Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Adecuación a ambientes Grid de dos aplicaciones de Computación de Alto Desempeño en el dominio de Física de Altas Energías y Simulación Atmosférica

Trabajo Especial de Grado presentado ante la Ilustre Universidad Central de Venezuela por la **Br. Esther González** para optar por el título de Licenciado en Computación.

Tutores:

Robinsón Rivas

Jaime Parada

Caracas, Octubre, 2009

Resumen

Hoy en día, las aplicaciones operativas o en desarrollo aumentan drásticamente la carga de trabajo de los computadores, exigiendoles mayores niveles de recursos para la ejecución de las mismas. La filosofía Grid , el alto poder de procesamiento que ofrecen hoy en día muchos computadores y las altas capacidades de almacenamiento disponibles en la actualidad, ofrecen una solución viable para la solución de estos problemas. El termino Grid se refiere a una infraestructura que permite la integración y el uso colectivo de computadores, redes y bases de datos que son propiedad y están administradas por diferentes instituciones.

El modelo hidrometeorológico BRAMS y el sistema de simulación AIRES son aplicaciones que requieren un alto poder de procesamiento, el objetivo principal de este trabajo es adaptar dichas aplicaciones a una plataforma Grid, en este caso de estudio se hará uso de la plataforma instalada en el Centro de Computación Paralela y Distribuida.

Dedicatoria

A mi bella abuelita, a mi madre, a toda mi familia.

Agradecimientos

Hay alguien muy especial a quien debo agradecer por haberme apoyado en todo momento, ayudarme en todo lo que le fue posible, sin ti no lo hubiese logrado, gracias por aclarar mis días malos, y hacer aun mejores los buenos, por toda tu comprensión y apoyo que no pudo ser mas incondicional. En segundo lugar, pero no menos importante, a mi madre, por ser simplemente el ser mas especial en mi vida, por amarme como lo hace, y entenderme mas allá de lo entendible. A mi tutor, el profesor Robinson, por comprender todos los inconvenientes presentados en el desarrollo de este trabajo de grado y presentarme siempre una solución. Al Ing. Rafael Mundaray, por siempre estar dispuesto a aclarar mis dudas sobre los modelos, y estar pendiente del desarrollo de la tesis. A Josi, por su loca pero incondicional amistad. Y a todas aquellas personas que de una u otra manera fueron parte de esta tesis, con su colaboración y apoyo.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Hoy en día, las aplicaciones operativas o en desarrollo aumentan drásticamente la carga de trabajo de los computadores, exigiendoles mayores niveles de recursos para la ejecución de las mismas, gracias a los avances alcanzados en computo grupal y distribuido, es posible contar con gran velocidad y poder de procesamiento a costos significativamente inferiores a los incurridos con la compra de grandes equipos, los cuales hasta hace poco tiempo eran los encargados de afrontar las tareas de calculo pesado en las empresas o instituciones. En este sentido, la filosofía Grid [2] ofrece una solución viable y robusta para estos problemas.

Los Grids computacionales ofrecen a las organizaciones grandes beneficios como el aumentar su capacidad de cómputo y almacenamiento al compartir los recursos existentes y de esta manera ahorrar recursos evitando la adquisición de nuevos equipos.

El termino Grid computacional se refiere a una infraestructura que permite la integración y el uso colectivo de computadores, redes y bases de datos que son propiedad y están administrados por diferentes institutos. El propósito es facilitar la integración de recursos computacionales. Universidades, laboratorios de investigación, empresas, etc. se asocian para formar un Grid computacional para lo cual utilizan algún tipo de software que implemente este concepto.

Para evaluar los Grid computacionales es importante tomar en cuenta los estándares como

medio para permitir la interoperabilidad de una infraestructura común.

Ian Foster[1], destacado investigador en Clusters y Grids, propone una lista que captura la esencia de lo que debería hacer un Grid:

- Coordinar los recursos que no están sujetos a un control centralizado: Un Grid computacional integra y coordina recursos y usuarios que están en diferentes dominios, diferentes unidades administrativas, etc.
- Utiliza estándares abiertos, protocolos de propósitos generales e interfaces: Un Grid computacional es construido con múltiples propósitos e interfaces que están dirigidos fundamentalmente a la autenticación, autorización, descubrimiento de recursos y acceso a los recursos. De otra manera se estaría trabajando con una aplicación específica para un sistema.
- Un Grid computacional utilizará los recursos que lo constituyen, los cuales deberán ser usados de una manera coordinada para entregar servicios de calidad: por ejemplo tiempo de respuesta, disponibilidad, seguridad y rendimiento al procesar los datos, además de la asignación de los múltiples tipos de recursos para satisfacer las complejas exigencias de los usuarios, de modo que la utilidad del sistema combinado sea perceptiblemente mayor que la suma de sus piezas.

1.1. Objetivos del trabajo

Existen diversos programas de calculo intensivo que necesitan de un poder de computo superior al que tienen disponible las instituciones, o investigadores que los utilizan. Dos de ellos son el simulador de lluvia de partículas AIRES[22] y el modelo hidrometeorologico BRAMS[10], el primero utilizado por el Departamento de Física en la Facultad de Ciencias de la UCV y el segundo por el Instituto Nacional de Meteorologia e Hidrología (INAMEH)[23]. Debido al gran volumen de datos que procesan estos programas, es necesaria una infraestructura capaz de procesar y almacenar de manera eficiente y eficaz dichos datos. El Grid computacional ofrece una alternativa para

cumplir con esta necesidad.

1.1.1. Objetivo General

Adaptar las aplicaciones AIRES y BRAMS para ser utilizadas en el Grid computacional UCV y realizar pruebas de funcionamiento de estas aplicaciones.

1.1.2. Objetivos Específicos

- Estudiar las aplicaciones AIRES y BRAMS, para conocer su funcionamiento y proceso de instalación.
- Adaptar las aplicaciones para poderlas ejecutar en el Grid computacional UCV.
- Desarrollar interfaces para facilitar el manejo de las aplicaciones una vez adaptadas al Grid computacional UCV.
- Realizar pruebas de funcionamiento y confiabilidad de las ejecuciones realizadas tanto en el Grid computacional UCV como contra el cluster, adicionalmente se tomaron metricas del tiempo de ejecucion en ambas infraestructuras.

Capítulo 2

Bases Teóricas

A continuación presentamos algunos conceptos de la infraestructura Grid, que nos servirán de base para comprender el funcionamiento y la interacción entre los distintos componentes de un Grid computacional.

2.1. Conceptos Generales

- **Servicios Web**: Un servicio web (en inglés, Web service) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de computadoras.

- **GLUE (Grid Laboratory for a Uniform Environment)**: El GLUE Schema tiene como objetivo definir un modelo de datos conceptuales común para ser usado por los recursos de un Grid. Permite describir una plataforma Grid de forma estandarizada,mostrando una visión uniforme de recursos y servicios a usuarios y servicios externos.

- **Sistemas Distribuidos**: Es una una colección de computadores generalmente separados físi-

camente y conectados entre si por una red de comunicaciones, cada maquina posee sus componentes de hardware y software que el usuario percibe como un solo sistema. El usuario accede a los recursos remotos de la misma manera que accede a recursos locales, las diferentes computadoras se comunican y coordinan sus acciones intercambiando mensajes. Los sistemas distribuidos deben ser muy confiables, ya que si un componente del sistema falla otro componente debe de ser capaz de reemplazarlo.[1]

- **Grid Computacional**: "Un Grid computacional es una infraestructura de hardware y software que provee acceso consistente a bajo costo a recursos computacionales de alto nivel."[1]. El termino "Grid Computing" sugiere entonces un paradigma similar a un tendido eléctrico en donde una diversidad de recursos contribuyen en la generación de energía que será compartida entre los usuarios. Por lo tanto, la clave de los esfuerzos es el de definir los estándares que van a permitir de manera sencilla el compartir todos los recursos computacionales, de almacenamiento y datos de manera que se pueda promover la adopción del Grid computacional en forma masiva.[2]

- **Organización Virtual**: Las organizaciones virtuales o VO por sus siglas en ingles son grupo o colecciones de organizaciones e individuos que permiten compartir recursos de una manera controlada, es decir, establecen reglas entre los consumidores y los proveedores de los recursos, definiendo claramente que recursos se comparten y bajo que condiciones, de modo que los miembros puedan colaborar para alcanzar una meta compartida.[4]

- **Middleware**: Es una capa intermedia de la arquitectura Grid, compuesta de un conjunto de software de conectividad que ofrecen servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas, funciona como una capa de abstracción de software distribuida.[7]

  El Middleware nos abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando un API (Application Programming Interface) para la fácil programación y manejo de aplicaciones distribuidas, los principales middlewares son:

- Globus: desarrollado por el Globus alliance[17].
- LCG: desarrollado por el CERN[18].
- gLite desarrollado por el EGEE[19].

- **Enabling Grids for E-sciencE (EGEE)**: El proyecto Enabling Grids for E-sciencE (EGEE)[9] comenzó en abril de 2004 y ha crecido rápidamente para ser una de las infraestructuras Grid más grandes del mundo utilizada para proyectos de investigación. Involucra mas de 27 países, mas de 20 mil CPUs, casi 200 sitios web y 10 petabytes de capacidad de almacenamiento en linea. La infraestructura da soporte a 7 áreas científicas y a más de 20 aplicaciones individuales. Se espera que en el futuro el Grid UCV forme parte de esta organización. En la figura 2.1 se representan las distintas organizaciones del proyecto en el mundo.

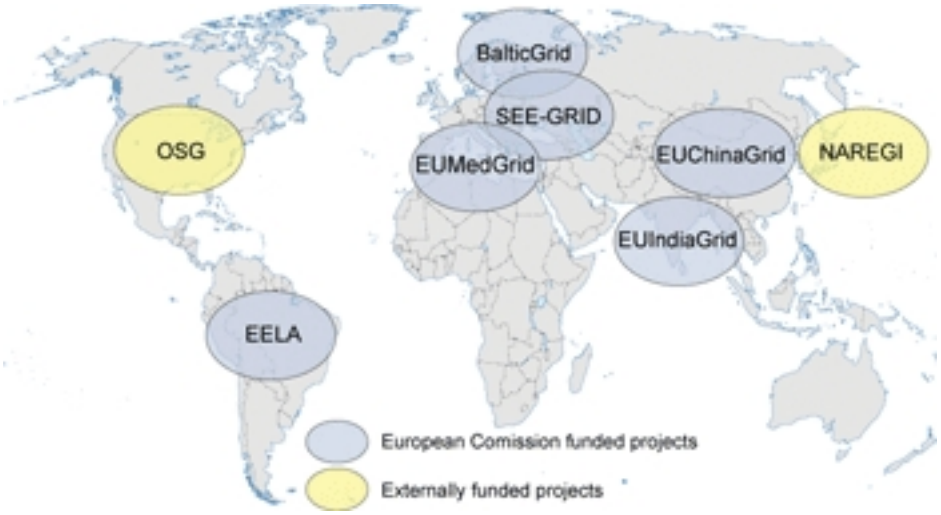

Figura 2.1: EGEE en el mundo

- **Gilda(Grid Infn Laboratory for Dissemination Activities)**: GILDA[8] es un banco de pruebas de Grid completamente operativo, permite que tanto los usuarios como los administradores del sistema accedan a una experiencia directa con los sistemas en Grid. En el contexto de EGEE, GILDA actúa como un componente crucial del proyecto del programa de infraestructura-f (infraestructura de formación), ayudando a transmitir el conocimiento y la experiencia, así como los recursos computacionales, a la comunidad científica y a la industria.

  El banco de pruebas GILDA consta de 20 sitios en tres continentes y utiliza hardware heterogéneo para actuar como un entorno de Grid "real". Está compuesto por todos los elementos de un proyecto Grid e incluye sistemas de prueba y monitorización, para que cualquier parte interesada pueda utilizar el banco de pruebas. Cuenta con una Organización Virtual (VO) y una Autoridad de Certificación (CA) real que concede un certificado de dos semanas para el uso de la infraestructura GILDA a modo de prueba.

  Para adaptar GILDA a las necesidades de infraestructura-f de EGEE, dispone de una serie de portales diferentes para diversos usos, entre los que incluyen portales básicos para usuarios nuevos y otros completos para seminarios y demostraciones avanzadas.

En el siquiente capítulo se describen los componentes principales del Middlware gLite, así como sus principales servicios y formas de interactuar con el mismo.

Capítulo 3

Middleware gLite

gLite[20] es un Middleware ligero, orientado a servicios, para plataformas Grid, nació del proyecto Enabling Grids for E-sciencE (EGEE), gLite es una evolución del proyecto LCG (LHC Computing Grid), actualmente esta basado en Globus Tool Kit 4(GTk4)[21], que no es mas que un conjunto de herramientas para construir Grids.

El principal objetivo de gLite es la inter-compatibilidad con diferentes sistemas. Es por esto, que el middleware trata de utilizar, lo mas que se pueda, protocolos e interfaces estandarizados, ponerse de acuerdo sobre las interfaces permite interactuar con otros middelware sin que los servicios estén implementados de la misma manera.

3.1. Componentes de gLite

EL Middleware gLite consta de una serie de componentes que permiten el envio, ejecución y almacenamiento de programas, entre otras funciones. A continuación se describe brevemente los principales componentes que se requieren para la implementación de un Grid computacional.

- **User Interface**: El User Interface (UI) son un conjunto de clientes y APIs, que los usuarios y las aplicaciones pueden usar para acceder a los recursos del Grid, básicamente es el punto

de acceso al Grid. Puede ser cualquier máquina en la que los usuarios tengan cuentas y sus certificados de usuario instalados. Desde una UI, el usuario puede autenticarse, acceder a los recursos del Grid, funcionalidades y servicios [6].

- **Workload Management System**: El Workload Management System (WMS), es el conjunto de componentes del Middleware gLite responsable de la distribución y gestión de Jobs en los recursos del Grid. El propósito del WMS es el de aceptar Jobs de los usuarios. El Resource Broker (RB) es la máquina donde el servicio del WMS se ejecuta, el RB ubica los archivos de entrada al Grid, especificados en la descripción del Job, utilizando un servicio llamado Data Location Interface (DLI), que proporciona una interfaz genérica a un catálogo de archivos, por último, el servicio de Logging and Bookkeeping (L&B) supervisa los trabajos manejados por el WMS, reúne eventos de muchos componentes del WMS y registra el status y la historia del trabajo [6].

- **Computing Element**: El Computing Element (CE) es el servicio que representan un recurso de cómputo, su función principal es el manejo de los Jobs (envío, control, etc), adicionalmente también esta provisto de otras capacidades, como por ejemplo, generar información sobre sus propias características y status. El CE expone una interfaz de servicios web, que pude ser usado por un cliente genérico, un usuario final que interactúa con el directamente, o por el WMS que busca un CE apropiado para un determinado trabajo. El CE se refiere a un conjunto, o grupo de recursos computacionales, gestionado por una LRMS o manejador de colas [6].

- **Storage Element**: El Storage Element (SE) es el encargado de proporcionar un acceso uniforme a los recursos de almacenamiento de datos, el SE puede controlar simples servidores de disco, arreglos de disco de gran tamaño o sistemas de almacenamiento masivo basado en cintas (Mass Storage Systems - MSS). La mayoría de los SE son gestionadas por un administrador de recursos de almacenamiento (Storage Resource Manager -SRM), un servicio del Middleware que proporciona la capacidad de migración transparente de archivos de disco a cinta, reserva de espacio, etc [6].

- **Working Nodes**: El Working Nodes (WN) es el servicio que se encarga de finalmente ejecutar los Jobs, una vez que al CE le es delegado un Job, este busca localmente su lista de WNs, y los manda a ejecutar en los mismos. Constantemente el WN esta actualizando el estado de ejecución del Job en su correspondiente CE, una vez que el Job se termina de ejecutar, el WN le delega la responsabilidad de su manejo al CE. Los WNs generalmente, están en redes privadas, son los nodos de los Cluster o en su defecto maquinas que solo seran usadas para el trabajo pesado, es por ello, que los WN solamente tienen comunicación con ellos mismos, y con el CE al que pertenecen [6].

En la figura 3.1 se observan los componentes principales de gLite y como interactuan.

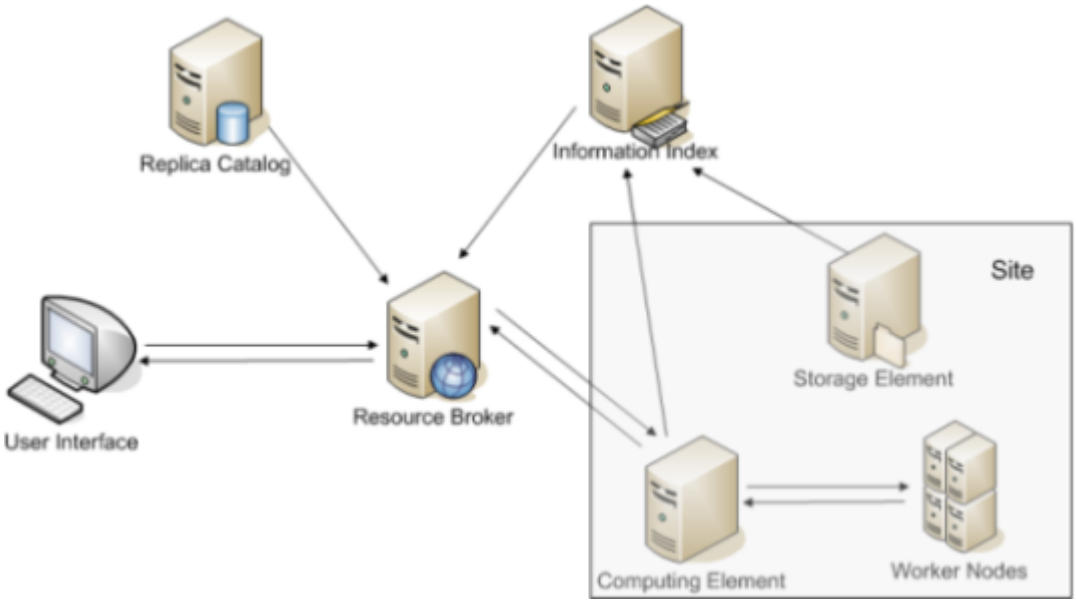

Figura 3.1: Componentes principales de gLite

3.2. Servicios de gLite

gLite es una arquitectura que esta constituida por una serie de elementos separados por su función, dichos funciones son:

- **Servicios de Seguridad**: Engloba los servicios de autenticación, autorización y auditoría, se encargan de la identificación de las entidades (usuarios, sistemas y servicios), permiten o niegan el acceso a servicios o recursos, y proveen información forense para el análisis de eventos de seguridad, adicionalmente también proporciona una funcionalidad para la confidencialidad de los datos y un servicio de conectividad dinámica [6].

- **Servicios de Información**: El servicio de información (IS) proporciona información sobre los recursos del Grid y su status. Esta información es esencial para la operación del Grid, ya que es a través del IS que los recursos son descubiertos. La información publicada también es utilizada para propósitos de monitoreo y contabilización[6].

  Gran parte de los datos publicados en el IS se ajustan al esquema GLUE, que define un modelo conceptual común de datos para ser utilizado para el monitoreo y descubrimiento de recurso en el Grid. Dos sistemas IS son utilizados en gLite:

  - Globus Monitoring and Discovery Service (MDS), utilizado para el descubrimiento de recurso y para publicar el status de recursos
  - Relational Grid Monitoring Architecture (RGMA), utilizada para contabilizar, monitorear y publicar información a nivel de usuario.

