Pulpo y la necesidad de un ambiente colaborativo para el estudio del cómputo paralelo en México

J.I. Hernández[1], Víctor Morales[2], Ramón Parra[3]
[1]Instituto de Ingeniería Eléctrica y Computación
Universidad Autónoma de Ciudad Juárez
Av. Del Charro No. 450 Nte., Col. Partido Romero. Cd. Juárez, Chihuahua. C.P. 32310
Israel.hernandez@uacj.mx[1], victor.morales@uacj.mx[2], rparra@uacj.mx[3]

Fecha de recepción: 5 de septiembre 2014
Fecha de aceptación: 2 de octubre 2014

Resumen

Existen aplicaciones cuya solución algorítmica, requiere del uso de una cantidad considerable de recursos computacionales (CPU, memoria y almacenamiento). Resolver estas aplicaciones en computadoras secuenciales pudiera generar costos considerables en términos de desempeño y tiempo. Por otro lado, avances recientes en tecnologías de redes permiten a un conjunto de computadoras conectadas en red, colaborar de manera coordinada en la solución de un problema particular. Esto ha impulsado el cómputo paralelo en red como una alternativa viable en la solución de aplicaciones complejas. La idea es particionar una tarea compleja en tareas más pequeñas que se ejecutan de manera coordinada entre las diferentes computadoras de la red. Los algoritmos de asignación de tareas a computadoras se vuelven fundamentales al buscar reducir el tiempo de ejecución de la aplicación particionada entre las computadoras de la red. Este artículo busca fomentar el estudio del cómputo paralelo entre las instituciones de educación superior en México. Pulpo es una herramienta de simulación creada para evaluar algoritmos de asignación de tareas en plataformas distribuidas. Se pretende que pulpo sea una herramienta útil a la comunidad académica interesada en el área y a su vez permita crear un ambiente colaborativo en el estudio del cómputo paralelo. Proporcionamos resultados experimentales y enseñamos a utilizar las librerías de pulpo con un ejemplo.

Palabras Clave

Cómputo distribuido, cómputo paralelo, algoritmos heurísticos, DAG scheduling.

1. Introducción

Existen aplicaciones cuya solución algorítmica requiere del uso de una cantidad considerable de recursos computacionales (CPU, memoria y almacenamiento). Pretender resolver estas aplicaciones en computadoras secuenciales pudiera ocasionar considerables costos en términos de desempeño y tiempo. Tales aplicaciones comienzan a ser más comunes en ámbitos académicos y algunos sectores empresariales. Ejemplos de este tipo de aplicaciones son el procesar una cantidad masiva de datos para encontrar el bosón de Higgs, modelos para predecir el clima, reconocimiento de patrones, modelos geofísicos para la industria petrolera, entre otros.

Por otro lado, avances recientes en tecnologías de redes, permiten a un conjunto de computadoras heterogéneas conectadas en red, compartir recursos y colaborar coordinadamente en la solución de un problema [3,7,13]. Esto ha impulsado la búsqueda de nuevos paradigmas de programación que pueden ser útiles en la solución de aplicaciones complejas. El presente proyecto considera el paradigma del cómputo paralelo [19,20] en red como una alternativa viable en la solución de aplicaciones algorítmicas complejas. La noción del cómputo paralelo es partir una tarea compleja en tareas más pequeñas tal que estas puedan asignarse y ejecutarse de manera coordinada entre las diversas computadoras que componen la red. La aplicación particionada se pueden representar por medio de diversos grafos, dependiendo de la relación que exista entre las diversas tareas que conforman la aplicación. Este proyecto considera aplicaciones particionadas que se pueden representar por medio de un grafo dirigido acíclico (DAG, por sus siglas en ingles), donde las aristas representan las tareas particionadas y los vértices representan una relación precedencia entre tareas. La complejidad de estas aplicaciones radica en considerar las restricciones de precedencia al momento de asignar y ejecutar las tareas que forman el DAG a las computadoras de la red.

Se ha demostrado que el problema de asignar tareas de un DAG a computadoras es NP-completo en la mayoría de los casos [1]. Esto ha inspirado a muchos investigadores a proponer algoritmos heurísticos de baja complejidad y fácil implementación para su solución. En la literatura se observa que conforme avanza la tecnología de redes se hace necesario desarrollar nuevos tipos de algoritmos heurísticos para explotar las características emergentes de las redes. De esta forma, una gran cantidad de algoritmos en la literatura consideran las computadoras de la red como dedicadas a la aplicación y con el mismo desempeño a lo largo del tiempo (ej. clusters) [4,6,8,12,14]. Algunos algoritmos recientes consideran redes con computadoras heterogéneas no dedicadas, distribuidas geográficamente y con desempeño variante a lo largo del tiempo (ej. grids, nubes, etc.) [5,10,11,17,18,21]. Estrategias de asignación de tareas en este tipo de plataformas se vuelven

fundamentales, ya que las redes actuales siguen está tendencia. Además, se necesitan desarrollar estrategias que permitan la tolerancia de fallas [11,16] y optimizar el problema del tráfico de datos en la red [2]. A pesar del esfuerzo mencionado, la naturaleza del problema ofrece oportunidades a la comunidad universitaria de contribuir en el entendimiento y dominio del problema.

