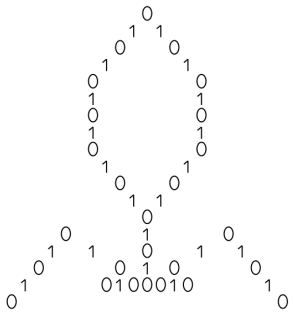
FuD-BOINC

Implementación de una capa de distribución del framework FuD utilizando como middleware a BOINC

Lucas Besso - Raúl Striglio

Universidad Nacional de Río Cuarto
Fac. de Cs. Exactas, Fco-Qcas y Naturales
Departamento de Computación



FuDePAN
Fundación para el Desarrollo de la Programación en Ácidos Nucleicos

28 de Diciembre de 2011

Temario

Fundación para el Desarrollo de la Programación en Ácidos Nucleicos

FuDePAN es una ONG (organización no gubernamental) en la cual se realizan investigaciones y desarrollos en bioinformática para aplicar a problemas biológicos asociados a la salud humana.

Utiliza la ciencia de la computación para:

- Mejorar vacunas.
- Mejorar tratamientos contra enfermedades como el VIH, o el virus Junín.

Fundación para el Desarrollo de la Programación en Ácidos Nucleicos

FuDePAN es una ONG (organización no gubernamental) en la cual se realizan investigaciones y desarrollos en bioinformática para aplicar a problemas biológicos asociados a la salud humana.

Utiliza la ciencia de la computación para:

- Mejorar vacunas.
- Mejorar tratamientos contra enfermedades como el VIH, o el virus Junín.

Motivación

Debido a que la fundación:

- intenta resolver problemas de alta complejidad realizando computación de alto rendimiento,
- cuenta con un framework para el desarrollo de aplicaciones distribuidas (**FuD**).
- necesita de recursos computacionales costosos para el cómputo de dichos problemas,
- es una organización sin fines de lucro,

Surgió la necesidad de:

Contar con una nueva implementación de la capa de distribución de **FuD** que permita a las aplicaciones, desarrolladas con este framework, hacer uso de la computación voluntaria.

Motivación

Debido a que la fundación:

- intenta resolver problemas de alta complejidad realizando computación de alto rendimiento,
- cuenta con un framework para el desarrollo de aplicaciones distribuidas (**FuD**).
- necesita de recursos computacionales costosos para el cómputo de dichos problemas,
- es una organización sin fines de lucro,

Surgió la necesidad de:

Contar con una nueva implementación de la capa de distribución de **FuD** que permita a las aplicaciones, desarrolladas con este framework, hacer uso de la computación voluntaria.

FuD

Definición

FuD (**F**uDePAN **U**biquitous **D**istribution) es un framework para automatizar la distribución de aplicaciones computacionales a través de disposiciones heterogéneas y dinámicas de nodos de procesamiento.

Características

- Posee una arquitectura *Master-Worker* con paralelismo de datos.
- Permite resolver cualquier problema computacional de manera paralela.
- El desarrollador que utilice **FuD** no necesita tener conocimientos de programación paralela.
- Abstrae a la aplicación del medio de distribución (BOINC, ANA, etc.)

FuD

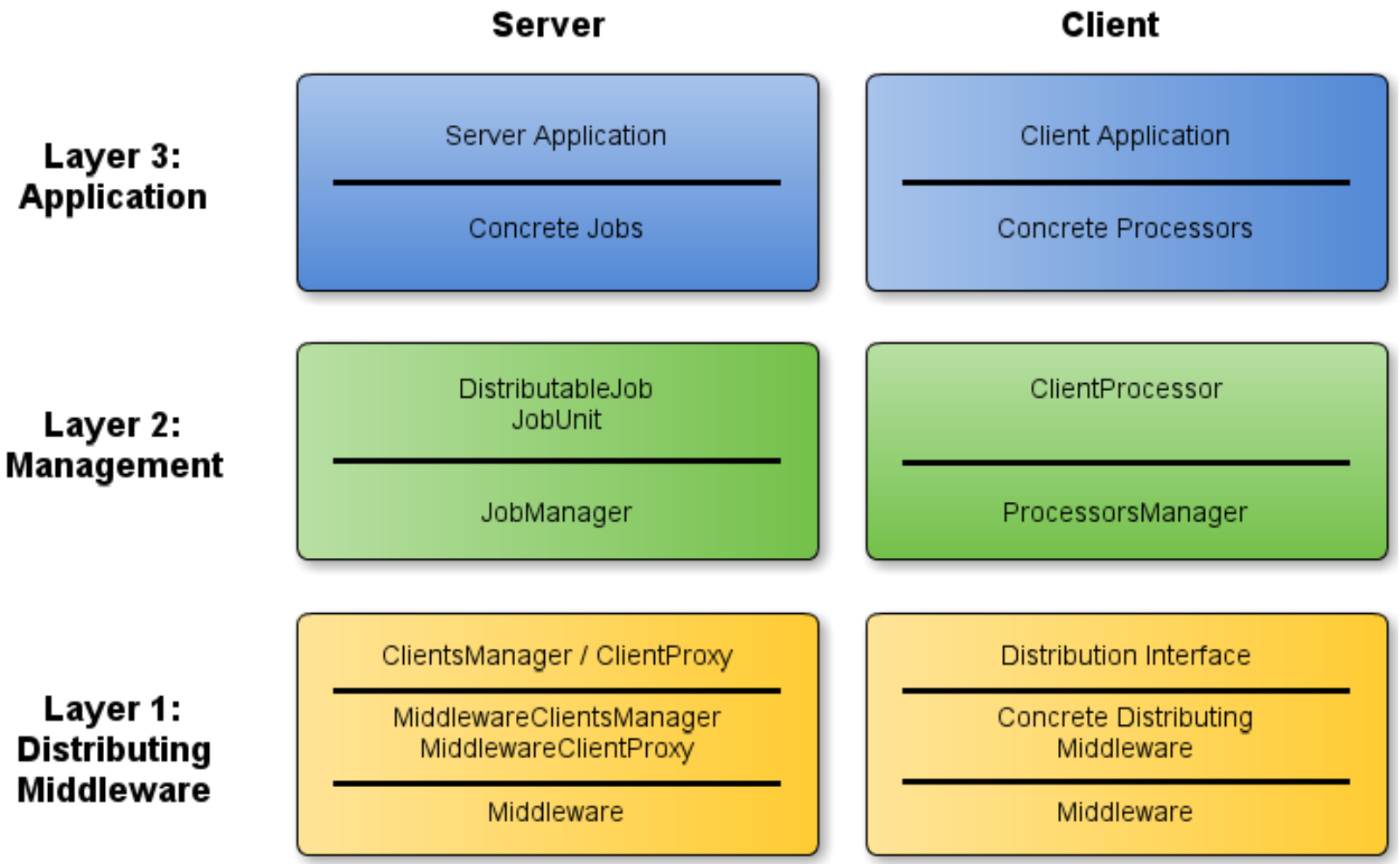
Definición

FuD (**F**uDePAN **U**biquitous **D**istribution) es un framework para automatizar la distribución de aplicaciones computacionales a través de disposiciones heterogéneas y dinámicas de nodos de procesamiento.

Características

- Posee una arquitectura *Master-Worker* con paralelismo de datos.
- Permite resolver cualquier problema computacional de manera paralela.
- El desarrollador que utilice **FuD** no necesita tener conocimientos de programación paralela.
- Abstrae a la aplicación del medio de distribución (BOINC, ANA, etc.)

Arquitectura



Trabajos de FuD

DistributableJob

Es un concepto de trabajo abstracto que encapsula cualquier tarea que será computada y puede ser subdividido en tareas más pequeñas llamadas **JobUnits**.

JobUnit

Representa una computación concreta y atómica que será llevada a cabo por alguno de los nodos de procesamiento. La tarea en sí es representada por un mensaje el cual será pasado a un cliente de procesamiento quien se encargará de computarla.

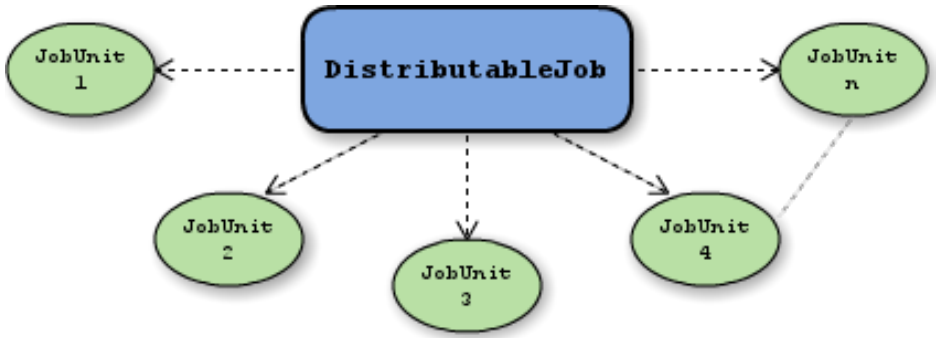
Trabajos de FuD

DistributableJob

Es un concepto de trabajo abstracto que encapsula cualquier tarea que será computada y puede ser subdividido en tareas más pequeñas llamadas **JobUnits**.

JobUnit

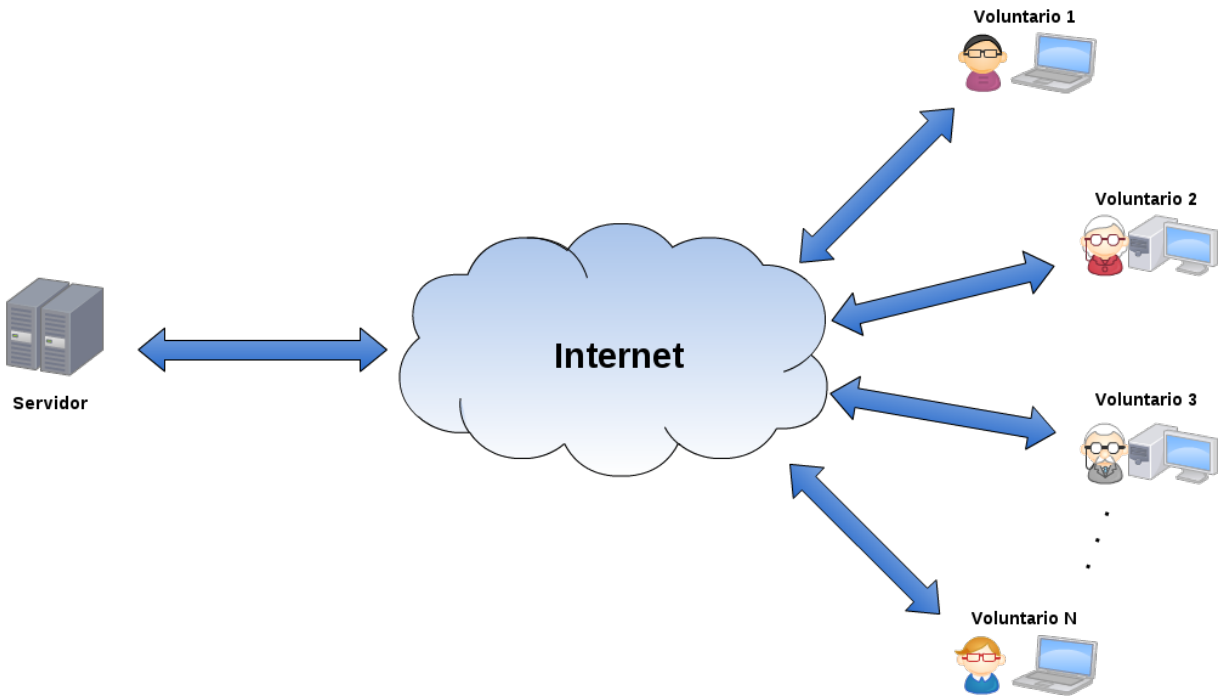
Representa una computación concreta y atómica que será llevada a cabo por alguno de los nodos de procesamiento. La tarea en sí es representada por un mensaje el cual será pasado a un cliente de procesamiento quien se encargará de computarla.



Computación Voluntaria

Definición

La computación voluntaria es un tipo de computación distribuida donde personas (voluntarios) donan los recursos libres de su ordenador a proyectos científicos mediante una conexión a internet.



