Multicomputador de Bajo Coste para Entornos Docentes

Carles Aliagas, Montse Garcia-Famoso, Roc Meseguer *Member, IEEE*, Pere Millán y Carlos Molina *Member, IEEE*

CÓMO REFERENCIAR ESTE ARTÍCULO:
C. Aliagas, M. Garcia-Famoso, R. Meseguer, P. Millán y C. Molina
"A Low Cost Multicomputer for Teaching Environments", en IEEE-RITA, Volumen YY, Número Z, pp. AA-BB, Enero 2020
DOI: https://doi.org/...

***Title*—A Low Cost Multicomputer for Teaching Environments**

***Abstract*— We propose a teaching resource based on HardKernel boards to build an MPI server of 256 cores. Although it is a resource with a relatively low performance, the aim is to have access to hundreds of processors in order to carry out scalability analysis, but above all, maintaining a good trade-off between performance, price, and energy consumption. We show details of the implementation, both at hardware and software levels, how this system is used in the context of a parallel programming course and, finally, the point of view of both teachers and students after using this new system.**

***Index Terms*—Parallel Machines, Multicore Processing, Multiprocessor Interconnection, Educational Activities**

I. Introducción

PARA poder adquirir las competencias de asignaturas de paralelismo y computación en los estudios del Grado en Ingeniería Informática, es necesario practicar y tener acceso a sistemas de computación paralelos y distribuidos. Se puede considerar que estándares como OpenMP, MPI y CUDA son los más habituales. Para poder utilizarlos se hace imprescindible tener acceso a hardware específico que permita realizar ejercicios y prácticas. Estos sistemas suelen tener precios elevados y, con frecuencia, el presupuesto limita el número de procesadores del sistema. Mientras que en el ámbito investigador los departamentos suelen tener acceso a sistemas de altas prestaciones, en el ámbito docente se hace difícil poder acceder a estos sistemas y se suele recurrir a equipos más modestos.

En la asignatura *Computación Paralela y Masiva* de cuarto curso del Grado en Ingeniería Informática de la Universidad Rovira i Virgili, objeto de este artículo, se utiliza un sistema de colas donde se organiza el acceso 24/7 (24 horas al día, 7 días a la semana) a servidores dedicados. De este modo, tanto en horas lectivas como en horas personales, siempre se tiene acceso a los servidores docentes.

Concretamente, en las prácticas se realizan ejercicios para máquinas de memoria compartida (MC) y para máquinas de memoria distribuida (MD). En el primer caso (MC) se programa con el estándar OpenMP y se utiliza una máquina con 2 procesadores Xeon E5-2660 a 2,2 GHz, disponiendo cada uno de ellos de 8 *cores* y con capacidad *multithread* (2 *threads*). De este modo, se pueden ejecutar hasta 32 procesos simultáneamente. En el segundo caso (MD) se estudia el modelo MPI de programación paralela y para realizar las ejecuciones se utiliza un sistema de 8 nodos con 2 procesadores Opteron 2210 a 1,8 GHz cada uno, disponiendo ambos de 2 *cores*. De este modo, se pueden ejecutar hasta 32 procesos simultáneamente.

En general, en entornos OpenMP con sistemas de memoria compartida, es relativamente fácil tener acceso a sistemas de 16-32 *cores* por un precio que puede ir de 5.000 € a 10.000 €. Del mismo modo, un entorno CUDA es también relativamente asequible. Una tarjeta gráfica de gama alta se puede adquirir por un precio de entre 1.000 € y 2.000 € y permite añadirla a un servidor para realizar los ejercicios de CUDA necesarios. Incluso podría utilizarse el mismo servidor para OpenMP y CUDA. En cambio, para entornos de programación MPI, al ser sistemas de memoria distribuida, permiten realizar mejores estudios sobre la escalabilidad de un problema, debido a que se tiene acceso a una cantidad de nodos, procesadores y *cores* bastante superior. Sin embargo, resulta demasiado caro adquirir sistemas que superen los 4 u 8 nodos y, por ello, los estudios de escalabilidad se quedan en el ámbito teórico.