La mejor estrategia para evaluar el desempeño de los algoritmos de asignación debería ser mediante su implementación en ambientes reales. Sin embargo esto se complica por las siguientes razones. 1) el desarrollo de una aplicación paralela real no es sencillo y se necesita invertir un tiempo considerable en la implementación. 2) aplicaciones paralelas en ambientes reales toman largos periodos de tiempo en ejecutarse y para que los resultados sean estadísticamente confiables se tendrían que ejecutar un número considerable de experimentos. 3) es difícil predecir el desempeño de los recursos en el tiempo. 4) es difícil conocer la configuración de las computadoras de la red, en especial en ambientes donde los recursos no son dedicados. 5) la naturaleza cambiante de los recursos dificulta obtener patrones repetitivos, los cuales son muy importantes en el contexto de la investigación.

Por lo anterior, hay una clara necesidad de desarrollar herramientas de simulación que permitan evaluar algoritmos de asignación en redes de manera realista. En [9,15] los autores reportan trabajos similares realizados en este sentido: El proyecto Simgrid desarrollado por el INRIA (Institut National de Reserche en Infomatique et en Automatique) en Francia y el proyecto Gridsim desarrollado por la Universidad de Melbourne en Australia. Pulpo contiene funcionalidades distintivas enfocadas en modelar plataformas computacionales con tendencias actuales. Tales funcionalidades permiten modelar la naturaleza dinámica de los recursos de red a lo largo del tiempo. Un caso particular es la posibilidad de modelar una falla controlada en algún recurso computacional de la red. Obviamente, los algoritmos de asignación de tareas deben de tener mecanismos para reaccionar a los ambientes dinámicos. Sin embargo, existen plataformas distribuidas recientes (ej., grids, clouds) que pueden ser extremadamente complejos de modelar en su totalidad y pulpo pudiera no ser suficiente para abstraer todos los detalles en la simulación (ej., sistemas por lotes, desperdicio de ancho de banda, etc.). Hasta el momento de la escritura del presente artículo, no existen trabajos similares en México.

Con el diseño de pulpo se pretende que un investigador en el área (i) tenga una herramienta con un nivel de abstracción adecuado para realizar su trabajo; (ii) implemente de manera rápida sus algoritmos de asignación; (iii) realice simulaciones más realistas que otros trabajos previos; (iv) pueda realizar pruebas de rendimiento que permitan evaluar su algoritmo con respecto a otros algoritmos; (v) genere resultados de simulación confiables.

Este artículo está organizado de la siguiente manera. La sección 2 muestra la visión general del proyecto. La sección 3 define el problema de asignación de tareas de una manera formal. La sección 4 presenta los resultados obtenidos. La sección 5

describe la API de pulpo y la sección 6 concluye el artículo con las conclusiones y trabajo futuro.

2. Visión general del proyecto

Pulpo es el resultado de un esfuerzo por fomentar el estudio del cómputo paralelo entre las diversas instituciones de educación superior en México. Anteriormente, se ha descrito la importancia que tiene el cómputo paralelo en la solución de problemas complejos. Es necesario que en México se busquen nuevas formas de avanzar en el estudio y comprensión de esta área del conocimiento. Experiencias propias permiten identificar algunas áreas de oportunidad.

1.- Sin generalizar, una buena parte de los investigadores nacionales están concentrados en escribir artículos para diversas revistas, porque así lo dictan las entidades encargadas de definir las políticas de tecnología del país. En cambio, si nuestros investigadores enfocaran sus esfuerzos en desarrollos tecnológicos que estén al alcance de la comunidad académica y empresarial, permitiría avanzar en la adquisición de experiencia, conocimiento y dominio del área.

2.- No existe un padrón con información de las líneas de investigación donde se desempeñan los investigadores nacionales que laboran tanto en universidades nacionales como extranjeras. De contar con tal padrón, se podrían crear vínculos de colaboración orientados a compartir experiencias y conocimiento en el área, así como en la creación de nuevos proyectos de investigación.

3.- La formación de estudiantes en carreras relacionadas con la computación considera un enfoque secuencial en el ámbito de la programación de computadoras, esto dificulta a los estudiantes el poder entender y aplicar el cómputo paralelo en la solución de una aplicación algorítmica.

