Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 8, No. 1, pp. 37-46, Junio 2021
Fecha recepción: 22/03/2021, Fecha aceptación: 15/08/2021

Despliegue de un Clúster HPC Utilizando Contenedores Docker

José Francisco Hernández[1], Gianfranco Alfredo Verrocchi[1], Yudith Cardinale[2,3], Ana Aguilera[4], Irvin Dongo[3,5]
kikohernandez92@gmail.com, gianfrancoverrocchi@gmail.com, ycardinale@usb.ve, ana.aguilera@uv.cl, ifdongo@ucsp.edu.pe

[1] Escuela de Informática, Universidad Católica Andrés Bello, Caracas, Venezuela
[2] Departamento de Computación, Universidad Simón Bolívar, Caracas, Venezuela
[3] Electrical and Electronics Engineering Department, Universidad Católica San Pablo, Arequipa, Peru
[4] Escuela de Ingeniería Informática, Universidad de Valparaíso, Valparaíso, Chile
[5] Univ. Bordeaux, ESTIA Institute of Technology, Bidart, France

Resumen: En ambientes de computación de alto rendimiento (HPC por sus siglas en inglés de *High Performance Computing*), una de las tareas más difíciles, tanto para los administradores de sistemas como para los usuarios finales, es preparar el ambiente de trabajo (e.g., configuración del clúster) y resolver las dependencias de software. En este contexto, este trabajo propone una solución de software, basada en la moderna tecnología de contenedores Docker, para construir un clúster HPC virtual con computadores personales de manera simple y automática, particularmente para trabajar bajo el paradigma de paso de mensajes con el estándar MPI (*Message Passing Interface*). La solución de software propuesta, se despliega de forma automatizada en virtualmente cualquier ambiente que soporte la instalación de la herramienta Docker, permite cubrir múltiples casos de uso y provee flexibilidad en cuanto a requerimientos de software, hardware y disponibilidad de equipos de cómputo. Para demostrar la flexibilidad y conveniencia de la solución propuesta, en este trabajo presentamos dos versiones de instalación: uno con el sistema operativo Alpine y librería de comunicación MPICH, y el segundo utiliza el sistema operativo Ubuntu y la librería de comunicación OpenMPI. Para evaluar la efectividad de la solución propuesta, se realizan pruebas con ambas versiones en dos ambientes de prueba: usando una red local (con tres computadores conectados con Ethernet) y usando cinco nodos virtualizados. Los resultados demuestran que la solución desarrollada puede ser desplegada eficientemente en distintos ambientes (real y virtualizado), lo que la hace ideal para fines académicos.

Palabras Clave: Cluster HPC; Contenedores; Docker; Instalación Automática; MPI.

Abstract: On High Performance Computing (HPC) environments, one of the hardest tasks for system administrators and developers, is to prepare their working environment (e.g., building and configuring a cluster) and resolve software dependencies. In this context, this work proposes a software solution based on the modern Docker container technology, to build a virtual HPC cluster in a simple and automated way using personal computers, particularly to work with the MPI (Message Passing Interface) standard. Our proposed software solution, can be deployed in an automated way in any environment that supports the Docker run time installation. It allows to cover multiple use cases and provides flexibility in terms of software requirements, hardware, and computer equipment availability. To demonstrate the flexibility and convenience of the proposed solution, we develop two versions of possible installation: one with Alpine operating system and MPICH and the second one with Ubuntu operating system and OpenMPI. To evaluate the effectiveness of the proposed solution, we test both versions in two different scenarios: one built with a local network (with three computers connect through Ethernet) and the other one with five nodes in virtual machines. Results demonstrate that the developed solution can be deployed efficiently across different environments (real and virtualized), which makes it ideal for academic purposes.

Keywords: HPC Cluster; Containers; Docker; Automatic Installation; MPI.

I. Introducción

La computación de alto rendimiento (HPC por sus siglas en inglés de *High Performance Computing*) implica el uso de plataformas multi-procesadores, particularmente clústers de computadores, y de software especializado como librerías de paso de mensajes (e.g., MPI, OpenMP) y manejadores de colas (e.g., Torque, Slurm). En ambientes HPC, por un lado, una de las tareas más difíciles es preparar el ambiente de trabajo (e.g.,

construcción y configuración del clúster) y resolver las dependencias de software. Generalmente encontramos conflictos en las versiones del software, así como con las características del sistema relacionadas con, por ejemplo, el sistema operativo, librerías, kernel; lo que conlleva a la instalación, en los recursos computacionales, de múltiples versiones de software y librerías especializadas. En consecuencia, se incrementa la complejidad de las tareas de los administradores del sistema y la inconformidad de los usuarios del ambiente de HPC, al tener que modificar y recompilar sus aplicaciones para que se ejecute en el ambiente HPC disponible.

Por otro lado, es común en un ambiente de clúster HPC que los usuarios y desarrolladores pretendan disponer, de manera exclusiva, de los recursos computacionales. Esto representa limitaciones, si las aplicaciones se ejecutan por mucho tiempo, privando al resto de los usuarios de su uso. Herramientas, como el planificador de tareas Slurm, evitan que procesos mal programados utilicen los recursos indefinidamente. Sin embargo, aún con estos mecanismos de control, es posible que estos procesos erráticos afecten el estado del clúster, impidiendo que futuros procesos funcionen con normalidad y generan la necesidad de auditar el sistema e incluso de causar interrupciones en el servicio.

La reciente tecnología de contenedores, y en particular, de contenedores Docker, se presenta como una solución potencial para simplificar el despliegue y administración del hardware y software requeridos para construir un clúster HPC. Docker es una plataforma amigable y muy popular, para ejecutar y administrar contenedores Linux. De hecho, actualmente, la vasta mayoría de herramientas provistas en contenedores, están empaquetadas en imágenes Docker. La naturaleza declarativa de la herramienta Docker y el aislamiento de procesos mediante el uso de contenedores `lxc`[1], permite entonces definir un ambiente de clúster que funcione de manera ininterrumpida, posea alta disponibilidad, y más importante aún, que permita que el estado del clúster sea replicable e independiente del ambiente en el que es desplegado. Esto permite que un ambiente de clúster pueda coexistir a la par de otras instancias del mismo con otras herramientas o configuraciones especializadas. Otro aspecto importante es que esta automatización abstrae las dificultades y necesidad de conocimiento técnico y costos de hardware requerido para implementar un ambiente de HPC.

Dados esos beneficios, recientemente existen varios trabajos que proponen el uso de contenedores Docker para el despliegue y configuración de clústers HPC [1]–[8]. Sin embargo, estas soluciones presentan limitaciones por ser de ámbito de aplicación específico y ofrecer una integración muy pobre de diversas tecnologías de HPC.

Con la intención de sobreponer tales limitaciones, en este trabajo de investigación proponemos una solución de software, basada en contenedores Docker, para construir un clúster HPC virtual con computadores personales. La solución propuesta ofrece varios ambientes de desarrollo y prueba de programas que se desenvuelven en el ámbito de la computación distribuida, computación paralela y computación de alto rendimiento, particularmente bajo el paradigma de pase de mensajes con el estándar MPI (*Message Passing Interface*). La solución propuesta permite desplegar un ambiente de clúster HPC a usuarios o desarrolladores que no cuenten con los recursos, tiempo o experiencia para conceptualizar, implementar y mantener un ambiente de clúster para el desarrollo de aplicativos que aprovechen un ambiente con estas características.