- **Servicios de manejo de Jobs**: Son aquellos servicios encargados de manejar los trabajos o Jobs, entre los mas importantes están el Computing Element (CE) y el Workload Management System (WMS). Aunque están relacionados principalmente con el manejo de Jobs, también se encargan de los eventos de autorización definidos en el VOMS (Virtual Organization Membership System)[6].

- **Servicios de manejo de Datos**: La unidad primaria para la gestión de datos en el Grid son los

archivos. En un ambiente Grid, los archivos pueden tener replicas en muchos sitios diferentes para mantener la consistencia, los archivos del Grid no pueden ser modificados después de que sean creados, sólo pueden leerse y borrarse[6].

Los archivos en el Grid pueden ser referidos por nombres diferentes: Grid Unique IDentifier (GUID), Logical File Name (LFN), Storage URL (SURL) and Transport URL (TURL). Mientras los GUIDs y LFNs identifican un archivo sin tomar en consideración su ubicación, el SURLs y TURLs contienen información sobre su ubicación, y cómo puede accederse a ellos.

Un archivo puede ser identificado sin ambiguedad por su GUID; este es asignado la primera vez que el archivo es registrado en el Grid, y es basado en el estándar UUID para garantizar su singularidad (UUIDs utiliza una combinación de una dirección MAC y un timestamp para asegurar que todo UUIDs sea distinto). El mapeo entre LFNs, GUIDs y SURLs es mantenido por un servicio llamado File Catalogue.

3.3. Seguridad en gLite

Autenticación La autenticación se refiere a la identificación de entidades (usuarios, sistemas y servicios), cuando se establece un contexto para el intercambio de mensajes entre los diferentes actores, es decir, que es el mecanismo que permite saber con quien estoy interactuando. Esta información se utiliza en muchas de las políticas de acceso a los recursos y la protección de datos, así como para la auditoría y la respuesta a incidentes.

El sistema esta basado en PKI(Public Key Infrastructure), o sistema de clave publica, bajo el estándar X.509, que usa el concepto de confianza en terceros, en este caso Autoridades Certificadoras (CA-Certification Authorities), estas CAs generan certificados o credenciales a los usuarios.

Para reducir vulnerabilidades, cuando un usuario se identifica mediante sus credenciales, se usan proxies de sus certificados para darles una validez limitada, generalmente de 24 o 48 horas. El hecho de usar proxies permiten una serie de ventajas, una de ellas es delegar el proxy a servicios

para que ellos actúen como si fuera el usuario original, ser almacenados en un ente externo, y ser renovados automáticamente al estar cercano a expirar[6].

Autorización La autorización se refiere a permitir o denegar acceso a servicios o recursos basados en políticas de acceso. El mayor problema con la autorización en un Grid es como manejar la superposición de políticas debido a los múltiples dominios administrativos (políticas de usuario, políticas propias de las VOs, etc), y como poder combinarlas.

Existen 3 modelos básicos de autorización, clasificados como Agent, Push y Pull. En el modelo Agent el usuario solamente interactúa con el servidor de autorizaciones, en el modelo Push el servicio de autorización delega tokens y el usuario delega esos tokens en el recurso que quiera usar, en el modelo Pull es el recurso quien pide los permisos al servidor de autorizaciones[6].

Los VO Membership Service (VOMS), son los servidores de autorización y están constituidos por una AA (Attribute Authority), donde un usuario tiene un conjunto de atributos sobre los recursos, y las políticas sobre esos recursos.

Acceso al Grid Todos los servicios de gLite pueden ser accedidos por medio de APIs y CLIs (Command Line Interface). La tendencia hacia los servicios web permite la generación automática de APIs, esto es sin embargo, tedioso y propenso a errores, pues con el tiempo puede ir variando sus definiciones, es por ello que se estila proveer de un API pre-generado. Estos APIs pueden ofrecer un nivel de funcionalidad de mas alto nivel que el mismo WSDL (Web Service Description Language). Para el acceso al Grid mediante gLite, es necesario acceder al UI (User Interface), el cual mediante el uso de CLIs, provee una serie de comandos para la generación de proxies, envio de Jobs y demás funcionalidades.

3.4. Interacción con el Grid

Autenticación en el Grid

Como primer paso, para que un usuario pueda hacer uso del Grid, debe tener instalado en su cuenta en el UI un certificado emitido por una Autoridad Certificadora (CA) confiable, el usuario puede crear un proxy para hacer uso del Grid, con el comando *grid-proxy-init* [6].

Con el Virtual Organization Membership Service (VOMS), el usuario puede crear un proxy con extensiones VOMS, indicando a cual organización pertenece junto con otra información relevante, el comando *voms-proxy-init* genera el grid proxy, contacta al servidor VOMS, obtiene los atributos del usuario y lo incluye en el proxy. Si se usa el comando sin argumentos funciona igual que el comando *grid-proxy-init*. Para crear un simple VOMS proxy se utiliza:

```
$ voms-proxy-init --voms [vo]
```

donde [vo] es la organización virtual del usuario.

La salida de este comando es similar a la mostrada en la figura 3.2:

```
Enter GRID pass phrase:
Creating temporary proxy ..................... Done
Contacting voms.ct.infn.it:15001 [/C=IT/O=INFN/OU=Host/L=Catania/CN=
   voms.ct.infn.it] "gilda" Done
Creating proxy ................................................
   Done
Your proxy is valid until Thu Oct  1 03:41:47 2009
```

Figura 3.2: Generación de proxy

Petición de Información El UI también ofrece comandos para pedir información sobre el estatus del Grid. Con el comando *lcg-infosites* se pueden solicitar los recursos disponibles para una organización virtual en particular. El comando *lcg-infosites* se utiliza de la siguiente manera:

```
$ lcg-infosites --vo <nombre de organizacion virtual> <componente>
```

Por ejemplo, el siguiente comando lista los CE disponibles en la VO gilda y el estado de sus CPUs.

```
$ lcg-infosites --vo gilda ce
```

obteniendo una salida similar a la mostrada en la figura 3.3:

```
bdii : gilda-bdii.ct.infn.it:2170
#CPU   Free TJobs Running Waiting ComputingElement
----------------------------------------------------------
  26    26    2       0       2    grid010.ct.infn.it:2119/jobmanager-lcgpbs-long
   6     6    1       0       1    gilda-ce.rediris.es:2119/jobmanager-lcgpbs-infinite
 104     0  102      93       9    ce1-egee.srce.hr:2119/jobmanager-sge-prod
  26    26    0       0       0    grid010.ct.infn.it:2119/jobmanager-lcgpbs-short
   6     6    0       0       0    gilda-ce.rediris.es:2119/jobmanager-lcgpbs-long
   6     6    1       0       1    gilda-ce.rediris.es:2119/jobmanager-lcgpbs-short
```

Figura 3.3: Comando lcg-infosites

Envío Simple de Jobs

En esta sección se describen los conceptos básicos del lenguaje usado para describir un Job. Se describirán los procedimientos para enviar, monitorear y recuperar las salidas de un Job.

Job Description Language El Job Description Language es un lenguaje de alto nivel, orientado a usuarios para la descripción y agregación de Jobs. Siendo el JDL un lenguaje extensible el usuario puede hacer uso de cualquier atributo para la descripción o especificación de requerimientos.

Sin embargo, solo un conjunto de atributos son tomados en cuenta por el WMS al momento de enviar un trabajo. Para poder enviar un Job al WMS es necesario el JDL. El JDL describe las características y limitaciones de un Job.

En la figura 3.4 se muestra un ejemplo de un JDL, que ejecuta un simple Job en el Grid:

```
Type = "Job";
JobType = "Normal";
Executable = "/bin/hostname";
StdOutput = "hostname.out";
StdError = "hostname.err";
OutputSandbox = {"hostname.err","
   hostname.out"};
Arguments = "-f";
ShallowRetryCount = 3;
```

Figura 3.4: Ejemplo de JDL

El atributo Executable especifica el comando que sera ejecutado en el WN. El atributo OutputSandbox indica los archivos que serán copiados en el UI al momento de terminar la ejecución del Job; normalmente estos archivos son las salidas del Job, sus nombres están definidos en los atributos StdOutput y StdError.

Flujo de un Job

En esta sección se describirá lo que sucede cuando un usuario envía un Job al Grid, y se explica la forma en que interactúan los diferentes componentes.

1. Después de obtener un certificado digital de una Autoridad Certificadora (CA), registrarse en una VO y obtener una cuenta en un UI, el usuario está listo para utilizar el Grid. Debe

iniciarse sesión en el UI y crear un proxy para autenticarse.

2. El usuario envía un Job desde el UI al WMS. En la descripción del Job se puede especificar si uno o más archivos serán copiados del UI al WN, estos son copiados inicialmente al WMS. Este conjunto de archivos son llamados Input Sandbox. Un evento es registrado por el L&B y el estatus del Job es SUBMITTED.

3. El WMS busca el mejor CE disponible para ejecutar el Job. Para hacer esto, es interrogado el Information Supermarket (ISM), una cache interna de información en el sistema actual es leído del BDII, para determinar el estatus de recursos computacionales y de almacenamiento. Otro evento es registrado por el L&B y el estatus del Job es WAITING.

4. El WMS prepara el Job para el envió, creando una script que será enviado, junto con otros parámetros, al CE seleccionado. Un evento es registrado por el L&B y el estatus del Job es READY.

5. El CE recibe la petición y envía el trabajo para la ejecución al LRMS local. Un evento es registrado por el L&B y el estatus del Job es SCHEDULED.

6. El LRMS maneja la ejecución de Jobs en los WN locales. Los archivos en el Input Sandbox son copiados del WMS a el WN disponible donde el trabajo es ejecutado. Un evento es registrado por el L&B y el estatus del Job es RUNNING.

7. Mientras el trabajo es ejecutado, archivos del Grid pueden ser accedidos directamente desde un SE utilizando los protocolos RFIO o gsidcap.

8. El trabajo puede producir nuevos archivos de salida que pueden ser cargados al Grid y estar disponible para ser usados por otros usuarios del Grid. Esto puede lograrse utilizando herramientas de Data Management. O copiandolos al Storage Element y registrandolo en un catálogo de archivo.

9. Si el trabajo termina sin errores, las salidas son transferidas al WMS. Un evento es registrado por el L&B y el estatus del Job es DONE.

10. En este momento, el usuario puede recuperar la salida del Job, copiandolas en el UI. Un evento es registrado por el L&B y el estatus del Job es CLEARED.

Envío de Jobs

Un Job puede ser enviado con el comando *glite-wms-Job-submit -d delegationId -o Jobidfile jdlname*, en la figura 3.5 se muestra un ejemplo del uso de este comando:

```
[esther@grid-ccpd03 esther]\$ glite-wms-Job-submit -d \$USER -o jobid hostname.jdl

Connecting to the service https://glite-rb3.ct.infn.it:7443/glite\_wms\_wmproxy\_server


====================== glite-wms-Job-submit Success ======================

The Job has been successfully submitted to the WMProxy
Your Job identifier is:

 https://grid-ccpd01.ucv.ve:9000/-fGJvzaJQRSUaC6vJH2prA


The Job identifier has been saved in the following file:
/home/esther/jobid

==========================================================================
```

Figura 3.5: Ejemplo de Job-submit

El archivo /home/user/jobid almacena el id de los trabajos que se envian, en caso de enviar otro trabajo, el id se anexara en una nueva linea del mismo archivo.

Estado de los Job

Para conocer el estado en el que se encuentra el Job existe el siguiente comando *glite-wms-Job-status*

este comando realiza consultas al L&B sobre el estado de cualquier Job cuyo id se encuentre en el archivo especificado, en la figura 3.6 se muestra un ejemplo:

```
[esther@grid-ccpd03 esther]\$ glite-wms-Job-status -i jobid


-------------------------------------------------------------
1 : https://grid-ccpd01.ucv.ve:9000/-fGJvzaJQRSUaC6vJH2prA
a : all
q : quit
-------------------------------------------------------------
Choose one or more jobId(s) in the list - [1-2]all::a

[esther@grid-ccpd03 esther]\$ glite-wms-Job-status -i jobid

*************************************************************
BOOKKEEPING INFORMATION:

Status info for the Job : https://grid-ccpd01.ucv.ve:9000/5-uIMIxVoVuj3XHQ9JugSA
Current Status:     Done (Success)
Logged Reason(s):
    - Job terminated successfully
Exit code:          0
Status Reason:      Job terminated successfully
Destination:        grid-ccpd04.ucv.ve:2119/jobmanager-lcgpbs-gilda
Submitted:          Mon Sep 21 13:29:44 2009 VET
Parent Job:         https://grid-ccpd01.ucv.ve:9000/-fGJvzaJQRSUaC6vJH2prA
*************************************************************
```

Figura 3.6: Ejemplo de Job-status

El comando se basa en el contenido del archivo que contiene los ids de los Jobs, mostrando una lista de los Jobs enviados. La opción -i especifica el archivo desde donde el comando tomara los JobIds. La salida mostrara el estado del Job así como, el CE donde se ejecuta el Job. Si se quiere conocer el estado de un Job especifico, conociendo su id, se puede solicitar el estado de únicamente ese Job, como se muestra en la figura 3.7:

```
[esther@grid-ccpd03 esther]$
glite-wms-Job-status https://grid-ccpd01.ucv.ve:9000/5-uIMIxVoVuj3XHQ9JugSA
*************************************************************
BOOKKEEPING INFORMATION:
Status info for the Job : https://grid-ccpd01.ucv.ve:9000/5-uIMIxVoVuj3XHQ9JugSA
Current Status:     Done (Success)
Logged Reason(s):
    - Job terminated successfully
Exit code:          0
Status Reason:      Job terminated successfully
Destination:        grid-ccpd04.ucv.ve:2119/jobmanager-lcgpbs-gilda
Submitted:          Mon Sep 21 13:29:44 2009 VET
Parent Job:         https://grid-ccpd01.ucv.ve:9000/-fGJvzaJQRSUaC6vJH2prA
```

Figura 3.7: Ejemplo de Job-status especifico

En el cuadro 3.1 se muestran los posibles estados de un Job y su definición.

Estado	Definición
Submitted	El Job ha sido enviado por el usuario, pero aun no ha sido transferido en la red.
Waiting	El Job ha sido transferido pero aun no ha sido procesado por el Workload Manager.
Ready	El Job ha sido asignado a un Computing Element pero aun no ha sido enviado.
Scheduled	El Job esta esperando en la cola del Computing Element.
Running	El Job esta siendo ejecutado.
Done	El Job a finalizado.
Aborted	El Job ha sido abortado por el WMS.
Cancelled	El Job ha sido cancelado por el usuario.
Cleared	El Output Sandbox ha sido transferido al User Interface.

Cuadro 3.1: Posibles Estados de un Job

Salida de los Jobs

Cuando el resultado de *glite-Job-status* es Done(Success), pueden ser recuperados usando el comando *glite-wms-Job-output*, como se muestra en la figura 3.8.

```
[esther@grid-ccpd03 esther]\$ glite-wms-Job-output -i jobid
_____________________________________________________________________

2 : https://grid-ccpd01.ucv.ve:9000/KFFCg4B0pfAs-ylYvXJ5BA
_____________________________________________________________________

Choose one or more jobId(s) in the list - [1-2]all (use , as separator or - for a range): 2
Connecting to the service https://grid-ccpd01.ucv.ve:7443/glite\_wms\_wmproxy\_server
================================================================================
                            JOB GET OUTPUT OUTCOME
Output sandbox files for the DAG/Collection :
https://grid-ccpd01.ucv.ve:9000/KFFCg4B0pfAs-ylYvXJ5BA
have been successfully retrieved and stored in the directory:
/tmp/jobOutput/despinozac\_KFFCg4B0pfAs-ylYvXJ5BA
================================================================================
```

Figura 3.8: Comando Job-output

El directorio donde se guarda la salida, puede ser cambiado por el usuario haciendo uso de la opción *–dir*, como se muestra en la figura 3.9.

```
[esther@grid-ccpd03 esther]\$ glite-wms-Job-output -i jobid --dir jobdir
_____________________________________________________________________

2 : https://grid-ccpd01.ucv.ve:9000/KFFCg4B0pfAs-ylYvXJ5BA
_____________________________________________________________________

Choose one or more jobId(s) in the list - [1-2]all (use , as separator or - for a range): 2
Connecting to the service https://grid-ccpd01.ucv.ve:7443/glite\_wms\_wmproxy\_server
================================================================================
Output sandbox files for the DAG/Collection :
https://grid-ccpd01.ucv.ve:9000/KFFCg4B0pfAs-ylYvXJ5BA
have been successfully retrieved and stored in the directory:
/tmp/jobOutput/jobdir
================================================================================
```

Figura 3.9: Ejemplo de Job-output

3.5. Implementación del Grid en la UCV

En esta sección se describirá de forma general la infraestructura de Grid instalada en la UCV. En el cuadro 3.2 se muestran los componentes de la infraestructura del Grid en la UCV.

Modelo Maquina	Servicio	Sistema Operativo	Características		
			CPU	Memoria	Disco Duro
PC	UI	Scientific Linux SL_4.7	Pentium III 1 Ghz	512 MB	40 GB
SUN WorkStation	WMS	Scientific Linux SL_4.7	2xAMD Opteron 2 GHz	2GB	72 GB
SUN WorkStation	CE	Scientific Linux SL_4.7	2xAMD Opteron 2 GHz	2GB	72 GB
SUN WorkStation	SE	Scientific Linux SL_4.7	2xAMD Opteron 2 GHz	2GB	72 GB
SUN Fire Z80	WN	Scientific Linux SL_4.4	2xAMD Opteron 2.2 GHz	4GB	72 GB
PC	BDIITOP	Scientific Linux SL_4.4	Celeron 1.7 Ghz	2 GB	40 GB

Cuadro 3.2: Infraestructura de Hardware

La figura 3.10 muestra los componentes gLite instalados en el Grid de la UCV.

