Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

Tipo de artículo: Artículo original
Temática: Programación paralela y distribuida
Recibido: 18/02/2020 | Aceptado: 1/04/2020

Biblioteca en C++ para trabajo con clusters

C++ library for work with clusters

Ing. Beatriz Valdés Díaz[1*] https://orcid.org/0000-0002-1534-7125
Dr. C. Fernando José Artigas Fuentes[2] https://orcid.org/0000-0003-4977-2135

[1]Departamento de Química. Facultad de Ciencias Naturales y Exactas. Universidad de Oriente. Cuba
[2]Centro de Estudios de Neurociencias y Procesamiento de Imágenes y Señales. Facultad de Ingeniería en Telecomunicaciones, Informática y Biomédica. Universidad de Oriente. Cuba

*Autor para la correspondencia: bvaldes@uo.edu.cu

RESUMEN

Los *clusters* de computadoras son importantes para el desarrollo de investigaciones científicas complejas. Estudios realizados evidencian que las herramientas empleadas en el proceso de trabajo con *clusters*, resultan complicadas para los usuarios menos relacionados con especialidades técnicas. Este trabajo presenta el diseño e implementación de una biblioteca en C++ orientada a los programadores, que organiza el proceso de trabajo con un *cluster* en una jerarquía de capas. Esto permite que el programador solo haga uso de funciones de alto nivel de abstracción, más cercanas al problema a resolver por el usuario. La biblioteca, además de permitir a los programadores integrar aplicaciones de escritorio específicas de los usuarios con *clusters*, viabiliza parte del trabajo con estos de forma desatendida y agiliza la realización de estudios de escalabilidad de programas paralelos. Asimismo, especifica los recursos del *cluster* y

características de aplicaciones necesarios para ejecutar los programas mediante recetas predefinidas. Como parte de la implementación se utilizó el componente de gestión de trabajo de la biblioteca GridMD, y las bibliotecas *boost_serialization* y *boost_filesystem*. Las pruebas realizadas para comprobar el funcionamiento de la biblioteca sobre un *cluster* con Linux y Slurm mostraron un buen desempeño sin errores identificados.

Palabras clave: Biblioteca; cluster; arquitectura por capas; HPC; C++.

ABSTRACT

Computer clusters are important for the development of complex scientific research. Some studies demonstrate that tools used on clusters are complicated for users less related to technical sciences. This paper presents the design and implementation of a C++ library targeted to programmers, which arranges the work with a cluster into layer architecture. This allows programmers to use high-level abstraction functions, closer to the user's problem to be solved. The library allows not only integrate user´s desktop applications with clusters but also makes part of the work with them in an unattended way and speeds up scalability studies of parallel programs. Likewise, specifies cluster resources and job features required to run the programs by using predefined recipes. As part of the implementation, the job manager component of the GridMD library, and the boost_serialization and boost_filesystem libraries were used. All the tests carried out to the library on a cluster with Linux and Slurm have shown a good performance without identified errors.

Keywords: library; cluster; layered architecture; HPC; C++.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

Introducción

La Computación de Alto Rendimiento (*High Performance Computing* o HPC en inglés) es un campo de trabajo que tiene como objetivo aprovechar al máximo las capacidades de una computadora. Para ello, utiliza técnicas especiales de programación con la finalidad de enfrentar problemas que requieren un modelado complejo, simulaciones en tiempos acotados o procesamiento de grandes cantidades de datos. Esta tecnología es fundamental en el desarrollo de actividades como el descubrimiento de fármacos, la predicción meteorológica, los estudios geológicos, la simulación de los efectos del cambio climático, entre otros. (Sterling, y otros, 2017)

Para problemas suficientemente complejos, HPC hace uso de supercomputadoras y técnicas de paralelización, que posibilitan acelerar la obtención de los resultados. Debido al alto costo de las supercomputadoras, HPC también se apoya en sistemas como los *clusters*. Un *cluster* es un conjunto de computadoras enlazadas por una red informática, y que se comportan como un solo sistema usando el poder de cómputo de procesadores combinados. Este sistema puede ser más rentable que un único equipo de rendimiento comparable. (Zakarya, y otros, 2017)

Teniendo en cuenta la habilidad de los usuarios para interactuar con los *clusters* estos pueden ser clasificados en tres grupos. Por una parte, están los usuarios expertos que son capaces de instalar esta tecnología, y dominan las herramientas necesarias para su uso, como la programación paralela, los comandos de sistema, gestores de recursos (Wu, y otros, 2017; Sidhu, 2016), y protocolos SSH y SCP. En segundo lugar, se encuentran los usuarios intermedios que tienen cierta habilidad para interactuar con *clusters*, pero prefieren emplear interfaces web como las descritas en (Ragni, 2013; Calegari, y otros, 2019; Cruz, 2018), que les permiten centrarse en trabajos de sus especialidades más que en aspectos de HPC. Finalmente, existe otro conjunto de usuarios básicos para los que aún estas soluciones resultan complicadas.

La mayoría de las tecnologías mencionadas están fuera del dominio de los usuarios menos relacionados con las ciencias técnicas. (Carnevale, y otros, 2014; Sampedro, y otros, 2017) A pesar de que las interfaces web citadas realizan las operaciones principales de interacción con un *cluster* desde una única herramienta, delegan a los usuarios la especificación de los recursos de hardware y particularidades de una aplicación determinada. Esto puede provocar la solicitud incorrecta de los recursos del *cluster*, y un uso ineficiente del mismo.