Nuestra solución extiende el trabajo propuesto en [5] y modifica e integra las soluciones parciales disponibles en [9], [10]; aprovechando la característica de auto-escalamiento de los servicios de descubrimiento y la herramienta de virtualización liviana de Docker. La solución propuesta se despliega de forma automatizada en virtualmente cualquier ambiente que soporte la instalación de la herramienta Docker. Esta solución está compuesta de varias imágenes, cuyo alcance cubre múltiples casos de uso y provee flexibilidad en cuanto a requerimientos de hardware, disponibilidad de equipos y necesidades del ambiente de desarrollo. En este trabajo presentamos el proceso detallado de construcción de los contenedores y las diferentes alternativas desarrolladas e integradas, así como las nuevas posibilidades que la solución resultante permite. Se especifican las consideraciones que se deben tomar al crear una imagen Docker y los pasos necesarios para desplegar un clúster HPC con nodos MPI, sobre un grupo de computadores personales. Nuestro objetivo es proveer la estrategia metodológica detrás de nuestra solución, de manera que otros interesados puedan adaptarla a diferentes requerimientos del sistema. Las variantes que definen los ambientes que ofrece esta solución forman parte de un mismo proyecto dentro de un repositorio de Docker Hub [11], en donde cada variante (imagen de Docker y código fuente) tiene un conjunto de herramientas completamente diferente una de la otra. Sin embargo, la arquitectura de despliegue se mantiene: un nodo maestro y N nodos esclavos.

Para evaluar el desempeño de la solución propuesta, realizamos una serie de pruebas de *benchmarking* comparando las versiones desarrolladas en diferentes ambientes: usando una red local (con tres computadores conectados con Ethernet) y usando cinco nodos virtualizados. Los resultados demuestran que la solución desarrollada puede ser desplegada eficientemente en distintos ambientes (real y virtualizado), lo que la hace ideal para fines académicos.

II. Computación de Alto Rendimiento y Contenedores: Preliminares

El término *High Performance Computing* (HPC), deriva de la utilización de equipos de computación y de procesamiento de datos en paralelo para resolver problemas computacionales complejos. Hoy día, se utiliza para resolver problemas avanzados, investigación científica, realizar predicciones del clima a través de modelado, simulación y análisis de los datos. Los clústers HPC son comúnmente denominados *Supercomputadores*, dada la enorme capacidad de procesamiento que poseen.

[1] LXC es una interfaz de espacio de usuario para las características de contenerización del kernel de Linux. A través de una poderosa API y herramientas simples, permite a los usuarios de Linux crear y gestionar fácilmente contenedores de sistemas o de aplicaciones.

En el mundo de HPC, uno de los paradigmas más usados para la implementación de las aplicaciones distribuidas y paralelas es el de pase de mensajes. En este contexto, el estándar MPI se destaca como una de las tecnologías más comúnmente usadas para software científico, generalmente escrito en C/C++ y Fortran. MPI está diseñado para soportar el modelo de programación paralelo SPMD (*Single Process Multiple Data*), bajo el paradigma maestro-esclavo: un único programa se ejecuta en todos los nodos, cada uno procesando diferentes datos (nodos esclavos), bajo la coordinación de un coordinador (nodo maestro).

Sin embargo, desplegar un clúster MPI para HPC puede resultar una tarea retadora, demandante de tiempo y dedicación, por parte de los administradores de sistemas, para satisfacer las necesidades de los usuarios, muchas veces requiriendo distintas versiones de las diversas librerías existentes. De aquí, la necesidad de proveer soluciones capaces de abstraer la dificultad de tales tareas de administración, de manera automática, simple y eficiente. La reciente tecnología de contenedores, y en particular de Docker, se presenta como una alternativa atractiva. Docker permite crear contenedores Linux para encapsular aplicaciones y sus respectivas dependencias para poder ejecutarlas, aliviando al usuario final de dichas tareas, y en este caso a los administradores de sistemas de HPC. Docker además permite "orquestar" (i.e., controlar la ejecución de un conjunto de contenedores con un fin específico) aplicaciones seccionadas en microservicios a través de su herramienta `Swarm`, que crea al momento de su ejecución un clúster de alta disponibilidad, para mantener en línea la aplicación en todo momento. `Swarm` también permite la adición y sustracción de nodos, sin perder su estado, siempre y cuando exista un quorum de $(n/2) + 1$ nodos[2] (siendo "n" la cantidad de computadores pertenecientes al `Swarm`) que estén declarados como *Managers* del clúster de alta disponibilidad.

III. Trabajos Relacionados

Actualmente, existen trabajos que exploran las posibilidades que introduce la herramienta Docker en el campo de la computación de alto rendimiento. En [4] los autores presentan un estudio comparativo entre el uso de máquinas virtuales y contenedores Docker para desplegar aplicaciones HPC, usando *benchmarks* bien conocidos. Sus resultados confirman las ventajas y beneficios que representan ambas técnicas en el mundo de HPC. Otros trabajos se concentran en mostrar cómo se puede integrar ambientes de MPI en clúster HPC a través de contenedores Docker [7].

Otros estudios más relacionados a este trabajo proponen soluciones para desplegar clúster de HPC con contenedores Docker. El trabajo presentado en [1] describe una solución con contenedores Docker para encapsular el ambiente de los nodos de un clúster, que al igual que nuestra solución, ofrece auto escalamiento de nodos y permite que varias instancias del clúster pueden coexistir. Sin embargo, difiere de nuestra solución en los siguientes aspectos: (i) fue ejecutado y concebido para hardware de clúster, en lugar de computadres personales; (ii) utiliza la red bridge; (iii) utiliza cónsul, un servicio distribuido de descubrimiento de servicios, para mantener la lista de nodos activos/conocidos en el archivo de host de mpi; el servicio está presente fuera de la solución, es decir, no está dentro de los contenedores; (iv) no posee soporte de almacenamiento centralizado; y (v) no utiliza un orquestador de contenedores, ni servicios, no está integrado ni automatizado el proceso de despliegue del clúster; sólo está automatizado el proceso de construcción del ambiente.

En [2] se realiza un análisis investigativo y exhaustivo sobre las implicaciones y decisiones técnicas, que conlleva desplegar un ambiente de HPC basado en contenedores de Docker, sin embargo plantean la implementación del concepto en un trabajo posterior [12]. Similar a nuestra propuesta, este trabajo, utiliza los contenedores de Docker para encapsular el ambiente de los nodos y un *script* de inicialización de los nodos. Las principales diferencias son: (i) la solución fue ejecutada y concebida para hardware de clúster; (ii) presenta un análisis más profundo sobre el impacto de Docker sobre el rendimiento en comparación con el hypervisor `kvm` y concluye que el *overhead* entre un ambiente con contenedores y uno real, es mínimo; (iii) soporta dos arquitecturas para el manejo de los procesos de MPI, una con un contenedor por nodo y otra con un contenedor por proceso (similar a nuestra solución propuesta); (iv) usan el *benchmark* de la librería LINPACK [13]; (v) no utiliza un orquestador de contenedores, ni servicios, no está integrado ni automatizado el lanzamiento, sólo la construcción del ambiente; y (vi) no posee soporte de almacenamiento centralizado.