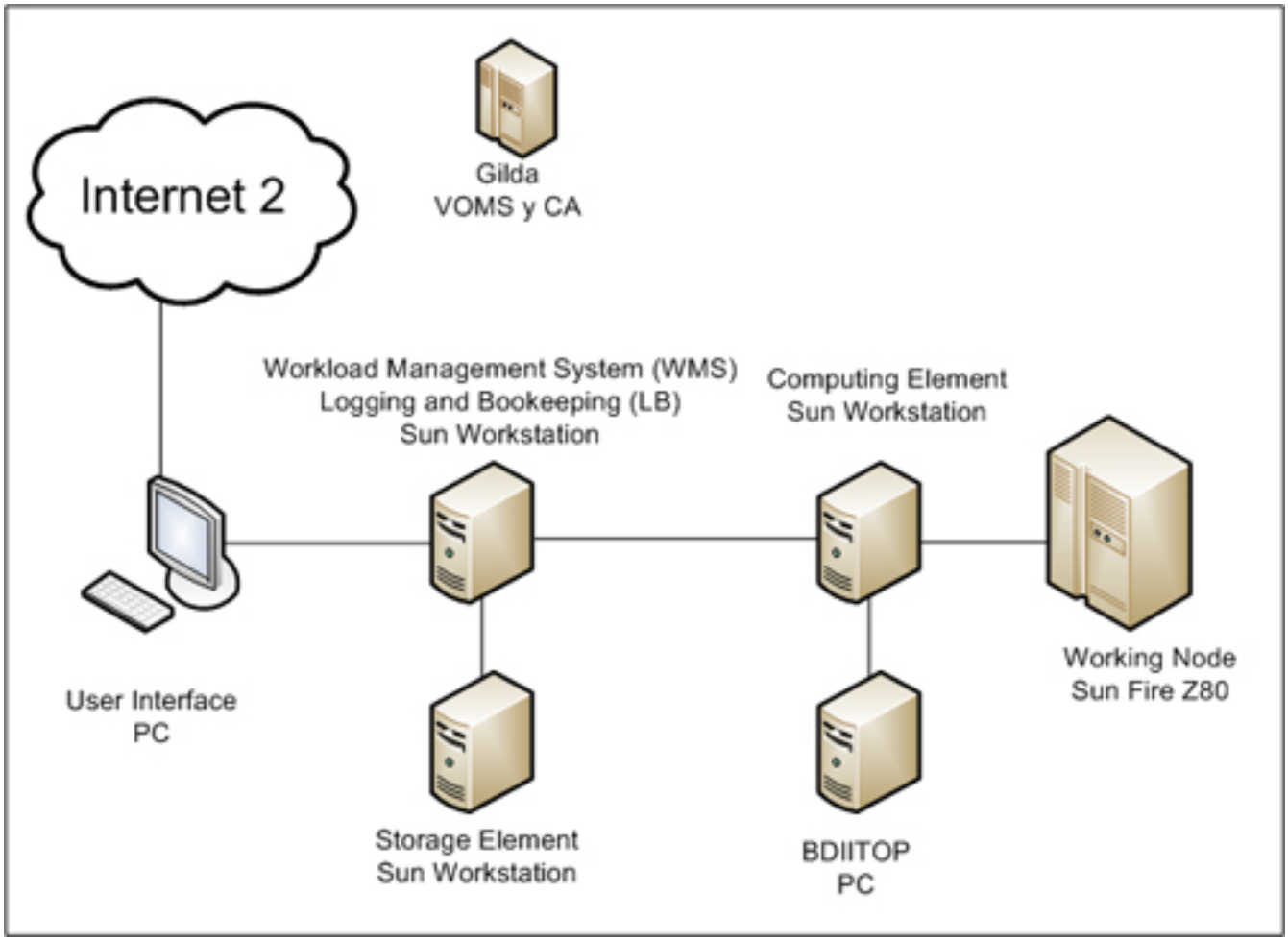

Figura 3.10: Componentes gLite en el Grid UCV

El User Interface (UI) es el punto de entrada al Grid, puede ser parte de la estación de trabajo del usuario. Sus responsabilidades son:

- Autenticación y Autorización de usuarios.
- Guarda los certificados del usuario.
- Generación del proxy de entrada al Grid.
- Proporcionar un CLI al usuario del Grid.

Proceso de instalación del UI. Para la instalación del UI se siguen los siguientes pasos:

1. Instalación de SL_4.7.
2. Establecer hostname de la máquina.
3. Configuración de NTP.
4. Descargar los repositorios de paquetes requeridos para el componente.
5. Instalación de java.
6. Instalación de los paquetes de bases gLite.

El Workload Management System (WMS) es el componente gLite que permite a los usuarios enviar Jobs, y realizar todas las tareas requeridas para su ejecución, sin exponer al usuario a la complejidad de el Grid. El WMS comprende un conjunto de componentes Grid middleware responsables de la distribución y manejo de tareas a través de los recursos del Grid. Sus responsabilidades son:

- Aceptar y satisfacer los Jobs de usuarios.
- Enviar Jobs a los CE que cumplan con los requerimientos del Job.

Proceso de instalación del WMS. Para la instalación del WMS se siguen los siguientes pasos:

1. Instalación de SL_4.7.
2. Establecer hostname de la máquina.
3. Configurar NTP.
4. Descargar los repositorios de paquetes requeridos para el componente.
5. Instalar java y autoridades certificadoras (yum).

6. Instalar paquetes WMS.

7. Adaptar archivo de configuración global (site-info.def).

8. Configurar usando yaim.

El Computing Element es el servicio central de un site. Representa un recurso de computo en el Grid, sus responsabilidades son:

- Gestionar Jobs (submission, control).
- Informar al WMS sobre el status de los Jobs (específicamente al componente L&B).
- Publicar la información del sitio (localización, colas, status de los CPUs, etc) a través del BDII.

Proceso de instalación del CE. Para la instalación del CE se siguen los siguientes pasos:

1. Instalación de SL_4.7.

2. Establecer hostname de la máquina.

3. Configurar NTP.

4. Descargar los repositorios de paquetes requeridos para el componente.

5. Instalar java y autoridades certificadoras (yum).

6. Instalar paquetes CE_torque y BDII (yum).

7. Instalar paquete gilda_utils (yum).

8. Adaptar archivo de configuración global (site-info.def).

9. Configurar usando yaim.

El Worker Node es una máquina donde efectivamente los Jobs son ejecutados, sus responsabilidades son:

- Ejecutar los Jobs.
- Informar al CE sobre el status de los Jobs.

Proceso de instalación del WN. Para la instalación del WN se siguen los siguientes pasos:

1. Instalación de SL_4.7
2. Establecer hostname de la máquina.
3. Configurar NTP.
4. Descargar los repositorios de paquetes requeridos para el componente.
5. Instalar java y autoridades certificadoras (yum).
6. Instalar paquetes WN_torque.
7. Instalar paquete gilda_utils y gilda_applications(yum).
8. Adaptar archivo de configuración global (site-info.def).
9. Configurar usando yaim.

En el siguiente capítulo se describen en detalle las aplicaciones utilizadas para ser adaptadas en el Grid computacional de la UCV.

Capítulo 4

Aplicaciones de Alto Rendimiento

Existen diversos grupos de investigación e institutos gubernamentales de nuestro país, que ejecutan aplicaciones de alto desempeño. Estas aplicaciones requieren de un gran poder de computo y capacidad de almacenamiento, por lo tanto, son ideales para ejecutarse en un ambiente Grid.

Las aplicaciones que serán utilizadas para adaptarse al ambiente grid son AIRES[22] utilizada por el departamento de física de la UCV y BRAMS[10] utilizada por el Instituto Nacional de Meteorologia e Hidrologia(INAMEH)[23].

A continuación se describen las aplicaciones AIRES y BRAMS.

4.1. AIRES

AIRES fue desarrollado por S. J. Sciutto perteneciente al Departamento de Física de la Universidad Nacional de la Plata, Argentina. Esta escrito completamente en FORTRAN y consiste en mas de 730 rutinas.

Descripción

AIRES (AIR-Shower Extended Simulations) es un conjunto de programas y subrutinas, para simular lluvias de partículas producidas despues de la incidencia de rayos cósmicos de alta energía en la atmósfera de la tierra, y para manejar todos los datos de salida asociados a la simulación. AIRES ofrece un espacio temporal para la propagación de partículas en un entorno realista, donde las características de la atmósfera, el campo geomagnético y la curvatura de la tierra se tienen en cuenta adecuadamente. Un procedimiento de muestreo estadístico es utilizado cuando el número de partículas en la lluvia es muy grande[5].

Las partículas que son tomadas en cuenta por la simulación de AIRES son: gammas, electrones, positrones, muones, piones, los kaones, mesones de ETA, lambda bariones, nucleones, antinucleones. La partícula primaria puede ser cualquiera de las mencionadas, con un rango de energías que va desde $1GeV$ hasta mas de $1ZeV$[1].

Manejo de la Simulación

Hay muchos parámetros que deben especificarse antes y durante la simulación. El IDL es una parte del sistema AIRES y consiste en unas 70 directivas que permitan un control eficaz de las simulaciones en un entorno amigable. Las directivas más comunes del IDL se describen en este capítulo.

Tareas, procesos y corridas

La simulación de lluvias de alta energía requiere de un uso intensivo de CPU que puede exigir días e incluso semanas de tiempo de procesador para poder ser completadas. El programa AIRES fue diseñado teniendo en cuenta este hecho por lo que incluye un mecanismo de “auto-guardado”, que guarda periódicamente en un archivo IDF (Internal Dump File) todos los datos relevantes de

[1]El símbolo eV, es una unidad de energía equivalente a la energía cinética que adquiere un electrón al ser acelerado por una diferencia de potencial en el vacío de 1 voltio. $1GeV = 10^9eV$

la simulación. En el caso de una falla del sistema, el proceso de simulación puede ser reiniciado a partir del ultimo “auto-guardado”, evitando así la perdida de todo el tiempo de simulación que había sido realizado, el procesamiento que existe entre dos auto-guardados consecutivos es llamado una *corrida*.

Una *tarea* representa un trabajo de simulación específico, definido en el IDL (por ejemplo, en una tarea se puede “simular diez lluvias de protones”), y como *proceso* identificamos a un proceso del sistema, que comienza cuando se invoca a AIRES y termina cuando se devuelve el control al sistema operativo.

Una tarea puede ser completada después de uno o más procesos, y puede haber una o más corridas en un proceso. El caso mínimo consiste en una tarea que termine en una sola corrida (sin “‘auto-guardado”) completada en un único proceso.

Input Directive Language (IDL)

AIRES lee sus directrices de entrada desde la consola, y usa un lenguaje común para recibir las instrucciones del usuario. Este lenguaje es llamado Input Directive Language (IDL). Estas instrucciones están escritas en un formato libre, conteniendo una instrucción por linea, y no pueden existir mas de 176 caracteres por linea.

Todas las instrucciones son analizadas, hasta que se llegue a un fin de archivo o una instrucción de finalización. La mayoría de las instrucciones se pueden colocar en cualquier orden dentro del archivo de entrada.

Ejemplo de un IDL Existen cuatro instrucciones que siempre deben ser definidas en el IDL antes de iniciar una simulación, el nombre de la tarea, la especificación del la partícula primaria, la energía, y el total de lluvias que serán simuladas.

Estas instrucciones pueden ser especificadas en un IDL de la siguiente manera:

```
Task ejemplo
Primary proton
PrimaryEnergy 150 TeV
TotalShowers 3
End
```

Parámetros de Entrada

Los parámetros de entrada de AIRES están divididos en secciones, de acuerdo con los diferentes tipos de variables que controlan el entorno computacional y físico de la simulación.

- **Control de Corrida:** Incluye todos los parametros que controlan las condiciones de la simulación, es decir, el numero total de lluvias, el numero de lluvias por corrida, el numero de corridas por proceso y el tiempo de CPU por corrida (El tiempo máximo). Las directivas que controlan estas variables son dinámicas, y por lo tanto pueden variar durante la simulación.

- **Nombre de Archivos:** A continuación se muestra una lista con los nombres de los archivos que se crean durante la simulación y una breve descripción:

  Archivos de Registro (<nombretarea>.lgf) : Este archivo tiene información sobre los acontecimientos que se presentaron durante las simulaciones. Contiene también, un resumen de los parametros de entrada. La mayoría de los datos incluidos en este archivo, también se escriben en la salida estándar.

  Archivo de Resumen(<nombretarea>.sry) : Incluye un resumen de las salidas. Por lo general datos de la simulación y todas las tablas que se imprimieron usando las directivas PrintTables del IDL.

  Archivos de datos de Exportación(<nombretarea>.tnnnn): Archivos de texto plano que contiene tablas de salida.

Archivo Script de Resumen de Tareas(<nombretarea>.tss) : Contiene un resumen de los datos de entrada y salida, escrito en un formato legible para otros programas.

Archivo de volcado binario(<nombretarea>.idf) : Este archivo contiene (en formato binario) todos los datos referentes a la simulación. Se actualiza periódicamente, durante el procesamiento de las tareas. Es usado para obtener datos relevantes de la simulación cuando esta ha culminado o durante la misma.

Archivo de volcado ASCII(<nombretarea>.adf) : Versión portable del archivo IDF, escrito al finalizar la tarea.

Archivos Comprimidos de Salida(<nombretarea>.grdpcles) : Estos archivos contienen datos detallados de las partículas. Gracias a la compresión utilizada para los datos, es posible salvar un gran número de registros de partículas usando una cantidad moderada de espacio en disco. El formato es universal, por lo que el archivo puede ser escrito en una maquina y procesado en otra maquina diferente.

Parámetros Básicos : Son los parámetros geométricos y físicos de las lluvias. Estas variables definen las condiciones iniciales de las simulaciones (partícula primaria, eje de inclinación, etc).

Parámetros Adicionales : Son otros parámetros de la lluvia, generalmente dependendientes del modelo usado.

Directivas IDL

En esta sección se definen las variables relacionadas con el control de procesos y ejecución. Por defecto estas variables (lluvias por corrida, numero de ejecuciones por proceso, y el tiempo de CPU por ejecución) tienen asignadas el valor “infinito”. Con esta asignación, el mecanismo de autoguardado para tolerancia a fallos estaría deshabilitado.

Esto puede ser aceptable para una simulación corta en un sistema de computación fiable. Para tareas mas pesadas es recomendable dividir la simulación en varios procesos. Vale la pena men-

cionar que las operaciones de autoguardado y restauración no alteran los resultados de las simulaciones.

Control de Ejecución Las directivas de IDL ShowersPerRun, MaxCpuTimePerRun y RunsPerProcess proveen un efectivo control sobre las condiciones de cálculo de las simulaciones. Los siguientes ejemplos ilustran cómo pueden ser utilizados.

```
RunsPerProcess 1
MaxCpuTimePerRun 2 hr
```

Estas dos instrucciones indican que debe comenzar una nueva ejecución cada dos horas de CPU. Como el numero de ejecuciones por procesos es 1, una nueva ejecución implicara también el comienzo de un nuevo proceso, en otras palabras, el archivo de entrada sera explorado cada dos horas de CPU, teniendo en cuenta posibles cambios en los parametros dinámicos de la simulación.

```
RunsPerProcess 4
ShowersPerRun 5
```

Aquí el tiempo máximo de CPU no está definida, lo que indica que no habrá límite de tiempo para que termine una ejecución. En su lugar, cada ejecución va a terminar después de concluir las simulaciones de cinco lluvias. El proceso terminará cuando se completen cuatro ejecuciones. Estas tres directivas pueden ser usadas simultaneamente:

```
RunsPerProcess 2
ShowersPerRun 2
MaxCpuTimePerRun 6 hr
```

Estas instrucciones indican que una ejecución terminara despues de seis horas de procesamiento o despues de completar dos lluvias, lo que suceda primero.

Directorios Usados por AIRES

El programa de simulación lee y escribe diversos archivos que contiene diferentes tipos de datos. Por defecto, todos los archivos generados por AIRES están ubicados en el directorio de

trabajo, definido como el directorio actual al momento de realizar la invocación de AIRES.

Cuando se desee cambiar la ubicación de los archivos generados por AIRES, se puede hacer uso de un conjunto de instrucciones del IDL que permiten controlar la localización de los archivos de AIRES. A continuación definiremos los diferente directorios usados por AIRES durante la simulación.

- **Globales:** Contienen los archivos de registro, IDF, ADF y archivos de resumen.
- **Comprimidos:** Contiene los archivos de salida comprimidos. Algunas veces es llamado simplemente directorio de salida.
- **Exportación:** Contiene toda los archivos de datos exportados.
- **Temporales:** Contiene la mayoría de los archivos internos que se generan durante las simulaciones.

Cuando no se especifica lo contrario, los archivos comprimidos y temporales son localizados en el directorio actual.

Por otro lado, los archivos exportados y globales son guardados por defecto en el directorio de salida. La directiva IDL FileDirectory permite un control total sobre los directorios usados durante la simulación de AIRES. Por ejemplo, las siguientes instrucciones indica cual sera el directorio temporal y el de exportación. Las especificaciones de directorios pueden ser direcciones absolutas o relativas, las direcciones relativas serán con respecto al directorio de trabajo.

```
FileDirectory Scratch /mytmpdir
FileDirectory Export /myexportdir
```

La siguiente instrucción especifica simultaneamente los directorios de salida, exportación y global.

```
FileDirectory All /mydir
```

Hay un conjunto adicional de directorios que pueden ser especificadas durante la lectura de los datos de entrada. Las siguientes instrucciones, por ejemplo:

```
InputPath /dir1:/dir2
InputPath Append /dir3
Input myinputfile.inp
```

le indican a AIRES que busque el archivo <miarchivo>.inp en los tres directorios especificados, usando la directiva InputPath, en caso de no conseguirlo en los tres directorios especificados, lo buscara en el directorio actual.

Sistema de Simulación AIRES

El sistema de simulación AIRES funciona solo en plataformas UNIX, provee herramientas para la revisión de los archivos de entrada, procesamiento secuencial y concurrente y registro de eventos.

Hay algunos parametros que modifican el comportamiento del sistema de simulación AIRES, la mayoría pueden ser configuradas por el usuario en el archivo de inicialización `.airesrc`, en la instalación estadar de AIRES este archivo estará ubicado en el directorio /home del usuario.

Revisión de los Datos de Entrada Las directivas IDL CheckOnly y Trace son usadas para indicar a AIRES que revise los archivos de entrada. El siguiente comando invocara a AIRES con el archivo de entrada <miarchivo>.inp.

```
$ airescheck -t <miarchivo>.inp
```

Manejo de Tareas Para iniciar la simulación se hace uso del comando :

```
$ airestask <miarchivo>.inp
```

este comando comprueba en primer lugar que el archivo <miarchivo>.inp existe, y luego crea una entrada en la cola correspondiente del sistema de simulación. Por último sera ejecutado el script *aireslaunch*.

El script *aireslaunch* detectará que hay una tarea pendiente por realizar por lo que preguntara al usuario si desea iniciar las simulaciones. En caso de respuesta positiva, el programa de simulación se iniciara con la entrada correspondiente, y se invocara repetidamente si es necesario hasta que la tarea sea completada. Todas las operaciones son completamente automáticas, sin intervención del usuario.

El siguiente comando creara una nueva entrada en la cola, y sera encolada. La ejecución de esta tarea se iniciará tan pronto como la anterior haya terminado. No hay límite en el número de tareas que pueden se encoladas en la cola del sistema de simulación.

```
$ airestask <mi_otro_archivo>.inp
```

Tareas Concurrentes En muchos casos es necesario procesar simultáneamente más de una tarea. El sistema de simulación AIRES proporciona ciertas herramientas para trabajar en estas circunstancias. La idea clave es definir más de una cola, y asignar a cada CPU una de ellas.

En los ejemplos anteriores, el comando *airestask* es invocado sin especificar ninguna cola. La cola por defecto es utilizada en caso de no especificarlo. En la configuración estándar hay 9 colas predefinidas, nombrados respectivamente, $1, 2, \cdots, 9$. La cola 1 es el valor por defecto. El comando

```
$ airestask -s 2 <myarchivo>.inp
```

creará una entrada en la cola 2.

El siguiente comando indicara que existe una simulación corriendo en la cola 2.

```
$ airesstatus 2
```

Instalación de AIRES

AIRES es actualmente distribuido solo para ambientes UNIX. El programa puede ser descargado de $http://www.fisica.unlp.edu.ar/auger/aires/$. En las plataformas UNIX se debe tener en cuenta, una vez descomprimido el paquete, los siguientes puntos:

1. El script doinstall instalara automáticamente el software.
2. El archivo config contiene todas las variables personalizables. Se debe editar este archivo antes de ejecutar doinstall.
3. Existirán dos directorios principales:

   Directorio de instalación raíz (denominado Iroot), es el directorio donde se ha descargado el archivo de la distribución (el directorio que contiene el script doinstall).