Una solución para mejorar en escalabilidad, es usar salas (o laboratorios) de ordenadores y así conseguir un número de nodos superior a 30 ordenadores. Esta solución tiene un grave problema: la exclusividad. Si bien se podrían reservar durante varias horas, sin duda se deben liberar pasado ese tiempo para que otras asignaturas los puedan utilizar. Con esta restricción, los alumnos no pueden realizar sus ejercicios fuera de horas de clase, teniendo que reducir de manera importante la complejidad de los mismos. Por otro lado, la red de interconexión que utilizan las salas de

Manuscrito enviado para revisar el 1 de diciembre de 2019.
English version received December, first, 2019.
Carles Aliagas, Montse Garcia-Famoso, Pere Millán y Carlos Molina del Departamento d'Enginyeria Informàtica i Matemàtiques de la Universitat Rovira i Virgili en Tarragona. {carles.aliagas,montse.garcia,pere.millan,carlos.molina}@urv.cat
Roc Meseguer del Departamento de Arquitectura de Computadores en la Universidad Politècnica de Catalunya en Barcelona. meseguer@ac.upc.edu

ordenadores estaría compartida con otros sistemas del centro educativo y, sin duda, perjudicaría de manera importante la comunicación MPI de los ejercicios a ejecutar, haciendo muy complejo obtener tiempos de ejecución invariables según la carga de la red. Dada esta problemática, resulta poco fiable hacer estudios de *speedup* y escalabilidad a partir de ordenadores en red no dedicados y no exclusivos.

Nuestro objetivo es construir un sistema para un entorno de programación MPI con dedicación exclusiva, de más de 250 *cores* y que resulte relativamente económico (entre 5.000€ y 6.000€). Además, este recurso usará la actual tendencia de trabajar con sistemas con alta relación entre rendimiento y bajo consumo de energía. De esta forma, los alumnos, aparte de aprender el estándard MPI, deberán adaptarse a los recursos que les ofrezca el sistema en cuanto a potencia de cálculo, memoria RAM y red de comunicación, y además podrán realizar estudios de escalabilidad variando el número de procesos desde pocas unidades hasta centenares de ellos.

Este trabajo es una versión ampliada y revisada del artículo "Minimulticomputador de Bajo Coste" [1] publicado en las XXV Jornadas sobre la Enseñanza Universitaria de la Informática (JENUI 2019) que tuvieron lugar en Murcia (España) en julio de 2019. En particular, las principales diferencias son las siguientes:

- *Título*, *Abstract*, Sección I (*Introducción*) y Sección VIII (*Conclusiones*) se han mejorado de acuerdo con el nuevo material incluido.
- Se ha añadido la sección II de *Trabajos Relacionados* y se han incluido y analizado nuevas referencias bibliográficas
- La Sección III (*Propuesta:Hardware*) y la Sección IV (*Propuesta:Software*) se han reorganizado, mejorando la descripción de la propuesta, tanto en términos hardware como en términos software.
- La Sección V (*Propuesta:Verificación*) se ha reorganizado para centrarla en la verificación del sistema, añadiendo un mejor análisis del consumo, temperatura y rendimiento de la solución propuesta. Además, se ha considerado un sistema con 8 nodos en lugar del sistema de 4 nodos inicialmente utilizado para las máquinas de memoria distribuida, haciendo que los resultados y análisis mostrados se hayan actualizado.
- La sección VI (*Uso del Sistema en el Contexto de la Asignatura*) es totalmente nueva. En ella se muestra el uso del sistema propuesto en el contexto de una asignatura de programación paralela. De este modo, se describe el modelo evaluativo de la asignatura y se comprueba el rendimiento del sistema con algoritmos inicialmente secuenciales y que los alumnos paralelizan en sus prácticas. Estos algoritmos se han ejecutado y analizado en todos los sistemas considerados en el trabajo (Xeon, Opteron y Odroid).
- La sección VII (*Valor Añadido del Nuevo Sistema*) es también una sección totalmente nueva. En ella se describe el valor añadido (cuantitativa y cualitativamente) que supone, tanto a alumnos como a profesores, utilizar en la asignatura el sistema de computación de bajo coste propuesto en este trabajo.