Los autores del trabajo presentado en [3], a diferencia del presentado en [2], determinaron en sus estudios (y en base a otros estudios previos referenciados en su trabajo) que sí existe un cuello de botella al ejecutar trabajos de MPI en ambientes de múltiples contenedores. Por lo que decidieron estudiar y diagnosticar las causas de este *overhead*, para luego desarrollar una librería que permita a los procesos de MPI comunicarse a través de memoria compartida, en lugar de por mensajes de red. Ese proyecto, al igual que los anteriores mencionados y el nuestro, utiliza contenedores Docker para encapsular el ambiente de los nodos. Las principales diferencias con nuestra solución son: (i) fue ejecutado y concebido para hardware de clúster, especialmente para ambientes en la nube; (ii) usa *Runtime Privilege*[3], lo que permite darle accesos al contenedor a todos los dispositivos y procesos del host; esto podría utilizarse en la solución propuesta en este trabajo, para darle acceso al contenedor (un nodo MPI como el maestro) al demonio de docker (encargado de desplegar la solución); un beneficio de ésto, sería que le otorgaría al maestro la capacidad de desplegar, apagar o destruir nodos del clúster sin tener que salir de la sesión o de la instancia de clúster; en el caso del trabajo presentado en [3], se utiliza esta característica para darle acceso a los contenedores a los dispositivos infiniband, esto para obtener las capacidades de

[2]Documentación sobre Algoritmo de consenso `Raft`: https://docs.docker.com/engine/swarm/raft

[3]Documentación sobre Runtime Privilege: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities

acceso a memoria compartida mencionadas anteriormente.

Un trabajo reciente en el contexto de procesamiento de imágenes de teledetección [14] utiliza la tecnología Docker para implementar un entorno HPC, necesitando solamente 5 minutos en la configuración de 16 servidores (*clusters*) en comparación de un día de trabajo en una configuración manual. La arquitectura consiste en un administrador de flujo de trabajo, un administrador de recursos y un calendarizador. En [15], los autores proponen Sarus, una herramienta para sistemas HPC que crea una instancia de contenedores con muchas funciones a partir de imágenes de Docker. Administra completamente el ciclo de vida del contenedor al extraer imágenes de los registros, administrar el almacenamiento de imágenes, crear el sistema de archivos raíz del contenedor y configurar *hooks* y espacios de nombres.

El proyecto presentado en [5] trata sobre una imagen basada en alpine con la implementación de MPICH, llamada `alpine-mpich`. El estudio de la arquitectura de esta solución permitió desarrollar más a fondo la imagen generada en el presente trabajo, llamada `ubuntu-openmpi`. En nuestra propuesta se utilizan algunas aspectos de [5] tales como el auto escalamiento de nodos, el uso de un *script* de inicialización de los nodos, la posibilidad de la coexistencia de instancias del clúster, el uso de contenedores Docker para encapsular el ambiente de los nodos, uso de un orquestador de contenedores, servicios, y la integración y automatización del lanzamiento y la construcción del ambiente. Así mismo, difiere en los siguientes aspectos: (i) no posee soporte de almacenamiento centralizado, de manera de que cuando se requiera ejecutar nuevos *scripts* se debe volver a construir la imagen de la solución; (ii) no posee soporte para ejecución de *scripts* en python; (iii) posee solamente un ambiente de desarrollo de aplicaciones distribuidas (alpine linux con la librería MPICH). Mientras que la solución aquí propuesta, además de ofrecer ese mismo ambiente, ofrece una versión basada en Ubuntu con la librería OpenMPI.

Se puede observar que la mayoría de los trabajos considerados se enfocan en la optimización, puesta en producción y el rendimiento del clúster. Esto es deseable, sin embargo, todos excepto el trabajo presentado en [5], requieren de hardware especializado para su utilización. De allí el valor de nuestra solución, ya que es más accesible y sencilla de utilizar, en equipos de cómputo de uso cotidiano.

IV. Arquitectura de la Solución Propuesta

La utilización de la herramienta Docker permite diseñar un clúster con una arquitectura distinta a la de un clúster tradicional, como puede observarse en la Figura 1. Además, ofrece una manera más ordenada de declarar los componentes de la arquitectura. El *script* de lanzamiento utiliza el archivo de despliegue para desplegar cualquier versión del clúster, en nuestro caso `alpine-mpich` o `ubuntu-openmpi`, como lo muestra la Figura 2. A continuación se describe brevemente los principales elementos de la arquitectura de un clúster HPC, desplegado en contenedores Docker.

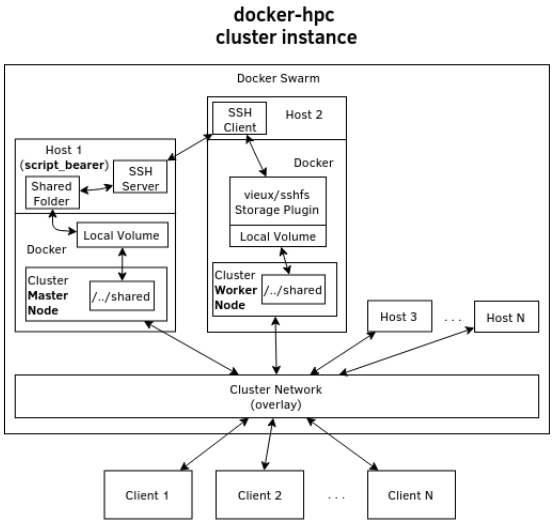

Figure 1: Despliegue de una Instancia del Clúster a lo Largo del Swarm

- **Contenedores, servicios y `stacks`:** Cada nodo es encapsulado en un contenedor que contiene todas las herramientas necesarias para su funcionamiento, dándole autonomía y aislamiento. Estos contenedores pueden pertenecer a dos servicios, uno que despliega el nodo maestro y otro que despliega un nodo esclavo junto con sus réplicas. A su vez, estos servicios forman parte de un `Stack`. Un `stack` es un grupo de servicios interrelacionados que pueden compartir dependencias o pueden ser orquestados para ser escalados en conjunto[4]. Ambos servicios pueden ser desplegados automáticamente utilizando un archivo de configuración `yaml` en donde éstos son definidos. Esto permite además que los servicios de los esclavos puedan ser replicados a voluntad, agregando o disminuyendo recursos al clúster.

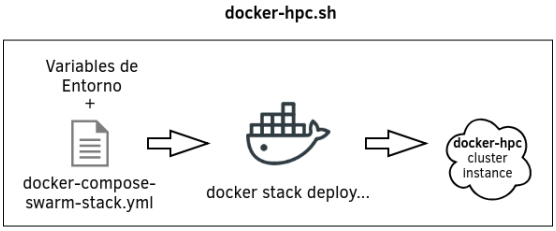

Figure 2: Estructura del *Script de* Lanzamiento del Clúster

- **Comunicación:** La comunicación entre estos nodos se realiza a través de una red virtual llamada `red overlay`, nativa de la herramienta Docker, que segmenta la red de manera lógica. Sin embargo, ésta no necesariamente tiene que ser local.
- **Almacenamiento:** Es deseable que en un ambiente de clúster exista un almacenamiento común para todos los nodos, de manera que los archivos fuente a ser ejecutados de manera paralela, estén disponibles para cada uno de ellos. Dado que los contenedores se ejecutan en diferentes computadores en un Swarm, la capacidad de Docker de

[4]Documentación sobre Stack en Docker: https://docs.docker.com/get-started/part5

montar `volúmenes` (directorios locales del host que son montados en una ruta dentro del contenedor) no podría usarse como almacenamiento común. Cada nodo tendría una versión local del punto de montaje diferente.