4.- El idioma es un factor que puede desmotivar a algunos estudiantes a incursionar en el cómputo paralelo. Parte de nuestro esfuerzo está encaminado a que los estudiantes puedan utilizar pulpo como herramienta y recibir retroalimentación en español que les permita proponer y desarrollar sus propios proyectos.

5.- Creación de métodos de enseñanza que permitan formar estudiantes e investigadores en los principios del cómputo paralelo.

6.- Investigadores nacionales relacionados con esta área laboran principalmente en las distintas universidades del país realizando trabajos académicos. Es necesario buscar mecanismos para acercar los investigadores a la industria nacional. Esto permitirá promover las ventajas del cómputo paralelo en la industria y obtener financiamiento para proyectos de desarrollo tecnológico.

La Fig. 1 muestra la visión que se busca promover con este proyecto en el área del cómputo paralelo. Se pretende desarrollar herramientas de software que nos permitan dominar el ciclo de vida de una aplicación paralela. Tal ciclo comienza con un análisis científico que permita definir el dominio de la aplicación. Posteriormente la tarea compleja se divide en tareas más pequeñas, cada una con su respectivo dominio.

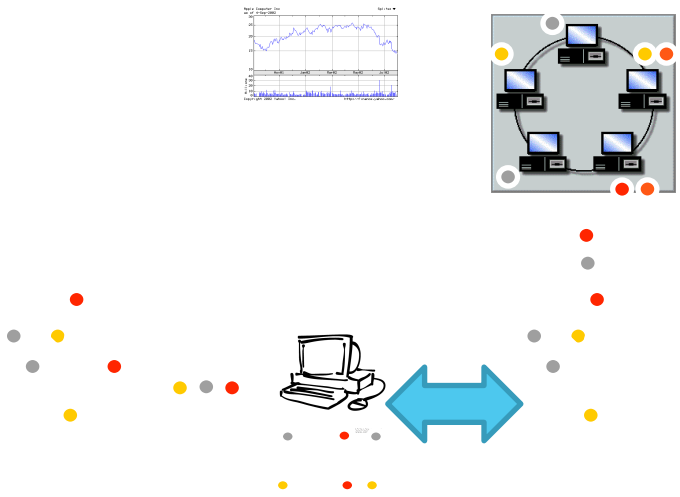


Fig 1. Ciclo de vida de una aplicación paralela

Con las tareas particionadas es necesario definir las dependencias y transferencias de datos entre las diferentes tareas formando el DAG. La complejidad en las aplicaciones tipo DAG radica en considerar las restricciones de precedencia entre las tareas al momento de asignar y ejecutar la aplicación entre las computadoras de la red. Por otro lado se debe tener visualizada la topología de la red donde el DAG se va a ejecutar. Antes de emprender la ardua labor de implementar la aplicación en real, es necesario recurrir a la simulación para encontrar patrones repetitivos que permitan entender y mejorar el desempeño la aplicación. Pulpo provee las funcionalidades y abstracciones necesarias para cumplir con tal objetivo. El análisis de las simulaciones pudiera ocasionar ajustes en la forma del DAG, de la topología de red ó del algoritmo de asignación. El siguiente paso es construir y ejecutar la aplicación paralela real teniendo presente los resultados de la simulación como base. Una

extensión de este proyecto busca desarrollar un conjunto de librerías de software que permitan a un investigador implementar una aplicación paralela real. Por último, la experiencia adquirida en el proceso debe retroalimentar el ciclo para futuras aplicaciones.

3. Definición formal del problema

Esta sección presenta una definición formal del problema de asignación de tareas de un DAG en redes con computadoras heterogéneas y dinámicas, características de las redes actuales que permitirán modelar escenarios realistas. Las definiciones en esta sección están alineadas con la modelación soportada en pulpo.

3.1 Definición de la Plataforma de Red

Los recursos que componen la red se representan por medio de un grafo no dirigido PD::(P, L, d, y) donde P es el conjunto de computadoras disponibles que forman la plataforma, pi($1 \leq i \leq |P|$). L es el conjunto de enlaces de comunicaciones que conectan un par de distintos procesadores, li($1 \leq i \leq |L|$) tal que l(pm, pn) $\in$ L denota un enlace de comunicación entre pm y pn. Para modelar la naturaleza cambiante en el desempeño de los recursos computacionales se utiliza d::P →[0..1] que denota el porcentaje de disponibilidad de cada computadora y y:: L→Float que denota el ancho de banda de cada enlace de comunicación. Esta definición permite considerar el caso extremo en que la disponibilidad de un recurso es igual a cero.