(Bogdanov, y otros, 2017) Igualmente, estas interfaces carecen de facilidades que permitan realizar tareas más complejas, como la ejecución de una misma aplicación con varios juegos de datos o recursos. Actualmente, en cada caso se debe definir y ejecutar múltiples veces el mismo trabajo, haciendo más complejo el uso de este servicio.

Este problema particular se puede resolver al tener aplicaciones escritas por desarrolladores para problemas específicos de los usuarios básicos (Potter, 2014), que aprovechen de manera transparente las potencialidades de un *cluster*. Con el fin de usar los recursos de *clusters* mediante estas aplicaciones, se deben emplear mecanismos de comunicación, habitualmente implementados en bibliotecas basadas en el protocolo SSH. (Adamiantiadis, y otros, 2020; Zadka, 2019; Golemon, 2020; Van, 2020) Estas bibliotecas realizan las tareas principales de comunicación con un servidor genérico, pero no están orientadas al trabajo específico con un *cluster*, donde se realizan operaciones de mayor complejidad.

Una solución más específica es la biblioteca GridMD (Valuev, y otros, 2015), que tiene como finalidad desarrollar aplicaciones distribuidas que se ejecutan en *clusters* y sistemas Grid. Entre sus características principales están la gestión de los trabajos, ejecución automática de *workflows*, y soporte para diferentes gestores de recursos y herramientas de acceso remoto. Sin embargo, para especificar los recursos del *cluster* requeridos por una aplicación, un programador necesita hacer uso de funciones de diversos niveles de abstracción de esta biblioteca.

Por este motivo, en el presente trabajo, se propone una nueva biblioteca en C++, basada en software libre, que organiza las diversas funcionalidades en una jerarquía de capas. Esto permite que un programador solo haga uso de funciones de alto nivel de abstracción, más cercanas al problema a resolver por el usuario. Esta propuesta, además de permitir a los programadores integrar aplicaciones de escritorio con *clusters*, realiza parte del trabajo con estos de forma desatendida. A diferencia de la GridMD, los recursos del *cluster* y otros detalles requeridos por los programas se especifican en recetas predefinidas. Adicionalmente, se ofrece un mecanismo para agilizar la realización de estudios de escalabilidad de programas paralelos. Como parte de la implementación se usó el componente de gestión de trabajo de la GridMD y las bibliotecas *boost_serialization y boost_filesystem*.

Este trabajo está compuesto por cuatro secciones, la primera ya descrita plantea el problema y objetivo de la investigación. La segunda expone la arquitectura de la biblioteca, especificando los criterios de diseño

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

empleados, y la independencia y jerarquía de las capas. En la tercera sección se muestra el flujo de trabajo con la biblioteca y las pruebas realizadas para corroborar la factibilidad y pertinencia de los resultados del trabajo. Por último, se presentan las conclusiones de la investigación y las recomendaciones futuras.

Métodos o Metodología Computacional

Arquitectura de la biblioteca

Como se explicó en el apartado anterior, el diseño de la biblioteca propuesta se basa en la programación por capas. El proceso de trabajo con un *cluster* se distribuye en tres capas que trabajan de forma independiente: *Jobs*, *Generation* y *Tasks* (Fig. 1). *Jobs* atiende las peticiones de los trabajos de los usuarios, *Generation* asigna los recursos a las tareas, y *Tasks* gestiona las tareas del *cluster* auxiliándose de funciones de la GridMD. Cada una de las capas solo llama a funciones definidas por la del nivel adyacente inferior y brinda servicios a la adyacente superior.

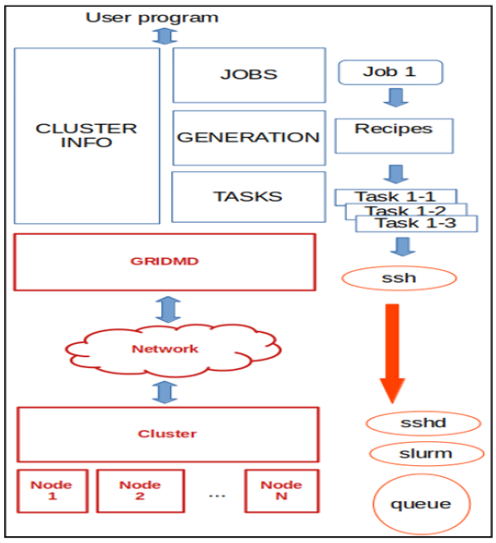


Fig. 1 - Arquitectura de la biblioteca.