La centralización del almacenamiento en un Swarm no es una característica o funcionalidad nativa de Docker. El desarrollo de dicha característica se encuentra fuera del alcance de este trabajo. Existen herramientas que implementan este requerimiento y pueden instalarse en forma de *plugins* de Docker, y dan soporte especializado a diferentes plataformas como Amazon Web Services u otros programas de orquestación de contenedores como Kubernetes[5]. Estos *plugins*, en la mayoría de los casos, requieren una instalación y configuración que no es amigable para usuarios inexpertos, o requiere de pasos extra que aumentan la complejidad de la instalación. Para que esta solución sea verdaderamente simple de utilizar y portable, se eligió un complemento muy simple de instalar y configurar: `docker-volume-sshfs` [16]. Se encarga de montar un directorio en un servidor remoto a través de la herramienta `sshfs`, y lo hace disponible como un volumen local para el uso de los contenedores (nodos trabajadores).

Otro beneficio de utilizar este *plugin* es que es compatible con el ambiente `Playwithdocker`, un *sandbox* temporal para la ejecución de proyectos que utilizan la herramienta Docker y su orquestador Swarm. Permitiendo desplegar la solución de este proyecto en un par de minutos. Es importante acotar que cuando el Swarm posee un solo nodo, el *script* de lanzamiento le indica a los nodos del clúster que utilicen un volumen local, ya que no tiene sentido montar de manera remota un directorio local.

A continuación se describen las diferentes versiones disponibles en la solución desarrollada.

V. Imágenes Docker Desarrolladas: Variantes de Clústers HPC

En este trabajo desarrollamos dos variantes de clústers HPC, en cuanto a sistema operativo base y librería de paso de mensajes MPI implementada: `alpine-mpich` y `ubuntu-openmpi`. Es importante destacar que en un proyecto de Docker pueden existir diferentes etiquetas o `tags` que permiten diferenciar las versiones entre sí, de acuerdo a criterios definidos por los desarrolladores como: tipos de herramientas presentes en la imagen, configuración, tamaño, entre otros. Para diferenciar las imágenes, se definió una nomenclatura basada en los componentes y herramientas más importantes en cada una. La nomenclatura es la siguiente:

xxx-yyy[-ppp][-zzz], donde:

- **xxx**: describe el sistema operativo base.
- **yyy**: indica la principal herramienta o ambiente de trabajo que se ejecutará al desplegar el clúster.
- **ppp**, **zzz**: representa la versión o acabado de la imagen (por ejemplo: *latest, base, onbuild*).

[5]Documentación sobre plugins en Docker: https://docs.docker.com/engine/extend/legacy_plugins/#volume-plugins

Los parámetros encerrados entre corchetes son opcionales. Por ejemplo, la imagen con el sistema operativo Ubuntu, con las herramientas `OpenMPI` y `Slurm` instaladas, se representaría como **ubuntu-openmpi-slurm-latest**, según esta nomenclatura. A continuación describimos las variantes que forman parte del los de ambientes disponibles en nuestra solución.

A. Versión alpine-mpich

Este ambiente despliega un clúster con el sistema operativo Alpine Linux[6], una distribución de Linux muy popular entre los desarrolladores de imágenes para Docker [17], [18] y los que necesitan una utilización óptima de recursos con un mínimo impacto en la seguridad y espacio en disco [19].

Sobre esta distribución se realiza la instalación de la implementación del estándar MPI, MPICH en la versión 3.2. La imagen correspondiente a esta versión, se basa en la propuesta en el trabajo presentado en [5] y disponible en [20]. A la versión disponible en [20], se le hicieron adaptaciones y extensiones para agregar funcionalidades y solucionar algunos problemas presentados en el despliegue (e.g., el clúster no queda en el estado deseado, ya que el nodo maestro del clúster es incapaz de comunicarse con los nodos esclavos). Las modificaciones realizadas se detallan a continuación.

- Solución de problemas de despliegue. Aunque la imagen disponible en [20] estaba correctamente armada, al desplegar el proyecto de acuerdo a la documentación, el nodo maestro del clúster no era capaz de comunicarse con los esclavos. Este problema fue ocasionado porque el *script* que mantiene actualizado el archivo de hosts, utilizaba la herramienta `nslookup` para mapear los esclavos existentes en la red (*overlay network*). De acuerdo a la documentación de Docker[7], para poder determinar cuáles son los hosts activos dentro de una *overlay network*, se tiene que consultar al dominio `tasks.`⟨`nombre del servicio`⟩. Sin embargo, ésto solo funcionó para un máximo de 10 hosts. Al escalar el servicio por encima de este número, el resto de los nodos no podía ser encontrado. Según algunas publicaciones, `nslookup` es una herramienta considerada antigua y en algunos casos obsoleta[8,9]. En esos trabajos se recomienda que se suspenda su uso y en su lugar se utilice la herramienta `dig`, que no posee la limitación antes mencionada. Luego de esta modificación la instancia del clúster fue capaz de reconocer todos los nodos de la red.
- Soporte de ejecución de *scripts* de python. Se instalaron las versiones 2.7 y 3 del *runtime* de Python, junto con su manejador de paquetes `pip` [21]. Finalmente, utilizando `pip`, se instaló el módulo para ejecución de tareas en paralelo `mpi4py` [22].

[6]https://alpinelinux.org

[7]https://docs.docker.com/network/overlay/#container-discovery

[8]https://www.linuxquestions.org/questions/linux-networking-3/why-nslookup-is-deprecated-122337

[9]http://blog.smalleycreative.com/linux/nslookup-is-dead-long-live-dig-and-host

B. Versión ubuntu-openmpi

A diferencia del ambiente anterior, en éste despliega un clúster con Ubuntu, un sistema operativo más comercial que Alpine Linux. En esta versión se encuentra instalada la implementación OpenMPI. La imagen utilizada de base para esta versión, forma parte del proyecto `Dispel4Py` disponible en [10]. Las adaptaciones y modificaciones realizadas a ese proyecto se describen a continuación.

- Reducción del tamaño en disco de la imagen original. Se eliminaron diversos paquetes que no son necesarios para la implementación correspondiente en este trabajo, permitiendo la creación de una imagen más ligera y en consecuencia, más fácil de descargar en los nodos a utilizar.
- Corrección en la exclusión e inclusión de interfaces de red para OpenMPI. La configuración de OpenMPI en los nodos carecía de parámetros de configuración de la capa de envío de datos de mensajes MPI (BTL por las siglas en inglés de *Byte Transfer Layer*) para interfaces TCP, causando *hangups* (interrupción de envío de mensajes) en MPI, al no "saber" qué interfaz utilizar para poder enviar y recibir los mensajes en paralelo. También se realizó la configuración en todos los nodos, de la interfaz a utilizar para los mensajes de control *Out-of-band* (OOB).
- Limitación de los procesos multi-hilo: Se estableció la configuración necesaria para que OpenMPI no trabaje en modo multi-hilo y permita dedicación exclusiva de sus nodos.
- Igual a lo corregido en la imagen de Alpine Linux, los nodos no eran capaces de interconectarse por una falla en la resolución de nombres y hosts. La falla se corrigió de la misma manera que para la imagen de `alpine-mpich`.
- Luego de estudiar el funcionamiento de la imagen de `alpine-mpich`, se desarrollaron una serie de *scripts* que le permiten a la instancia de esta imagen, mantener actualizado el archivo de *hosts* a medida que los nodos se incorporan o son eliminados del clúster.