3.2 Definición de la Aplicación Patrocinada

La aplicación particionada se representa por medio de un DAG AP::(V, E, q, t). V representa el conjunto de tareas que componen la aplicación vi($1 \leq i \leq |V|$). $E \subseteq V \times V$ es el conjuntos de arcos dirigidos que conectan distintos pares de tareas ei($1 \leq i \leq |E|$), así e(vi,vj) denota una transferencia de datos de vi a vj y a la vez una precedencia que indica que vj no puede comenzar a ejecutarse hasta que vi termine su ejecución y envié sus datos respectivos a vj. Por conveniencia se define Pred(vi) para denotar el subconjunto de tareas que directamente preceden a vi y Succ(vi) para denotar el subconjunto de tareas que directamente suceden a vi. Aquella tarea vi tal que |Pred(vi)| = 0 es llamada tarea de entrada y |Succ(vi)|= 0 es llamada tarea de salida. Usamos q::VxV→ int para describir el costo de la transferencia de datos, tal que q (vi, vj) denota la cantidad de datos a ser transferidos de vi a vj. Considerando que los procesadores son heterogéneos, los tiempos estimados de ejecución se representan como t::VxP → Int, donde t (vi, pm) denota el costo de ejecución estimado de la tarea vi en el procesador pm.

3.3 El Algoritmo de Asignación

Los algoritmos de asignación se enfocan en la generación de una planificación

(Scheduling) de tareas en las computadoras de la red, buscando optimizar una función objetivo, que por lo regular es minimizar el tiempo estimado de ejecución (makespan) de la aplicación. La planificación de tareas se puede representar como una función ASIGNA::V → P , la cual asigna tareas a procesadores. De esta forma, ASIGNA(vm,pj) denota que la tarea vm se asigna al procesador pj.

3.4 Mecánica de la Simulación

Pulpo considera que el DAG tiene una tarea de entrada y una tarea de salida. En caso de que un DAG particular pudiera tener más de una tarea de entrada, este se puede modelar agregando una tarea dummy conectada a las diversas tareas de entrada, cuyo costo de cómputo y comunicación sea cero. El mismo proceso se sigue en el caso de que existan varias tareas de salida.

Para coordinar la ejecución de las tareas, utilizamos diferentes tipos de status que permiten determinar la situación de una tarea en cualquier momento a lo largo de la simulación. Los status que puede tener una tarea son los siguientes: 0-creada, 1-asignada, 2-lista para ejecutarse, 3-ejecutándose, 4-pausada, 5-terminada, 6-falla. La tarea de entrada es la primera en ejecutarse y de ahí comienzan a ejecutarse el resto de las tareas. La tarea de salida es la última tarea en ejecutarse.

Cuando una tarea vi comienza a ejecutarse en un procesador pn, se puede utilizar TI(vi, Pn) y TF(vi, Pn) para denotar el tiempo de inicio y tiempo final de ejecución de vi en pn respectivamente. El TI de una tarea de entrada es cero y para las demás tareas se calcula de la siguiente forma.

$$TI(vi, pn) = \max \{ DISP(pn), \max_{vk \in Pred(vi)} (TF(vk, pk) + C(vk, pk, vi, pn)) \} \quad (1)$$

Donde en la primer parte de la ecuación se determina DISP(pn) para indicar el tiempo más cercano en el cual pn está listo para ejecutar vi. La siguiente parte de la ecuación hace referencia al tiempo en que vi recibe la totalidad de los datos transferidos de parte de sus predecesores y por lo tanto se encuentra lista para ejecutarse. Esto se obtiene considerando las tareas predecesoras inmediatas a vi, el tiempo final (TF) en que estas terminan de ejecutarse en su respectivo procesador pk y el tiempo que tarda para transferir los datos necesarios de pk al procesador en consideración pn. El TF de vi en pm se determina por

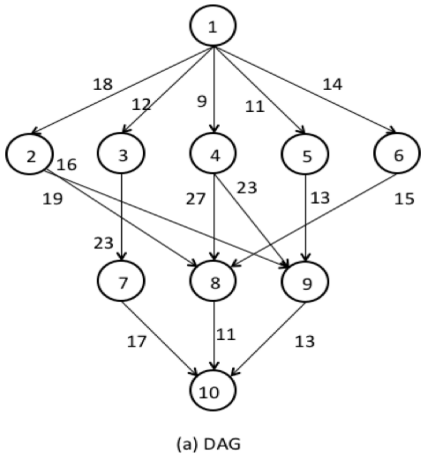
$$TF(vi, pm) = TI(vi, pm) + t(vi, pm) \quad (2)$$

que indica que el tiempo final de ejecución de una tarea vi en pm equivale al tiempo inicial de ejecución más el tiempo que esta tarda en ejecutarse en el procesador asignado pm. El makespan de la aplicación está determinado por el tiempo de ejecución de vsalida (última tarea del DAG).