   El directorio raíz de AIRES (denominado Aroot), es el directorio de más alto nivel para los archivos instalados. El valor predeterminado sera en el home del usuario. Los directorios Iroot y Aroot puede o no ser el mismo directorio.
4. Se debe tener instalado compiladores de C (cc, gcc, etc) y fortran 77 (f77, fort77, etc), estos compiladores deben estar correctamente asignados al PATH del usuario.

Procedimiento de Instalación: Para instalar la aplicación se siguen los siguientes pasos:

1. Se deben tener permisos de escritura en los directorios Iroot y Aroot, y en todos sus subdirectorios.
2. Entrar al directorio Iroot, y editar el archivo de configuración. Establecer todas las variables necesarias, de acuerdo al tipo de instalación deseada. Es obligatorio seleccionar una y sólo una plataforma.
3. Ingresar el siguiente comando si se esta instalando AIRES por primera vez:

```
$ doinstall 0
```

o el siguiente comando si se esta haciendo una reinstalación del software

```
$ doinstall 1
```

Se iniciara la instalación del software, utilizando los datos que se establecieron en el paso 2. Esto puede tomar algunos minutos. Se mostrara un mensaje en el terminal indicando si la instalación fue o no exitosa.

4. Ingresar el siguiente comando para verificar si el programa se está ejecutando y esta correctamente configurado en el PATH del usuario.

```
$ AIRES
```

La salida del comando debe ser similar a :

```
>>>>
>>>> This is AIRES version V.V.V (dd/MM/yyyy)
>>>> (Compiled by . . . . .
>>>> USER: uuuuu, HOST: hhhhhhh, DATE: dd/MM/yyyy
>>>>
> dd/Mmm/yyyy hh:mm:ss. Reading data from standard input unit
```

5. Entrar al home del usuario y verificar la existencia del archivo .airesrc.

Si estos pasos son completados satisfactoriamente, AIRES esta instalado en su sistema.

4.2. BRAMS

BRAMS (Brazilian Regional Atmospheric Modeling System)[10] fue desarrollado por un conjunto de instituciones; ATMET (Atmospheric Meteorological and Environment Technologies) [11] , IME/USP (Instituto de Matemática y Estadística de la Universidad de Sao Paulo) [12], IAG/USP (Instituto de Astronomía Geofísicas y Ciencias Estadísticas de la Universidad de Sao Paulo) [13] y CPTEC/INPE (Centro de Predicción del Tiempo y Estudios Climatológicos) [14], financiado por FINEP (Brazilian Funding Agency) [15], con el propósito de crear una nueva versión del modelo RAMS (Regional Atmospheric Modeling System) [16], para ser usada en trópicos.

BRAMS es un modelo de mesoescala, en estos modelos de predicción numérica del tiempo, la resolución horizontal y vertical es suficiente para pronosticar fenómenos meteorológicos de mesoescala. Dichos fenómenos, que a menudo son producto del forzamiento de la topografía o de los litorales, o están relacionados con la convección, presentan algunos de los mayores retos a la hora de hacer el pronóstico. Los fenómenos climáticos severos, incluidos los tornados. La visibilidad, la turbulencia, el tiempo que percibimos y el estado del mar pueden variar enormemente en distancias de pocos kilómetros y sus repercusiones pueden ser enormes [16].

Instalación de BRAMS en un sistema Linux

En la pagina web[10] del BRAMS existen dos versiones del modelo, una version basica y otra para usuario avanzados.

La versión basica contiene un archivo binario ejecutable que ejecutara de manera secuencial el BRAMS en computadoras con un sistema linux. Esta versión incluye todos los datos de entrada requeridos para la ejecución del modelo, así como la salida esperada, para que el usuario pueda verificar la correcta instalación del modelo.

La versión para usuarios avanzados sólo contiene el código fuente y las instrucciones para la compilación, esta versión soporta la ejecución en paralelo del modelo.

Para iniciar la instalación de BRAMS es necesario descomprimir el paquete descargado previamente:

```
$ tar -xzvf brams4.2-serial.tar.gz
```

Luego de ser descomprimido es creada una estructura de directorio cuya raíz es el directorio de inicio de BRAMS. Para verificar la instalación, se ejecuta BRAMS por primera vez:

```
$ ./runBRAMS.sh
```

El script runBRAMS.sh ejecuta los siguientes pasos:

- Convertir archivos globales CPTEC en un formato legible por BRAMS.
- Generar archivos de superficie para el área deseada.
- Generar las condiciones iniciales para el área deseada.
- Ejecutar de forma secuencial la generación del pronostico.
- Postprocesamiento y generación de imágenes.

Para cada uno de estos pasos, se mostrará una salida por la consola. Si la ejecución se completa con éxito, se mostrará por consola el siguiente mensaje:

```
$ "BRAMS execution ends successfully"
```

y deben generarse dos imágenes .gif, topo.gif y temp.gif dentro del directorio RAMPOST60, estas imagenes se muestran a cntinuación:

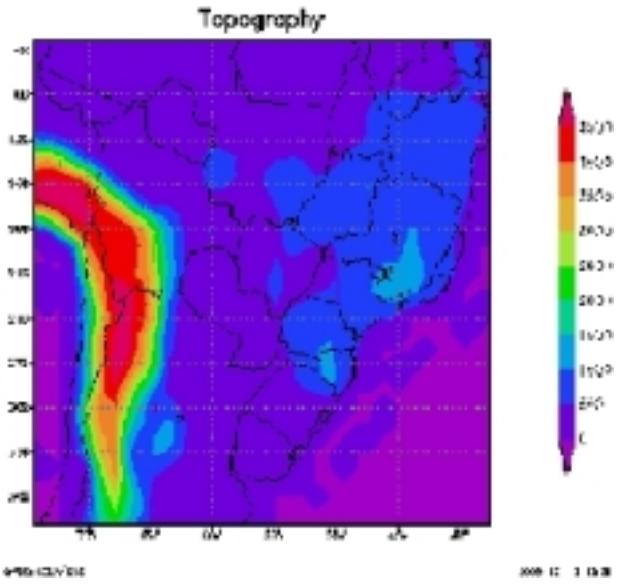

Figura 4.1: Imagen del archivo Topo

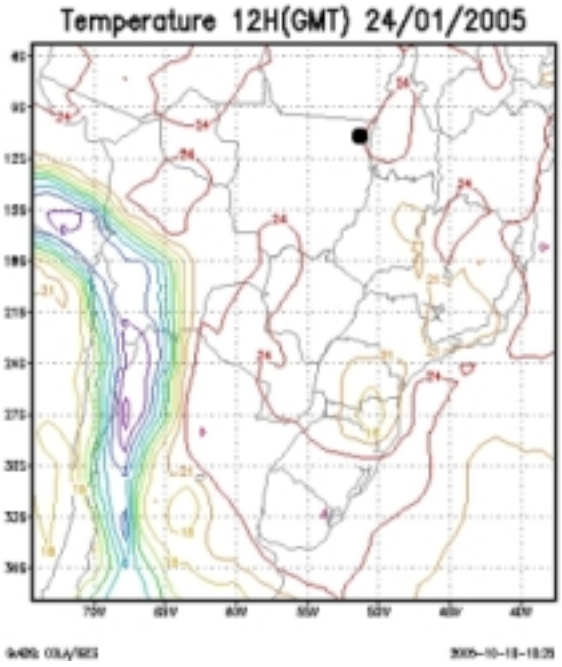

Figura 4.2: Imagen del archivo temp

Ejecución de BRAMS

Es fundamental para entender cómo se realiza la ejecución de BRAMS conocer el archivo Ramsin. Ramsin es el archivo de entrada con las especificaciones para la ejecución de BRAMS.

El script runBRAMS.sh se ejecuta tres veces, cada una con un archivo Ramsin específicos:

- RAMSIN_sfc genera archivos de superficie.
- RAMSIN_vfile genera los limites y condiciones iniciales.
- RAMSIN_initial genera el pronostico.

Estos tres archivos sólo difieren en el modo de ejecución.

El primer paso para la ejecución de BRAMS es transformar los archivos que contienen el estado del tiempo de la atmósfera pronosticado por el CPTEC en un formato de archivo que entienda BRAMS. CPTEC produce archivos en formato GRIB que contienen el estado de la atmósfera en una grilla Gaussiana, que abarca América del Sur. BRAMS no entiende este formato de archivo, el espera un archivo ASCII con una grilla regular que contenga el estado de la atmósfera. La transformación de los archivos se realiza con grib2_to_dp.x.

El segundo paso es generar los archivos con la topografía, la temperatura superficial del mar y cubierta vegetal para la zona deseada. El script realiza este paso mediante la ejecución de BRAMS, guiándose por el archivo RAMSIN_sfc. Los archivos de entrada son los que están ubicados en los directorios SST, topo10km, Topo y veg_usgs. Los archivos resultantes son almacenados en el directorio de datos. Estos archivos tienen el formato SST-Brams-W-XXXX, NDVI-Brams-N-AAAA, SFC-Brams-X-ZZZZ y Toph-Brams-S-wwww.

El tercer paso es generar el estado inicial de la atmósfera y condiciones de frontera, durante el tiempo de pronostico para la zona deseado. El script realiza este paso mediante la ejecución de BRAMS, guiándose por el archivo de entrada RAMSIN_vfile. Los archivos de entrada para este paso son los archivos de salida de la ejecución de grib2_to_dp, BRAMS lee estos archivos, reduce

su contenido al área deseada y genera los archivos ivbrams-V-ZZZZ, que son almacenados en el directorio ivar.

El cuarto paso es realizar el pronóstico para el área deseada y por un periodo de tiempo limitado. El script realiza este paso mediante la ejecución de BRAMS, haciendo uso del archivo de entrada RAMSIN_initial.

Existen dos tipos de salidas de BRAMS, histórica y de análisis:

- Los archivos históricos se almacenan en el directorio H, que contiene toda la información necesaria para los archivos de control de BRAMS. Estos son nombrados hist-H-AAAA.
- Los archivos de análisis se almacenan en el directorio que contiene el estado de la atmósfera, según lo pronosticado por BRAMS. Estos se denominan anal-A-ZZZZ.

Post Procesamiento y Generación de Imágenes

Los archivos de análisis no son adecuados para la interpretación humana. La fase de post procesamiento selecciona los campos que se deseen en los archivos de análisis y genera archivos adecuados para la visualización.

El post procesamiento es realizado por el ejecutable *ramspost60.x* ubicado en el directorio RAMSPOST60, los archivos de entrada son los archivos de análisis. Los archivos de salida son los result_XXX ubicados en el directorio RAMSPOST60.

La generación de imágenes es realizada por el programa Grads, teniendo como archivos de entrada result_XXX generando los archivos de salida topo.gif y temp.gif, todos en el directorio RAMSPOST60. La figura 5.3 muestra el flujo de ejecución de BRAMS.

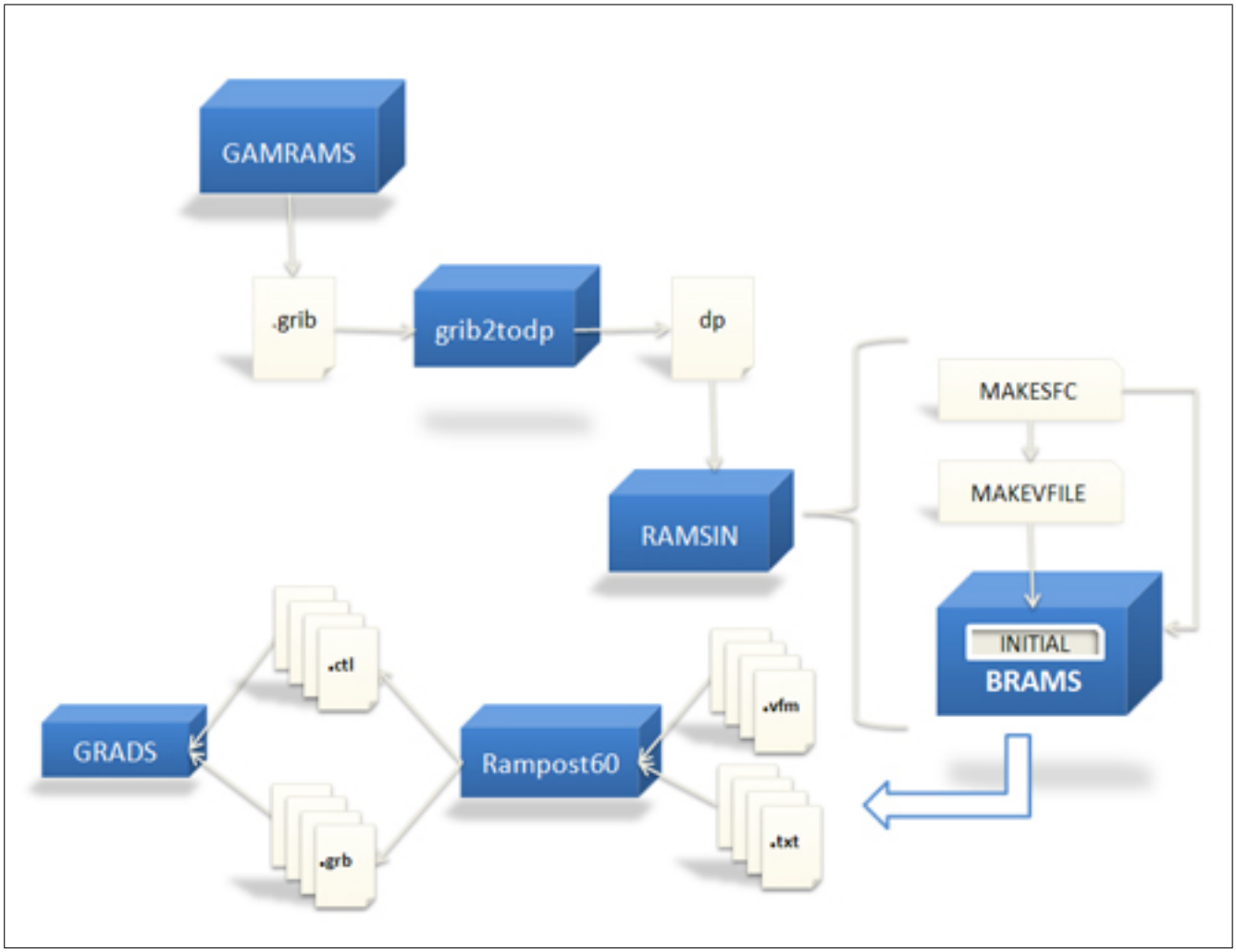

Figura 4.3: Flujo de ejecución de BRAMS

GBRAMS

GBRAMS (Climatology Studies Using BRAMS in a Grid Computing Environment) es un proyecto conformado por tres grupos de investigación (LAC/INPE, CPTEC/INPE and II/UFRGS), cada uno de ellos cuenta con un cluster dedicado a realizar las pruebas necesarias. Los propósitos de este proyecto son:

- Evaluar la viabilidad del uso de la infraestructura Grid para la ejecución de modelos numéricos meteorológicos.
- Generar tres años de modelaje meteorológico usando el BRAMS.
- Los datos generados serán para tres regiones de Brasil: Norte, Noreste y Sur/Sureste.
- Compara tres middleware: Globus, OAR/CIGRI y OURGRID.

La interacción entre el meteorólogo y el Grid es realizada a través de un portal web, cuyo objetivo es permitir el envió de trabajos y la recuperación de los resultados. Cada nuevo trabajo es almacenado en una base de datos. El scheduler está encargado de buscar los trabajos en la base de datos y ejecutarlos en una maquina del Grid. Después de la ejecución, los resultados estarán disponibles en el portal web.

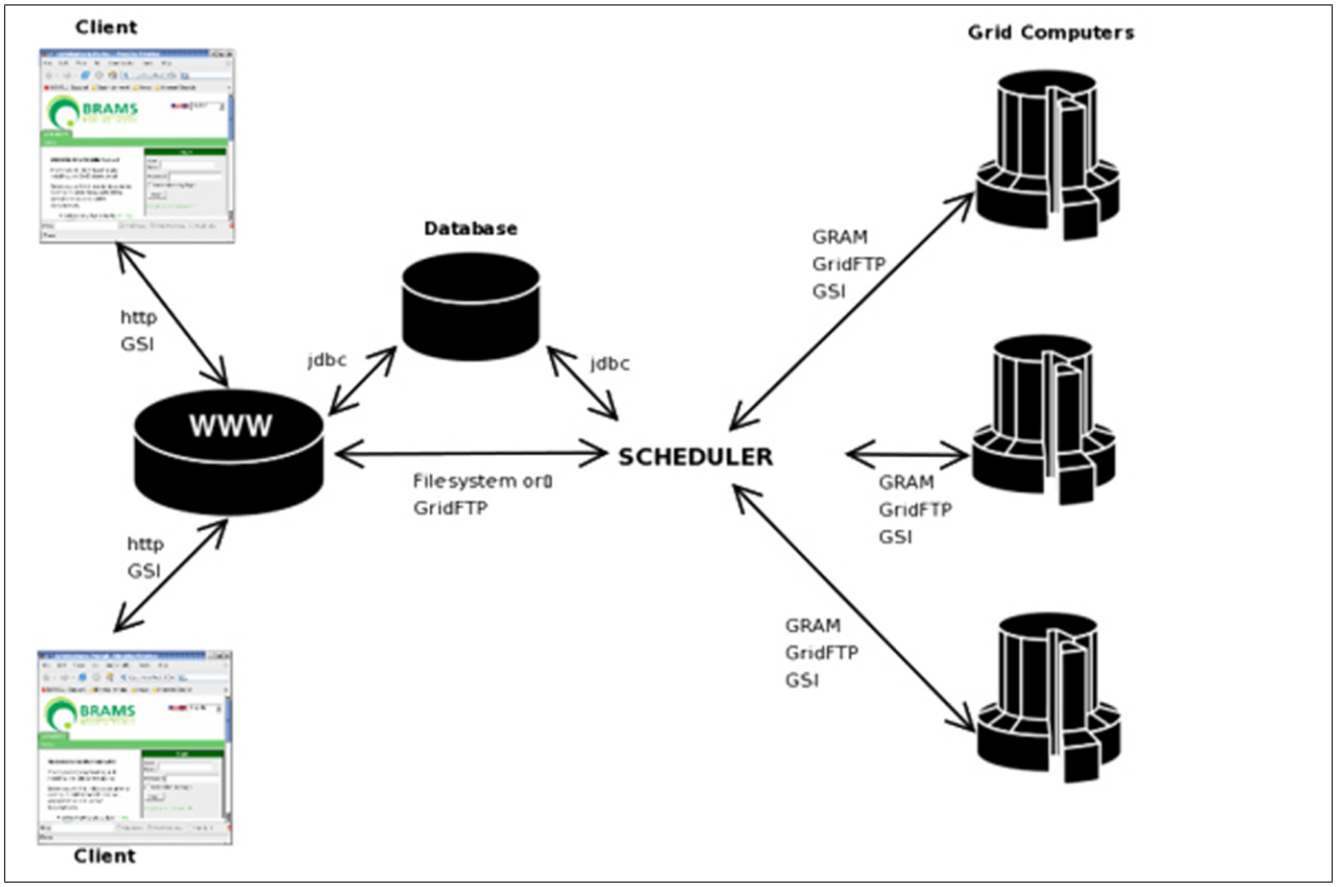

Figura 4.4: Proyecto GBRAMS

El proyecto escogerá uno de los tres middlewares probados, para ser usado en los tres nodos del Grid (cluster en LACA/INPE, CPTEC/INPE e II/UFRGS). Se espera que el proyecto GBRAMS valide el uso de un ambiente Grid para realizar un alto número de simulaciones en extensiones geográfica usadas para estudios climatológicos. En la figura 4.4 se muestra el flujo de ejecución de GBRAMS.