Computación Voluntaria

Ventajas:

- Los voluntarios pueden colaborar desde cualquier punto geográfico.
- Hace posible la existencia de proyectos de investigación a bajos costos.
- Los voluntarios no necesitan ser usuarios avanzados de computadoras.
- Permite obtener y superar el poder de cálculo de una súper computadora.

Computación Voluntaria

Desventajas:

- Aumenta el consumo de energía de los ordenadores clientes debido a que éstos reducen su consumo en su tiempo libre.
- Para lograr un buen desempeño del proyecto es necesario apelar a medios de comunicación para reclutar voluntarios.
- Los voluntarios pueden confiar en los proyectos pero el proyecto no puede confiar en ellos.

Berkeley Open Infrastructure for Network Computing (BOINC)

BOINC es una plataforma de software de código abierto para computación voluntaria y computación grid, cuyo principal propósito es hacer posible la investigación científica.

Características

- Los voluntarios pueden computar desde distintas plataformas.
- Los usuarios pueden participar en varios proyectos a la vez.
- Sólo se asignan tareas a voluntarios que cuenten con los recursos necesarios.
- Asignación de trabajos bajo demanda por parte de los clientes.

Berkeley Open Infrastructure for Network Computing (BOINC)

BOINC es una plataforma de software de código abierto para computación voluntaria y computación grid, cuyo principal propósito es hacer posible la investigación científica.

Características

- Los voluntarios pueden computar desde distintas plataformas.
- Los usuarios pueden participar en varios proyectos a la vez.
- Sólo se asignan tareas a voluntarios que cuenten con los recursos necesarios.
- Asignación de trabajos bajo demanda por parte de los clientes.

Trabajos BOINC

Los trabajos de BOINC se encapsulan bajo dos conceptos importantes:

Workunit

Describe y representa la tarea a ser computada por un cliente del proyecto. Cada una está asociada a una aplicación en particular y puede tener uno o más archivos de entrada relacionados.

Result

Es una instancia de una workunit que se desea resolver. Precisamente es este result el que será enviado a los clientes de BOINC para su procesamiento. Cada workunit debe tener asociado al menos un result.

Trabajos BOINC

Los trabajos de BOINC se encapsulan bajo dos conceptos importantes:

Workunit

Describe y representa la tarea a ser computada por un cliente del proyecto. Cada una está asociada a una aplicación en particular y puede tener uno o más archivos de entrada relacionados.

Result

Es una instancia de una workunit que se desea resolver. Precisamente es este result el que será enviado a los clientes de BOINC para su procesamiento. Cada workunit debe tener asociado al menos un result.

Trabajos BOINC

Los trabajos de BOINC se encapsulan bajo dos conceptos importantes:

Workunit

Describe y representa la tarea a ser computada por un cliente del proyecto. Cada una está asociada a una aplicación en particular y puede tener uno o más archivos de entrada relacionados.

Result

Es una instancia de una workunit que se desea resolver. Precisamente es este result el que será enviado a los clientes de BOINC para su procesamiento. Cada workunit debe tener asociado al menos un result.

Proyecto BOINC

Un **proyecto BOINC** es el encargado de manegar todo el proceso de computación distribuida. Permite la administración de cuentas de usuarios, de aplicaciones y define cómo van a ser distribuidos los trabajos.

Componentes de un proyecto:

- Una base de datos
- Un sitio web
- Un servidor conformado por:
  - Generadores de trabajos *(work generators)*
  - Aplicaciones
  - Utilidades *(ej: start, stop, status, xadd, update_version)*
  - Demonios *(ej: validator, assimilator, file_deleter, db_purge)*
- Un archivo de configuración

Todo proyecto se identifica por la URL de su sitio web.

Proyecto BOINC

Un **proyecto BOINC** es el encargado de manegar todo el proceso de computación distribuida. Permite la administración de cuentas de usuarios, de aplicaciones y define cómo van a ser distribuidos los trabajos.

Componentes de un proyecto:

- Una base de datos
- Un sitio web
- Un servidor conformado por:
  - Generadores de trabajos *(work generators)*
  - Aplicaciones
  - Utilidades *(ej: start, stop, status, xadd, update_version)*
  - Demonios *(ej: validator, assimilator, file_deleter, db_purge)*
- Un archivo de configuración

Todo proyecto se identifica por la URL de su sitio web.

Proyectos conocidos

- Seti@home (http://setiathome.ssl.berkeley.edu/)
- Climate Prediction (http://climateprediction.net/)
- LHC@Home (http://lhcathomeclassic.cern.ch/sixtrack/)
- Einstein@Home (http://einstein.phys.uwm.edu/)

Lista completa de proyectos BOINC:
http://boinc.berkeley.edu/wiki/Project_list/

Cliente BOINC y BOINC Manager

Cliente BOINC (core client)

Es la aplicación cliente de BOINC que corre como demonio en las computadoras de los participantes.

Sus principales funciones son:

- Mantener una conexión directa con el servidor del proyecto para recibir trabajos y enviar resultados.
- Descargar la aplicación necesaria para la computación de cada trabajo.
- Realizar la computación de cada tarea.
- Administrar la distribución de los recursos locales entre los proyectos adheridos.

BOINC Manager

Es una aplicación que ofrece una simple interfaz gráfica mediante la cuál los voluntarios pueden controlar el **cliente BOINC**.

Cliente BOINC y BOINC Manager

Cliente BOINC (core client)

Es la aplicación cliente de BOINC que corre como demonio en las computadoras de los participantes.

Sus principales funciones son:

- Mantener una conexión directa con el servidor del proyecto para recibir trabajos y enviar resultados.
- Descargar la aplicación necesaria para la computación de cada trabajo.
- Realizar la computación de cada tarea.
- Administrar la distribución de los recursos locales entre los proyectos adheridos.

BOINC Manager

Es una aplicación que ofrece una simple interfaz gráfica mediante la cuál los voluntarios pueden controlar el **cliente BOINC**.

Cliente BOINC y BOINC Manager

Cliente BOINC (core client)

Es la aplicación cliente de BOINC que corre como demonio en las computadoras de los participantes.

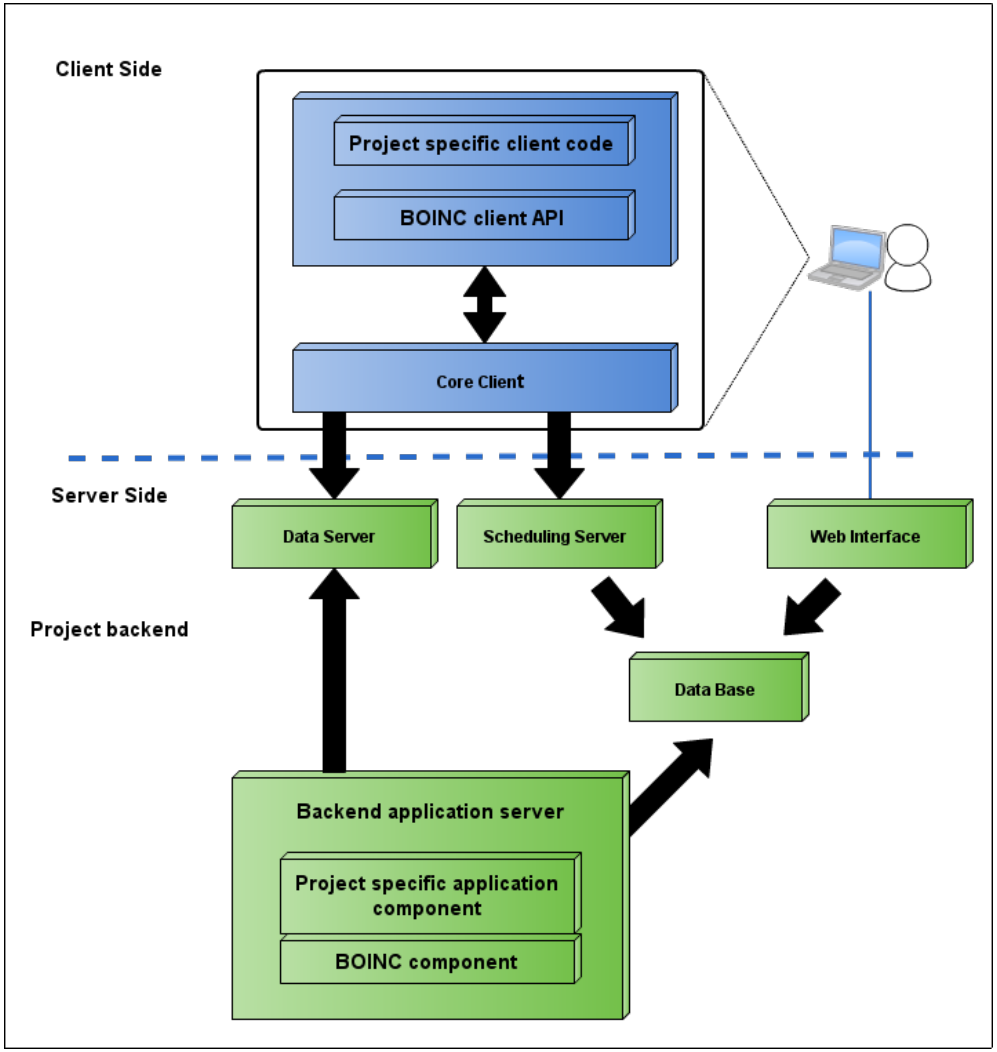
Sus principales funciones son:

- Mantener una conexión directa con el servidor del proyecto para recibir trabajos y enviar resultados.
- Descargar la aplicación necesaria para la computación de cada trabajo.
- Realizar la computación de cada tarea.
- Administrar la distribución de los recursos locales entre los proyectos adheridos.

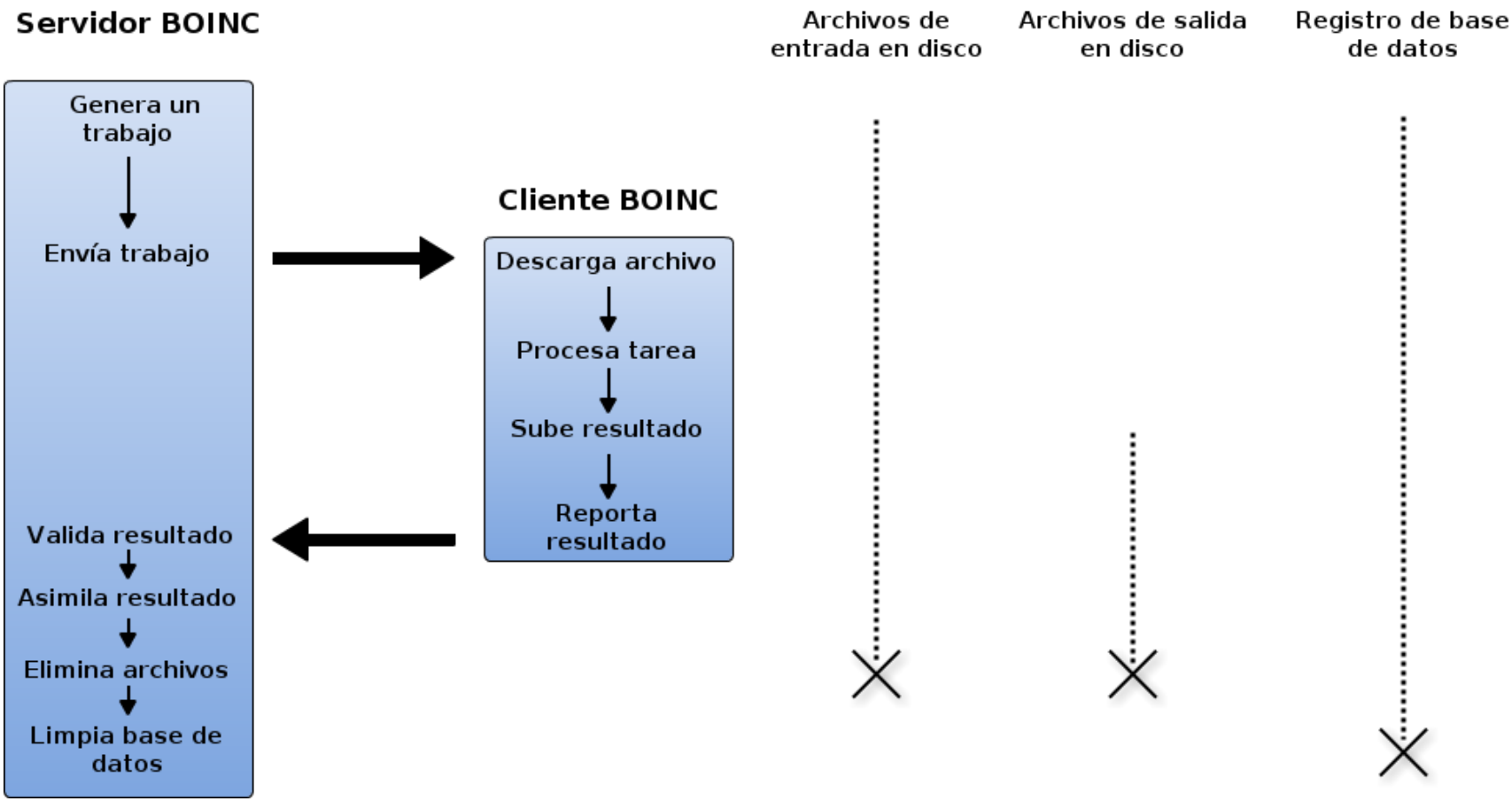
BOINC Manager

Es una aplicación que ofrece una simple interfaz gráfica mediante la cuál los voluntarios pueden controlar el **cliente BOINC**.