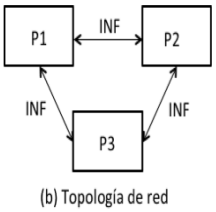
makespan = TF (vsalida) (3)

4. Resultados

En esta sección evaluamos la API de Pulpo y presentamos algunos resultados obtenidos con la herramienta. Para nuestro propósito vamos a utilizar el artículo en [14] donde se describen los algoritmos heurísticos HEFT (Heterogeneous Earliest-Finish-Time) y CPOP (Critical-Path-on-a-Processor). Estos algoritmos utilizan el ejemplo de la Fig. 2 para generar su respectiva planificación. Al igual que la mayoría de los algoritmos de asignación con enfoque heurísticos, para mantener baja la complejidad del algoritmo, no se considera el tráfico de la red sino que se considera un número infinito de enlaces de comunicación. Esto significa que una tarea que finaliza su ejecución, siempre será de capaz de encontrar un enlace de comunicación disponible para transferir datos a sus sucesores. Como se muestra más adelante, esta consideración puede provocar inconsistencias al momento de evaluar los algoritmos.



(a) DAG



(b) Topología de red

Task	P1	P2	P3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

(c) Tiempo de ejecución estimados

Fig. 2. DAG con 10 tareas y una topología de red con un infinito número de enlaces

De acuerdo a lo reportado en [14], las planificaciones generadas por ambos algoritmos se muestran en la Fig. 3. Se observa que HEFT tiene un makespan igual a 80 y CPOP tiene un makespan de 86. De esto se deduce, que para este ejemplo y en un ambiente que considera un número infinitos de enlaces de comunicación, HEFT es mejor que CPOP en un 7.07%. Sin embargo, si se toman del ejercicio las mismas planificaciones generadas por HEFT y CPOP y los evaluamos en pulpo en un ambiente más realista, donde todos los enlaces de comunicación entre computadoras tienen un ancho de banda limitado, en este ejemplo consideramos un ancho de banda igual a uno, lo que significa que en una unidad de tiempo, un enlace de comunicación puede transmitir una unidad de datos. En la Fig. 4 se observan los resultados

generados por pulpo: HEFT (makespan = 107) y CPOP (makespan = 100). Esto significa que en un ambiente más realista CPOP es mejor que HEFT en un 7%, resultado opuesto al obtenido en el ejercicio anterior con el escenario poco realista. Utilizar escenarios más realistas en una simulación puede ayudar a evaluar de una mejor manera los algoritmos de asignación.