La adopción de este estilo de programación facilita la actualización por separado de cada capa sin afectar el funcionamiento de las restantes, ya que cada nivel cuenta con una función específica. En caso de ser necesario, se puede escalar la arquitectura de la biblioteca, o sustituir alguna capa con una implementación alternativa. (Sarasty, 2016)

Asimismo, este esquema de desarrollo propicia que cada capa cuente con un nivel de abstracción diferente. En este caso, las capas de nivel inferior manejan problemas relacionados con el *cluster*, y las de mayor nivel tratan los específicos del usuario. De este modo el programador, haciendo uso de la capa de más alto nivel, puede enfocarse en el desarrollo de aplicaciones específicas del campo de trabajo de los usuarios. En otro sentido, a medida que se procesan los datos, estos se van transformando y obteniendo con un significado diferente en cada capa. Esto se puede notar al gestionar trabajos compuestos por varias subtareas. El estado de estas últimas se controla de manera independiente en la capa *Tasks*, y se informa al usuario de un estado general del trabajo primario en la capa *Jobs*.

No obstante, existen funciones generales para todas las capas, lo que significa que mantienen el mismo resultado en todos los niveles. Por ejemplo, las funciones que ofrecen información de los recursos del *cluster*, tales como la cantidad de nodos y núcleos existentes. Estas funciones se implementaron en la capa lateral *Cluster_Info*.

Descripción de las capas

Capa Jobs

La capa *Jobs* permite al usuario de la biblioteca (un programador) especificar y solicitar la ejecución de nuevos trabajos en el *cluster*. Esta capa valida la información que describe al trabajo, crea e inserta el trabajo en el contenedor de trabajos y envía los datos a la capa de *Generation*. Luego retorna el identificador del nuevo trabajo, un número entero mediante el cual se obtiene el estado del trabajo o la ruta donde se encuentran los resultados de su ejecución en la computadora del usuario. Todo ello, realizando llamadas a funciones de la capa *Generation*.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

La capa *Jobs* se compone de las clases *Job* y *JobManager*. La clase *Job* (Fig. 2a) permite especificar las características de un trabajo como el nombre, comando de ejecución, y ficheros de entrada y salida. A su vez, cuenta con atributos para indicar si lo que se suministró fue un código fuente que requiere ser compilado antes de ejecutarse. Para ello, permite definir el comando de compilación, el camino de los ficheros del código fuente y el nombre del ejecutable resultante. La clase posee métodos para el acceso y modificación de cada atributo mencionado, y la gestión del trabajo. Estos últimos invocan a los métodos análogos de la clase *JobManager*.

La clase *JobManager* (Fig. 2b) gestiona trabajos, ya sean secuenciales, paralelos o experimentos. Se considera como experimento a un trabajo compuesto por varias subtareas, cada una de las cuales ejecuta el mismo programa sobre recursos diferentes del *cluster*, ya sea variando la cantidad de nodos o de núcleos. Para ello, cuenta con funciones que permiten la creación, envío, monitoreo y eliminación de los trabajos de los usuarios. Tiene como atributos la estructura *Job_User* que almacena los datos de acceso al *cluster* tales como el nombre o número IP del mismo y las credenciales del usuario, un objeto para comunicarse con la clase *Generation*, el identificador del último trabajo creado, y un contenedor con los identificadores e instancias respectivas de la clase *Job*.

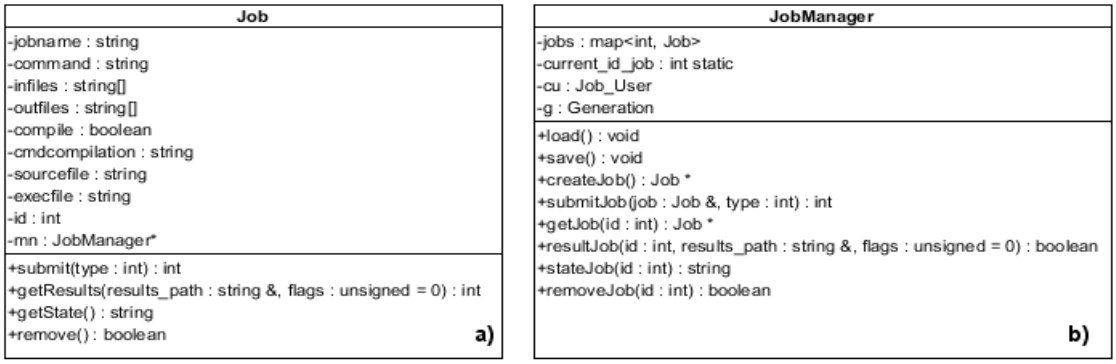


Fig. 2 - Diagramas UML de las clases: a) Job b) JobManager.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

El listado de trabajos y el identificador se guardan en el fichero *jobsStore* mediante la serialización de la clase *Job*, utilizando la biblioteca *boost_serialization*. Esto permite que ante fallas o reinicios de la aplicación del usuario se recuperen las características de los trabajos sin conectarse al *cluster*. En este sentido, cuando se crea la instancia de la clase *JobManager*, si existe el fichero mencionado, se carga de este el listado de los trabajos y se le asigna a ese objeto (método *load*). Cada vez que se modifica el listado de trabajos, se actualiza el fichero con el método *save*. Para garantizar el éxito de las operaciones sobre *jobsStore*, el método *save* ignora posibles señales inesperadas que puedan ocurrir, como las interrupciones de terminación de programa *SIGINT* y *SIGTERM*. Una vez finalizada la actualización del fichero, se reasigna la acción predeterminada de cada señal.