Componentes de BOINC

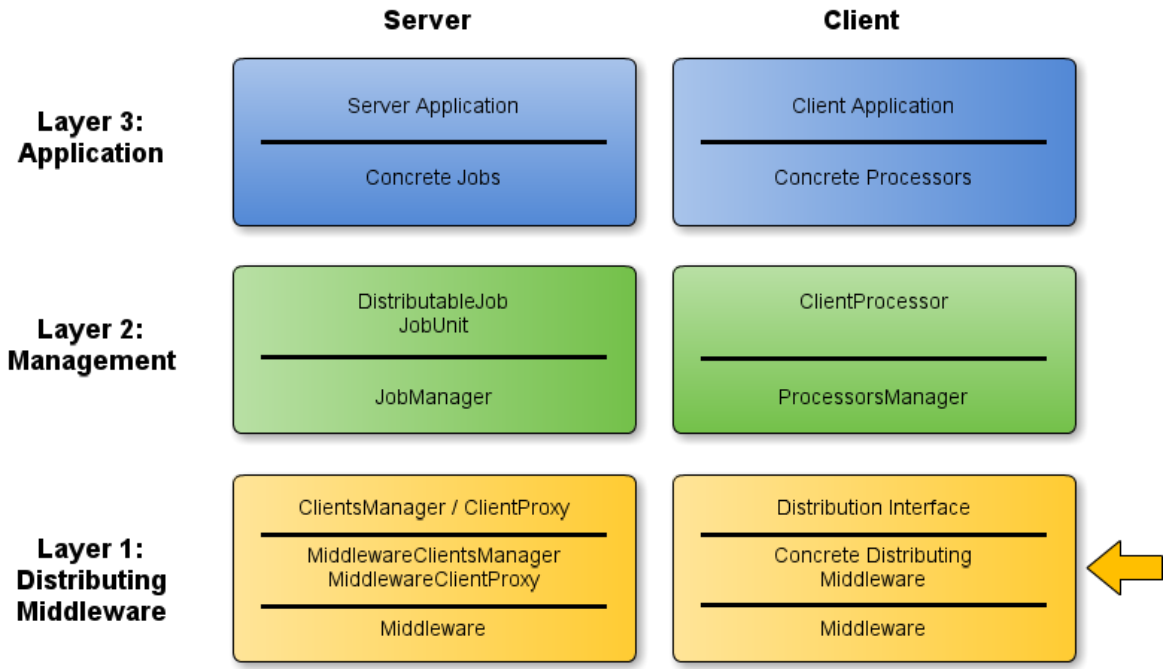


Interacción entre servidor y cliente BOINC

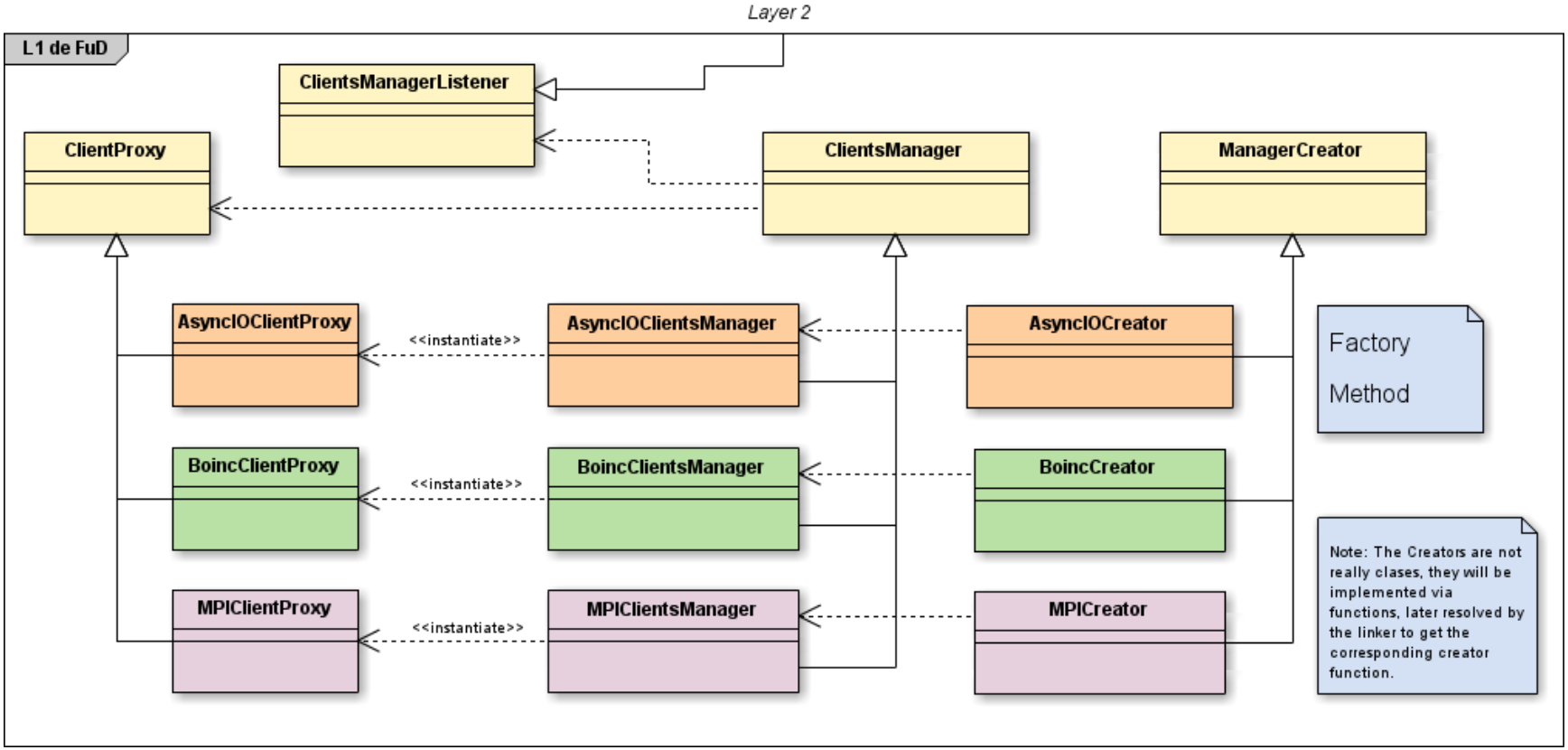


¿Qué es FuD-BOINC?

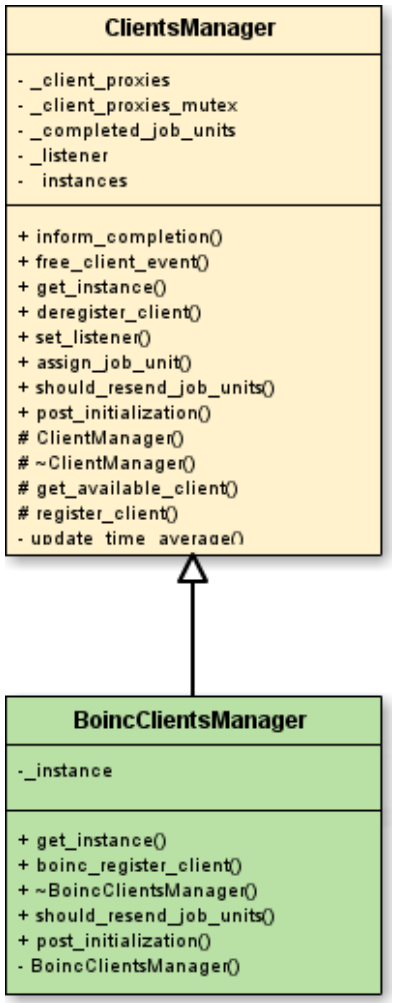
- Una nueva implementación de la capa L1 de **FuD**.
- Permite que aplicaciones desarrolladas con **FuD** puedan correr sobre proyectos BOINC.
- Permite abstraer al desarrollador de aplicaciones **FuD** de la API de BOINC.



Capa de distribución: lado servidor



Lado servidor: BoincClientsManager



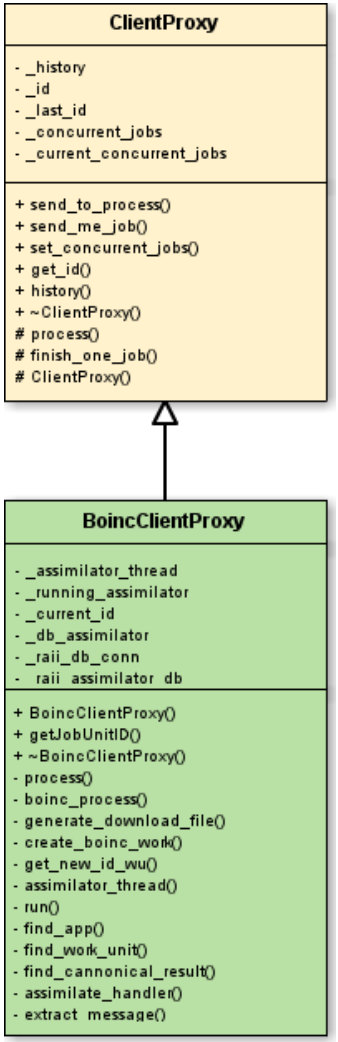
BoincClientsManager

Hereda de la clase `ClientsManager` de **FuD**.

Responsabilidades

- Realizar el registro de un único cliente.
- Comunicar al manejador de trabajos la disponibilidad del cliente conectado.