II. Trabajos relacionados

Una de las primeras aproximaciones para implementar sistemas paralelos en entornos docentes consistió en la conexión de computadores personales mediante alguna red de área local a finales de los años 90. Chhabra y Singh [2] fueron de los primeros en aplicar este enfoque, proponiendo el uso combinado de los computadores personales disponibles en la intranet de los campus universitarios. A partir de allí, multitud de universidades siguieron esa misma tendencia en sus asignaturas de programación paralela. Por ejemplo, Apon *et al*. [3] describen en su trabajo varias experiencias de uso de *clusters* de computación, basados en PCs/*Workstations* en distintas universidades, dependiendo de los objetivos concretos de las asignaturas que en cada una de ellas se imparten (Arquitectura de Computadores, Procesamiento Paralelo, Sistemas Paralelos, etc). Incluso, se han propuesto enfoques con unos pocos computadores personales (apenas 4) y de muy bajo coste, como el que plantean Xu y Su [4] para la Universidad de Shanghai de Ciencias de la Ingeniería. En general, todos estos sistemas están enfocados a trabajar el procesamiento paralelo en entornos de memoria distribuida, pero en algunos casos también es aplicable a entornos de memoria compartida.

Una década después, este tipo de entornos es el que aún se acostumbra a utilizar en ámbitos docentes de asignaturas que introducen a los alumnos en la programación paralela, usando ahora placas GPU (*Graphics Processing Units*) junto a procesadores comerciales de propósito general. Así pues, Giménez [5] describe su experiencia en el curso “Introducción a la Programación Paralela” del Departamento de Informática y Sistemas de la Universidad de Murcia, que se centra en entornos y herramientas de programación paralela como OpenMP, MPI y CUDA. Para ello utiliza un *cluster* de 6 nodos con un total de 64 *cores* y con tarjetas GPU y Xeon Phi para cada nodo (10 GPUs y 2 Xeon Phi). Por su parte, Santamaría *et al*. [6] en el curso de Arquitectura de Computadores de la Ingeniería de Telecomunicación de la Universidad de Jaén, utilizan ordenadores con placas GPU NVIDIA GeForce 8800 GT, así como el *toolkit* CUDA 2.3, pero la experiencia es aplicable a otras alternativas de paralelización (MPI, OpenMP).

Una alternativa atractiva, en contrapartida a los entornos físicos, son los sistemas *clusters* implementados en la nube. En este sentido, Ivica *et al*. [7] presentan StarHPC, un entorno desarrollado por el MIT para el aprendizaje de algoritmos y programación paralela, que proporciona una imagen de máquina virtual configurada para la programación paralela en tecnologías OpenMP y OpenMPI. Adicionalmente, en el MIT también se ha desarrollado el proyecto StarCluster [8] que ofrece un *cluster* dinámico alojado en el *Elastic Cloud Computing* (EC2) de Amazon y que permite el uso de aplicaciones y sistemas informáticos distribuidos y paralelos. Del mismo modo, Gómez-Folgar *et al*. [9] proponen utilizar *cloud computing* como base para el aprendizaje de MPI, en este caso basado en *CloudStack*. La ventaja de este tipo de sistemas es que ofrecen una gran flexibilidad y una alta disponibilidad. Además, las

instituciones educativas pueden reducir costes, ya que eliminan la necesidad de disponer de servidores, sistemas de almacenamiento y de refrigeración dedicados, al margen de minimizar las tareas de los administradores de sistemas.