Luego de haber culminado con el desarrollo de las imágenes, para realizar pruebas y comprobar el correcto funcionamiento del clúster desplegado, se utilizaron dos programas sencillos de prueba: `mpi-hello-world.py` y `ring.py`. Estos programas se ejecutaron para probar la conectividad de los nodos esclavos con el maestro (`mpi-hello-world.py`) y para probar el correcto funcionamiento del envío y recepción de mensajes (en `ring.py` los nodos se comunican siguiendo una topología lógica de anillo). Dichos programas resultaron en ejecuciones exitosas, lo que permitió proseguir al despliegue de la solución y ejecución de pruebas sintéticas para evaluar el rendimiento, cuyos resultados se encuentran en la Sección VII.

VI. Despliegue de la Solución

En esta sección se describen los pasos a seguir para que un usuario pueda instalar, desplegar y utilizar un clúster en sus equipos personales. Antes de realizar el despliegue de alguna de las versiones disponibles en nuestra solución, es necesario instalar una versión de Docker superior a 1.13 (preferiblemente 18.06 en adelante). Si se quiere aprovechar el poder de cómputo de varios hosts, el programa debe estar presente en todos ellos, para aprovechar el orquestador de contenedores Docker Swarm[10]. Adicionalmente se requerirá que al menos uno de los nodos cuente con un servidor `ssh` activo. Finalmente se deben seguir los pasos que se muestran a continuación para asegurarse de que el *plugin* de almacenamiento está instalado y funciona correctamente.

A. Preparación del Ambiente

Para explicar el proceso de despliegue se usarán dos máquinas: `bob` y `alice`. Ambas corriendo el sistema operativo Linux. `bob` albergará un contenedor con el nodo maestro del clúster (y es el que debe tener un servidor `ssh` activo de manera nativa) y `alice` hospedará los contenedores ejecutándose como nodos esclavos (debe poder tener acceso como cliente al servidor `ssh`, esto es necesario para que el *plugin* funcione correctamente). En la Figura 3 se ejemplifica este proceso. `bob` posee un archivo `cureofcancer.c`, al cual `alice` quiere tener acceso para utilizarlo dentro de sus contenedores, además desea que los cambios sean persistentes. Utilizando el *plugin* de almacenamiento podrá hacer disponible ese directorio remoto a otros contenedores locales, debido a que este crea un volumen local de `docker` que mantiene los cambios en la computadora de `bob` luego de que los contenedores de `alice` hayan sido destruidos.

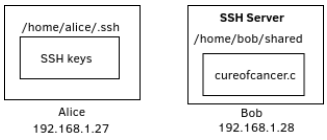

Figure 3: Elementos Presentes en los *Hosts* para Instalación del *Plugin*

En la máquina cliente (`alice`) se deberán ejecutar los siguientes comandos:

1) `ssh-keygen`
2) `ssh-copy-id bob@192.168.1.28`
3) `docker plugin install --grant-all-permissions vieux/sshfs sshkey.source=/home/alice/.ssh`
4) `docker volume create -d vieux/sshfs -o sshcmd=bob@192.168.1.28:/home/bob/shared sshvolume`
5) `docker run -it -v sshvolume:/brilliant busybox ls /brilliant`

La secuencia de comandos anterior, en el caso de las máquinas `bob` y `alice`, deberá generar la siguiente salida (obviando la salida de los comandos de instalación):

```
alice@client /home/alice $

cureofcancer.c
```

[10]https://docs.docker.com/get-started/part4/#prerequisites

B. Pasos para Levantar el Clúster HPC

Continuamos utilizando como referencia los computadores de `bob` y `alice`.

1) Iniciar el Swarm en el host que posee el servidor `ssh` activo (`bob`) con la orden `docker swarm init`; si éste posee más de una interfaz de red deberá especificar cuál desea utilizar mediante el parámetro `--advertise-addr string`, en donde `string` deberá ser la dirección `ip` o el nombre de la interfaz. Si se desea utilizar los recursos de varios equipos, cada uno deberá unirse al Swarm previamente creado utilizando el token generado por `bob` en este paso (ver Figura 4).
2) Clonar el código del repositorio que contiene las versiones de la solución, disponible en [11] en el nodo que posee el servidor `ssh` activo (`bob`). El *script* de lanzamiento permite elegir y lanzar posteriormente alguna de esas versiones (ver Figura 5).

Figure 4: Agregando Nodos al Swarm

3) Como se detalla en la documentación del repositorio de este proyecto [23], el *script* `docker-hpc.sh` sin parámetros, desplegará un clúster con el ambiente Alpine Linux y la implementación del estándar MPI, MPICH, provisto de un maestro y cuatro nodos esclavos (Figura 5). En caso de que se quiera modificar las imágenes que componen a los ambientes de la solución, será necesario modificar el usuario y repositorio en el archivo de configuración `docker-hpc.conf` y proveer los parámetros para construir y subir la imagen al nuevo repositorio de DockerHub.
   Al finalizar la descarga o construcción y publicación de la imagen, el *script* esperará a que el nodo maestro esté disponible (Figura 6) y se autenticará automáticamente utilizando las llaves SSH presentes en el directorio de cada variante (Figura 7). Si se desea lanzar el clúster con la versión de `ubuntu-openmpi`, se debe especificar la bandera `-i` seguida del nombre de la versión a utilizar, en ese caso `ubuntu-openmpi` (ver Figura 8).
4) Finalmente, si se desea eliminar la instancia del clúster antes creada, se pasa el parámetro `-d` al *script* de lanzamiento, seguido del nombre de la imagen utilizada para crear el ambiente. Si no se provee ningún parámetro, por defecto se utiliza `alpine-mpich`.
   Si se desea aumentar o reducir el tamaño del clúster, tan soló es necesario proveer la bandera `-s` seguida del número de trabajadores deseado como lo muestra la Figura 9. Esto también se puede conseguir en el paso 3 utilizando la bandera `-n` junto con un número de réplicas.

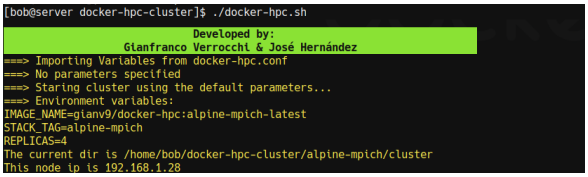

Figure 5: Ejecutando el *Script* de Lanzamiento del Clúster

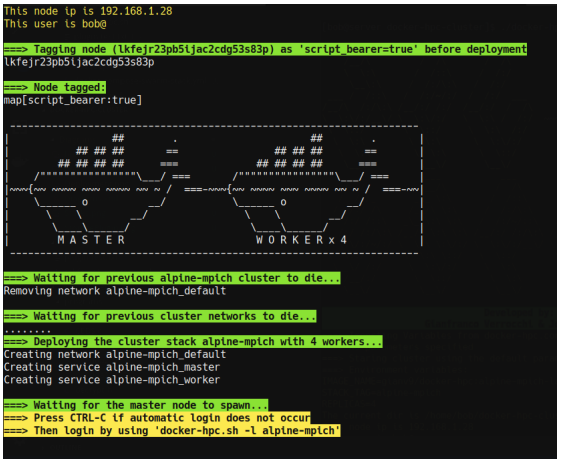

Figure 6: Estado del Despliegue del Clúster

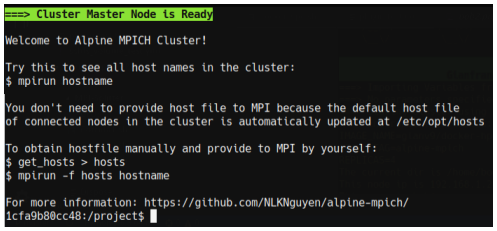

Figure 7: Instancia de Clúster `alpine-mpich` Desplegada Exitosamente

VII. Resultados Experimentales

Para la evaluación de nuestra propuesta, se desplegó una instancia de cada versión de la solución, en dos ambientes: un ambiente con una red local (tres nodos heterogéneos pertenecientes a un Swarm clúster interconectados vía cables Ethernet) y un ambiente remoto con máquinas virtualizadas utilizando el ambiente *Play With Docker*[11] (cinco nodos homogéneos).