Capa Generation

Esta capa recibe del nivel superior la información que identifica a un trabajo. A partir de esta información, y usando recetas predefinidas (Fig. 3), genera una o varias tareas asociadas al mismo, empleando los servicios de la capa *Tasks.* Estas recetas son ficheros que contienen las necesidades de recursos del *cluster* asociadas a los programas que se invocan en el trabajo. A continuación, la capa informa a la capa superior el estado general de ejecución del trabajo, a partir de los estados individuales de cada tarea obtenidos de la capa *Tasks.* Finalmente, devuelve a *Jobs* los resultados de la ejecución de las mismas, ya sean parciales o finales.

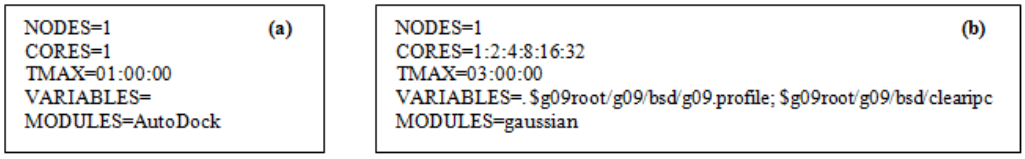


Fig. 3 - Ficheros de configuración de trabajos: a) Secuencial programa AutoDock b) Experimento programa gaussian.

La capa está compuesta por la clase *Generation* (Fig. 4) que cuenta con las estructuras y métodos necesarios para la conversión de un trabajo en sus tareas equivalentes. Por ejemplo, la estructura *Generation_User* almacena las credenciales del usuario y datos del *cluster*, mientras que la estructura *Generation_Resources*

mantiene información de los recursos del *cluster* requeridos por todas las tareas de un trabajo. La clase maneja una carpeta con las recetas separadas para cada tipo de trabajo (*experiment*, *parallel*, *secuencial*). El método *getRecipeFor* busca en la carpeta definida con la variable *scripts_path*, la receta correspondiente al programa establecido en el comando del trabajo. De no existir, utiliza la receta creada por defecto para una aplicación general. Luego, se insertan las tareas en el contenedor de la clase *TaskManager*. Cada tarea recibe un identificador formado de la siguiente manera: *idtrabajo_nodes_cores*.

Los métodos restantes de la clase *Generation* permiten obtener el estado y los resultados parciales o totales de un trabajo. Si el trabajo está compuesto por varias tareas, puede transitar por uno de los estados siguientes:

1. PendingSubmit: en proceso de envío, al menos una tarea en espera de ser enviada al *cluster*
2. Submitted: todas las tareas han sido enviadas al *cluster* y están en espera de ejecución
3. Running: al menos una tarea ejecutándose en el *cluster*, el resto distintas de estado *Failed*
4. Failed: todas las tareas fallidas
5. Completed: todas las tareas completadas
6. Result: todos los resultados de las tareas se encuentran en la computadora del usuario
7. *PartialFailed*: al menos una tarea fallida, el resto distinta de estado *Failed*
8. *PartialCompleted*: al menos una tarea completa, el resto ejecutándose
9. *PartialResult*: hay resultados de al menos una de las tareas en la computadora del usuario

Sin embargo, si al trabajo está asociada una sola tarea, los estados de ambos coinciden con los anteriores descritos, con excepción de los estados parciales.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

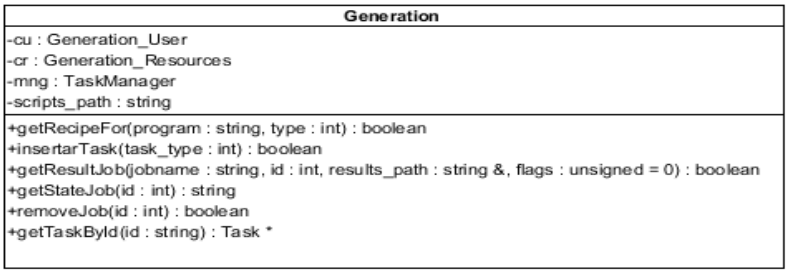


Fig. 4 - Diagrama UML de la clase Generation.

Es importante destacar, que conjuntamente con los resultados de un experimento, opcionalmente se devuelve un fichero con los valores del tiempo de ejecución, la eficiencia y la aceleración, para todas las subtareas finalizadas correctamente en el *cluster*. Si al solicitar los resultados ya se ha obtenido el tiempo de ejecución sobre un solo procesador, se devuelven además los valores correspondientes a la curva ideal de ejecución, y los valores de aceleración y eficiencia correspondientes. De lo contrario, las curvas de valores obtenidos se completan con valores estimados para aquellas tareas que no han terminado. En caso de que el *cluster* tenga instalado el programa *gnuplot*, este fichero es utilizado por la capa *Task* para generar las imágenes correspondientes a las curvas de los valores reales, ideales o estimados. Estas gráficas se devuelven en la carpeta de resultados del trabajo en la computadora del usuario. Esto facilita realizar estudios de escalabilidad de programas paralelos.

Capa Tasks

La capa *Tasks* almacena los datos de las tareas generadas por los trabajos y gestiona cada una de ellas de forma individual. Esta capa recibe solicitudes de compilación, inserción, recuperación y modificación de tareas desde la capa *Generation*, e interactúa con la GridMD para ejecutar dichas operaciones directamente en el *cluster*. Está formada por las clases *Tasks* y *Tasks_Manager*.