En el capítulo 4 se describe en detalle los pasos seguidos para la adaptación de AIRES y BRAMS en la plataforma Grid de la UCV.

Capítulo 5

Adaptación de Aplicaciones en el Grid

5.1. AIRES en el Grid

En este trabajo ha sido adaptado el sistema de simulación AIRES a un ambiente Grid, permitiendo a los usuarios realizar simulaciones teniendo una mínima interacción con el middleware gLite.

Se ofrecen dos alternativas para realizar la ejecución, una por linea de comando, ejecutando un script, o mediante una pagina web, que se encargara de llamar a los scripts necesarios.

Scripts de ejecución en el Grid UCV

Para poder realizar las simulaciones, el usuario deberá ejecutar un script en la consola de linux.

Los script que realizan todas las tareas necesarias, para ejecutar Aires en el grid son:

- **script_aires.sh**: Script encargado de generar el JDL, enviar el trabajo al grid, consultar status, y guardar la salida en el UI. Este script es ejecutado en el UI.
- **aires_nodo.sh**: Script encargado de ejecutar la simulación en el WN asignado. La forma en

la que se ejecuta es especificada en el JDL generado por el script_aires.sh.

El código que se mostrara mas adelante, se encarga de la creación del JDL, recordemos que el JDL esta constituido por una serie de atributos que contienen información sobre los requerimientos necesarios por el Job y algunas acciones que deben ser realizadas por el WMS para programar el Job.

Algunos atributos del JDL son obligatorios, en caso de que falte algunos de ellos el WMS no podrá manejar la petición, es por esto, que a pesar de que la realización del JDL puede ser una tarea relativamente sencilla; para evitar errores en los valores asignados a los atributos, y facilitar la tarea de enviar un trabajo al grid, el JDL es creado de forma automática por script_aires.sh, solo es necesario pasarle como argumento el archivo de entrada para la simulación de Aires.

La ejecución del script se realiza de la siguiente manera:

```
\$ ./script_aires.sh archivo\_entrada.inp
```

Antes de explicar el código del script ejecutado en el UI por el usuario, es importante conocer las principales variables utilizadas, tal como se describe en el cuadro 5.1.

Variable	Descripción
JOBS	Nombre del archivo JDL que se creara, y que sera enviado al WMS.
LOGS	Nombre del de entrada .inp
SHELL	Contiene el estado de éxito o error del ultimo comando ejecutado.
TASKNAME	Nombre de la tarea a ejecutar en el WN, los archivos de salidas de la simulación empezaran con este nombre.
ID	ID del Job enviado.
DIR	Nombre del directorio donde se guardaran las salidas.

Cuadro 5.1: Variables script_aires.sh

El código encargado de realizar la creación del JDL, envío del Job, consulta de status, y recuperación de la salida se muestra a continuacion.

```
cat > $JOBS << EOF
[
 Type = "collection";
 nodes = {
EOF
variable=`echo [` echo $variable>> $JOBS
contador=0
for i in `cat $LOGS`
   do

              TASKNAME=`awk '/Task/ { print $2 }' $i`
              contador=`expr $contador + 1`
              if [ $contador -gt 1 ]
              then
                 variable=`echo ],[`
                 echo $variable>> $JOBS
              fi
              variable1=`echo JobType = \"Normal\";`
              echo $variable1 >> $JOBS
              variable2=`echo NodeName = \"node$contador\";`
              echo $variable2 >> $JOBS
              variable3=`echo Executable = \"/bin/sh\";`
              echo $variable3 >> $JOBS
              variable4=`echo Arguments = \"nodo_aires.sh $i $PARALELO
                 \";`
              echo $variable4 >> $JOBS
              variable5=`echo InputSandbox = {\"nodo_aires.sh\"};`
              echo $variable5 >> $JOBS
```

```
28                variable6=‘echo StdOutput = \"salida$contador\"\;‘
29                echo $variable6 >> $JOBS
30                variable7=‘echo StdError  = \"error$contador\"\;‘
31                echo $variable7 >> $JOBS
32                variable8=‘echo OutputSandbox = \{\"$TASKNAME’.sry’\",\"
                     $TASKNAME’.grdpcles’\",\"salida.txt\",\"
                     salida$contador\",\"error$contador\"\}\;‘
33                echo $variable8 >> $JOBS
34                variable9=‘echo ShallowRetryCount = 1\;‘ echo $variable9
                     >> $JOBS
35
36    done
37  variable=‘echo ]‘
38  echo $variable>> $JOBS
39  variable=‘echo }\;‘
40  echo $variable>> $JOBS
41  variable=‘echo ]‘
42  echo $variable>> $JOBS
```

A continuación se describen los puntos mas importantes de este código:

- Linea #1 se crea un archivo cuyo nombre es el contenido de la variable $JOBS, y se deja abierto para escribir en el.
- Linea #9 se crea un ciclo, que recorrerá todo el archivo $LOG.
- A partir de la linea # 21 y hasta la #37 se realizan las especificaciones de los atributos del JDL.
    - Type: Representa el tipo de requerimiento descrito en el JDL. Sus posibles valores son:

      Job: Un simple Job.

      DAG: Un grafo acíclico de Jobs dependientes.

Collection: Un conjunto de Jobs independientes.

- JobType: Es un string que especifica el tipo de Job descrito por el JDL. Sus posibles valores son:

  Normal: Un simple Job.

  Interactive: Un Job cuyo flujo estándar es enviado al cliente que lo envió. (a Job whose standard streams are forwarded to the submitting client)

  MPICH: Un Job ejecutable en paralelo que utilice MPICH.

  Partitionable: Un Job que este compuesto por una serie de pasos independientes, que pueda ejecutarse en paralelo.

  Checkpointable: Un Job capaz de salvar su estado, de modo que su ejecución pueda ser suspendida y reanudada después, desde el mismo punto donde fue detenida.

  Parametric: Un Job cuyo JDL contiene atributos parametrizados.

- NodeName:
- Executable: Representa el comando a ejecutar en el WN, se debe especificar la ruta absoluta del comando.
- Arguments: Son todos los argumentos que se pasaran por linea del comando al momento de ejecutar el Job en el WN.
- InputSandbox: Es una lista de archivos necesarios para la ejecución del Job, estos archivos deben estar ubicados en UI. Los archivos especificados en esta variable estarán disponible en la WN donde se ejecute el Job.
- StdOutput: Este atributo especifica los nombres los archivos donde se guardara la salida del Job. Estos archivos estarán disponibles para el usuario cuando finalice la ejecución del Job, y deben estar especificado en OutputSandbox.
- StdError: En este atributo se especifica el nombre del archivo donde se almacenara la salida de error del Job ejecutado, en caso de que exista alguna.
- OutputSandbox: Es una lista de archivos, que son generados por el Job cuando es ejecutado en el WN, y que estarán disponibles para el usuario cuando termine la ejecución.

- ShallowCount: Es un entero que representa el numero máximo de jobs que serán reenviados en caso de alguna falla en algún componente del grid.

Luego de generar el JDL correspondiente al Job que sera ejecutado, se procede a enviar el Job al WMS.

```
1  SHELL=0
2  glite-wms-Job-submit -d $USER -o jobid $JOBS > submit.txt
3  SHELL=`echo $?`
4  ID=`awk 'NR == 10 { print $1 }' submit.txt`
5  if [ "$SHELL" != 0 ]
6  then
7          echo El comando glite-wms-Job-submit ha fallado
8          exit 1
9  else
10         echo $ID
11         cat submit.txt
12 fi
```

- Linea #2 Se realiza el envio del trabajo, la opción -d especifica el usuario al que fue delegado el proxy, la opción -o especifica que se almacenará el ID del Job en el archivo especificado "jobid", y por ultimo se envía el JDL creado.
- Linea #3 Se obtiene el ID del Job del archivo submit.txt donde fue almacenada la salida del comando anterior.
- Linea #5 Se verifica si el ultimo comando ejecutado (glite-wms-Job-submit) lanzo algún error, en caso de suceder esto se finaliza el programa, en caso contrario se muestra la salida del comando por consola.

Despues de realizar el envío del Job, se realizan consultas de su estado, como se muestra en figura 5.1.

```
1  while [ "$STATUS" != "Done" ]
2  do
3          glite-wms-Job-status --verbosity 0 $ID > status$File
4          SHELL=`echo $?`
5          AUX=$STATUS
6          STATUS=`awk '/Current/ {if(NR < 9) print $3 }' status$File`
7          if [ "$AUX" != "$STATUS" ]
8          then
9                  echo
10                 echo "Status Actual ----> $STATUS"
11         fi
12 done
13 if [ "$SHELL" != 0 ]
14 then
15         echo El comando glite-wms-Job-status ha fallado
16         exit 1
17 else
18         glite-wms-Job-status --verbosity 0 $ID
19 fi
```

Figura 5.1: Consulta de Estado del Job

Este código realiza dentro de un ciclo consultas del estado del Job enviado, el ciclo terminara solo cuando el estado del Job sea “Done” o el comando gLite genere algún error.

- Linea #3 Se realiza la consulta del estado. La opción –verbosity especifica el nivel de detalle en el que se mostrara la salida del comando, este nivel es el de menor detalle posible.
- Linea #6 Se obtiene del archivo status$File (donde fue guardada la salida del comando glite-wms-Job-status), el estado especifico del Job, cuando este estado sea “Done”, se terminara la consulta del estado, y se procederá a obtener la salida.

- Lineas #13 a la #19 En este if se verifica que la salida del comando glite-wms-Job-status no haya fallado, en caso de no fallar se mostrara al usuario la salida completa del comando.

Finalmente, el script obtiene la salida de la ejecución de Aires, con el comando *glite-wms-Job-output*, especificándole con la opción –dir el directorio donde deberá ser almacenado.

```
1  DIR=${ID:32}
2  if [ "$SHELL" == 0 ]
3  then
4          glite-wms-Job-output --dir $DIR $ID
5
6  fi
```

El script ejecutado en el WN, encargado de iniciar la simulación de Aires es el siguiente:

```
1  #!/bin/sh
2
3  cp /usr/local/bin/airesrc /$HOME/.airesrc
4
5  ruta=$PWD
6
7  cp PE* $HOME
8  cd $HOME
9  mkdir aires
10 cp PE* aires
11
12 cd aires
13
14 /usr/local/bin/airesstatus $2 > status.out
15
```

```
STATUS=`awk '/currently/ { print $7 }' status.out`
while [ "$STATUS" != "NOT" ]
do
        AUX=$STATUS
        /usr/local/bin/airesstatus $2 > status.out
        STATUS=`awk '/currently/ { print $7 }' status.out`
        if [ "$AUX" != "$STATUS" ]
        then
                echo
                echo "Status Actual ----> $STATUS"
        fi
done

/usr/local/bin/airesstatus $2

cd ..
/bin/tar -czvf aires.tar.gz aires
cp aires.tar.gz $ruta
```

Este código, se encarga de copiar el archivo airesrc en el HOME del usuario actual, ya que es un requisito para poder ejecutar la simulación, la existencia de este archivo en el HOME del usuario que lo vaya a ejecutar. Luego de esto, se hace un llamado a *airestask* especificándole con la opción -s el numero de procesadores a utilizar. Luego, se realiza un ciclo preguntado el estatus de la ejecución, para así garantizar que *nodo_aires.sh* culminara solo cuando la simulación haya finalizado.

5.2. BRAMS en el Grid

Para adaptar BRAMS al Grid se siguió un procedimiento similar al de AIRES. Se crearon dos scripts, uno a ejecutar en el UI y otro en el WN, el script que se ejecuta en el UI es muy similar al script de AIRES ejecutado en el UI.

Scripts de ejecución en el Grid UCV

A continuación se mostraran y explicaran los fragmentos de códigos mas importantes del script.

```
cat > $JOBS << EOF
Type = "Job";
JobType = "Normal";
Executable = "/bin/sh";
Arguments = "nodo_brams.sh";
InputSandbox = {"BRAMS_$FECHA.tar.gz","nodo_brams.sh"};
StdOutput = "brams.out";
StdError = "brams.err";
OutputSandbox = {"BRAMS_$FECHA.tar.gz","brams.err","brams.out"};
ShallowRetryCount = 1;
EOF
```

En este primer paso, se genera el JDL, con los parametros descritos en la sección anterior. En el OutputSandbox e InputSandbox se especifica un archivo BRAMS_$FECHA.tar.gz, este archivo contiene en el primer caso los archivos de entrada y el segundo todas las salidas de la ejecución del modelo, la variable $FECHA corresponde a la fecha para la cual se realiza el pronostico.

```
SHELL=0
glite-wms-Job-submit -d $USER -o jobid $JOBS > submit.txt
```

```
SHELL=`echo $?`
ID=`awk 'NR == 10 { print $1 }' submit.txt`
if [ "$SHELL" != 0 ]
then
        echo El comando glite-wms-Job-submit ha fallado
        exit 1
else
        echo $ID
        cat submit.txt
fi
```

Luego de realizar la creación del JDL, se procede a enviar el Job al WMS, al igual que en AIRES se hace uso del comando *glite-wms-Job-submit*, especificando con la opción -d el usuario que lo esta enviando, y con -o el archivo donde se almacenara el ID del Job.

La salida se almacena en el archivo submit.txt, para luego, haciendo uso del programa awk, extraer del archivo el ID del Job enviado. Al igual que en los casos anteriores, se verifica que la salida del comando *glite-wms-Job-submit* no haya generado ningún error.

```
STATUS=''
AUX=''
File=${ID:32}
while [ "$STATUS" != "Done" ]
do
        glite-wms-Job-status --verbosity 0 $ID > status$File
        SHELL=`echo $?`
        AUX=$STATUS
        STATUS=`awk '/Current/ {if(NR < 9) print $3 }' status$File`
        if [ "$AUX" != "$STATUS" ]
        then
```

```
12                  echo
13                  echo "Status Actual ----> $STATUS"
14          fi
15  done
16  if [ "$SHELL" != 0 ]
17  then
18          echo El comando glite-wms-Job-status ha fallado
19          exit 1
20  else
21          glite-wms-Job-status --verbosity 0 $ID
22  fi
```

Despues de enviar el Job, se realizan consultas constantes del estado del mismo, para evitar que el usuario tenga que estar frente a la consola, escribiendo el comando y solicitando el estado. Es por esto, que el ciclo que inicia en la linea #4 se encarga de consultar el estado del Job, verificar si la salida genero un error, y mostrar al usuario cada vez que el estado del Job cambie.

```
1  DIR=${ID:32}
2  if [ "$SHELL" == 0 ]
3  then
4          glite-wms-Job-output --dir $DIR $ID
5
6  fi
```

Cuando el estado del Job sea “Done”, indicara que este ya ha culminado su ejecución, por lo que solicitamos las salidas de este Job, la salida sera almacenada en un directorio ubicado en el HOME del usuario, este directorio tendrá como nombre parte del ID del Job enviado.

Por ejemplo, si el ID del Job es:

```
https://grid-ccpd01.ucv.ve:9000/TJEbLEEr4XxNJNa_7vMCvw.
```

El nombre del directorio sera:

```
 TJEbLEEr4XxNJNa_7vMCvw.
```

El script ejecutado en el WN, se encarga de ejecutar todas las fases necesarias para realizar el pronostico. Este script realiza una serie de modificaciones en archivos de configuración necesarios para la ejecución del modelo.

El script realizara la fase de preprocesamiento, procesamiento, y postprocesamiento. La fase de preprocesamiento, es la encargada de tomar los archivos que contienen el estado del tiempo de la atmósfera y transformarlos a un formato que sea legible para BRAMS. Para esto se utiliza grib2dp, que convierte archivos en formato grib y los transforma a un formato RALPH2.

```
1 cp -r $VAR/GAMRAMS/"$FECHA"_00Z_GAMRAMS/* $VAR/BRAMS-4.2/grib2dp
2 ./grib2dp.x
```

En estas lineas, se copian los archivos descargados del modelo global (contienen el estado del tiempo de la atmósfera) a las carpeta donde grib2dp los transformara. Luego de copiarlos, se ejecuta el programa.

Al tener los archivos en un formato legible para BRAMS, se puedo iniciar la fase de procesamiento. El archivo RAMSIN es el encargado de especificarle a BRAMS todos los parametros necesarios para su ejecución, este archivo debe ser modificado para cada ejecución, primero se debe cambiar la fecha, y despues modificar la variable RUNTYPE, que debe tomar tes valores, MAKESFC, MAKEVFILE e INITIAL, en el orden indicado. Por cada una de las modificaciones de la variables RUNTYPE se ejecutara el modelo, generando diversos archivos, como se muestra en la figura 5.2.

```
sed -e 's/IMONTH1\s*=\s*[0-9]*/IMONTH1  =  '"$MES"'/' RAMSIN >> RAMSIN2
rm RAMSIN
mv RAMSIN2 RAMSIN

sed -e 's/IDATE1\s*=\s*[0-9]*/IDATE1   =  '"$DIA"'/' RAMSIN >> RAMSIN2
rm RAMSIN
mv RAMSIN2 RAMSIN

sed -e 's/IYEAR1\s*=\s*[0-9]*/IYEAR1  =  '"$ANIO"'/' RAMSIN >> RAMSIN2
rm RAMSIN
mv RAMSIN2 RAMSIN
```

Figura 5.2: Archivos generados

En esta parte del código, se realizan las modificaciones de la fecha en el archivo RAMSIN.

```
sed -e '9 s/INITIAL/MAKESFC/' RAMSIN >> RAMSIN2
rm RAMSIN
mv RAMSIN2 RAMSIN

./brams_PARALELO -f RAMSIN
```

Aquí primero se realiza la modificación en el RAMSIN y luego se ejecuta por primera vez el modelo, teniendo a RUNTYPE = MAKESF.

```
sed -e '9 s/MAKESFC/MAKEVFILE/' RAMSIN >> RAMSIN2
rm RAMSIN
mv RAMSIN2 RAMSIN

./brams_PARALELO -f RAMSIN
```

Al igual que en el paso anterior, se realiza la modificación del RUNTYPE en el archivo RAMSIN, en este caso se le asigna MAKEVFILE, y se ejecuta el modelo.

```
sed -e '9 s/MAKEVFILE/INITIAL/' RAMSIN >> RAMSIN2
rm RAMSIN
mv RAMSIN2 RAMSIN

./brams_PARALELO -f RAMSIN
```

Luego se realiza el ultimo paso de la fase de procesamiento, ejecutando el modelo con la variable RUNTYPE = INITIAL. Esta ultima ejecución es la que realiza el pronostico como tal, utilizando archivos generados por las ejecuciones previas.

