping with the revision of this paper.

References

Barros, A., Brasileiro, F., Farias, G., Germano, F., Nóbrega, M., Ribeiro, A. C., Silva, I., and Teixeira, L. (2015). Using fogbow to federate private clouds. *Salão de Ferramentas do XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Brasileiro, F. V., Araújo, E., Voorsluys, W., Oliveira, M., and de Figueiredo, F. (2007). Bridging the high performance computing gap: the OurGrid experience. In *CCGRID*, pages 817–822. IEEE Computer Society.

Clinio, A. and Almeida, I. (2015). CSDK: Uma forma de modularização de componentes do desktop no CSBase. Relatório Técnico do Instituto Tecgraf/ PUC-Rio.

de Lima, M., Ururahy, C., de Moura, A., Melcop, T., Cassino, C., dos Santos, M. N., Silvestre, B., Reis, V., and Cerqueira, R. (2006). CSBase: A framework for building customized Grid environments. In *Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, WETICE '06, pages 187–194.

Erwin, D. (2002). UNICORE – a Grid computing environment. *Concurrency and Computation: Practice and Experience*, 14(13-15):1395–1410.

EUBrazilCC (2015). EUBrazil Cloud Connect. `http://eubrazilcloudconnect.eu`.

Forum, O. G. (2008). Distributed resource management application API specification 1.0. `www.ogf.org/documents/GFD.133.pdf`.

Gentzsch, W. (2001). Sun Grid Engine: Towards creating a compute power grid. In *CCGRID*, pages 35–39. IEEE Computer Society.

Gomes, A., Bastos, B., Medeiros, V., and Moreira, V. (2014). Experiences of the Brazilian national high-performance computing network on the rapid prototyping of science gateways. *Concurrency and Computation: Practice and Experience*.

Ierusalimschy, R. (2013). *Programming in Lua*. Lua.org.

Jette, M. A., Yoo, A. B., and Grondona, M. (2002). SLURM: Simple linux utility for resource management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pages 44–60. Springer-Verlag.

Maia, R. and Roenick, H. (2015). Secure authentication in integrations of systems with OpenBus. Relatório Técnico do Instituto Tecgraf/ PUC-Rio.

Petrostreamz (2010). Pipe-it. `http://petrostreamz.com/pipe-it`.

Santos, M. and Cerqueira, R. (2006). GridFS: Targeting data sharing in Grid environments. In *Proceedings of the 6th International Workshop on Distributed Shared Memory*, CCGRID '06, Cingapura.

Staples, G. (2006). TORQUE resource manager. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA. ACM.

Tejedor, E. and Badia, R. M. (2008). COMP superscalar: Bringing GRID superscalar and GCM together. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '08, pages 185–193.

Thain, D., Tannenbaum, T., and Livny, M. (2005). Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356.