Lado servidor: BoincClientProxy



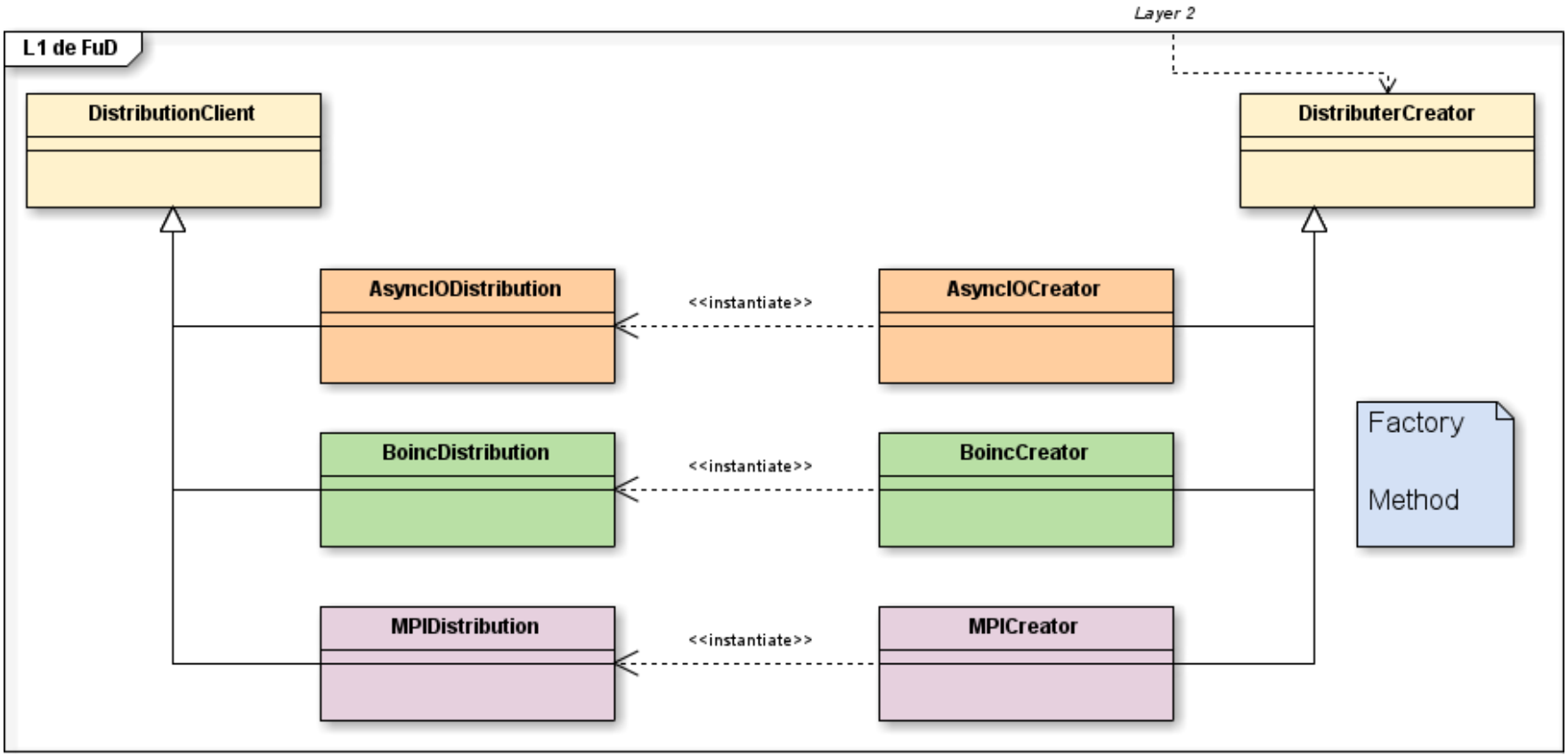
BoincClientProxy

Hereda de la clase `ClientProxy` de **FuD** y representa un cliente conectado con la particularidad que siempre está disponible.

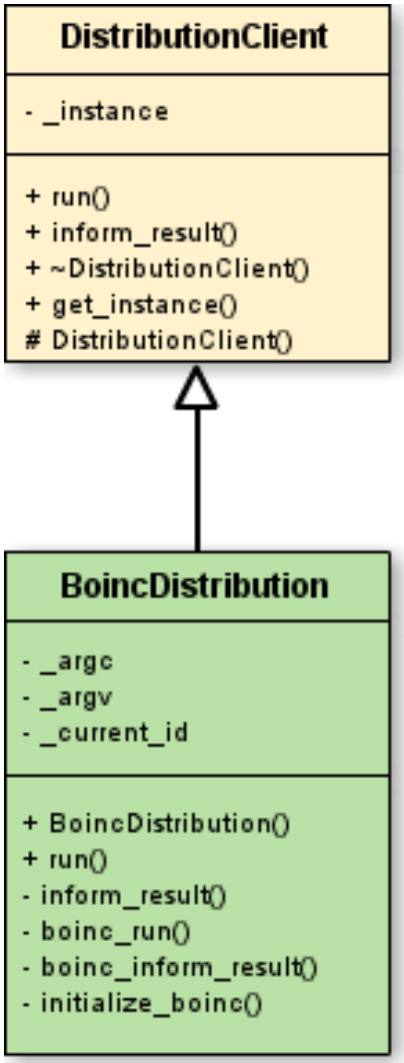
Responsabilidades

- Mantener una conexión directa con la base de datos del proyecto BOINC.
- Generar un trabajo de BOINC a partir de una `JobUnit`.
- Obtener los resultados enviados por los clientes e informarlos a L2.

Capa de distribución: lado cliente



Lado cliente: BoincDistribution



BoincDistribution

Hereda de la clase DistributionClient de **FuD** y provee la funcionalidad necesaria para que la aplicación pueda correr satisfactoriamente sobre el cliente BOINC.

Responsabilidades

- Extraer la `JobUnit` del archivo de entrada brindado por el **cliente de BOINC**.
- Envíar la `JobUnit` a la capa L2 para su procesamiento.
- Escribir el resultado de la tarea en un archivo de salida para que el **cliente de BOINC** lo informe al servidor.

API de BOINC

Servidor

Del lado servidor de FuD-BOINC se debió utilizar la API de BOINC para realizar las siguientes tareas:

- Leer el archivo de configuración del proyecto BOINC:

```
config.parse_file()
```

- Establecer una conexión con la base de datos:

```
boinc_db.open(config.db_name, config.db_host, config.db_user,
    config.db_passwd);
```

API de BOINC

Cliente

Del lado cliente de FuD-BOINC se debió utilizar la API de BOINC para realizar las siguientes tareas:

- Inicializar los diagnósticos de BOINC:

```
int boinc_init_diagnostics(int flags);
```

- Inicializar la aplicación cliente de BOINC para que sea propiamente ejecutada por el cliente de BOINC:

```
boinc_init();
```

- Finalizar la aplicación cliente de BOINC:

```
int boinc_finish(int status);
```

Servidor: envío de trabajos

FuD provee el método `process()` mediante el cual se envían `JobUnits` a clientes.

En FuD-BOINC, el método `process()` se encarga de traducir una `JobUnit` a un trabajo de BOINC mediante los siguientes pasos:

1. Serializar la `JobUnit` de **FuD** dentro de un archivo binario.
2. Leer la plantilla utilizada por BOINC en donde se especifican las características de la **workunit** a crear.
3. Invocar al método `create_boinc_work()`.

Servidor: envío de trabajos

FuD provee el método `process()` mediante el cual se envían `JobUnits` a clientes.

En FuD-BOINC, el método `process()` se encarga de traducir una `JobUnit` a un trabajo de BOINC mediante los siguientes pasos:

1. Serializar la `JobUnit` de **FuD** dentro de un archivo binario.
2. Leer la plantilla utilizada por BOINC en donde se especifican las características de la **workunit** a crear.
3. Invocar al método `create_boinc_work()`.

Servidor: crear trabajo de BOINC

create_boinc_work()

El método `create_boinc_work()` es el encargado de crear un nuevo trabajo de BOINC.

Las tareas llevadas a cabo por éste método se pueden simplificar en los siguientes pasos:

1. Crear el nombre que identifica la **workunit**.
2. Asociar la nueva tarea a la aplicación en ejecución.
3. Crear el trabajo invocando al método `create_work()` de BOINC.

```
create_work( wu, wu_template,
             RE_TEMPLATE.c_str(),
             config.project_path(RE_TEMPLATE.c_str()),
             input_files, NRO_INFILES, config );
```

Servidor: obtener resultados de clientes

FuD provee el método `inform_completion()` para informar los resultados de las `JobUnit` recibidos desde los clientes a las capas superiores.

Para recibir y manejar los resultados enviados por los clientes de BOINC, se debió integrar el comportamiento del demonio **assimilator** como parte del comportamiento de FuD-BOINC.

¿Cómo?

Se implementó un nuevo hilo de ejecución dentro del servidor FuD-BOINC para que se encargue de chequear la existencia de nuevos resultados no asimilados.

Servidor: obtener resultados de clientes

FuD provee el método `inform_completion()` para informar los resultados de las `JobUnit` recibidos desde los clientes a las capas superiores.

Para recibir y manejar los resultados enviados por los clientes de BOINC, se debió integrar el comportamiento del demonio **assimilator** como parte del comportamiento de FuD-BOINC.

¿Cómo?

Se implementó un nuevo hilo de ejecución dentro del servidor FuD-BOINC para que se encargue de chequear la existencia de nuevos resultados no asimilados.

Servidor: obtener resultados de clientes

FuD provee el método `inform_completion()` para informar los resultados de las `JobUnit` recibidos desde los clientes a las capas superiores.

Para recibir y manejar los resultados enviados por los clientes de BOINC, se debió integrar el comportamiento del demonio **assimilator** como parte del comportamiento de FuD-BOINC.

¿Cómo?

Se implementó un nuevo hilo de ejecución dentro del servidor FuD-BOINC para que se encargue de chequear la existencia de nuevos resultados no asimilados.

Servidor: asimilador de tareas

El thread assimilator cumple con las siguientes funciones:

- Buscar en la base de datos workunits no asimiladas
- Verificar si se encontró resultado canónico para dichas workunits
- Invocar al método `assimilate_handler()`, una vez confirmada la existencia de un resultado canónico

assimilate_handler()

Extrae el resultado de la workunit del archivo de salida enviado por el cliente BOINC para luego informarlo mediante el método `inform_completion()` de FuD.

Servidor: asimilador de tareas

El thread assimilator cumple con las siguientes funciones:

- Buscar en la base de datos workunits no asimiladas
- Verificar si se encontró resultado canónico para dichas workunits
- Invocar al método `assimilate_handler()`, una vez confirmada la existencia de un resultado canónico

assimilate_handler()

Extrae el resultado de la workunit del archivo de salida enviado por el cliente BOINC para luego informarlo mediante el método `inform_completion()` de FuD.

Servidor: asimilador de tareas

Ciclo del `assimilator_thread()`:

```
app = find_app(NAME_APP, _db_assimilator);
while(_running_assimilator)
{
    if (find_work_unit(app, wu) == true)
    {
        if ( find_cannonical_result(wu,canonical_result) == true )
        {
            assimilate_handler(wu,canonical_result);
            update_wu_state(wu, WuDone);
        }
     }
     sleep(SLEEP_INTERVAL);
}
```

Tareas más relevantes del método `assimilate_handler()`:

```
std::string* msg = extract_message(output_file);
ClientsManager::get_instance()->inform_completion(getJobUnitID(),
    msg);
```

Cliente: computación de una tarea

FuD provee el método `run()` el cual debe ser implementado con las tareas específicas que debe realizar el cliente para llevar a cabo el cómputo de las JobUnits.

En FuD-BOINC, el método `run()` se remite a obtener la `JobUnit` de **FuD** a partir del trabajo de BOINC mediante los siguientes pasos:

1. Leer el archivo binario que contiene encapsulada la JobUnit de FuD
2. Extraer el mensaje correspondiente a la JobUnit de FuD
3. Invocar al método `deliver_message()` quien enviará el contenido a las capas superiores para su computación.

Cliente: computación de una tarea

FuD provee el método `run()` el cual debe ser implementado con las tareas específicas que debe realizar el cliente para llevar a cabo el cómputo de las JobUnits.

En FuD-BOINC, el método `run()` se remite a obtener la `JobUnit` de **FuD** a partir del trabajo de BOINC mediante los siguientes pasos:

1. Leer el archivo binario que contiene encapsulada la JobUnit de FuD
2. Extraer el mensaje correspondiente a la JobUnit de FuD
3. Invocar al método `deliver_message()` quien enviará el contenido a las capas superiores para su computación.

Cliente: informar resultado

FuD provee el método `inform_result()` cuya función es enviar el resultado de la computación al servidor.

En FuD-BOINC se remite a invocar al método `boinc_inform_result()` el cual se encarga de las siguientes tareas:

1. Obtener el resultado de la computación
2. Escribir dicho resultado en un archivo de salida asociado al trabajo de BOINC

Cliente: informar resultado

FuD provee el método `inform_result()` cuya función es enviar el resultado de la computación al servidor.