Al finalizar el procesamiento, se inicia la fase de postprocesamiento. En esta fase se toman los archivos generados por la fase de procesamiento y se transforman en archivos con extensión .ctl y .grb, estos pueden ser visualizados con la herramienta grads.

```
sed -e 's/anal-A-[0-9]*-/anal-A-'"$ANIO"'-/' ramspost.inp >> ramspost2.
   inp
rm -f ramspost.inp
mv ramspost2.inp ramspost.inp

sed -e 's/GPREFIX\s*=\s*.[0-9]*_/GPREFIX   = \x27'"$FECHA"'_/' ramspost
   .inp >> ramspost2.inp
rm -f ramspost.inp
mv ramspost2.inp ramspost.inp

./ramspost60.x
```

5.3. Aplicación Web

En esta sección se describen las funcionalidades presentes en la aplicación web, que facilita y proporciona un ambiente amigable al usuario, al cual enviar trabajos al Grid y recuperar sus respectivas salidas. La arquitectura de la aplicación web se muestra en la figura 5.3

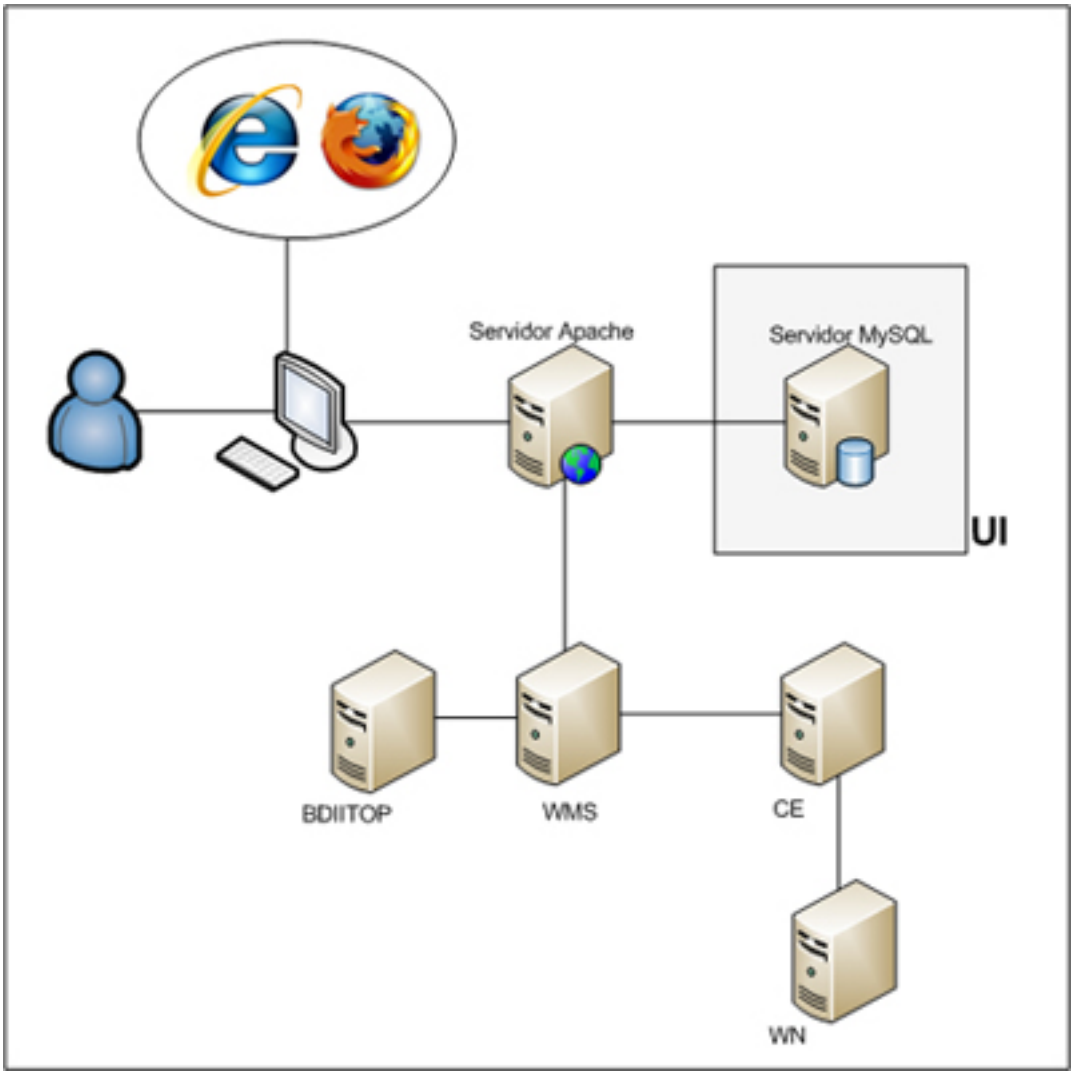

Figura 5.3: Arquitectura aplicación Web

Las funcionalidades de esta aplicación son:

1. Permitir a usuarios que no poseen cuenta solicitar una. Antes de crear la cuenta, es necesario

que el usuario descargue un certificado valido de Gilda, tal como se muestra en la figura 5.4.

2. Al momento de registrarse, el usuario debe esperar recibir un correo, indicando que su certificado ya fue instalado, a partir de ese momento, podrá acceder a la página, y ejecutar las dos aplicaciones que han sido adaptadas al Grid.

Figura 5.4: Formulario de Registro

En la figura 5.5 se muestra el inicio de sesion, este es el punto de acceso al sistema. Una vez autenticado, el usuario pasa a la página de entrada que se muestra en la figura 5.6, esta página permite acceder a las funcionalidades que se brinda a los usuarios autenticados. En este módulo son tomados en cuenta todos los requerimientos de seguridad que se deben observar para ejecutar trabajos en el Grid. Como consecuencia de la autenticación, se genera un proxy que habilita al usuario a utilizar el Grid.

Figura 5.5: Inicio de sesión

En la imagen 5.6 se observa la página en la que el usuario puede elegir entre las dos aplicaciones que han sido adaptadas al Grid, adicionalmente se muestra información sobre el proxy generado.

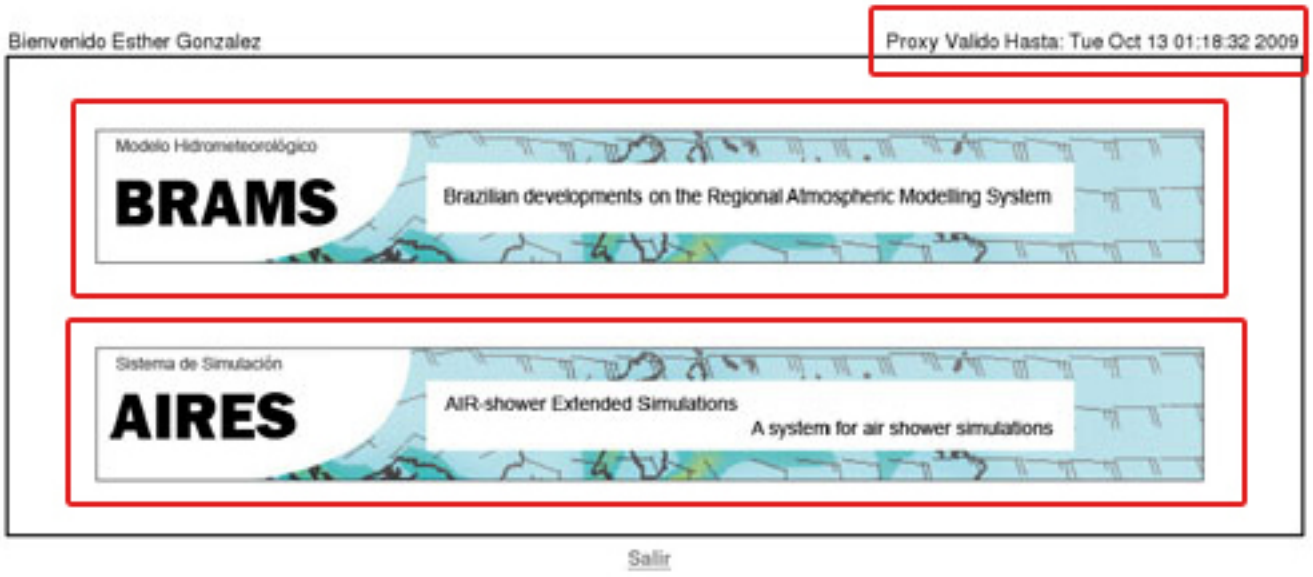

Figura 5.6: Menú de aplicaciones

Las imagenes 5.7 y 5.3 corresponde a la página que permite el envio de trabajos y recuperación de las salidas. La primera imagen corresponde a la vista de la pagina dedicada al sistema AIRES, y la segunda corresponde al modelo BRAMS.

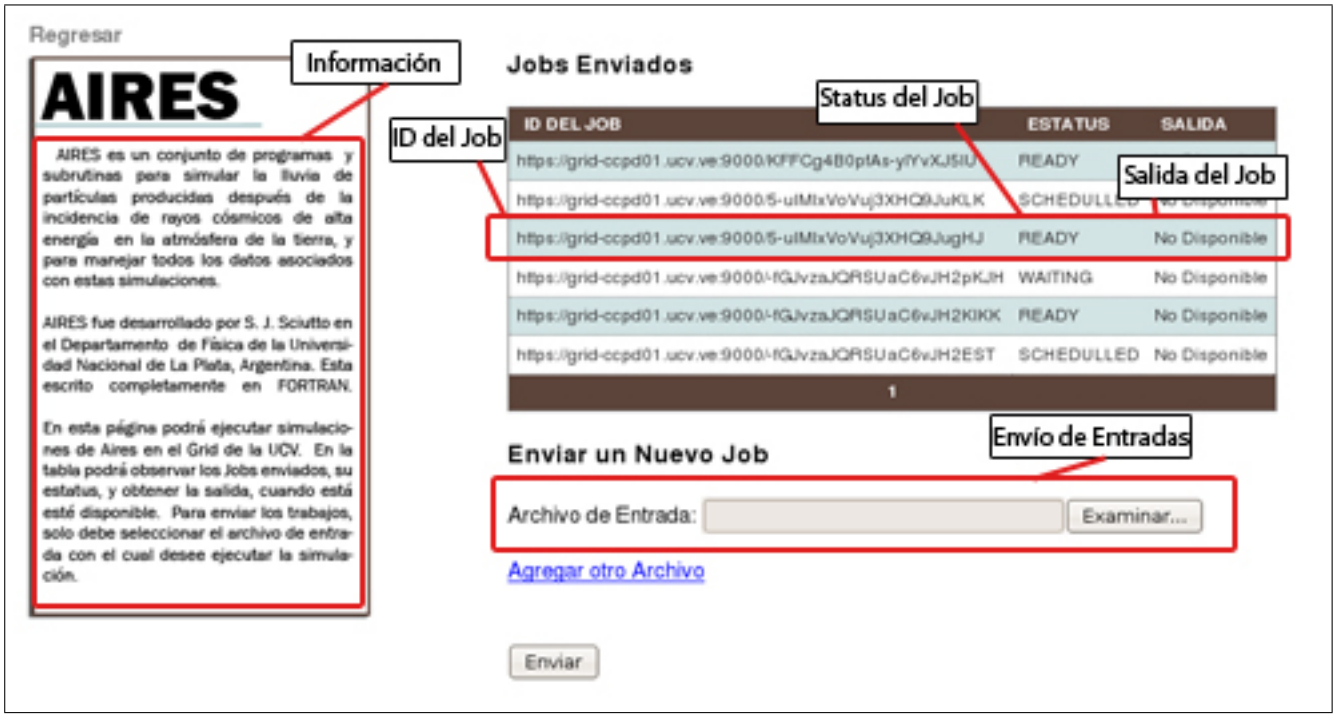

Figura 5.7: AIRES

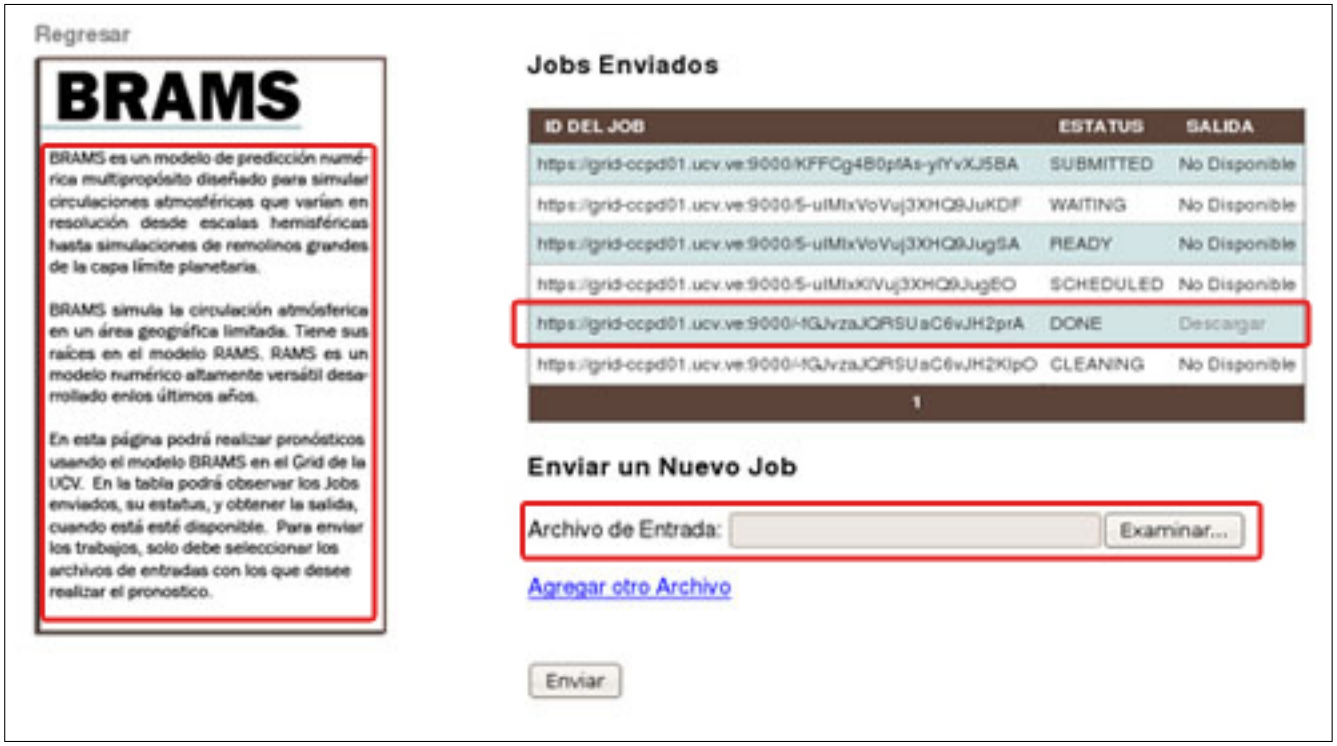

Figura 5.8: BRAMS

En la tabla se puede visualizar el ID del Job enviado, el estatus del Job, y obtener la salida, en caso de que haya terminado su ejecución. Debajo de la tabla, se encuentra un pequeño formulario, desde el cual se pueden subir los datos de entrada para la ejecución de el modelo BRAMS o el sistema AIRES, para el primero es necesario enviar un archivo comprimido con todos los datos globales de la fecha para la que se desee realizar el pronostico, para el segundo se debe enviar un simple archivo IDL con las directivas necesarias para la ejecución.

En el siguiente capítulo se muestran los resultados de la ejecución en el Grid UCV de las aplicaciones AIRES y BRAMS.

Capítulo 6

Pruebas de Funcionamiento

En este capitulo se describen las pruebas realizadas a las aplicaciones AIRES y BRAMS, tanto en una infraestructura Grid como en una infraestructura de Cluster.

6.1. Descripción de las Infraestructuras

Infraestructura Grid

Toda los equipos de la infraestructura Grid utilizada para la realización de estas pruebas tienen como Sistema Operativo Linux, distribución Scientific Linux SL release 4.4 (Beryllium), y las especificaciones que se muestran en el cuadro 6.1.

Modelo Maquina	Características		
	CPU	Memoria	Disco Duro
PC	Pentium III 1 Ghz	512 MB	40 GB
SUN WorkStation	2 x AMD Opteron 2 GHz	2GB	72 GB
SUN WorkStation	2 x AMD Opteron 2 GHz	2GB	72 GB
SUN WorkStation	2 x AMD Opteron 2 GHz	2GB	72 GB
SUN Fire Z80	2 x AMD Opteron 2.2 GHz	4GB	72 GB
PC	Celeron 1.7 Ghz	2 GB	40 GB

Cuadro 6.1: Especificaciones del Grid UCV

Infraestructura Cluster

El cluster utilizado para estas pruebas, es un cluster beowulf, con seis nodos, cada uno de ellos tiene como Sistema Operativo Ubuntu Server 8.04 (Hardy), sus especificaciones se muestran en el cuadro 6.2:

Modelo Maquina	Características		
	CPU	Memoria	Disco Duro
PC	Pentium III 1 Ghz	256MB	40 GB
PC	Pentium III 1 Ghz	256MB	40 GB
PC	Pentium III 1 Ghz	256MB	40 GB
PC	Pentium III 1 Ghz	256MB	40 GB
PC	Pentium III 1 Ghz	256MB	40 GB
PC	Pentium III 1 Ghz	256MB	40 GB

Cuadro 6.2: Especificaciones del cluster beowulf en el CCPD

6.2. Descripción de la pruebas de AIRES

Pruebas en el Cluster

Para la realización de las pruebas del cluster el procedimientos es el siguiente:

1. En primer lugar, se levantan los servicios de la librería mpi.

   ```
   $ mpdboot -n 6
   ```

   Donde 6 es el numero de procesadores en los que se ejecutaran las pruebas.

2. Luego se realiza las especificaciones de los parámetros de entrada, en los archivos IDL correspondientes a cada prueba. Los parametros modificados son los siguientes:

   - RunsPerProcess Número de corridas por cada proceso.
   - ShowersPerRun Número de lluvias a simular en cada corrida.
   - TotalShowers Número total de lluvias a simular, en la multiplicación de RunsPerProcess por ShowersPerRun.
   - PrimaryParticle La partícula primaria, con la que se inicia la simulación.

A continuación, se hace un llamado a la función *airestask*, especificándole el número de procesadores y el archivo de entrada a utilizar.

```
$ airestask -s <archivo>.inp
```

Este proceso se repite para cada una de las pruebas, con un archivo de entrada diferente.

Partícula	Lluvias	Corridas	T. Ejecución
Gamma	50	10	214
Gamma	40000	100	4797
Gamma Proton	40000	100	3128
Proton	30000	100	1576
Iron	60000	100	2632
Gamma	500	100	1898
Gamma	4000	100	516
Proton	20000	100	2149
Iron	30000	100	516
Gamma	90000	100	11312

Cuadro 6.3: Pruebas AIRES en el Cluster (tiempos expresados en segundos)

En el cuadro 6.3 se pueden ver los resultados obtenidos para cada una de las pruebas. En las tres primera columnas se muestran los parámetros modificados en los archivos de entrada, el número de ejecuciones y numero de lluvias son los valores que más influyen en cada simulación.

A continuación se muestran las pruebas realizadas en ambientes Grid.

Pruebas en el Grid

Las pruebas realizadas en el Grid siguieron los siguientes pasos.

1. Generación y delegación del proxy, haciendo uso del script proxy.exp

   ```
   $ expect proxy.exp
   ```

2. Luego se realiza las especificaciones de los parametros de entrada, en los archivos IDL correspondientes a cada prueba. Los parametros que fueron modificados son los siguientes:

- RunsPerProcess Numero de corridas por cada proceso.
- ShowersPerRun Numero de lluvias a simular en cada corrida.
- TotalShowers Numero total de lluvias a simular, en la multiplicación de RunsPerProcess por ShowersPerRun.
- PrimaryParticle La partícula primaria, con la que se inicia la simulación.