La utilización de sistemas de bajo coste para mejorar algún aspecto de las aulas o laboratorios de las universidades que ofertan algún tipo de ingeniería, es una práctica cada vez más habitual. Y es que, ante la falta de financiación en las universidades, el personal docente acostumbra a agudizar el ingenio. En este sentido, y sin pretender ser exhaustivos, podemos encontrar multitud de ejemplos que siguen este enfoque. De este modo, Asín y Pastor [10] del Departamento de Electrónica de la Universidad de Alcalá, desarrollan una plataforma robótica que facilita el aprendizaje sin necesidad de laboratorios presenciales, mediante el uso de dispositivos de bajo coste y con placas desarrolladas en el propio departamento. Por otro lado, Rodríguez *et al*. [11] utilizan placas Raspberry Pi, en las aulas docentes de la Universidad Nacional de La Matanza (Buenos Aires), para implantar un sistema de puntos de acceso de red. De este modo, los alumnos (sin necesidad de acceso a Internet) podrán responder a cuestionarios de la asignatura desde sus dispositivos móviles. Más centrados en soluciones para las asignaturas de Ingeniería Informática, Catalán *et al*. [12] en el Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza, proponen un modelo de programación para Raspberry Pi. Para ello, conectan un ordenador de sobremesa y dos placas Raspberry Pi a través de Internet, para mostrar el funcionamiento de esas placas. Por otro lado, Ortega *et al*. [13] del Departamento de Informática de la Universidad de Almería, utilizan placas Raspberry Pi y placas Arduino para demostrar el potencial de este tipo de sistemas de bajo coste y, sobretodo, para motivar y despertar el interés de los alumnos. Así pues, proponen experimentos como activar un relé con una llamada telefónica, construir un piano con frutas o incluso montar un *cluster* con 4 Raspberry Pi. Más recientemente, Johnston *et al*. [14] analizan el potencial de los *clusters* basados en SBCs (*Single Board Computers*) como alternativa a los *clusters* tradicionales en términos de coste y consumo. En cualquier caso, estas propuestas no están encaminadas a desarrollar o analizar ningún tipo de aplicación basada en programación paralela.

Si nos centramos en la programación paralela MPI y el uso de sistemas *low-cost*, podemos encontrar varias propuestas. Pfalzgraf y Driscoll [15], con el objetivo de disponer de un *cluster* de bajo coste (menos de 1.000 US$) para la formación en computación de altas prestaciones (HPC, *High Performance Computing*), construyen un sistema de 25 nodos (Raspberry Pi Model B) interconectados mediante un *switch* Ethernet. El sistema se gestiona mediante software libre (*open source*), lo cual permite mantener un coste final asequible. Para probar el sistema utilizan 3 códigos Python+MPI con operaciones básicas de álgebra lineal, variando el tamaño n del problema (tamaño de los vectores, matrices). Los resultados presentan un comportamiento muy similar a sistemas HPC de coste mucho más elevado. De forma similar, Rachmawan *et al*. [16] usan un *cluster* de bajo coste formado por 32 placas PandaBoard ES, 2 *switches* y 2 fuentes de alimentación adaptadas, con librería de comunicación OpenMPI. La novedad en este caso reside en que los estudiantes también participan en el diseño y montaje del hardware, y no sólo en la programación. La formación HPC se organiza en 7 cursos y las encuestas realizadas por los alumnos muestran las preferencias de éstos por la formación práctica que es posible usando este *cluster*. Un punto problemático de estos sistemas de bajo coste para la formación HPC, y que no queda reflejado en la mayoría de artículos, es la configuración de los diferentes componentes hardware y software. Doucet y Zhang [17] exponen detalladamente el montaje y configuración del hardware, software y comunicaciones de un *cluster* educativo con 20 Raspberry Pi.