Los tres computadores usados para el ambiente con la red local son `Link`, `Falco` y `Wolf`. Los tres son *Managers* del Swarm debido a que es requerido por Docker para su correcto funcionamiento. En el ambiente remoto con máquinas virtuales, se usó una plantilla pre-existente de tres *Managers* y dos trabajadores para el `Swarm`.

En la Figura 10 se muestra las especificaciones de los equipos utilizados para desplegar la instancia local del clúster (primera fila y segunda fila con primera columna), junto con las especificaciones de las instancias de *Play With Docker* (segunda fila con segunda columna).

En cada versión de la solución se utilizó la herramienta de pruebas sintéticas OSU Microbenchmarks, *benchmark* de MPI que mide el desempeño de varias operaciones [24]:

[11] https://labs.play-with-docker.com – *Play With Docker* es una plataforma en línea, que ofrece de forma gratuita contenedores trabajando como instancias de máquinas virtuales.

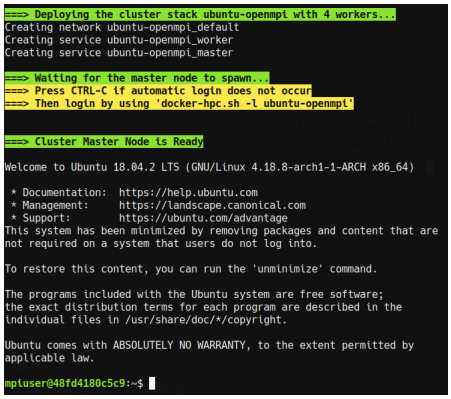

Figure 8: Instancia del Clúster `ubuntu-openmpi` Desplegada Exitosamente

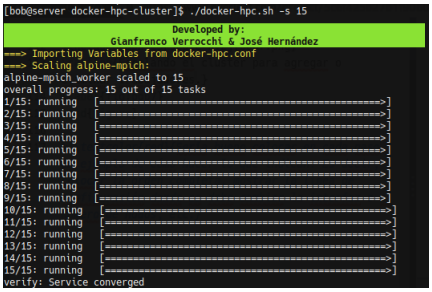

Figure 9: Escalando el Clúster para Agregar o Eliminar Trabajadores

- *collective*: Pruebas sintéticas de funciones colectivas de MPI.
- *one-sided*: Pruebas sintéticas de funciones unilaterales y bidireccionales de MPI.
- *point-to-point*: Pruebas sintéticas de funciones punto a punto de MPI.
- *startup*: Pruebas de arranque y chequeo de librerías.

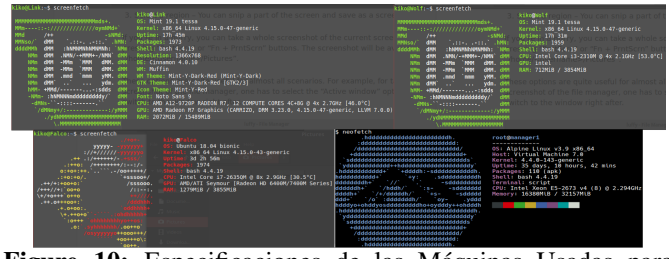

Figure 10: Especificaciones de las Máquinas Usadas para el Ambiente de Red Local: `Link`, `Falco` y `Wolf`, y el Ambiente Remoto: `Play With Docker`

Para el lanzamiento, en ambos casos el clúster que fue desplegado usando el *script* de la Figura 2, y los pasos descritos en la Sección VI. El clúster desplegado está compuesto de cinco contenedores: un coordinador y cuatro esclavos de MPI.

Las Figuras 11 a 21 muestran de manera gráfica los resultados de la ejecución de las pruebas más significativas (aquellas que prueban funciones referenciales de MPI como *scatter* y *broadcast* o que presentan diferencias significativas en cuanto a latencia entre las ejecuciones en los dos diferentes ambientes), usando la versión `alpine-mpich`. No se muestran los resultados de la versión `ubuntu-openmpi` dado que son muy similares. Los tiempos y el ancho de banda están expresados en base logarítmica para apreciar mejor los resultados.

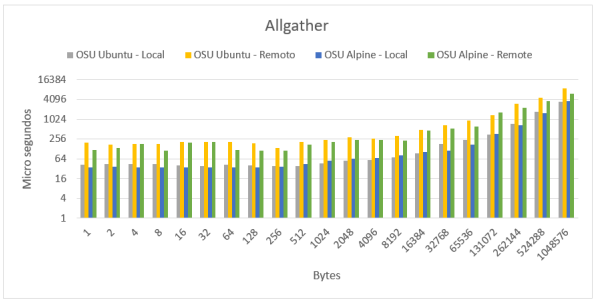

Figure 11: Prueba *allgather* con OSU Microbenchmarks

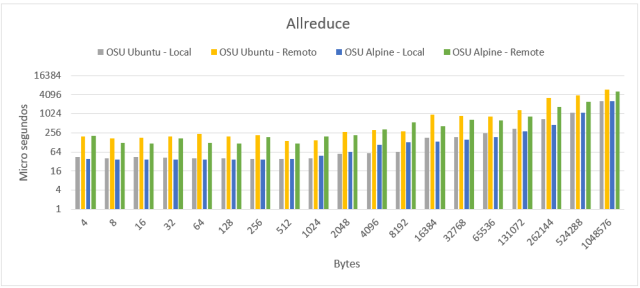

Figure 12: Prueba *allreduce* con OSU Microbenchmarks

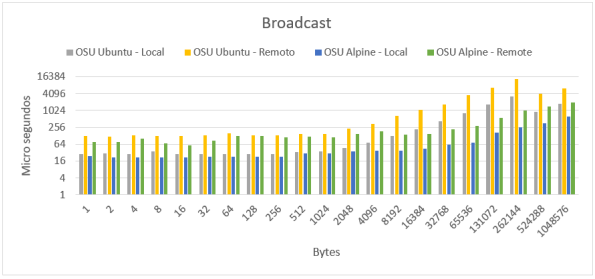

Figure 13: Prueba *broadcast* con OSU Microbenchmarks

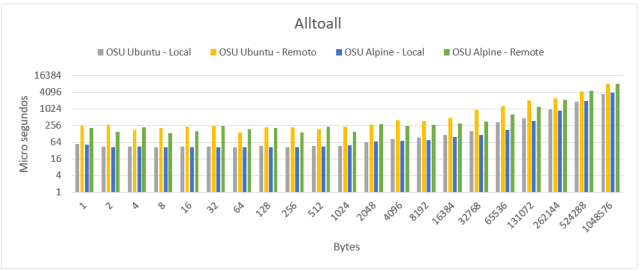

Figure 14: Prueba *alltoall* con OSU Microbenchmarks

Los resultados detallados de éstas y el resto de las pruebas, así como los obtenidos en `ubuntu-openmpi` de la ejecución de OSU Microbenchmarks y otro *benchmark*, Intel MPI Benchmarks)[12], que por incompatibilidades no se puedo ejecutar en