Al tener los archivos de entrada para cada una de las ejecuciones, se ejecuta el *script_brams.sh*, este script se encargara de realizar el envió, consulta de estatus, y obtención de salidas, de el Job enviado.

```
$ ./script_brams.sh <entrada>
```

Este proceso se repite para cada una de la pruebas.

Partícula	Lluvias	Corridas	T. Ejecución	T. Total
Gamma	50	10	63	1841
Gamma	40000	100	2845	5080
Gamma Proton	40000	100	1898	3652
Proton	30000	100	926	3373
Iron	60000	100	1523	3642
Gamma	500	100	647	2710
Gamma	4000	100	284	2087
Proton	20000	100	1257	3010
Iron	30000	100	281	1504
Gamma	90000	100	6399	8389

Cuadro 6.4: Pruebas de AIRES Grid (tiempos expresados en segundos)

En el cuadro 6.4 se visualizan los resultados obtenidos en el Grid, se muestran los mismos parametros que en la tabla con los resultados del Cluster, a diferencia de que se añade un tiempo total, que es el que indica el tiempo tomado desde el envío del Job hasta la recepción de las salidas.

Comparación de la ejecución de Aires en el Cluster y en el Grid

Partícula	Lluvias	Corridas	Cluster	Grid	
			T. Ejecución	T. Ejecución	T. Total
Gamma	50	10	214	63	1841
Gamma	40000	100	4797	2845	5080
Gamma Proton	40000	100	3128	1898	3652
Proton	30000	100	1576	926	3373
Iron	60000	100	2632	1523	3642
Gamma	500	100	1898	647	2710
Gamma	4000	100	516	284	2087
Proton	20000	100	2149	1257	3010
Iron	30000	100	516	281	1504
Gamma	90000	100	11312	6399	8389

Cuadro 6.5: Pruebas de AIRES en el Cluster y en el Grid (tiempos expresados en segundos).

En el cuadro 6.5 podemos ver los resultados de las pruebas, en el Cluster y en el Grid. Los resultados reflejan que el tiempo de ejecución es mucho mas cortos en el Grid que en el Cluster, sin embargo, el tiempo total (tomamos el tiempo de ejecución, como tiempo total en el Cluster), es mas corto en el Cluster, debido principalmente al tiempo que toma el WMS en actualizar el estatus de los Jobs enviados y los tiempos de autenticación y autorización necesarios para utilizar el Grid. AIRES soporta la ejecución de tareas concurrentes en equipos de memoria compartida, es por esto que los tiempos de ejecución en el cluster son significativamente superiores, ya que el cluster usa memoria distribuida y no compartida por lo que AIRES se ejecuta en un solo nodo del cluster.

En cuanto a los resultados obtenidos en cada una de las simulaciones, estos fueron distintos debido a que existen factores aleatorios involucrados en cada simulación. Además los resultados

se ven influenciados por el hecho de que las maquinas del Grid son AMD Opteron de 64 bits mientras que las del Cluster son Pentium III de 32 bits, lo cual afecta la precisión de los resultados obtenidos.

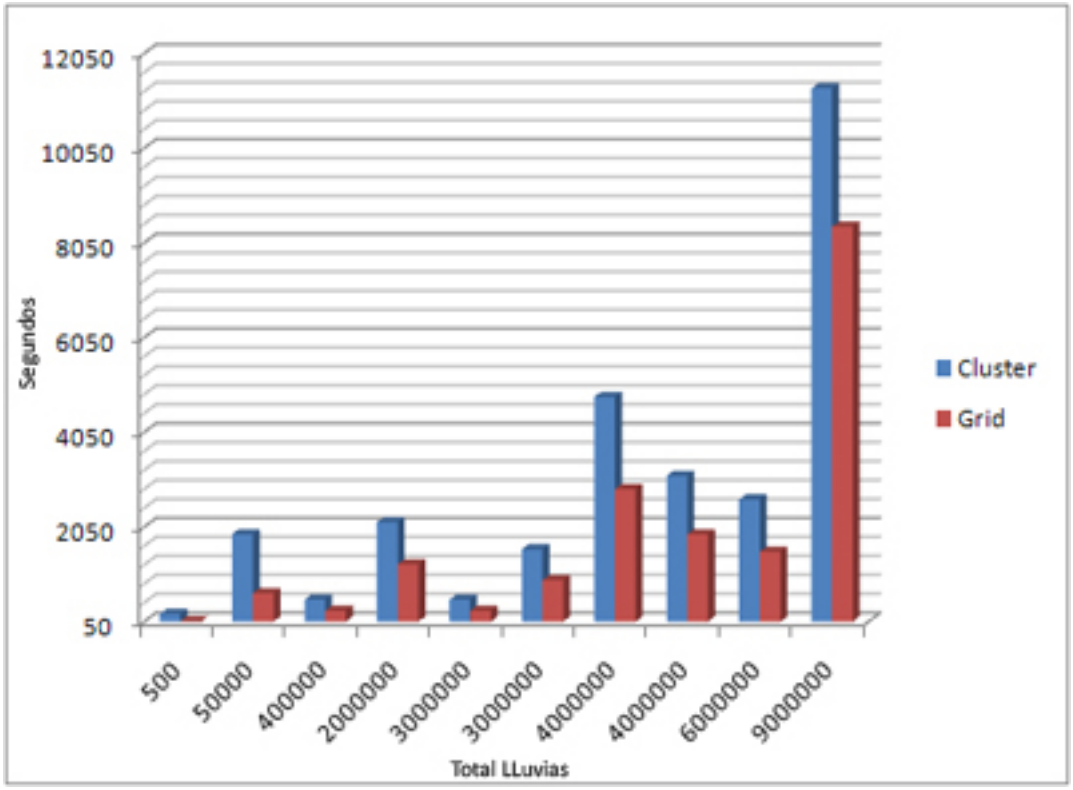

Figura 6.1: Gráfico Pruebas AIRES en el Cluster y Grid

En el gráfico 6.1 se visualizan los resultados de los tiempos de ejecución de AIRES en el cluster y en el Grid. Se puede notar la diferencia en el tiempo dependiendo de las lluvias a simular, así como el rendimiento del cluster ante la infraestructura Grid .

6.3. Descripción de la pruebas de BRAMS

Pruebas en el Cluster

La realización de las pruebas en el cluster se realizaron de la siguiente forma:

La obtención de los datos de entradas para la fecha deseada, es realizada haciendo uso del script *get_gamrams.sh*, este script descarga los datos globales desde el ftp://ftp1.cptec.inpe.br/modelos/io/tempo/global

```
$ ./get_gamrams.sh
```

Al tener los datos de entrada, si el día para el que se correrá el modelo es diferente al día actual, hace falta asignar las variables relacionadas en el script *nodo_brams.sh*, al tener estas variables con los valores adecuados se ejecuta el script.

```
$ ./nodo_brams.sh
```

Esto se repite para cada una de las pruebas.

Fecha Pronostico	Tiempo de Ejecución
15 Septiembre 2009	4327
29 Septiembre 2009	4392
30 Septiembre 2009	4281
01 Octubre 2009	4162
08 Octubre 2009	4374
02 Octubre 2009	4220
07 Octubre 2009	4348
17 Octubre 2009	4365
18 Octubre 2009	4290
19 Octubre 2009	4257

Cuadro 6.6: Pruebas de Brams en el Cluster (tiempos expresados en segundos)

En el cuadro 6.6 se muestran los tiempos obtenidos para la ejecución del modelo BRAMS en el cluster, se realizaron diez pruebas, cada una realiza el pronostico climático para días distintos, se puede notar que los tiempos para cada una de las ejecuciones son muy similares.

Pruebas en el Grid

Las pruebas en el Grid se realizaron haciendo uso de los scripts desarrollados para la adaptación de esta aplicación al Grid de la UCV.

1. El primero paso es la obtención de los datos de entrada para la fecha deseada, esto se realiza haciendo uso del script *get_gamrams.sh*, este script descarga los datos globales desde el ftp donde son publicados los datos.

   ```
   $ ./get_gamrams.sh
   ```

   Al tener los archivos de entradas, estos se comprimen, para ser enviados como parametros de entradas.

   ```
   $ tar -cvf BRAMS_<fecha>.tar.gz <Datos Descargado>
   ```

2. Generación y delegación del proxy, haciendo uso del script proxy.exp

   ```
   $ expect proxy.exp
   ```

3. Luego, se realiza la ejecución del script *script_brams.sh*, este script realiza la generación del JDL, envío del Job al WMS, consulta de estatus, y recuperación de la salida.

   ```
   $ ./script_brams.sh BRAMS_<fecha>.tar.gz
   ```

Estos pasos se repiten para cada una de las pruebas.

Fecha Pronostico	Tiempo Ejecución	Tiempo Total
15 Septiembre 2009	4186	5292
29 Septiembre 2009	4241	6354
30 Septiembre 2009	4130	5671
01 Octubre 2009	4013	5852
08 Octubre 2009	4248	5422
02 Octubre 2009	4015	5077
07 Octubre 2009	4091	5403
17 Octubre 2009	4248	5792
18 Octubre 2009	4039	6112
19 Octubre 2009	4027	5348

Cuadro 6.7: Pruebas de Brams en el Grid (tiempos expresados en segundos)

En el cuadro 6.7 se muestran los resultados obtenidos en el Grid, se observa la similitud en los tiempos de cada una de las pruebas, y se puede observar el tiempo de demora que existe entre la terminación de la ejecución y la obtención de los resultados.

Comparación Brams Cluster y Grid

	Cluster	Grid	
Fecha Pronostico	Tiempo de Ejecución	Tiempo Ejecución	Tiempo Total
15 Septiembre 2009	4327	4186	5292
29 Septiembre 2009	4392	4241	6354
30 Septiembre 2009	4281	4130	5671
01 Octubre 2009	4162	4013	5852
08 Octubre 2009	4374	4248	5422
02 Octubre 2009	4220	4015	5077
07 Octubre 2009	4348	4091	5403
17 Octubre 2009	4365	4248	5792
18 Octubre 2009	4290	4039	6112
19 Octubre 2009	4257	4027	5348

Cuadro 6.8: Pruebas Brams Cluster y Grid (tiempos expresados en segundos)

En en el cuadro 6.8 se puede observar que los tiempos en el cluster son menores, por lo que concluimos, que esta aplicación explota de una forma mas eficiente sus capacidades para ejecutarse en paralelo haciendo uso de la librería MPI. Sin embargo la diferencia entre el tiempo de ejecución no es muy significativa, esto debido a las capacidades de cada una de las maquinas donde se ejecuto el modelo.

En el Grid los tiempos de comunicación sumados a los procesos de autenticación y autorización necesarios para usar la infraestructura, hacen que la obtención de las salidas tarden mas tiempo, lo que no significa que la ejecución del programa en si haya sido mas lenta.

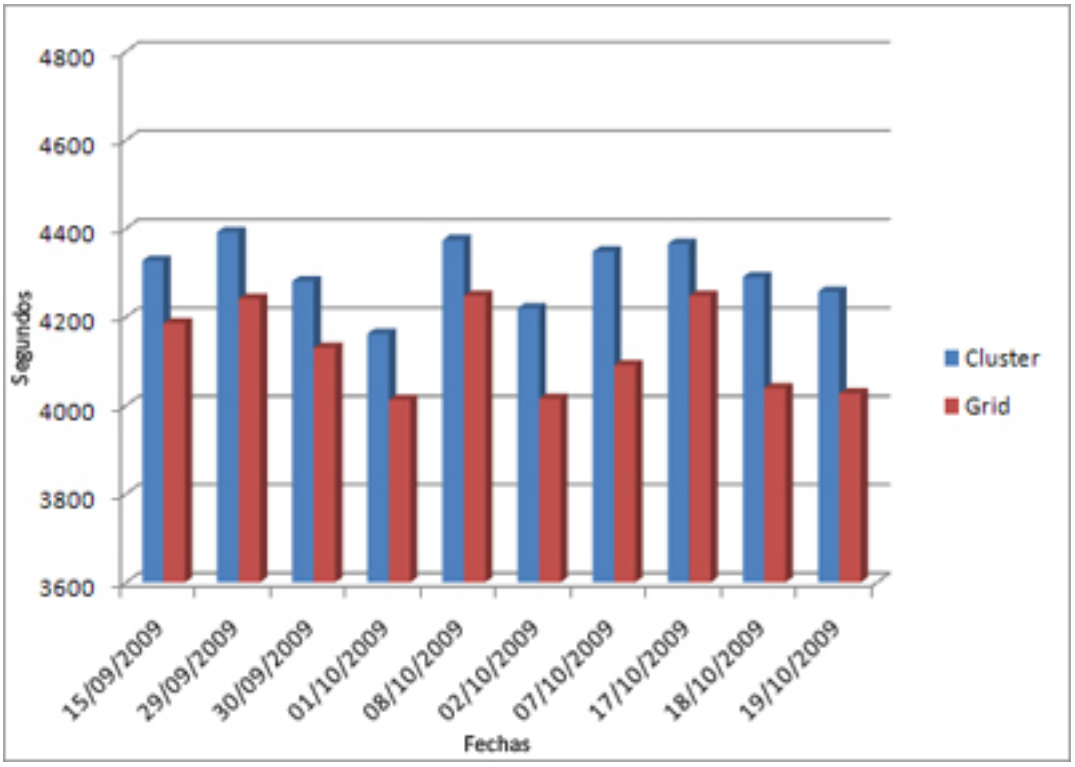

Figura 6.2: Gráfico Pruebas BRAMS en el Cluster y Grid

En el gráfico 6.2 se puede observar la diferencia entre el tiempo de ejecución de BRAMS en el cluster y en el Grid, se puede notar que esta diferencia es relativamente uniforme en cada una de las pruebas.

En el siguiente capítulo se mostraran las conclusiones del trabajo realizado.

Conclusiones y Recomendaciones

El objetivo principal de este trabajo fue la adaptación de AIRES y BRAMS, dos aplicaciones de alto rendimiento al Grid, especificamente al Grid de la UCV. Esto se hizo debido a la gran complejidad que aún representa la interacción de los usuarios con la infraestructura Grid, lo cual dificulta la migración y ejecución de aplicaciones de alto rendimiento a este ambiente. Con la gridificación (adaptación de aplicaciones al Grid) de este tipo de aplicaciones, se pueden beneficiar, tanto instituciones como científicos, que necesiten de un gran poder de computo para la realización de sus actividades. Permitiendoles la ejecución de estas aplicaciones sin tener que incurrir en los costos necesarios para la compra de equipos con gran poder de procesamiento o almacenamiento. Para poder realizar este proceso de adaptación, se realizo un estudio detallado del funcionamiento de cada una de las aplicaciones, teniendo finalmente, ambas aplicaciones adaptadas al Grid, contando con dos formas de ejecución, una mediante la ejecución de script en la consola de comando, o mediantes una interfaz web que permite el envío, monitoreo y recepción de salida de los Jobs enviados.

Por ejemplo en el área de meteorología existe suficiente aplicaciones y problemas de interés como para que el desarrollo de aplicaciones Grid pueda tener éxito. Sin embargo, no se dispone de una infraestructura computacional y técnica apropiada para plantear proyectos Grid a nivel nacional, otras áreas que puede verse beneficiada con la gridificacion de aplicaciones, es la industria petrolera, la cual maneja una gran cantidad de datos que requieren ser procesados en el menor tiempo posible.

Se pudo observar, que una desventaja en la ejecución de una aplicación en un ambiente Grid, es el tiempo que toma los procesos de autenticación y autorización, así como la demora que se presenta en la actualización del estatus de los Jobs enviados, retrasando así la obtención de los resultados. Sin embargo, a pesar de este tiempo, sigue siendo viable y beneficioso la adaptación de aplicaciones de alto rendimiento a un ambiente Grid, primero, porque para aplicaciones que pueden durar días e incluso semanas ejecutándose, este tiempo puede ser despreciable; y segundo, porque al contar con una mayor cantidad de WN, se podrán obtener mejores resultados, haciendo

que valga la pena sacrificar este tiempo necesario para usar el Grid.

La plataforma del Grid UCV permitió la instalación, ejecución de las aplicaciones AIRES y BRAMS, presentado diversos inconvenientes durante el desarrollo de este trabajo, es por esto que se recomienda realizar mejoras a la infraestructura Grid instalada en la UCV, así como continuar con la gridificacion de aplicaciones de alto rendimiento. Añadir mas funcionalidades a las interfaces realizadas, para permitir al usuario tener mayor control sobre los Jobs.

Con la realización de este trabajo se obtuvieron los siguientes resultados:

- Adaptación al Grid de la UCV del sistema de simulación AIRES.
- Adaptación al Grid de la UCV del modelo BRAMS.
- Desarrollo de scripts de gratificación de AIRES y BRAMS.
- Desarrollo de una interfaz web para el envío, monitoreo, y recuperación de salidas de Jobs de AIRES y BRAMS.
- Realización de pruebas de funcionamiento y consistencia de los resultados de AIRES y BRAMS en el cluster y en el Grid de la UCV.

Al culminar este trabajo, se logro completar exitosamente todos los objetivos propuestos inicialmente, teniendo una plataforma Grid en la UCV con dos aplicaciones de alto rendimiento disponibles para su utilización.