En FuD-BOINC se remite a invocar al método `boinc_inform_result()` el cual se encarga de las siguientes tareas:

1. Obtener el resultado de la computación
2. Escribir dicho resultado en un archivo de salida asociado al trabajo de BOINC

Cliente: informar resultado

Parte del código de `boinc_inform_result()`:

```
std::string body = ProcessorsManager::get_instance()->
    get_return_message();

OutputMessage bos;
bos << _current_id << body;

// File to upload to server.
std::ofstream out;

//enable the exceptions
out.exceptions ( std::ofstream::failbit | std::ofstream::badbit );
out.open(file_name.c_str(), std::ios::binary);

// Get the message with result of computation to send back to server
out << bos.str();
```

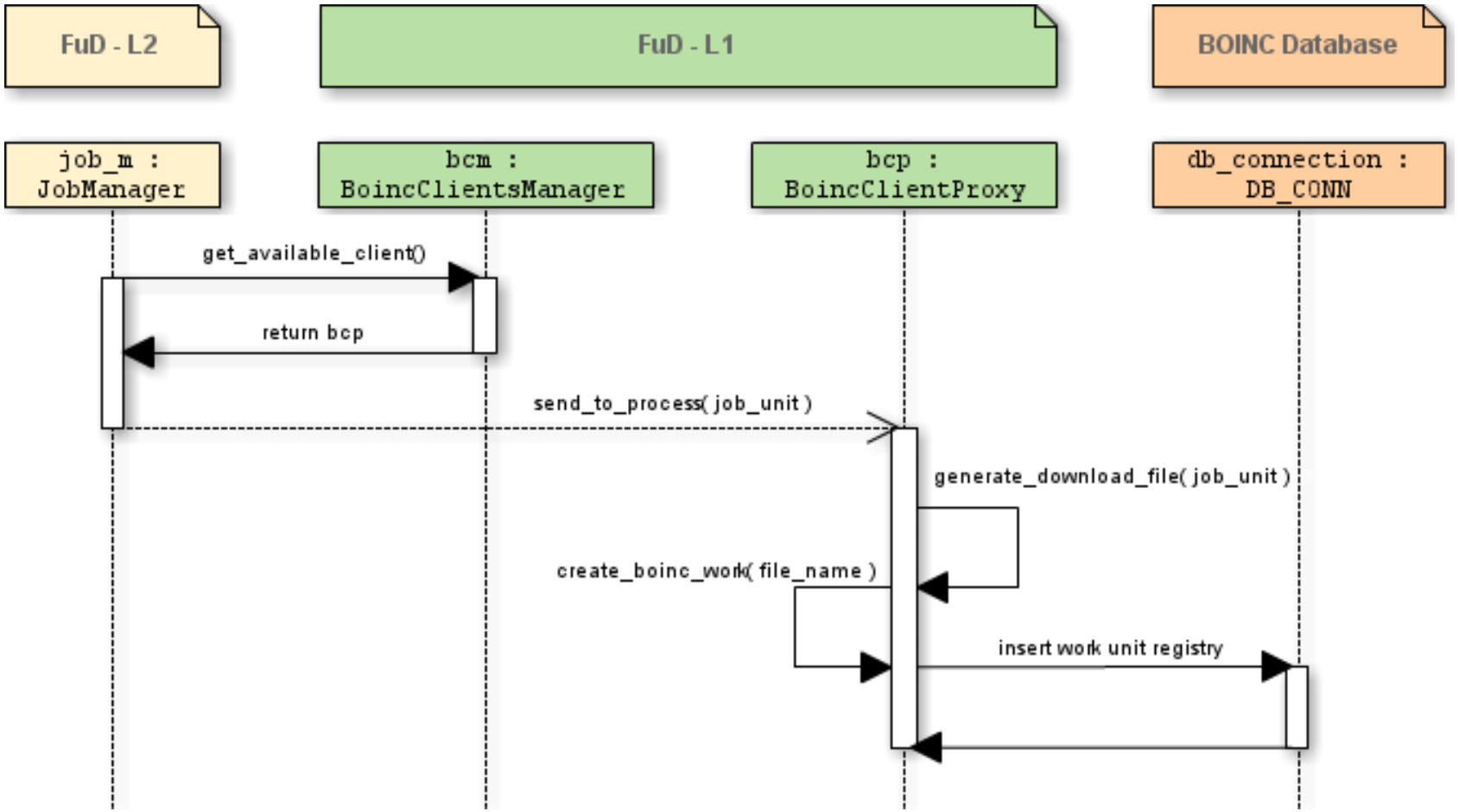
Cliente: compilación en Windows

Fue necesario compilar el cliente de FuD-BOINC para sistemas operativos Windows ya que actualmente son los más utilizados.

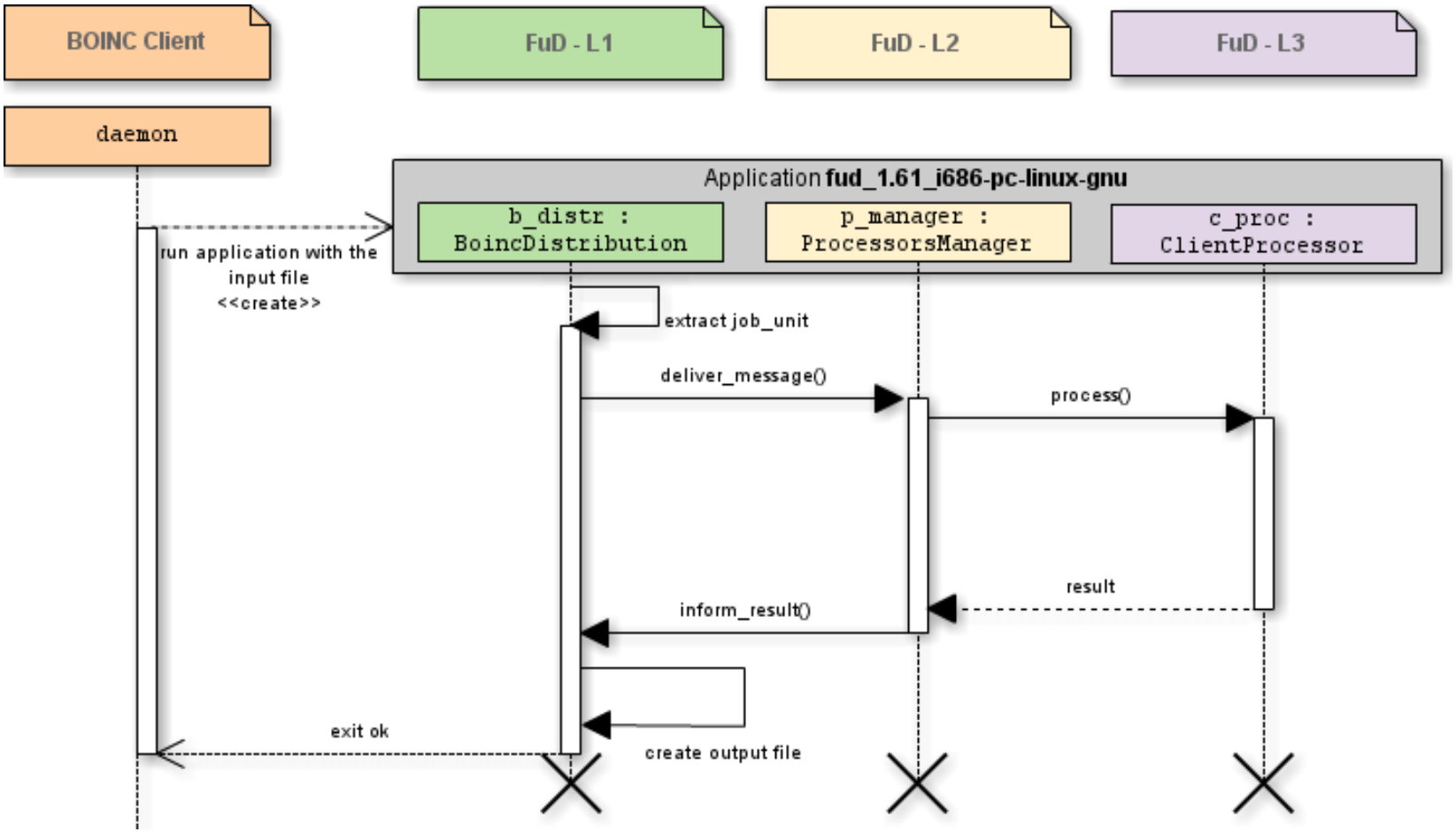
Pasos para la compilación:

1. Compilar las librerías BOINC.
2. Generar y configurar un proyecto solución de *Visual Studio* para la compilación de FuD-BOINC.
3. Resolver dependencias con las librerías: boost, pthread, mili y MySQL
4. Modificar el tipo de `include` realizado en algunos archivos del cliente **FuD**.
5. Compilar el cliente de **FuD**.

Servidor: envío de una JobUnit



Cliente: computación de una JobUnit



Rediseño de FuD

Durante el desarrollo de este proyecto se debieron efectuar diversos cambios en el diseño e implementación original de FuD para poder integrar esta nueva capa de distribución.

- Redeclaración del método `create_distribution_client()`
- `JobManager` post initialization
- Reenvío de `JobUnits` configurable
- Múltiples `JobUnits` a clientes

Redeclaración del método create_distribution_client()

Método provisto por **FuD** para la creación de un cliente de distribución.

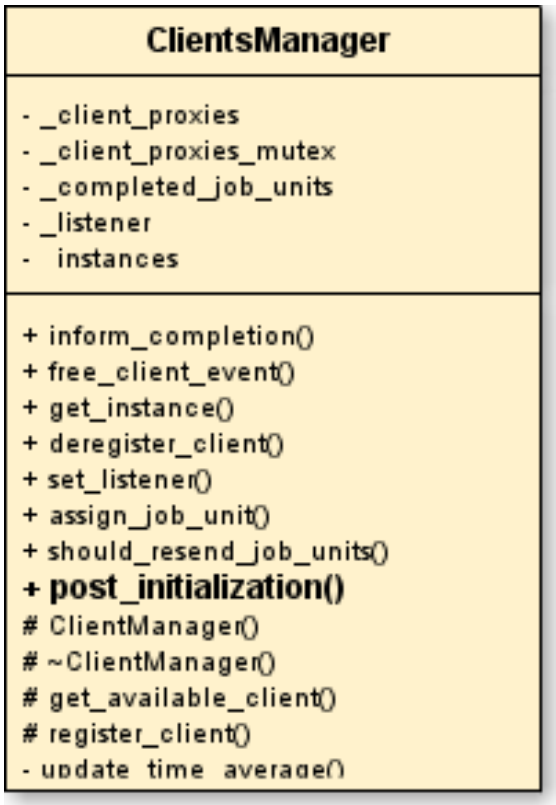
Prototipo original:

```
DistributionClient* create_distribution_client(
        std::string address = "127.0.0.1", Port port = 31337);
```

Prototipo actual:

```
DistributionClient* create_distribution_client(
        int argc, char** argv);
```

JobManager post initialization



- Se agregó un nuevo método en la clase `ClientsManager` de **FuD**.
- Por defecto, su implementación es vacía.
- El método sólo es invocado al final del constructor de `JobManager`:

```
_clients_manager->post_initialization()
```

Redefinición de post_initialization()

Declaración y definición en ClientsManager

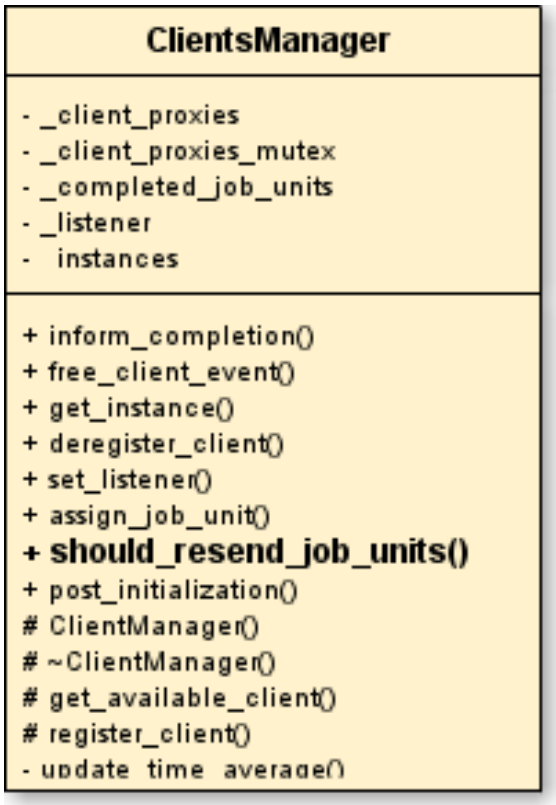
```
virtual void post_initialization() {};
```

Redefinición en BoincClientsManager

```
void BoincClientsManager::post_initialization()
{    boinc_register_client();    }
```

```
void BoincClientsManager::boinc_register_client()
{
    // Init the client_proxy.
    boinc_log_debug(std::string("Registering BOINC with FuD."));
    BoincClientProxy * client = new BoincClientProxy();
    client->set_concurrent_jobs(UNLIMITED_JOBS);
    // Register the unique client proxy.
    register_client(client);
}
```

Reenvío de JobUnits configurable



- Se agregó un nuevo método en la clase ClientsManager de **FuD**.

```
virtual bool should_resend_job_units() = 0;
```

- Definición del método en BoincClientsManager:

```
bool BoincClientsManager::should_resend_job_units()
{
    return false;
}
```

Reimplementación de handle_free_client_event()

```
170 if (_jobQueue.empty())
171 {
172     if (_clients_manager->should_resend_job_units())
173     {
174         if (! _pendingList.empty())
175         {
176             if (_clients_manager->assign_job_unit(*_pendingList.front()))
177             {
178                 //send this one to the back, act as Round Robin
179                 _pendingList.push_back(_pendingList.front());
180                 _pendingList.pop_front();
181             }
182             else
183                 syslog(LOG_NOTICE, "Error sending JobUnit %u from Pending List to a client.",
                        _pendingList.front()->get_id());
184         }
185     }
186 }
187 else
188 {
189     if (_clients_manager->assign_job_unit(*_jobQueue.front()))
190     {
191         syslog(LOG_NOTICE, "Sending JobUnit %u to pending list.", (_jobQueue.front()->get_id()))
                ;
192         _pendingList.push_back(_jobQueue.front());
193         _jobQueue.pop_front();
194     }
195     else
196         syslog(LOG_NOTICE, "Error sending JobUnit %u from Job Queue to a client.", _jobQueue.
                front()->get_id());
197 }
```

Múltiples JobUnits a clientes

Problema

Los clientes sólo pueden procesar a lo sumo de a una JobUnit por vez.