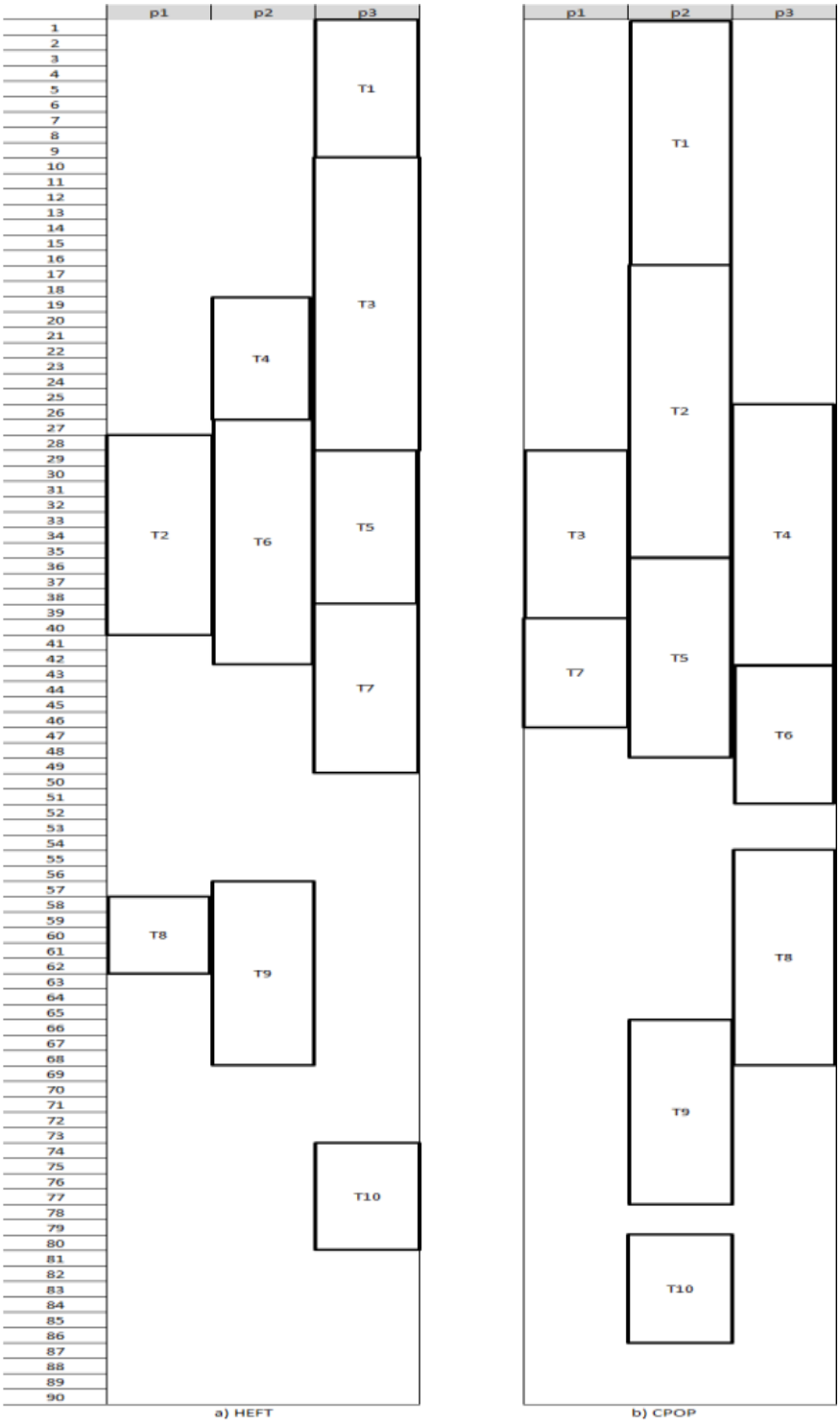


Fig. 3. HEFT (makespan = 80) y CPOP (makespan = 86)

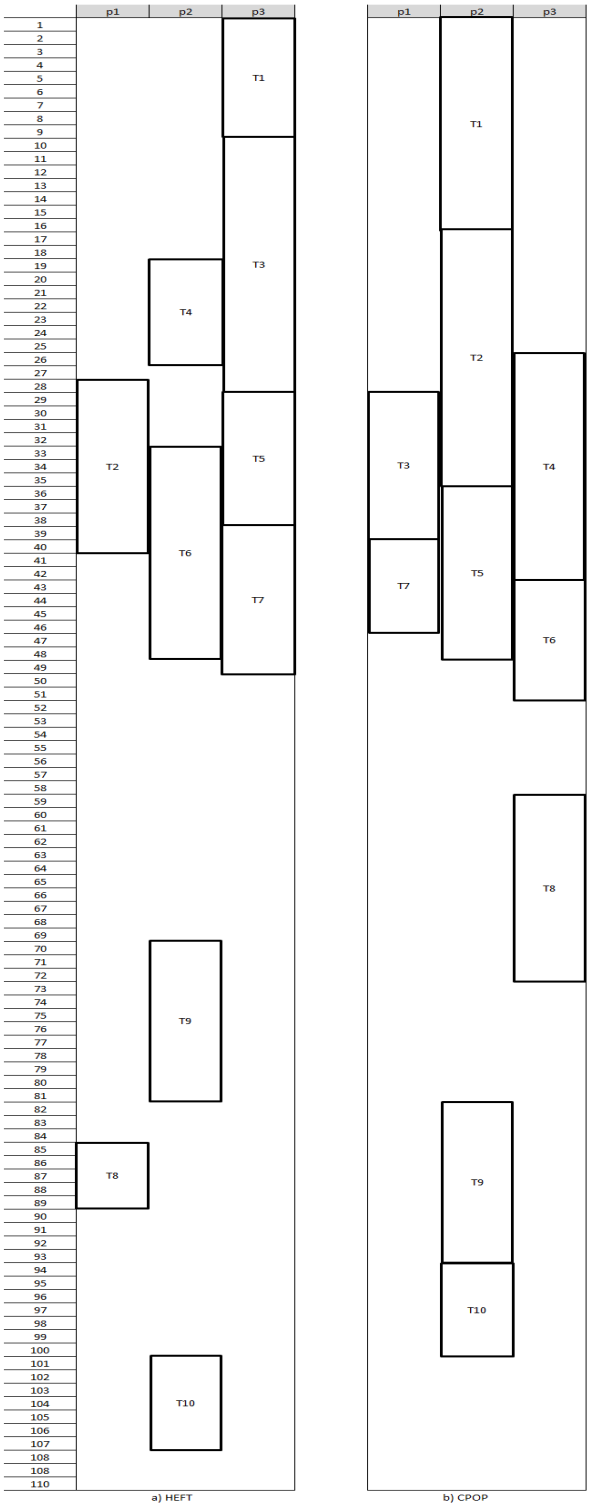


Fig. 4. Usando Pulpo: HEFT (makespan = 107) y CPOP (makespan = 100)

5. API de Pulpo

Pulpo fue implementado en phyton utilizando programación orientada a objetos. Tal decisión se debe a que la estructura de phyton simplifica la programación de aplicaciones, permitiendo que estudiantes/investigadores puedan proponer e implementar sus proyectos de una manera rápida.