La clase *Task* (Fig. 5a) representa a una tarea asociada a un trabajo. Además de los atributos que almacenan la información originada en la capa *Jobs*, posee otros atributos que son generados por la propia capa o por la GridMD. Entre los generados por la GridMD se encuentran el identificador y estado asociados a la tarea real

en ejecución sobre el *cluster*. Generados por la propia capa están el identificador del trabajo al que pertenece la tarea, un nuevo identificador único de la tarea que es usado por las capas superiores, el estado general, el camino de resultados, y el error en caso de fallos. Los estados de las tareas son los mismos definidos en la clase *Job* de la GridMD, con la excepción del estado *FAILEDSUBMIT*, agregado para manejar posibles errores de envío de los trabajos. Si una tarea falla, es ejecutada nuevamente hasta que se complete de manera exitosa o se alcance el máximo de reintentos. Es por ello que son usados dos identificadores diferentes para una misma tarea. El primero identifica a la tarea actual en ejecución sobre el *cluster* mientras que el segundo mantiene una identidad única para las capas superiores.

La clase *Task* cuenta, además, con estructuras similares a las del nivel superior para almacenar las credenciales de los usuarios y los recursos de las tareas. En este caso, solo se representan los recursos de una sola tarea y no de varias como en la capa *Generation*.

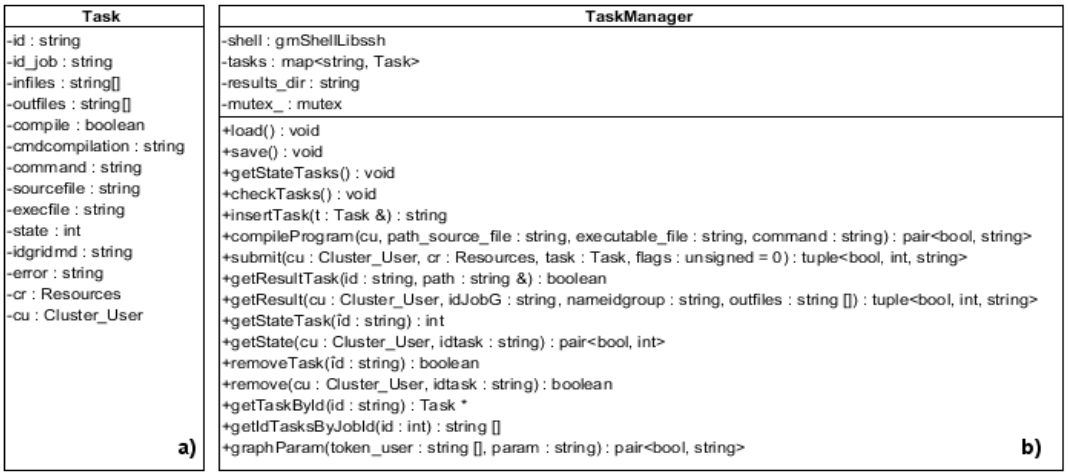


Fig. 5 - Diagramas UML de las clases: a) Task b) TaskManager.

La clase *TaskManager* (Fig. 5b) tiene un contenedor con los identificadores e instancias respectivas de la clase *Task*, y funciones para gestionarlo. Este se guarda en el fichero *tasksStore* mediante la serialización de la clase *Task*. Los métodos *save* y *load* son semejantes a los de la capa *Jobs*, pero referidos a las tareas.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

TaskManager usa las clases de la GridMD: *gmShellLibssh* para acceder al *cluster* y *gmSLURMManager* para gestionar las tareas.

Además, la clase cuenta con un hilo adicional de trabajo que chequea periódicamente el estado de las tareas, y en dependencia del mismo realiza una operación determinada que puede llevar a la actualización del estado de las mismas. Esto posibilita efectuar parte del trabajo con el *cluster* de forma desatendida, ya que el monitoreo de las tareas y la descarga de los resultados hacia la carpeta destino, que es definida con la variable *results_dir*, se realizan automáticamente, sin que sea solicitado por el programa usuario. El hilo principal y el de trabajo operan protegidos por un semáforo que evita accesos simultáneos a las estructuras de datos compartidas.

Capa Cluster_Info

Está compuesta por la clase *Cluster_Info* (Fig. 6) que obtiene información del *cluster* tal como el estado general del mismo, estado de un nodo de cálculo en específico, la partición por defecto, y de una partición determinada el total de núcleos, nodos y núcleos por nodos.

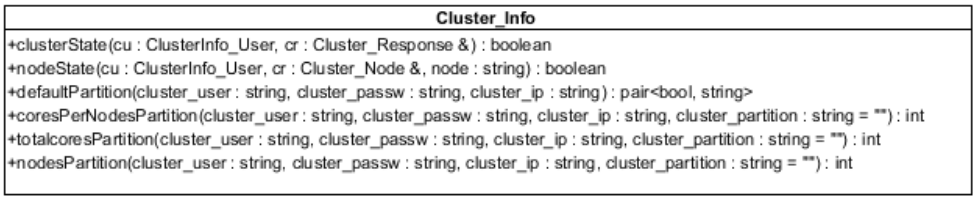


Fig. 6 - Diagrama UML de la clase Cluster_Info.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

Resultados y discusión

Para comprobar el funcionamiento de la biblioteca, se desarrolló un programa que representa el flujo de trabajo necesario para ejecutar dos aplicaciones de modelado molecular, sobre un *cluster* (Universidad de Oriente, 2019) con sistema operativo Linux y gestor de recursos *Slurm*. Para ello, se realizaron llamadas a las funciones principales de la capa *Jobs* (Fig. 7). La implementación del programa se realizó utilizando C++, cmake y gcc. El programa funciona de la siguiente manera: en primer lugar, se especifican las credenciales del usuario y los datos del *cluster* sobre el que se van a ejecutar las aplicaciones. Luego se crea una instancia de la clase *JobManager* con la estructura que almacena los datos anteriores.