Anexo A

Scripts de Autenticación

A continuacion se muestran los scripts encargado de la creación y delegación del proxy, en primer lugar el script *proxy.exp*, permite intertactuar con la consola. El script, invoca al script *proxy.sh*, que se encarga de generar y delegar el proxy.

```
#!/usr/bin/expect
set timeout -1

spawn ./proxy.sh

expect "Enter GRID pass phrase:"
send "mifrase\r"

expect eof
exit
```

```
#!/bin/sh
SHELL="0"

voms-proxy-init --voms gilda > proxy.txt
SHELL=`echo $?`
```

```
6
7  if [ "$SHELL" != "0" ]
8  then
9          echo "No se pudo generar el proxy para el usuario $USER"
10
11         exit 1
12 else
13         valido=`awk '/until/ { print $7 , $8 , $9 , $10}' proxy.txt`
14         php5 /var/grid/scripts/actualizar_proxy.php $USER $valido
15         echo "Proxy Generado para el usuario $USER $valido"
16         echo
17         sleep 10
18         glite-wms-job-delegate-proxy -d $USER
19         SHELL=`echo $?`
20         if [ "$SHELL" != "0" ]
21         then
22                 echo "No se pudo delegar el proxy al usuario $USER"
23                 exit 1
24         else
25                 echo "Proxy delegado al usuario $USER"
26         fi
27
28 fi
29 rm -f proxy.txt
```

Anexo B

Scripts AIRES

A continuación se muestran los scripts realizados para la ejecución de AIRES en el Grid. Primero el script *script_aires.sh*, es el encargado del la creacion del JDL, envio, consulta de status, y recuperación de las salidas del Job. Luego, se muestra el script encargado de la ejecución del Job en el WN, *nodo_aires.sh*.

```
#!/bin/sh

JOBS='aires_jobs.jdl'
LOGS=$1
PARALELO='1'
SHELL='1'
rm -f $JOBS
clear
echo \##############################################
echo \####.........Creando el JDL.......... ....####
echo \##############################################
cat > $JOBS << EOF
[
Type = "collection";
nodes = {
```

```
EOF
variable=`echo [`
echo $variable>> $JOBS
contador=0
for i in `cat $LOGS`
   do
    tam=`echo ${#i}`
    if [ "$tam" -gt "2" ]
    then
              TASKNAME=`awk '/Task/ { print $2 }' $i`
              contador=`expr $contador + 1`
              if [ $contador -gt 1 ]
              then
                 variable=`echo ],[`
                 echo $variable>> $JOBS
              fi
              variable1=`echo JobType = \"Normal\";`
              echo $variable1 >> $JOBS
              variable2=`echo NodeName = \"node$contador\";`
              echo $variable2 >> $JOBS
              variable3=`echo Executable = \"/bin/sh\";`
              echo $variable3 >> $JOBS
              variable4=`echo Arguments = \"nodo_aires.sh $i $PARALELO
                 \";`
              echo $variable4 >> $JOBS
              variable5=`echo InputSandbox = {\"nodo_aires.sh\", \"$i
                 \"};`
              echo $variable5 >> $JOBS
              variable6=`echo StdOutput = \"salida$contador\";`
              echo $variable6 >> $JOBS
              variable7=`echo StdError  = \"error$contador\";`
```

```
45                echo $variable7 >> $JOBS
46                variable8=`echo OutputSandbox = \{\"aires.tar.gz\",\"
                     salida.txt\",\"salida$contador\",\"error$contador
                     \"\}\;`
47                echo $variable8 >> $JOBS
48                variable9=`echo ShallowRetryCount = 1\;`
49                echo $variable9 >> $JOBS
50        else
51                PARALELO=$i
52        fi
53    done
54  variable=`echo ]`
55  echo $variable>> $JOBS
56  variable=`echo }\;`
57  echo $variable>> $JOBS
58  variable=`echo ]`
59  echo $variable>> $JOBS
60  echo "------------------Se ha genrado el JDL-----------------"
61  echo "--------------Presiones 1 para visualizarlo -----------"
62  echo "--------------Presiones 2 para continuar---------------"
63  #read teclado
64
65  #if [ "$teclado" == "1" ]
66  #then
67  #cat aires_jobs.jdl
68  #fi
69
70  #echo "------------------Presione una Tecla para Continuar
       -----------------"
71  #read teclado
72  clear
```

```
73 echo \##############################################
74 echo \####.....Enviando el Job.................####
75 echo \##############################################
76 SHELL=0
77 glite-wms-job-submit -d $USER -o jobid $JOBS > submit.txt
78 SHELL=`echo $?`
79 ID=`awk 'NR == 10 { print $1 }' submit.txt`
80 if [ "$SHELL" != 0 ]
81 then
82         echo El comando glite-wms-job-submit ha fallado
83         exit 1
84 else
85         echo $ID
86         cat submit.txt
87 fi
88 rm -f submit.txt
89 php5 /var/grid/scripts/guardar_job.php $ID SUBMITTED - 0 $USER
90 sleep 5
91 clear
92 echo \##############################################
93 echo \####........Consulta de Status............####
94 echo \##############################################
95
96 STATUS=''
97 AUX=''
98 File=${ID:32}
99 while [ "$STATUS" != "Done" ]
100 do
101         glite-wms-job-status --verbosity 0 $ID > status$File
102         SHELL=`echo $?`
103         AUX=$STATUS
```

```
104         STATUS=`awk '/Current/ {if(NR < 9) print $3 }' status$File`
105         if [ "$AUX" != "$STATUS" ]
106         then
107                 echo
108                 echo "Status Actual ----> $STATUS"
109                 php5 /var/grid/scripts/actualizar_param.php $ID estatus
                        $STATUS
110         fi
111 done
112 if [ "$SHELL" != 0 ]
113 then
114         echo El comando glite-wms-job-status ha fallado
115         exit 1
116 else
117         glite-wms-job-status --verbosity 0 $ID
118 fi
119 rm -f status$ID
120 clear
121 echo \##############################################
122 echo \####.....Obteniendo la Salida.............####
123 echo \##############################################
124 DIR=${ID:32}
125 tar -cvf $DIR".tar.gz" $DIR
126 if [ "$SHELL" == 0 ]
127 then
128         glite-wms-job-output --dir $DIR $ID
129         php5 /var/grid/scripts/actualizar_param.php $ID salida "$PWD/""
              $DIR"".tar.gz"
130
131 fi
132 exit 0
```

```
#!/bin/sh

cp /usr/local/bin/airesrc /$HOME/.airesrc

ruta=$PWD
echo en directorio `pwd`

cp PE* $HOME
cd $HOME
mkdir aires
cp PE* aires
echo "ls aires"
ls

cd aires

/usr/local/bin/airestask -s $2 $1

ls -la

/usr/local/bin/airesstatus $2

/usr/local/bin/airesstatus $2 > status.out

STATUS=`awk '/currently/ { print $7 }' status.out`
while [ "$STATUS" != "NOT" ]
do
        AUX=$STATUS
        /usr/local/bin/airesstatus $2 > status.out
```

```
30          STATUS=`awk '/currently/ { print $7 }' status.out`
31          if [ "$AUX" != "$STATUS" ]
32          then
33                  echo
34                  echo "Status Actual ----> $STATUS"
35          fi
36  done
37
38  /usr/local/bin/airesstatus $2
39
40  cd ..
41  /bin/tar -czvf aires.tar.gz aires
42  cp aires.tar.gz $ruta
43  cd $ruta
44  ls
```

Anexo C

Scripts BRAMS

A continuación se muestran los scripts realizados para la ejecución de BRAMS en el Grid. Primero el script *script_brams.sh*, es el encargado del la creación del JDL, envío, consulta de status, y recuperación de las salidas del Job. Luego, se mostrara el script encargado de la ejecución del Job en el WN, *nodo_brams.sh*.

```
#!/bin/sh

JOBS='brams_jobs.jdl'
FECHA=`date --date= "+%Y%m%d"`
rm -f brams_jobs.jdl
clear
echo \###############################################
echo \####.........Creando el JDL.......... ....####
echo \###############################################
TAR="$FECHA"_00Z_GAMRAMS.tar
cat > $JOBS << EOF
Type = "Job";
JobType = "Normal";
Executable = "/bin/sh";
Arguments = "nodo_brams.sh";
```

```
16 InputSandbox = {"nodo_brams.sh"};
17 StdOutput = "brams.out";
18 StdError = "brams.err";
19 OutputSandbox = {"BRAMS_$FECHA.tar.gz","brams.err","brams.out"};
20 ShallowRetryCount = 1;
21 EOF
22 echo "------------------Se ha genrado el JDL-----------------"
23 echo "--------------Presiones 1 para visualizarlo -----------"
24 echo "--------------Presiones 2 para continuar---------------"
25 read teclado
26
27 if [ "$teclado" == "1" ]
28 then
29 cat brams_jobs.jdl
30 fi
31 echo "------------------Presione una Tecla para Continuar
      -----------------"
32 read teclado
33 clear
34 echo \################################################
35 echo \####.....Enviando el Job..................####
36 echo \################################################
37 SHELL=0
38 glite-wms-job-submit -d $USER -o jobid $JOBS > submit.txt
39 SHELL=`echo $?`
40 ID=`awk 'NR == 10 { print $1 }' submit.txt`
41 if [ "$SHELL" != 0 ]
42 then
43         echo El comando glite-wms-job-submit ha fallado
44         exit 1
45 else
```

```
46          echo $ID
47          php5 /var/www/guardar_job.php $ID SUBMITTED - 1 $USER
48          cat submit.txt
49  fi
50  rm -f submit.txt
51  sleep 5
52  clear
53  echo \##############################################
54  echo \####........Consulta de Status............####
55  echo \##############################################
56
57  STATUS=''
58  AUX=''
59  File=${ID:32}
60  while [ "$STATUS" != "Done" ]
61  do
62          glite-wms-job-status --verbosity 0 $ID > status$File
63          SHELL=`echo $?`
64          AUX=$STATUS
65          STATUS=`awk '/Current/ {if(NR < 9) print $3 }' status$File`
66          if [ "$AUX" != "$STATUS" ]
67          then
68                  echo
69                  echo "Status Actual ----> $STATUS"
70                  php5 /var/www/grid/actualizar_param.php $ID estatus
                        $STATUS
71          fi
72  done
73  if [ "$SHELL" != 0 ]
74  then
75          echo El comando glite-wms-job-status ha fallado
```

```
        exit 1
else
        glite-wms-job-status --verbosity 0 $ID
fi
rm -f status$ID
clear
echo \##############################################
echo \####.....Obteniendo la Salida.............####
echo \##############################################
DIR=${ID:32}
if [ "$SHELL" == 0 ]
then
        glite-wms-job-output --dir $DIR $ID
        php5 /var/www/grid/actualizar_param.php $ID salida "$PWD""$DIR"
fi
exit 0
```

```
#!/bin/bash
echo \##############################################
echo \##........................................##
echo \##...SCRIPT PARA LA EJECUCION DE BRAMS......##
echo \##........................................##
echo \##############################################
MYDIR=$PWD
VAR=/usr/local/BRAMS
dias=$1;
FECHA=`date --date=''"$dias"' days ago' +"%Y%m%d"`
FECHA2=`date --date=''"$dias"' days ago' +"%d%m%Y"`
ANIO=`date --date=''"$dias"' days ago' +"%Y"`
MES=`date --date=''"$dias"' days ago' +"%m"`
```

```
DIA=`date --date=''"$dias"' days ago' +"%d"`

mkdir -p $VAR/SALIDAS_MODELOS/BRAMS_$FECHA
cd $VAR/SALIDAS_MODELOS/BRAMS_$FECHA
echo `date` > date.out

TAR=$2
echo \################################################
echo \##.....INICIANDO PARTE 1...................##
echo \################################################


tar -xvz $TAR

cp $VAR/GAMRAMS/"$FECHA"_00Z_GAMRAMS/* $VAR/BRAMS-4.2/grib2dp
cd $VAR/BRAMS-4.2/grib2dp
ls
./grib2dp.x
ls

echo \################################################
echo \##.........FIN PARTE 1... ..................##
echo \################################################


echo \################################################
echo \##.........INICIANDO PARTE 2................##
echo \################################################


cd $VAR/BRAMS-4.2/brams_paralelo
```

```
45 pwd
46
47 sed -e 's/IMONTH1\s*=\s*[0-9]*/IMONTH1  =  '"$MES"'/' RAMSIN >> RAMSIN2
48 rm RAMSIN
49 mv RAMSIN2 RAMSIN
50
51 sed -e 's/IDATE1\s*=\s*[0-9]*/IDATE1   =  '"$DIA"'/' RAMSIN >> RAMSIN2
52 rm RAMSIN
53 mv RAMSIN2 RAMSIN
54
55 sed -e 's/IYEAR1\s*=\s*[0-9]*/IYEAR1  =  '"$ANIO"'/' RAMSIN >> RAMSIN2
56 rm RAMSIN
57 mv RAMSIN2 RAMSIN
58
59 chmod 777 RAMSIN
60 echo \##############################################
61 echo \##...Aqui ejecuto brams paralelo MAKESFC....##
62 echo \##############################################
63
64 sed -e '9 s/INITIAL/MAKESFC/' RAMSIN >> RAMSIN2
65 rm RAMSIN
66 mv RAMSIN2 RAMSIN
67
68 chmod 777 RAMSIN
69
70 ./brams_PARALELO -f RAMSIN
71
72 echo \##############################################
73 echo \##..Aqui ejecuto brams paralelo MAKEVFILE...##
74 echo \##############################################
75
```

```
76 sed -e '9 s/MAKESFC/MAKEVFILE/' RAMSIN >> RAMSIN2
77 rm RAMSIN
78 mv RAMSIN2 RAMSIN
79
80 chmod 777 RAMSIN
81 #mpirun -machinefile /home/estudiantes/egonzalez/mpd.hosts -np 6 ./
      brams_PARALELO -f RAMSIN
82 ./brams_PARALELO -f RAMSIN
83 echo \##############################################
84 echo \##...Aqui ejecuto brams paralelo INITIAL....##
85 echo \##############################################
86
87 sed -e '9 s/MAKEVFILE/INITIAL/' RAMSIN >> RAMSIN2
88 rm RAMSIN
89 mv RAMSIN2 RAMSIN
90
91 chmod 777 RAMSIN
92
93 echo Aqui ejecuto mpirun..
94 #mpdtrace
95 #mpirun -machinefile /home/estudiantes/egonzalez/mpd.hosts -np 6 /usr/
      local/BRAMS/BRAMS-4.2/build/brams-opt -f RAMSIN
96 ./brams_PARALELO -f RAMSIN
97
98 echo \##############################################
99 echo \##.........FIN PARTE 2.............. ...##
100 echo \##############################################
101
102
103
104 echo \##############################################
```

```
105 echo \##.........INICIANDO PARTE 3..............##
106 echo \##############################################
107 cd $VAR/BRAMS-4.2/brams_paralelo/Ramspost60
108 pwd
109
110 echo primersed
111
112 sed -e 's/anal-A-[0-9]*-/anal-A-'"$ANIO"'-/' ramspost.inp >> ramspost2.
       inp
113 rm -f ramspost.inp
114 mv ramspost2.inp ramspost.inp
115
116 echo segundosed
117
118 sed -e 's/GPREFIX\s*=\s*.[0-9]*_/GPREFIX   = \x27'"$FECHA"'_/' ramspost
       .inp >> ramspost2.inp
119 rm -f ramspost.inp
120 mv ramspost2.inp ramspost.inp
121
122 echo cambiopermisos
123
124 chmod 777 ramspost.inp
125
126 echo corroramspot
127 ./ramspost60.x
128
129 echo \##############################################
130 echo \##.........FIN PARTE 3.................##
131 echo \##############################################
132
133 echo \##############################################
```

```
134 echo \##..........INICIANDO PARTE 4.............##
135 echo \##############################################
136
137 echo
138 echo ............COPIANDO ARCHIVOS.............
139 #mkdir -p $VAR/SALIDAS_MODELOS/BRAMS_$FECHA
140 mkdir -p $VAR/SALIDAS_MODELOS/BRAMS_$FECHA/GAMRAMS
141 mkdir -p $VAR/SALIDAS_MODELOS/BRAMS_$FECHA/brams_paralelo
142 mkdir -p $VAR/SALIDAS_MODELOS/BRAMS_$FECHA/brams_paralelo/A
143 mkdir -p $VAR/SALIDAS_MODELOS/BRAMS_$FECHA/brams_paralelo/A/RAMPOST60
144
145 chmod 777 $VAR/SALIDAS_MODELOS/BRAMS_$FECHA
146
147 cp $VAR/GAMRAMS/"$FECHA"_00Z_GAMRAMS/* $VAR/SALIDAS_MODELOS/
      BRAMS_$FECHA/GAMRAMS
148 cp $VAR/BRAMS-4.2/brams_paralelo/A/* $VAR/SALIDAS_MODELOS/BRAMS_$FECHA/
      brams_paralelo/A/
149 cp $VAR/BRAMS-4.2/brams_paralelo/Ramspost60/"$FECHA"_VENEZUELA_g1.ctl
      $VAR/SALIDAS_MODELOS/BRAMS_$FECHA/brams_paralelo/A/RAMPOST60/
150 cp $VAR/BRAMS-4.2/brams_paralelo/Ramspost60/"$FECHA"_VENEZUELA_g2.ctl
      $VAR/SALIDAS_MODELOS/BRAMS_$FECHA/brams_paralelo/A/RAMPOST60/
151 cp $VAR/BRAMS-4.2/brams_paralelo/Ramspost60/"$FECHA"_VENEZUELA_g1.gra
      $VAR/SALIDAS_MODELOS/BRAMS_$FECHA/brams_paralelo/A/RAMPOST60/
152 cp $VAR/BRAMS-4.2/brams_paralelo/Ramspost60/"$FECHA"_VENEZUELA_g2.gra
      $VAR/SALIDAS_MODELOS/BRAMS_$FECHA/brams_paralelo/A/RAMPOST60/
153
154 rm $VAR/BRAMS-4.2/grib2dp/dp*
155 rm $VAR/BRAMS-4.2/grib2dp/GAMRAMS*
156 rm $VAR/BRAMS-4.2/brams_paralelo/A/*
157 rm $VAR/BRAMS-4.2/brams_paralelo/H/*
158 rm $VAR/BRAMS-4.2/brams_paralelo/ivar/*
```

```
159 rm $VAR/BRAMS-4.2/brams_paralelo/Ramspost60/*.ctl
160 rm $VAR/BRAMS-4.2/brams_paralelo/Ramspost60/*.gra
161
162
163 echo \##############################################
164 echo \##.........FIN PARTE 4.....................##
165 echo \##############################################
166 cd $VAR/SALIDAS_MODELOS/BRAMS_$FECHA
167 /bin/tar -czvPf BRAMS_$FECHA.tar.gz $VAR/SALIDAS_MODELOS/BRAMS_$FECHA
168 cp BRAMS_$FECHA.tar.gz $MYDIR
169 echo `date` >> date.out
170 echo \##############################################
171 echo \##............FIN CORRIDA...................##
172 echo \##############################################
```

Bibliografía

[1] I. Foster, C. Kesselma. The Grid: Blueprint for a New Computing Infraestructure. Morgan Kaufman, 2005.

[2] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. International J. Supercomputer Applications, 2001.

[3] I. Foster, C. Kesselman and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002. http:// citeseer.ist.psu.edu/foster02physiology.html.

[4] I. Foster. What is the Grid? A Three Point Checklist, Argonne National Laboratory University of Chicago ,2002.

[5] S. J. Sciutto. A system for air shower simulations, Departamento de Física Universidad Nacional de La Plata, 2002.

[6] S. Burke, S. Campana, E. Lanciotti, P. Mendez. gLite 3.1 User Guide Manuals Series, Enabling Grids for E-sciencE-II, 2009.

[7] F. Martin, A. Villafranca. Introduccion a la Tecnologia Grid, Asociación Profesional del Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado, 2005.

[8] Grid Infn Laboratory for Dissemination Activities, https://gilda.ct.infn.it/.

[9] E-science Grid Facility for Europe and Latin America, http://www.eu-eela.eu/.

[10] Brazilian developments on the Regional Atmospheric Modelling System, http://brams.cptec.inpe.br/.

[11] Atmospheric Meteorological and Environment Technologies, http://www.atmet.com/.

[12] Instituto de Matemática y Estadica de la Universidad de Sao Paulo, http://www.ime.usp.br/.

[13] Instituto de Astronomia Geofisicas y Ciencias Estadisticas de la Universidad de Sao Paulo, http://www.iag.usp.br/.

[14] Centro de Prediccion del Tiempo y Estudios Climatologicos,http://www.cptec.inpe.br/.

[15] Brazilian Funding Agency, http://www.finep.gov.br/.

[16] Regional Atmospheric Modeling System, http://rams.atmos.colostate.edu/.

[17] The Globus Alliance, http://www.globus.org/.

[18] European Organization for Nuclear Research, http://www.cern.ch/.

[19] Enabling Grids for E-science, http://public.eu-egee.org/activities/index.html.

[20] Lightweight Middleware for Grid Computing, http://glite.web.cern.ch/glite/.

[21] The Globus Toolkit, http://www.globus.org/toolkit/.

[22] AIRshower Extended Simulations, http://www.fisica.unlp.edu.ar/auger/aires/.

[23] Instituto Nacional de Meteorología e Hidrología. http://www.inameh.gob.ve/web/.