Por otra parte, múltiples artículos técnicos como Cloutier *et al*. [18], Aroca y Garcia [19] y Göddeke *et al*. [20], no asociados directamente a la experiencia docente sino desde el punto de vista de rendimiento paralelo, han estudiado la viabilidad de montaje de un *cluster* usando sistemas SoC (*System on Chip*). Los artículos se centran en la comparación de rendimiento, consumo energético y coste, pero al basarse el estudio mayoritariamente en procesadores ARM de 32 bits, que tienen poca memoria (de 256 KB a 1 GB), el rendimiento que proporcionan es bastante bajo. Por mucho que tengan un coste bajo y consuman poca energía, al establecer una relación rendimiento/coste y rendimiento/energía, tal como hacen Cloutier *et al*. [18], no llegan a superar la relación que ofrecen sistemas clásicos basados en procesadores de alto rendimiento tipo Xeon y Opteron. Aroca y Garcia [19] concluyen, por su parte, que los sistemas ARM son eficientes a nivel energético cuando ejecutan aplicaciones basadas en servicios de Base de Datos como MySQL, pero que no lo consiguen con aplicaciones que requieren cálculos en coma flotante. Por último, Göddeke *et al*. [20] comparan la ejecución de aplicaciones HPC en dos *clusters*, uno con 32 nodos con procesadores Xeon X5550 y otro con 96 nodos con procesadores ARM Cortex-A9, y concluyen que, a pesar del bajo consumo de los procesadores ARM, al tener menos potencia de cálculo requieren más tiempo de ejecución, no siendo entonces competitivos a nivel energético, comparado con los procesadores Xeon. Además, concluyen que sería necesaria más del doble de potencia de cálculo en coma flotante para empezar a ser competitivos en la relación rendimiento/energía.

En este trabajo utilizamos procesadores ARM Cortex-A53 con más potencia de cálculo y, yendo un paso más allá, queremos construir un sistema de varias decenas de nodos, que permita a los alumnos analizar la escalabilidad de los entornos de desarrollo MPI, a un precio razonable. No nos planteábamos el montaje del sistema *cluster* SoC hasta que el mercado no ofreció sistemas con una potencia, RAM y conectividad suficientes para obtener una mejor relación rendimiento/coste. Para ello, aprovechamos los desarrollos más recientes de placas de bajo coste y con mayores prestaciones que las utilizadas en los trabajos relacionados. En nuestro caso, como se verá en la siguiente sección,

usamos placas Odroid más potentes, con el doble de RAM (2 GB), con memoria eMMC (más rápida que las tarjetas SD de otros trabajos) y conexiones Gigabit Ethernet (10 veces más rápidas que la Fast Ethernet de Raspberry), conectados a 2 *switches* Gigabit. Los 4 *cores* de cada Odroid (más potentes que los de las Raspberry Pi) permiten a los alumnos realizar pruebas de escalabilidad con hasta 256 procesos, lo cual no es posible en los trabajos relacionados de carácter docente.

III. Propuesta: Hardware

A. Alternativas

Tal y como ya se ha comentado, nuestra propuesta se centra en sistemas de procesadores de bajo consumo y con una capacidad de cálculo razonable, que puedan ejecutar un sistema operativo (SO) completo desde el primer momento. Si bien lo anterior incluye ordenadores personales, éstos se descartan porque se quiere un sistema que pueda ser fácilmente aglutinado con otros, que ocupe el mínimo espacio posible y que tenga un consumo de energía reducido. Del mismo modo, se descartan sistemas empotrados/embebidos porque no permiten ejecutar fácilmente un SO Linux completo. Por lo tanto, se decide analizar sistemas SoC que tengan características parecidas a los basados en Raspberry Pi [21].