A continuación, se crean dos instancias de la clase *Job* que representan una tarea secuencial para el programa *Autodock* y un experimento para el programa *Gaussian*. A partir de este punto la biblioteca se encarga de ejecutar ambos trabajos en el *cluster* y monitorear la ejecución de los mismos. Mientras tanto el usuario puede continuar insertando otros trabajos o pidiendo información del estado de un trabajo enviado con anterioridad. Si los trabajos finalizan de forma exitosa en el *cluster*, se obtiene la ruta de resultados en la computadora del usuario. En caso de ocurrir un error o no haber terminado alguno de los trabajos, se devuelve el estado fallido y una descripción del error.

Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

```
#include "jobmanager.h"
int main(){
    Job_User cu;
    cu.cluster_ip="ip_cluster";
    cu.cluster_user="user_cluster";
    cu.cluster_passw="passw_user";
    cu.local_workspace="/home/beatriz/Descargas";

    JobManager mn(cu);
    Job *job=mn.createJob();
    std::vector<std::string> infiles;
    infiles.push_back("/home/beatriz/Descargas/autodock");
    job->setJobname("autodock");
    job->setCommand("cd autodock; autodock4 -p dock.dpf -l dock.dlg");
    job->setOutfiles({"dock.dlg"});
    job->setInfiles(infiles);
    std::cout<<"Id de trabajo: "<<job->submit(Job::SECUENCIAL)<<std::endl;

    Job *job2=mn.createJob();
    job2->setJobname("gaussian");
    job2->setCommand("g09 < gaussian_memCompartida2.com");
    job2->setInfiles({"/home/beatriz/Descargas/gaussian/gaussian_memCompartida2.com"});
    std::cout<<"Id de trabajo: "<<job2->submit(Job::EXPERIMENT)<<std::endl;

    std::map<int, Job> jobs=mn.getJobs();
    std::string resultado;
    while(true){
        for (std::map<int, Job>::iterator it = jobs.begin(); it != jobs.end(); ++it){
            std::cout<<"Id de trabajo: "<<(*it).second.getId()<<std::endl;
            std::cout<<"Nombre de trabajo: "<<(*it).second.getJobname()<<std::endl;
            std::cout<<"Comando de trabajo: "<<(*it).second.getCommand()<<std::endl;
            std::string state=(*it).second.getState();
            if (state=="Result"){
                if ((*it).second.getResults(resultado)) std::cout<<"Resultado: "<<resultado<<std::endl;
            }
        }
    }
}
```

Fig. 7 - Ejemplo de flujo de trabajo con la biblioteca.

Para definir las credenciales de autenticación en el *cluster* se declara una estructura de tipo *Job_User* que debe especificar como mínimo el IP o nombre de dominio del *cluster*, nombre de usuario y contraseña, y la carpeta local de trabajo. Opcionalmente, se pueden indicar el puerto y la partición, o hacer uso de los configurados por defecto en el *cluster*. Esta estructura es utilizada para definir la instancia de la clase *JobManager*. Usando esta última se crean dos objetos de la clase *Job* que representan a los trabajos, mediante el método *createJob*. Posteriormente, se utilizan los métodos de modificación de la clase *Job* (*setJobname*, *setOutfiles*, *setCommand*, *setInfiles*) para especificar las características de los trabajos. En caso de no definir el fichero de salida, se asume como tal la salida estándar del programa correspondiente al trabajo.

Una vez definidos los datos de cada trabajo, estos son ejecutados llamando al método *submit* de la clase *Job*, pasando como parámetro el tipo de trabajo (Job::SECUENCIAL, Job::EXPERIMENT, Job::PARALLEL). Como resultado se obtiene el identificador de cada trabajo, un número entero que se incrementa por cada llamada al método *submit*, o -1 en caso de ocurrir algún error. El identificador es utilizado para obtener el

estado (Job::getState) o la ruta de los resultados (Job::getResults) de los trabajos. En caso de producirse algún error en la ejecución de los trabajos se devuelve el estado fallido (*Failed*) y una descripción del error.

Como resultado de la ejecución de los trabajos, se obtuvieron los resultados mostrados en la figura 8. En el caso del programa *AutoDock* se devuelve solamente una carpeta que contiene el fichero de salida *dock.dlg*. En cambio, para el programa *gaussian*, al ser un experimento, se obtuvo una carpeta por cada subtarea ejecutada en el *cluster* (Fig. 9), el fichero *Parametros.txt* con los tiempos de ejecución, aceleración y eficiencia, además de las gráficas asociadas en formato png (Fig. 10).