Un escenario de simulación es definido al crear una instancia de la clase scenario con el respectivo nombre del escenario. La modelación de las tareas del DAG se hace creando instancias de la clase task especificando el nombre de la tarea respectiva. Posteriormente el método addDependancy permite crear una relación de precedencia entre dos tareas, así como para indicar la cantidad de datos a transmitir entre ambas tareas. El conjunto de las precedencias creadas resulta en el DAG. La modelación de una topología de red particular es a partir de la creación de instancias de la clase processor con su respectiva disponibilidad y latencia. Posteriormente se utiliza el método addLink para crear un enlace de comunicación entre dos procesadores con su respectivo ancho de banda. La naturaleza cambiante de los recursos en el tiempo se puede modelar de la siguiente manera: En el caso de las computadoras se utiliza el método addChangeCapacity cuyo primer parámetro es el tiempo y el segundo parámetro es la capacidad que cambia en el tiempo definido. En el caso de los enlaces de comunicación se tiene el método addChangeLinkBandwidth para modelar un cambio en algún punto del tiempo del ancho de banda. Para modelar la asignación de tareas a computadoras heterogéneas se usa el método addSchedule de una instancia task, donde el primer parámetro es una instancia tipo procesador al cual se asigna la tarea y el segundo parámetro representa el tiempo que tarda la tarea en ejecutarse en el procesador. La simulación se realiza al llamar el método simulation de una instancia tipo scenario.

La Fig. 5 muestra un ejemplo de pulpo considerando la representación del DAG de la Fig. 2. Observamos la creación de las tareas del DAG con sus respectivas dependencias y transferencias de datos. La topología está compuesta por tres computadoras con sus respectivas capacidades, las cuales están totalmente conectadas por un enlace de comunicación con su respectivo ancho de banda. Se incluyen algunos cambios en recursos a través del tiempo que son reflejados en la simulación. También se muestra la forma de asignar una tarea a una computadora para su posterior ejecución.

```
from classes import *
from dag import *
from integrity import *
from simulation import *

''' creación del escenario'''
SC = scenario("Escenario")

''' tasks'''
t1=dag.addTask("t1")
t2=dag.addTask("t2")
t3=dag.addTask("t3")
t4=dag.addTask("t4")
t5=dag.addTask("t5")
t6=dag.addTask("t6")
t7=dag.addTask("t7")
t8=dag.addTask("t8")
t9=dag.addTask("t9")
t10=dag.addTask("t10")

''' DAG dependencies '''
dt0 = dag.addDependancy(t1,t2, 18)
dt1 = dag.addDependancy(t1,t3, 12)
dt2 = dag.addDependancy(t1,t4, 9)
dt3 = dag.addDependancy(t1,t5, 11)
dt4 = dag.addDependancy(t1,t6, 14)
dt5 = dag.addDependancy(t2,t8, 19)
dt6 = dag.addDependancy(t2,t9, 16)
dt7 = dag.addDependancy(t3,t7, 23)
dt8 = dag.addDependancy(t4, t8, 27)
dt9 = dag.addDependancy(t4,t9, 23)
dt10 = dag.addDependancy(t5,t9, 13)
dt11 = dag.addDependancy(t6,t8, 15)
dt12 = dag.addDependancy(t7,t10, 17)
dt13 = dag.addDependancy(t8,t10, 11)
dt14 = dag.addDependancy(t9,t10, 13)

''' processors '''
p1=net.addProcessor("p1", 1.0, 0.0)
p2=net.addProcessor("p2", 1.0, 0.0)
p3=net.addProcessor("p3", 1.0, 0.0)

''' network '''
L1 = net.addLink(p1,p2, 1.0, 1.0)
L2 = net.addLink(p1,p3, 1.0, 1.0)
L3 = net.addLink(p2,p3, 1.0, 1.0)

''' changes in capacity over time '''
p1.addChangeCapacity(100, 2)
L1.addChangeLinkBandwidth(150,3)
L3.addChangeLinkBandwidth(200,3)

''' schedule '''
t1.addSchedule(p2,16)
t2.addSchedule(p2,19)
t3.addSchedule(p1,11)
t7.addSchedule(p1,7)
t4.addSchedule(p3,17)
t5.addSchedule(p2,13)
t9.addSchedule(p2,12)
t6.addSchedule(p3,9)
t8.addSchedule(p3,14)
t10.addSchedule(p2,7)

''' simulacion '''
SC.simulation()
```