A continuación se describen las diferentes alternativas consideradas (los detalles adicionales se pueden consultar en el cuadro I) para seleccionar un sistema completo de procesador de bajo coste:

- Raspberry Pi-3B: es la tercera revision del diseño original y el más popular en sistemas SoC. Merece su estudio por tener un diseño claro y altas prestaciones. Se descarta por no disponer de Gigabit Ethernet y por su baja capacidad de memoria RAM.
- Banana Pi-M2-Berry: alternativa equivalente, que destaca por tener Gigabit Ethernet, pero con menos potencia de cálculo.
- Banana Pi-M3: opción superior a M2-Berry, con el doble de *cores* y memoria, almacenamiento eMMC más rápido, con un precio más elevado, pero no suficientemente potente.
- Orange Pi-plus-2E: con un precio razonable, destaca por mejorar en memoria, almacenamiento y comunicación, pero tiene poca potencia de cálculo.
- Odroid-XU4: es el sistema más potente, con 8 *cores* (4 Cortex A15 2,1 GHz + 4 *cores* A7 1,5 GHz) pero tiene un precio elevado.
- Odroid-C1+: sistema que compite directamente con Raspberry Pi-3B. Es una buena opción, pero el mismo fabricante ofrece un modelo superior más interesante y potente.
- Odroid-C2 [22]: este sistema aglutina todas las ventajas de los sistemas anteriores a un precio razonable. Mejora en memoria, comunicación, almacenamiento y potencia de cálculo.

B. Placa seleccionada: Odroid-C2

La opción seleccionada ha sido: Odroid-C2. En este caso esta placa cumple con lo mínimo que considerábamos para poder montar el sistema: Procesador ARM con suficiente rendimiento (4 *cores* por placa), suficiente memoria RAM (2 GB) y ancho de banda bastante aceptable (Gigabit Ethernet). Para ver las prestaciones de la placa elegida, la Figura 1 muestra una comparativa de ella con diversas alternativas: una placa más barata (Odroid-C1+), una placa más cara (Odroid-XU4) y una placa muy popular en el mercado (Raspberry Pi-3B). Estos valores se han obtenido de la página web del fabricante HardKernel® [22] y muestran el rendimiento de la CPU (Figura 1a), la velocidad de transferencia del acceso al almacenamiento (Figura 1b) y la velocidad de transferencia de las comunicaciones (Figura 1c).

Como se puede observar, las prestaciones de la placa escogida sobresalen del resto, a excepción de la placa XU4 que tiene un rendimiento superior, pero que tiene un coste económico de más del doble. La placa seleccionada tiene un precio de 46 US$ con una relación rendimiento/precio muy buena. Sus características son:

- Amlogic ARM® Cortex®-A53 (ARMv8) 1.5 GHz quad core CPUs
- Mali™-450 GPU (3 Pixel-processors + 2 Vertex shader processors)
- 2 GByte DDR3 SDRAM
- Gigabit Ethernet
- HDMI 2.0 4K/60Hz display
- H.265 4K/60FPS and H.264 4K/30FPS VPU
- 40 pin GPIOs + 7 pin I2S
- eMMC5.0 HS400 Flash Storage slot / UHS-1 SDR50 MicroSD Card slot
- 4 USB 2.0 Host, 1 USB OTG (power+data)
- Infrared (IR) Receiver
- Ubuntu 16.04 or Android 6.0 Marshmallow based on Kernel 3.14LTS