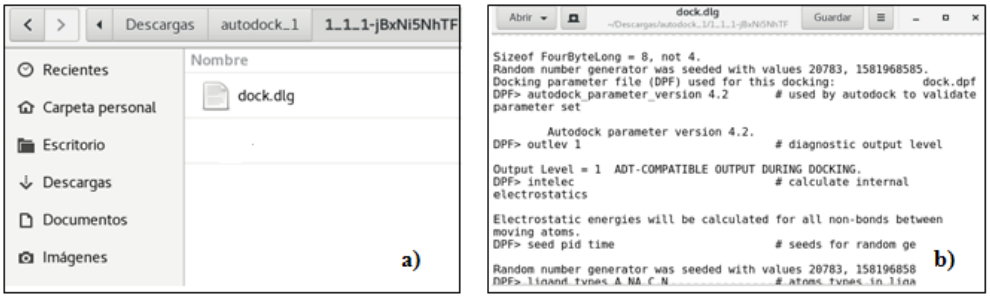
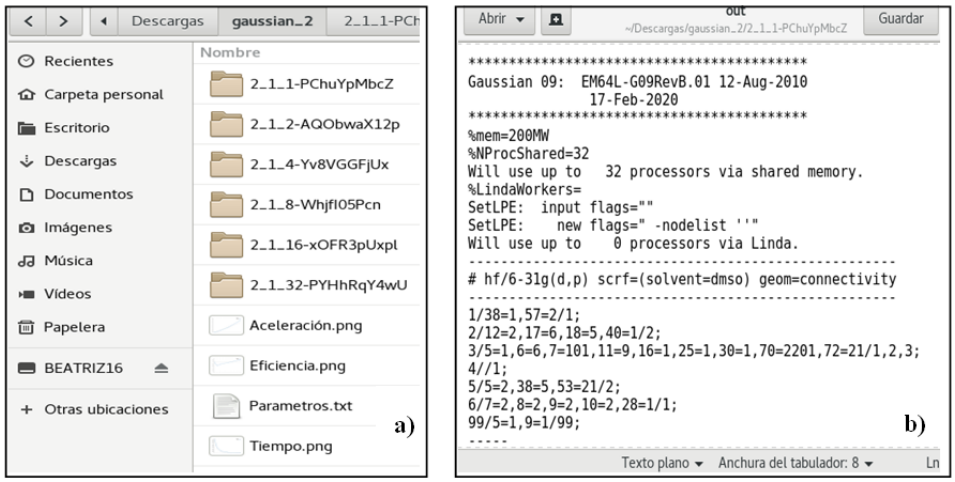


Fig. 8 - Resultado de trabajo secuencial correspondiente al programa AutoDock a) Carpeta con resultado b) Fragmento de fichero de salida dock.dlg.



Revista Cubana de Ciencias Informáticas
Vol. 14, No. 2, Abril-Junio, 2020
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 15-33

Fig. 9 - Resultado de trabajo de tipo experimento correspondiente al programa Gaussian a) Carpeta con resultado de cada subtarea y b) Fichero de salida de la tarea correspondiente a un procesador.

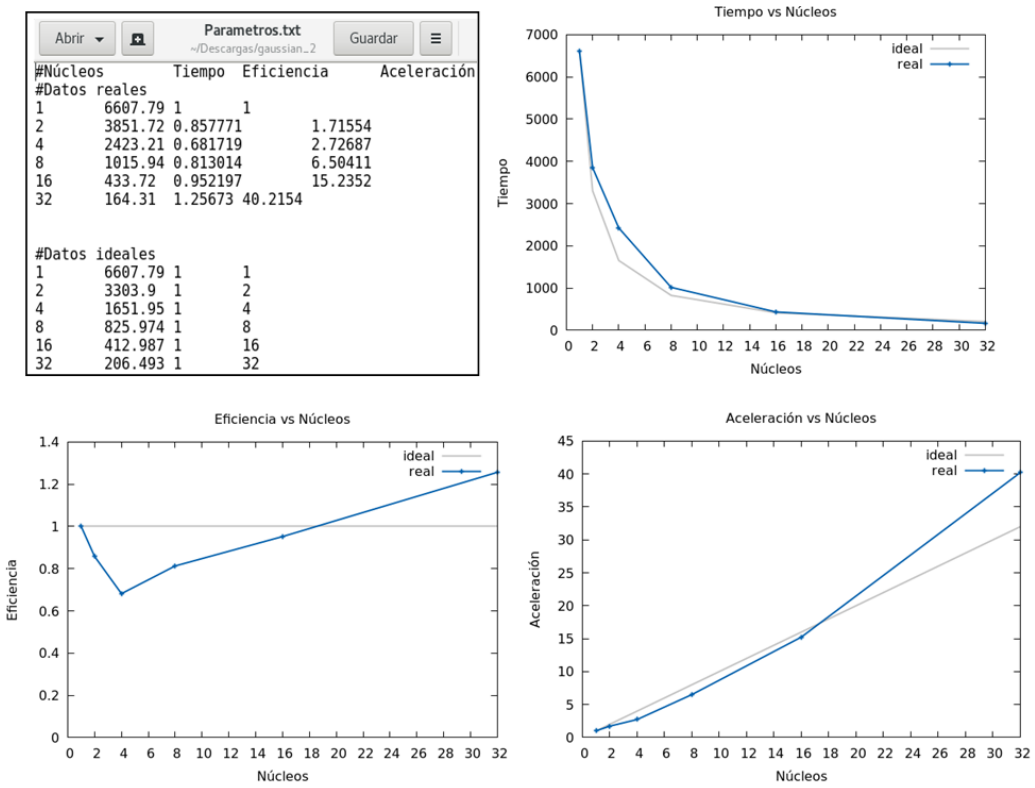


Fig. 10 - Fichero y gráficas con los tiempos de ejecución, aceleración y eficiencia de experimento con Gaussian.