[12]https://software.intel.com/content/www/us/en/develop/articles/intel-mpi-benchmarks.html

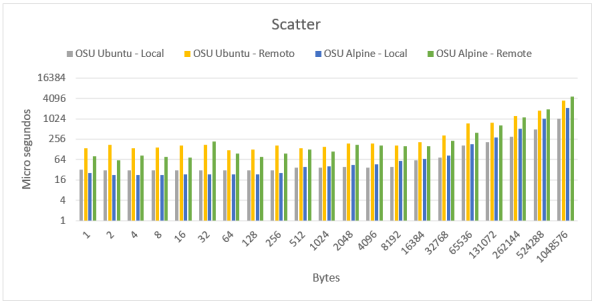

Figure 15: Prueba *scatter* con OSU Microbenchmarks

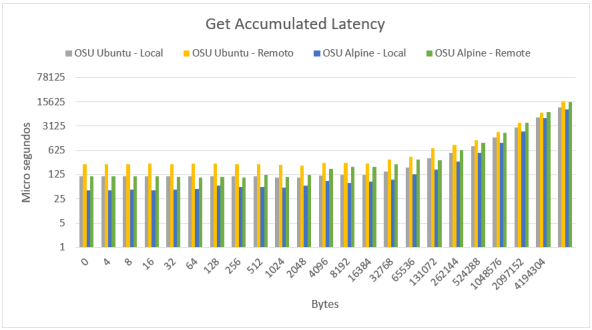

Figure 16: Prueba *get accumulated latency* con OSU Microbenchmarks

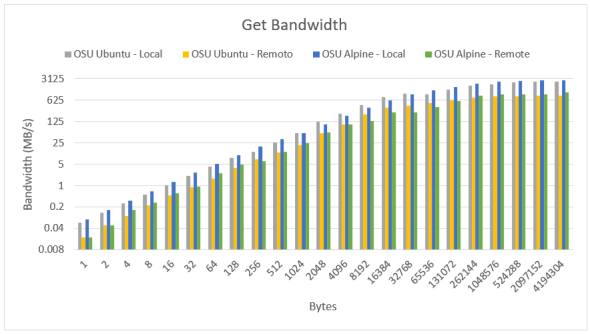

Figure 17: Prueba *get bandwidth* con OSU Microbenchmarks

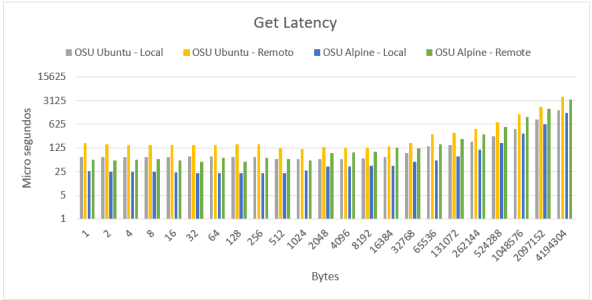

Figure 18: Prueba *get latency* con OSU Microbenchmarks

la versión `alpine-mpich`, se encuentran disponibles en el repositorio de este proyecto [23].

Los OSU Microbenchmarks fueron lo suficientemente ligeros para ejecutarlos sin problema a través de ambos ambientes de prueba y con ambas versiones de la solución. Los resultados muestran que para el ambiente de red local hay menos latencia y mejor desempeño de las operaciones MPI, en particular para cargas bajas. Esto se debe principalmente a que el ambiente `PlayWithDocker` utiliza una arquitectura `Docker In Docker`, que permite ejecutar una instancia de Docker dentro de un contenedor, en este caso para dar el efecto de tener múltiples máquinas virtuales. Se presume que ésto contribuyó al aumento de la latencia. Sin embargo, el ambiente remoto virtualizado permitió localizar mejor los recursos dada la homogeneidad de los nodos del clúster deplegado, mientras que la distribución de las tareas en los nodos heterogéneos de la red local no fue balanceada, dada la diferencia en la cantidad de núcleos de los procesadores utilizados. Adicionalmente, es importante enfatizar que los resultados obtenidos fueron similares para ambas versiones disponibles en nuestra solución: `ubuntu-openmpi` y `alpine-mpich`.

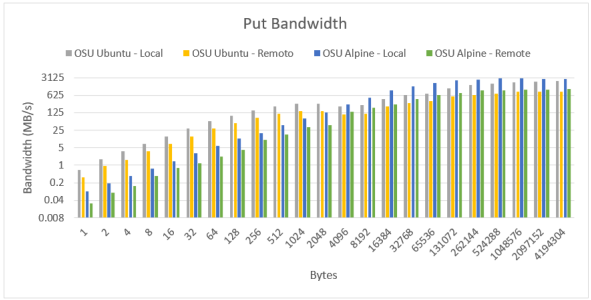

Figure 19: Prueba *put bandwidth* con OSU Microbenchmarks

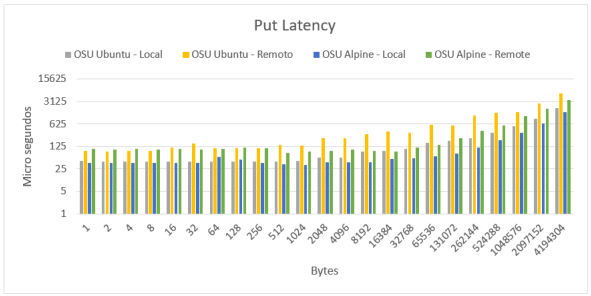

Figure 20: Prueba *put latency* con OSU Microbenchmarks

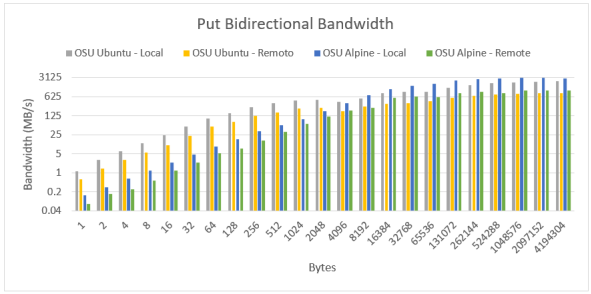

Figure 21: Prueba *bidirectional bandwidth* con OSU Microbenchmarks

Estos resultados demuestran que nuestra solución puede realizar trabajos pesados en ambos tipos de ambiente, sin embargo es más recomendable utilizar el ambiente remoto virtualizado para cargas pesadas debido a su homogeneidad, balance de carga, mientras que el ambiente local es ideal para cargas ligeras puesto que no tiene la carga extra de la virtualización. Para casi todos los comandos mpi, Ubuntu y Alpine Remotos son más lentos que Ubuntu y Alpine locales, como es de esperarse. Sin embargo, para unos pocos comandos resulta

lo inverso. Estos resultados es necesario verificarlos con más pruebas para identificar las razones de este comportamiento, que en principio pueden ser causados por accesos locales a discos que resultan más lentos que el acceso a la red.