Solución

La idea general consiste en:

1. Permitir configurar la cantidad máxima de tareas en simultáneo que un cliente puede ejecutar.
2. Llevar un registro de la cantidad de tareas que un cliente se encuentra computando en un determinado momento.
3. Determinar la disponibilidad de un cliente.
4. Si el cliente se encuentra disponible, generar un nuevo evento de cliente libre.

Múltiples JobUnits a clientes

Problema

Los clientes sólo pueden procesar a lo sumo de a una JobUnit por vez.

Solución

La idea general consiste en:

1. Permitir configurar la cantidad máxima de tareas en simultáneo que un cliente puede ejecutar.
2. Llevar un registro de la cantidad de tareas que un cliente se encuentra computando en un determinado momento.
3. Determinar la disponibilidad de un cliente.
4. Si el cliente se encuentra disponible, generar un nuevo evento de cliente libre.

Múltiples JobUnits a clientes

Problema

Los clientes sólo pueden procesar a lo sumo de a una JobUnit por vez.

Solución

La idea general consiste en:

1. Permitir configurar la cantidad máxima de tareas en simultáneo que un cliente puede ejecutar.
2. Llevar un registro de la cantidad de tareas que un cliente se encuentra computando en un determinado momento.
3. Determinar la disponibilidad de un cliente.
4. Si el cliente se encuentra disponible, generar un nuevo evento de cliente libre.

Múltiples JobUnits a clientes

Problema

Los clientes sólo pueden procesar a lo sumo de a una JobUnit por vez.

Solución

La idea general consiste en:

1. Permitir configurar la cantidad máxima de tareas en simultáneo que un cliente puede ejecutar.
2. Llevar un registro de la cantidad de tareas que un cliente se encuentra computando en un determinado momento.
3. Determinar la disponibilidad de un cliente.
4. Si el cliente se encuentra disponible, generar un nuevo evento de cliente libre.

Múltiples JobUnits a clientes

Problema

Los clientes sólo pueden procesar a lo sumo de a una JobUnit por vez.

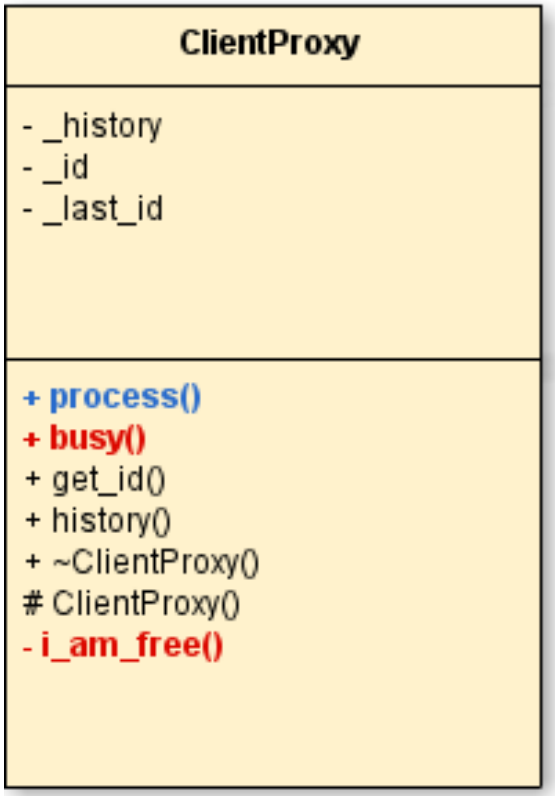
Solución

La idea general consiste en:

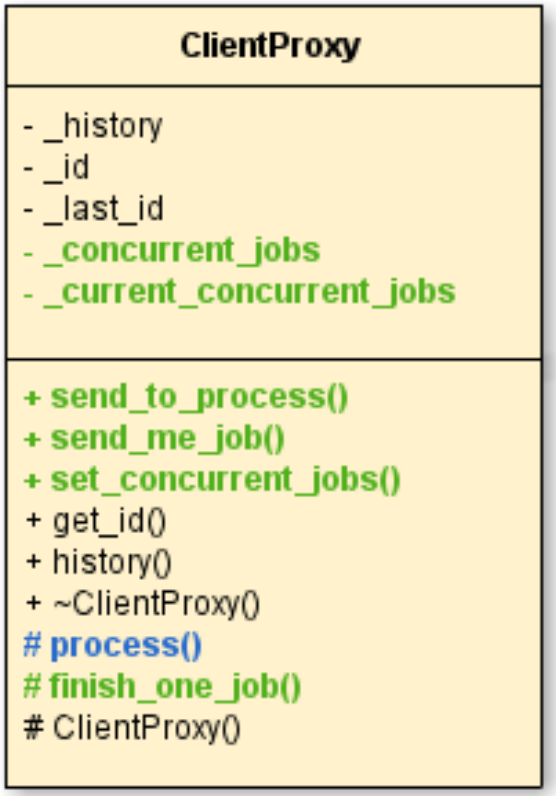
1. Permitir configurar la cantidad máxima de tareas en simultáneo que un cliente puede ejecutar.
2. Llevar un registro de la cantidad de tareas que un cliente se encuentra computando en un determinado momento.
3. Determinar la disponibilidad de un cliente.
4. Si el cliente se encuentra disponible, generar un nuevo evento de cliente libre.

ClientProxy

Diseño Original

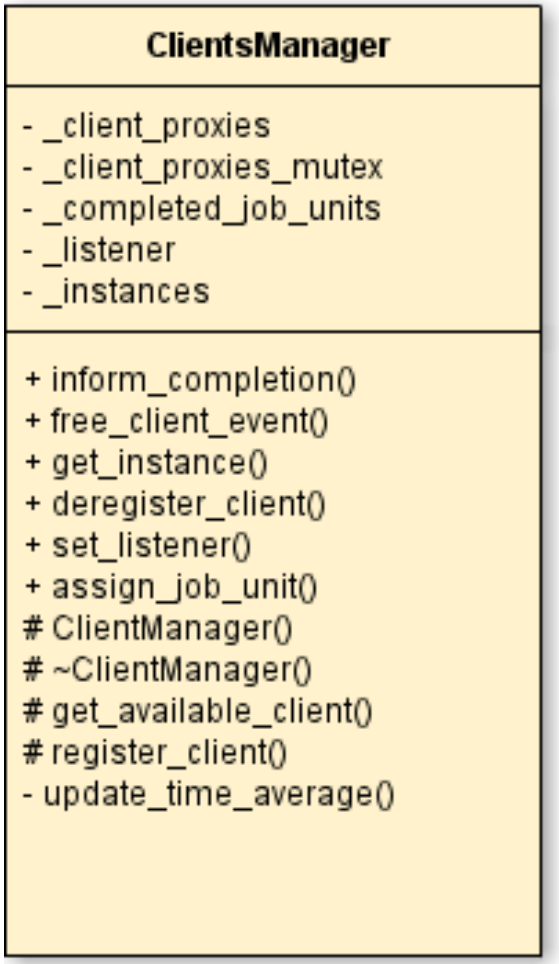


Diseño Actual



ClientsManager

Diseño Original



Diseño Actual

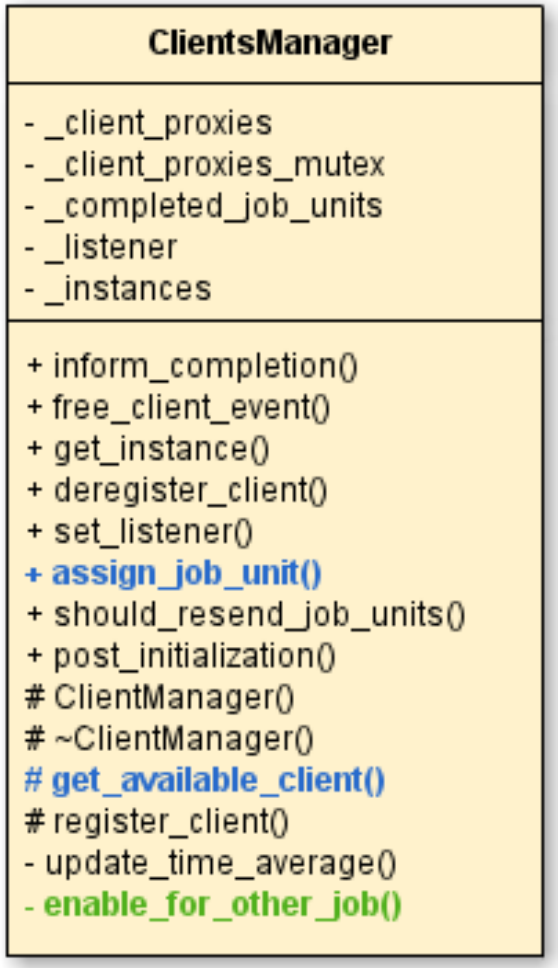
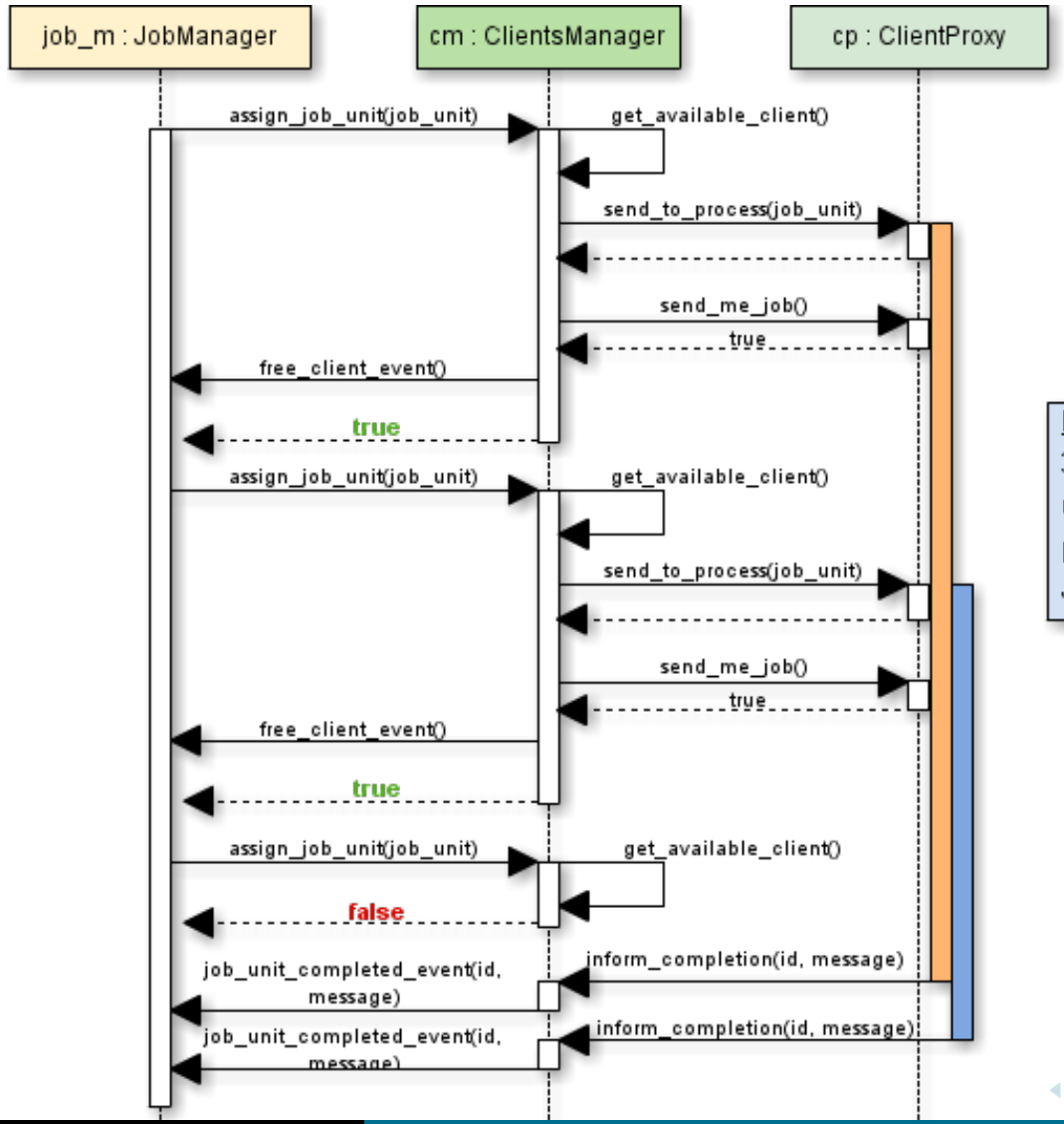