Conclusiones

El principal resultado del trabajo ha sido la implementación de una biblioteca en C++ orientada a los programadores, que organiza el proceso de trabajo con un *cluster* en una jerarquía de capas. La biblioteca permite a los programadores integrar aplicaciones de escritorio que de manera transparente al usuario ejecuten programas sobre *clusters*, usando funciones de alto nivel de abstracción. Además, la biblioteca

viabiliza parte del trabajo con *clusters* de forma desatendida, agiliza la realización de estudios de escalabilidad de programas paralelos, y especifica los recursos del *cluster* y otros detalles necesarios para ejecutar los programas mediante recetas predefinidas. Se corroboró el funcionamiento de la biblioteca a partir de la ejecución de dos programas de modelado molecular en un *cluster* con Linux y Slurm.

Como desarrollo futuro se prevé la validación de la biblioteca en Windows, ya que todas las dependencias utilizadas son multiplataforma. Además, se permitirá el uso sobre *clusters* con otros gestores de recursos.

Referencias

ADAMANTIADIS, A.; et al. The SSH library**.** [En línea]. 2020, [Consultado el: 20/01/2020] Disponible en: https://www.libssh.org.

BOGDANOV, A.; et al. User interface for a computational cluster: resource description approach. En: CEUR Workshop Proceedings, 2017, p. 145-149.

CALEGARI, P.; LEVRIER, M.; BALCZYŃSKI, P. Web Portals for High-performance Computing: A Survey. ACM Trans. Web, 2019, 13 (1): p. 2-36.

CARNEVALE, T.; et al. The neuroscience gateway portal: High performance computing made easy. BMC Neuroscience, 2014, 15 (S1): p. 101.

CRUZ, H., Making Supercomputing Available to All Cuban Researchers. Computing in Science & Engineering, 2018, 20 (3): p. 25-30.

GOLEMON, S. libssh2. [En línea]. [Consultado el: 20/01/2020] Disponible en: https://www.libssh2.org.

POTTER, J. ParWeb: a front-end interface for cluster computing. Honors Theses, Western Michigan University, Michigan, 2014.

RAGNI, M. Force. [En línea]. 2013, [Consultado el: 22/01/2020] Disponible en: https://www.ragni.me/Force/.

SAMPEDRO, Z.; HAUSER, T.; SOOD, S. Sandstone HPC: A domain-general gateway for new HPC users. En: Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact. ACM, 2017, p. 33.

SARASTY, H. Documentación y análisis de los principales frameworks de arquitectura de software en aplicaciones empresariales. Tesis Doctoral, Universidad Nacional de La Plata, 2016.

SIDHU, R., et al. Machine Learning Based Datacenter Monitoring Framework. Master Theses, The University of Texas, 2016.

STERLING, T.; ANDERSON, M.; BRODOWICZ, M. Chapter 1 Introduction. En: High performance computing: Modern Systems and Practices. San Francisco, C.A.: Morgan Kaufmann Publishers Inc, 2017. p. 1-12.

UNIVERSIDAD DE ORIENTE. Guía de Usuario del HPC UO. [En línea]. 2019, [Consultado el: 22/01/2020] Disponible en: https://wiki.hpc.uo.edu.cu/doku.php?id=hpc-guia#referencia_en_publicaciones.

VALUEV, I. A.; MOROZOV, I. V. Managing Dynamical Distributed Applications with GridMD Library. En: International Conference on Computational Science and Its Applications. Springer, Cham, 2015, p. 272-289.

VAN, J. sshj. [En línea]. 2020, [Consultado el: 20/01/2020] Disponible en: https://github.com/hierynomus/sshj.

WU, X.; LONG, X. Implementation of a global GPU management plugin for Slurm. En: 2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT). IEEE, Ghaziabad, 2017, p. 1-5.

ZADKA, M. Paramiko. En: DevOps in Python. Berkeley, CA: Apress, 2019, p. 111-119.

ZAKARYA, M.; GILLAM, L. Energy efficient computing, clusters, grids and clouds: A taxonomy and survey. Sustainable Computing: Informatics and Systems, 2017, 14: p. 13-33.

Conflicto de interés

Los autores declaran que no existen conflictos de intereses.

Contribuciones de los autores

Dr. C. Fernando José Artigas Fuentes: Participó en la concepción y diseño de la investigación, así como la revisión y aprobación del artículo. Además, contribuyó en la redacción del artículo y en el análisis e interpretación de los resultados.

Ing. Beatriz Valdés Díaz: Participó el diseño e implementación del software, realización de los experimentos, análisis e interpretación de los resultados y redacción del artículo.