Aún cuando la solución que proponemos puede ser usada en cualquier hardware disponible o incluso en un ambiente virtualizado, su uso está orientado a fines académicos más que para ambientes de producción reales. En este sentido, nuestra solución de despliegue automático de un cluster HPC basada en contenedores Docker, no pretende competir con un ambiente de clúster tradicional HPC en términos de velocidad o capacidad de procesamiento, dada la naturaleza del hardware sobre el cual fue probado y para el cual fue concebido (computadores personales interconectados principalmente a través de cables Ethernet). Sin embargo, su arquitectura le otorga las siguientes características (la mayoría no disponibles en un ambiente de clúster tradicional), que en un ambiente académico son de mucho beneficio:

- **Mayor accesibilidad:** Facilita el aprendizaje de conceptos de computación distribuida a estudiantes e investigadores que generalmente, no tienen acceso a un ambiente de este tipo.
- **Ambiente de pruebas portable (*sandbox*):** Debido a que Docker soporta la mayoría de los sistemas operativos, es posible ejecutar una o más instancias del clúster en cualquiera de ellos. Permitiendo a los desarrolladores ejecutar sus programas independientemente de la plataforma en la que se encuentren. Esto es cierto siempre y cuando sus computadores tengan un sistema operativo de 64 bits, bien sea Linux (soporte nativo de contenedores `lxc`) o posean soporte de virtualización en Windows o Mac (`docker-toolbox` y `docker-for-mac`, respectivamente).
- **Maleable, dinámico:** Modificar la configuración de lanzamiento (*script* de lanzamiento y archivo `yaml` (estándar de serialización de datos amigable), disponible para todos los lenguajes de programación o las herramientas y configuraciones del sistema (`Dockerfile`), bastará para modificar la arquitectura y ambiente del clúster. Además, la herramienta de orquestación de contenedores, Docker Swarm, se encarga de utilizar los recursos disponibles, incluso si varias en el tiempo (se agregan o se quitan computadores), manteniendo consistente el estado del clúster.

VIII. Conclusiones y Trabajo Futuro

Es este trabajo presentamos una solución de software basada en contenedores Docker para el despliegue automático de clústers HPC. Se generaron dos imágenes posibles de despliegue: `alpine-mpich` o `ubuntu-openmpi`. Presentamos la estrategia que usamos para generar las imágenes, cuyo enfoque puede ser usado para incorporar otras imágenes con diferentes sistemas operativos y librerías de comunicación. Ambas versiones se evaluaron en dos escenario diferentes, con una red local y con nodos virtualizados en un servidor remoto. Luego de estudiar los resultados obtenidos, se puede apreciar que la solución desarrollada es más recomendable para entornos académicos que para entornos investigativos, sin embargo es de gran utilidad para ambos casos. Existe el caso borde de poseer una arquitectura física lo suficientemente potente y homogénea, donde sea posible realizar el despliegue y cumplir con los requerimientos de cargas mucho más pesadas fácilmente. En ese caso se podría alcanzar un nivel de ambiente de producción, integrando la librería [3].

Actualmente trabajamos en la integración de la herramienta `slurm`, para generar una nueva versión. También se esta investigando sobre el despliegue de un registro local con las imágenes del clúster, además de la inclusión una interfáz web de usuario para gestión de instancias del clúster.

References

[1] H.-E. Yu and W. Huang, "*Building a Virtual HPC Cluster with Auto Scaling by the Docker*," CoRR: arXiv preprint arXiv:1509.08231, vol. abs/1509.08231, pp. 1–4, 2015.

[2] J. Higgins, V. Holmes, and C. Venters, "*Orchestrating Docker Containers in the HPC Environment*," in International Conference on High Performance Computing, pp. 506–513, Springer, 2015.

[3] J. Zhang, X. Lu, and D. K. Panda, "*High Performance MPI Library for Container-based HPC Cloud on InfiniBand Clusters*," in 2016 45th International Conference on Parallel Processing (ICPP 2016), pp. 268–277, IEEE, 2016.

[4] M. T. Chung, N. Quang-Hung, M.-T. Nguyen, and N. Thoai, "*Using Docker in High Performance Computing Applications*," in 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE 2016), pp. 52–57, IEEE, 2016.

[5] N. Nguyen and D. Bein, "*Distributed MPI Cluster with Docker Swarm Mode*," in 2017 IEEE 7th annual Computing and Communication Workshop and Conference (CCWC 2017), pp. 1–7, IEEE, 2017.

[6] A. Azab, "*Enabling Docker Containers for High-performance and Many-task Computing*," in 2017 IEEE International Conference on Cloud Engineering (IC2E 2017), pp. 279–285, IEEE, 2017.

[7] M. de Bayser and R. Cerqueira, "*Integrating MPI with Docker for HPC*," in 2017 IEEE International Conference on Cloud Engineering (IC2E 2017), pp. 259–265, IEEE, 2017.

[8] N. Zhou, Y. Georgiou, M. Pospieszny, L. Zhong, H. Zhou, C. Niethammer, B. Pejak, O. Marko, and D. Hoppe, "*Container Orchestration on HPC Systems through Kubernetes*," *Cloud Computing*, vol. 10, pp. 1–14, Dec. 2021.

[9] N. Nguyen, "*Alpine MPICH*," May 2018.

[10] R. Filgueira, "*docker.openmpi*," Aug. 2015.

[11] G. A. Verrocchi Carias and J. F. Hernández Balestrini, "*docker-HPC-cluster*," Feb. 2019.

[12] J. Higgins, V. Holmes, and C. Venters, "*VCC: A Framework for Building Containerized Reproducible Cluster Software Environments*," The Journal of Open Source Software, vol. 2, 03 2017.

[13] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary, "*HPL - A Portable Implementation of the High-performance Linpack Benchmark for Distributed-memory Computers*," Feb. 2018.

[14] F. Hu and J. Zhang, "*A Docker-based HPC Architecture for RS Imagery Processing*," in IOP Conference Series: Earth and Environmental Science, vol. 509, p. 012026, IOP Publishing, 2020.

[15] L. Benedicic, F. A. Cruz, A. Madonna, and K. Mariotti, "*Sarus: Highly Scalable Docker Containers for HPC Systems*," in International Conference on High Performance Computing, pp. 46–60, Springer, 2019.

[16] V. Vieux, "*sshFS Docker Volume Plugin*."

[17] B. Christner, "*Docker Official Images are Moving to Alpine Linux*," May 2016.

[18] S. Bhartiya, "*Meet Alpine Linux, Docker's Distribution of Choice for Containers - The New Stack*," Mar. 2017.

[19] N. Janetakis, "*Benchmarking Debian vs Alpine as a Base Docker Image*," Oct. 2018.

[20] N. Nguyen, "*Alpine-MPICH*," 2017.

[21] P. Gedam, "*pip - The Python Package Installer*," Feb. 2019.

[22] L. Dalcin, "MPI for Python," Feb. 2019.

[23] G. A. V. Carias and J. F. H. Balestrini, "*Project Info*," Mar. 2019.

[24] D. K. D. Panda, "*MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE*," Mar. 2019.