Diagrama de secuencia: multiple jobs



Escenario: el servidor debe enviar 3 JobUnits. En ese momento hay un único cliente conectado el cual puede procesar un máximo de 2 JobUnits al mismo tiempo.

Parallel-Clusterer

Parallel-clusterer

Es una aplicación utilizada por la organización FuDePAN. Fue desarrollada utilizando el framework FuD y realiza la siguiente tarea:

Tarea:

Dado un conjunto de diferentes cadenas principales o esqueletos de una proteína (cada uno representado por un vector de átomos), agrupa estos elementos bajo similitudes geométricas de las posiciones de sus átomos. El resultado final es un conjunto de agrupaciones o clusters donde cada uno tiene esqueletos de proteínas cuya estructura geométrica es muy similar.

Parallel-Clusterer

Parallel-clusterer

Es una aplicación utilizada por la organización FuDePAN. Fue desarrollada utilizando el framework FuD y realiza la siguiente tarea:

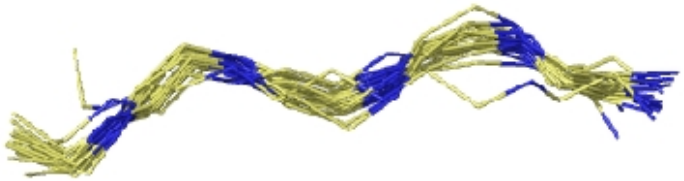
Tarea:

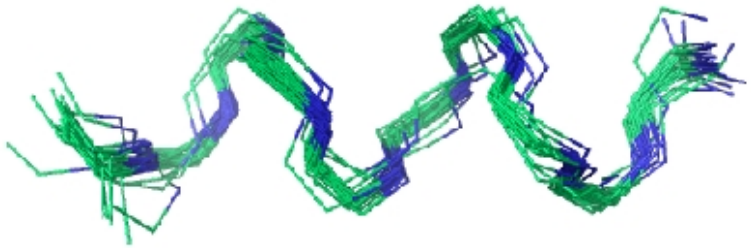
Dado un conjunto de diferentes cadenas principales o esqueletos de una proteína (cada uno representado por un vector de átomos), agrupa estos elementos bajo similitudes geométricas de las posiciones de sus átomos. El resultado final es un conjunto de agrupaciones o clusters donde cada uno tiene esqueletos de proteínas cuya estructura geométrica es muy similar.

Parallel-clusterer

Estructura de un esqueleto de proteína:

Clusters resultantes:





Análisis de rendimiento

Es importante realizar el análisis de rendimiento de una aplicación distribuida con el propósito de estudiar los costos/beneficios que ésta posee ante la comparación con su versión secuencial.

Se muestra el análisis de rendimiento de la aplicación Parallel-Clusterer compilada con FuD-BOINC en términos de las siguientes métricas:

- Aceleración
- Eficiencia
- Overhead
- Costo

La ejecución se llevo a cabo variando de 1 a 5 clientes conectados.

Tiempo de ejecución

Tiempo secuencial (T_s)

Se define como el intervalo de tiempo que transcurre desde que un programa secuencial se inicia hasta que finaliza.

Tiempo distribuido (T_p)

Se calcula desde que se inicia hasta que el último nodo finaliza su ejecución.

Factores que afectaron al tiempo de ejecución de la aplicación distribuida:

- Tiempo de descarga de archivos de entrada y de la aplicación.
- Tiempo de subida de los archivos resultantes.
- Tiempo entre requerimientos al servidor.

Tiempo de ejecución

Tiempo secuencial (T_s)

Se define como el intervalo de tiempo que transcurre desde que un programa secuencial se inicia hasta que finaliza.

Tiempo distribuido (T_p)

Se calcula desde que se inicia hasta que el último nodo finaliza su ejecución.

Factores que afectaron al tiempo de ejecución de la aplicación distribuida:

- Tiempo de descarga de archivos de entrada y de la aplicación.
- Tiempo de subida de los archivos resultantes.
- Tiempo entre requerimientos al servidor.

Tiempo de ejecución

Tiempo secuencial (T_s)

Se define como el intervalo de tiempo que transcurre desde que un programa secuencial se inicia hasta que finaliza.

Tiempo distribuido (T_p)

Se calcula desde que se inicia hasta que el último nodo finaliza su ejecución.

Factores que afectaron al tiempo de ejecución de la aplicación distribuida:

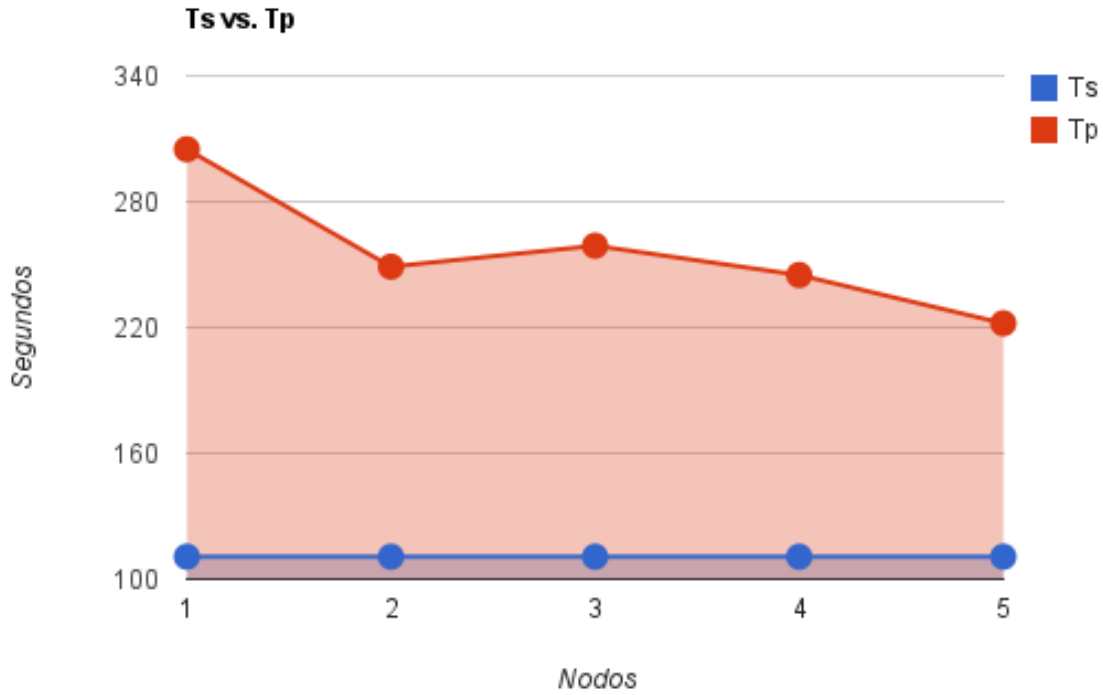
- Tiempo de descarga de archivos de entrada y de la aplicación.
- Tiempo de subida de los archivos resultantes.
- Tiempo entre requerimientos al servidor.

Tiempo de ejecución

Tiempo secuencial (T_s) vs. Tiempo paralelo (T_p)

$T_s = 110.7$ segundos.

p	T_p
1	305
2	249
3	259
4	245
5	222



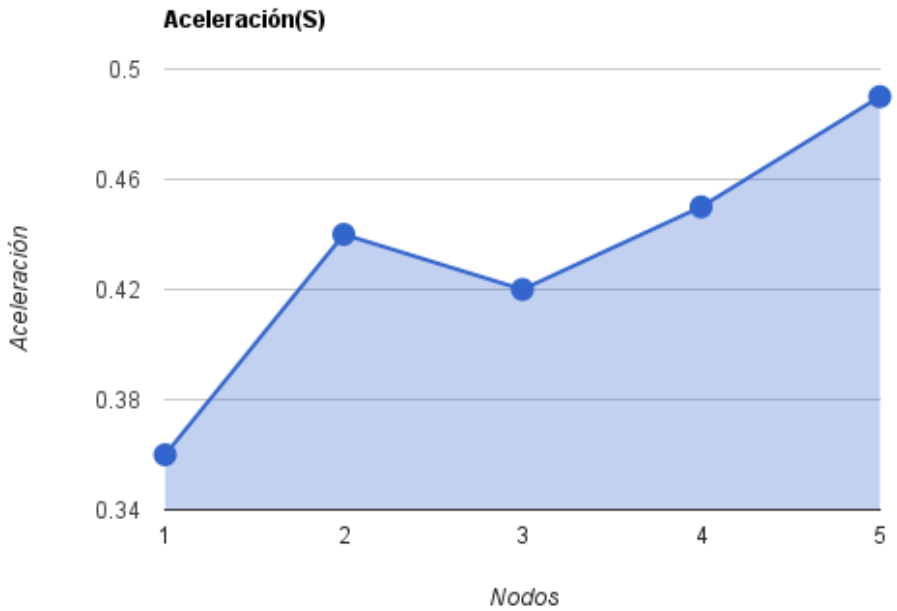
Aceleración

Definición

Es una medida que arroja el beneficio relativo de resolver un problema en paralelo. Indica la ganancia de velocidad obtenida.

Se define como: $S = T_s / T_p$

p	S
1	0.36
2	0.44
3	0.42
4	0.45
5	0.49



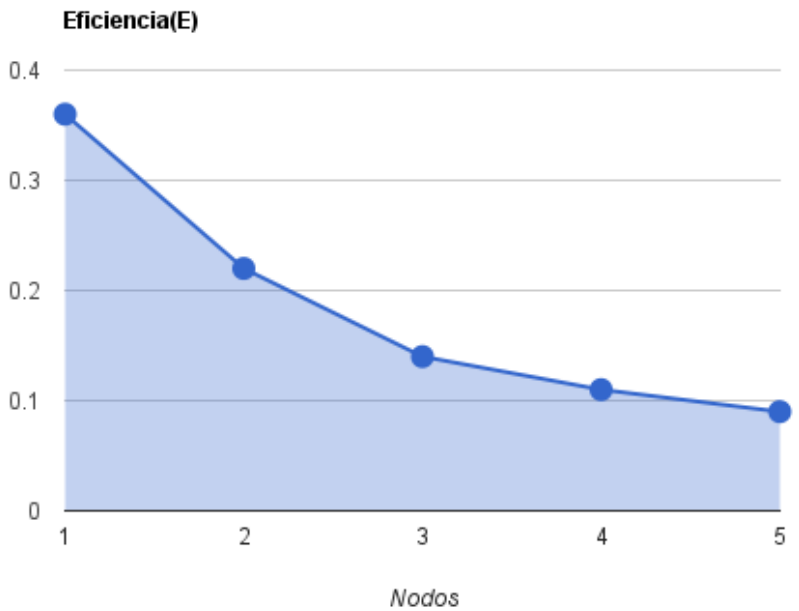
Eficiencia

Definición

Indica el grado de utilidad de cada elemento de procesamiento.