Fig. 5. Ilustración de la API de Pulpo

5. Conclusiones y trabajo futuro

En este artículo se expuso la necesidad de contar con un ambiente colaborativo para el estudio y comprensión del cómputo paralelo en México. Se describió a pulpo, una herramienta de simulación para evaluar algoritmos de asignación de tareas que forman un DAG en una red de computadoras con características heterogéneas y dinámicas. Pulpo provee funcionalidades y abstracciones necesarias para la implementación y evaluación de algoritmos de asignación en ambientes realistas. Resaltamos la importancia de pulpo cuando algoritmos de la literatura utilizan escenarios poco realistas y esto puede ocasionar inconsistencias en sus evaluaciones. Pulpo puede apoyar a robustecer y entender el comportamiento de los algoritmos de evaluación al incluir funcionalidades para configurar escenarios realistas. Esfuerzos futuros se va a centrar en cuatro aspectos: La confiabilidad de los resultados, depuración de posibles errores, colaborar con la comunidad académica interesada en

el área y buscar patrones que nos ayuden a entender las posibles divergencias de una aplicación simulada en pulpo y ejecutada en ambientes reales.

Referencias

[1] Adam, T., Chandy, K., and Dickson, J. A comparison of list scheduling for parallel processing systems. Communications of the ACM, 17(12):685–690. (1974b).
[2] Agarwal, T., Sharma, A., and Kale, L Topology-aware task mapping for reducing communication contention on large parallel machines. IEEE/IPDPS, page 10 pp. . (2006).
[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets", Journal of Network & Computer Applic., 23(3): 187-200 (1999).
[4] Beaumont, O., Legrand, A., Marchal, L., and Robert, Y. Independent and divisible tasks scheduling on heterogeneous star-shaped platforms with limited memory. Proceedings of the Conference on Parallel,Distributed and Network-Based Processing(Euromicro-PDP'05), pages 179–186. (2005).
[5] Eun-Kyu B., Yang-Suk K., Jin-Soo K., Deelman, E.,"BTS: Resource capacity estimate for time-targeted science workflows", Journal of Parallel Distrib. Comput. (JPDC), 71(6): 848-862 (2011).
[6] Eshaghian, M. and Wu, Y., "Mapping heterogeneous task graphs onto heterogeneous system graphs", In Proceedings of Heterogeneous Computing Workshop (HCW'97), pages 147–160, 1997.
[7] Foster, I., Kesselman, C., and Tuecke, S, "The anatomy of the grid: Enabling scalable virtual organizations", International Journal on Supercomputer Applications, 15(3):200–222 (2001).
[8] Gerasoulis, A. and Yang, T., "A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors", Journal of Parallel and Distributed Computing, 16(4):276–291 (1992).
[9] GridSim,"The GridSim project homepage", (2010) http://www.cloudbus.org/gridsim/.
[10] Hernandez, I. and Cole, M., "Reactive grid scheduling of dag applications", In Proceedings of the 25th IASTED(PDCN), Acta Press, pages 92–97, 2007a.
[11] Hernandez, I. and Cole, M., "Reliable DAG scheduling with rewinding and migration", In Proc.of the First International Conference on Networks for Grid Applications (GridNets), ACM Press, pages 1-8,2007b.
[12] Kwok, Y. and Ahmad, I., "Static algorithms for allocating directed task graphs to multiprocessors", ACM Computing Surveys, 31(4):406–471 (1999).
[13] Mell, P. and Grance, T., "The NIST definition of cloud computing", National Institute of Standards and Technology Special Publication 800-145, 7 pages, (2011).
[14] Topcuoglu, H., "Performance-effective and low-complexity task scheduling for heterogeneous computing", IEEE Transactions on Parallel and Distributed Systems, 13(3):260–274 (2002).
[15] Simgrid , "The simgrid project homepage", http://simgrid.gforge.inria.fr/. (2009)
[16] Sven K., Riddle S. and Zinn D. "Improving Workflow Fault Tolerance through Provenance-Based Recovery". Scientific and Statistical Database Management, Lecture Notes in Computer Science Vol. 6809, 2011, pp 207-224
[17] Olteanu A., Pop F., Dobre C. "Re-scheduling and Error Recovering algorithm for Distributed Environments" .Sci. Bull., Series C., Vol. 73, Iss 1, 2011.

[18] Olteneanu A., Pop F., Dobre C. "A dynamic rescheduling algorithm for resource management in large scale dependable distributed systems". Elsevier Computers & Mathematics with Applications, Vol. 63, Iss 9, pp1409-1423, 2012.
[19] Pacheco P., "An Introduction to Parallel Computing", ISBN-10: 0123742609, Morgan Kaufmann, 1st edition (January 21, 2011)
[20] Razdan S., "Fundamentals of Parallel Computing", ISBN-10: 1842658808, Alpha Science International Ltd, 1st edition (August, 2014)
[21] Vouk M., "On High-Assurance Scientific Workflows". International Symposium on High-Assurance Systems Engineering (HASE), 2011 IEEE 13th.