Se define como: $E = S/p$

p	E
1	0.36
2	0,22
3	0,14
4	0,11
5	0,09



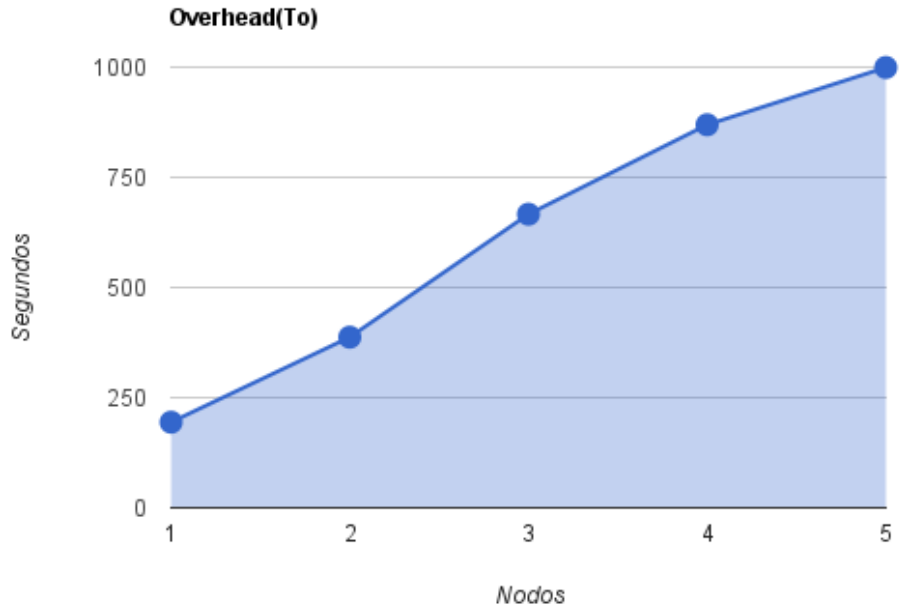
Overhead

Definición

Indica el trabajo adicional que realiza un programa paralelo respecto a la solución secuencial equivalente.

Se define como: $T_o = p * T_p - T_s$

p	T_0
1	194.3
2	387.3
3	666.3
4	869.3
5	999.3



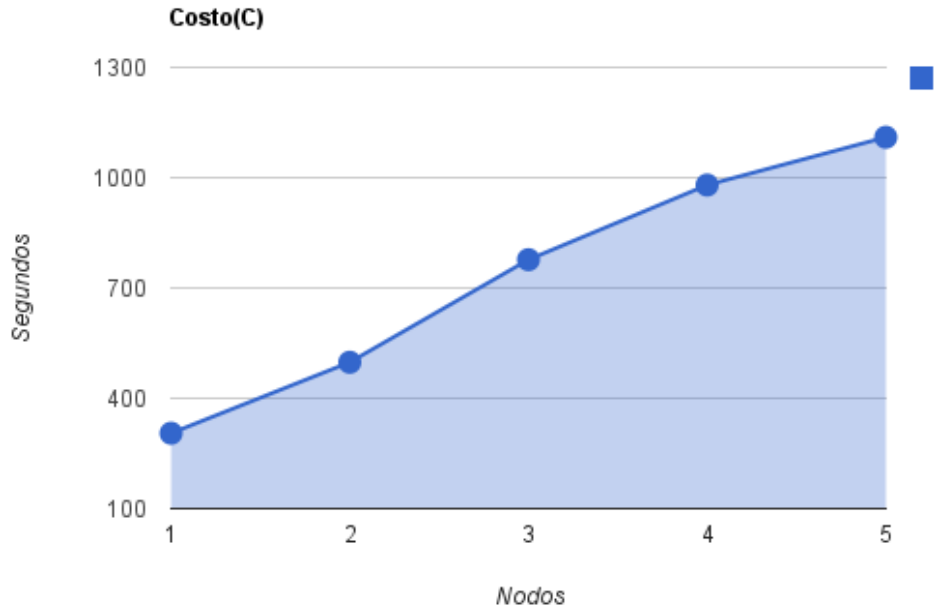
Costo

Definición

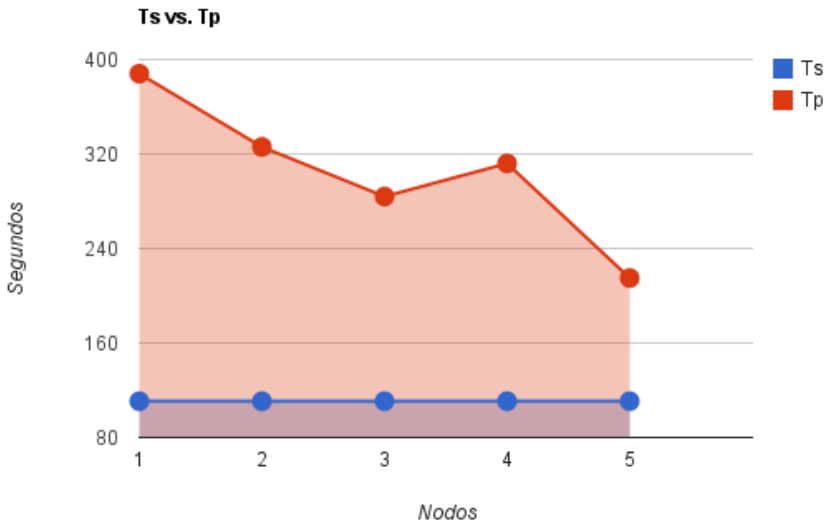
Refleja la suma de los tiempos de ejecución de cada unidad de procesamiento.

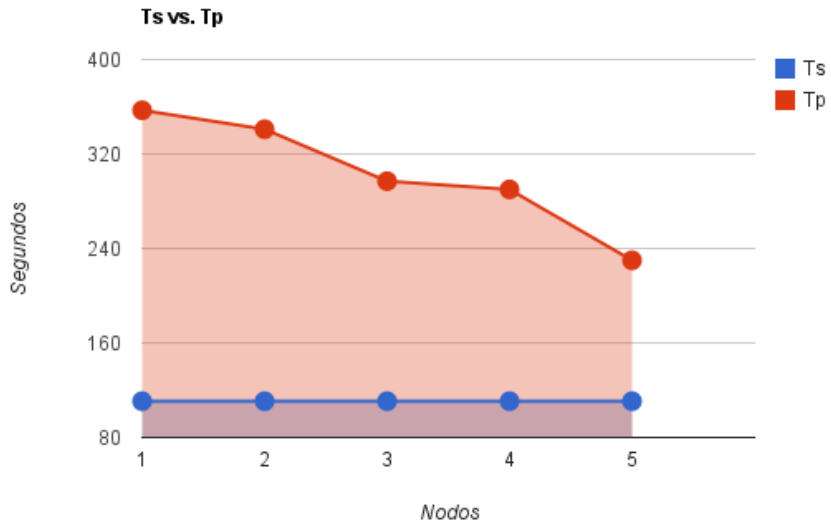
Se define como: $C = p * T_p$

p	C
1	305
2	498
3	777
4	980
5	1110



Otros resultados: tiempos de ejecución





Conclusión

- La nueva capa de distribución, junto con el rediseño efectuado, mantiene compatibilidad con el modelo original de **FuD**.
- Los desarrolladores pueden implementar aplicaciones clientes de FuD-BOINC que pueden correr tanto en Linux como en Windows.
- FuD-BOINC debería ser utilizado por aplicaciones que requieran gran cantidad de procesamiento.
- La investigación realizada permitió desenvolvernos satisfactoriamente ante un tema desconocido: computación voluntaria con BOINC.

Conclusión

- La nueva capa de distribución, junto con el rediseño efectuado, mantiene compatibilidad con el modelo original de **FuD**.
- Los desarrolladores pueden implementar aplicaciones clientes de FuD-BOINC que pueden correr tanto en Linux como en Windows.
- FuD-BOINC debería ser utilizado por aplicaciones que requieran gran cantidad de procesamiento.
- La investigación realizada permitió desenvolvernos satisfactoriamente ante un tema desconocido: computación voluntaria con BOINC.

Conclusión

- La nueva capa de distribución, junto con el rediseño efectuado, mantiene compatibilidad con el modelo original de **FuD**.
- Los desarrolladores pueden implementar aplicaciones clientes de FuD-BOINC que pueden correr tanto en Linux como en Windows.
- FuD-BOINC debería ser utilizado por aplicaciones que requieran gran cantidad de procesamiento.
- La investigación realizada permitió desenvolvernos satisfactoriamente ante un tema desconocido: computación voluntaria con BOINC.

Conclusión

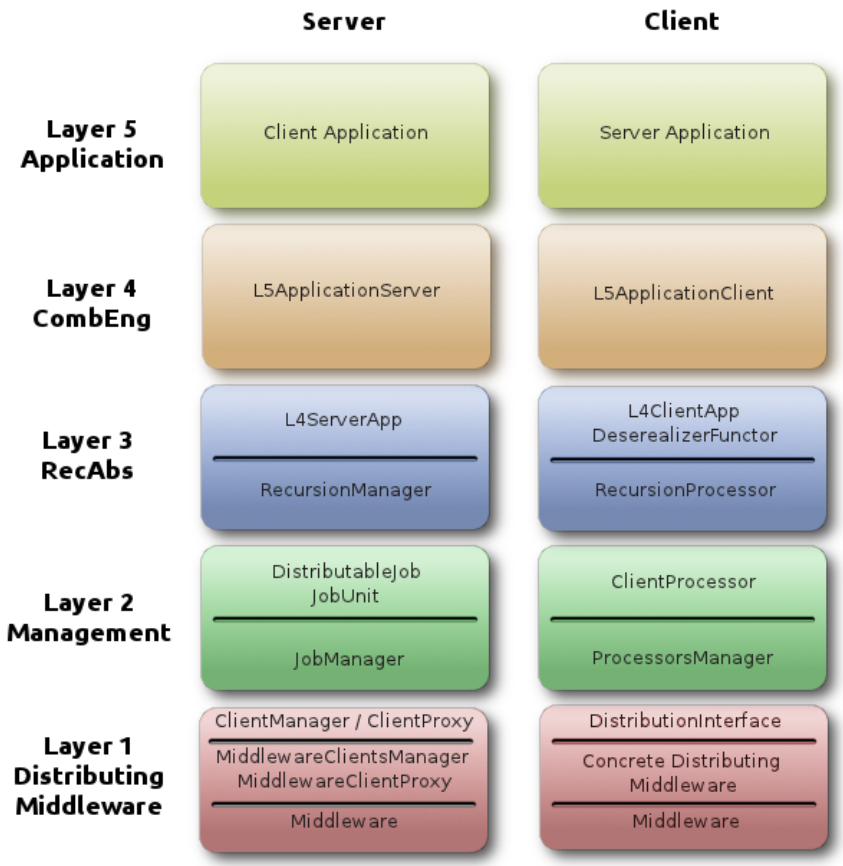
- La nueva capa de distribución, junto con el rediseño efectuado, mantiene compatibilidad con el modelo original de **FuD**.
- Los desarrolladores pueden implementar aplicaciones clientes de FuD-BOINC que pueden correr tanto en Linux como en Windows.
- FuD-BOINC debería ser utilizado por aplicaciones que requieran gran cantidad de procesamiento.
- La investigación realizada permitió desenvolvernos satisfactoriamente ante un tema desconocido: computación voluntaria con BOINC.

Trabajos a futuro

- Instruir y profundizar los conocimientos sobre la administración y configuración de un proyecto BOINC.
- Lanzar oficialmente un proyecto de computación voluntaria **FuDePAN@HOME**.
- Implementar un screensaver con un logo personalizado de FuDePAN.
- Agregar la entrega de créditos a los clientes respecto de la cantidad de trabajos procesados.

Trabajos a futuro

- Integrar el funcionamiento de las nuevas capas **L3** y **L4** de **FuD** con esta capa de distribución.



¿Preguntas?

¡Gracias!