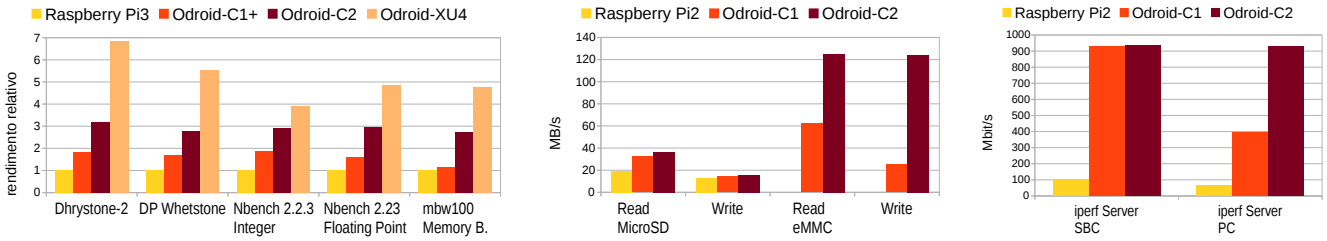
De entre todas sus características, destacamos la potencia de cálculo del ARMv8 (3 veces más potente que Raspberry Pi3), los 2 GB de RAM (2 veces superior), la velocidad en el acceso al almacenamiento en eMMC (120 MB/s) y la velocidad de transferencia en red de la Gigabit-Ethernet (900 Mbit/s). Quizás, el único inconveniente de esta placa es su falta de conectividad inalámbrica de serie, ya que viene sin WiFi ni Bluetooth, aunque es posible añadirle complementos opcionales a través de sus puertos USB que cumplan esa función. En cualquier caso, nuestra intención es realizar la conexión de todas las placas mediante cable.

C. Red de comunicación

La red de interconexión es uno de los factores más importantes que definen el rendimiento de los algoritmos paralelos de memoria distribuida. En este sistema no se ha optado por una solución de alto rendimiento, sino que se ha limitado a ofrecer una solución que obtenga el máximo rendimiento de las tarjetas de comunicación de las placas. Aunque la comunicación mediante USB puede ofrecer buenos rendimientos, éstos están limitados en nuestro caso al

Cuadro I: comparativa SoC. Características técnicas relevantes

Placa	Procesador	ARM	*Cores*	Frecuencia	MEM	Disco	Ethernet
Raspberry Pi-3B	Cortex-A53	v8	4	1,2 GHz	1 GB DDR2	MicroSD	Fast
Banana Pi-M2-Berry	Cortex-A7	v7	4	1,0 GHz	1 GB DDR3	MicroSD	Gigabit
Banana Pi-M3	Cortex-A7	v7	8	1,8 GHz	2 GB DDR3	eMMC	Gigabit
Orange Pi-plus-2E	Cortex-A7	v7	4	1,5 GHz	2 GB DDR3	eMMC	Gigabit
Odroid-XU4	Cortex-A15/A7	v8/v7	4/4	2/1,5 GHz	2 GB DDR3	eMMC	Gigabit
Odroid-C1+	Cortex-A5	v7	4	1,5 GHz	1 GB DDR3	eMMC	Gigabit
Odroid-C2	Cortex-A53	v8	4	1,5 GHz	2 GB DDR3	eMMC	Gigabit



(a) rendimiento CPU (b) velocidad almacenamiento (c) velocidad Ethernet

Figura 1: comparativa placas SoC (*fuente del fabricante HardKernel*®)

estándar USB 2.0 cuya velocidad máxima es de 480 Mbps (siendo la velocidad efectiva de un dispositivo Gigabit-Ethernet USB 2.0 no superior a 300 Mbps). Dada esta limitación, se ha optado por comunicar las placas mediante la Gigabit-Ethernet incorporada, obteniendo así unas velocidades efectivas cercanas a los 900 Mbps.

Las placas se han interconectado mediante conmutadores dedicados. En este caso, la forman dos conmutadores de la marca Zyxel modelo GS1900-48, que es una solución suficiente y económica. Sus principales características son:

- Switching capacity (Gbps): 100
- Forwarding rate (Mpps): 74
- Packet buffer (byte): 1.5 M
- MAC address table: 8 K
- Jumbo frame (byte): 9 K

D. Ensamblaje de placas

El multicomputador que se propone consta de: 64 Odroid-C2 con 8 GB de almacenamiento eMMC cada uno, 8 fuentes de alimentación con 10 salidas USB cada una de las fuentes, 2 conmutadores Zyxel y cables Ethernet categoría 5E. Al disponer cada placa Odroid de 4 *cores*, hace que se disponga de un sistema de 256 *cores* en total.

El montaje del multicomputador consistió en unir físicamente un Odroid al lado del otro y conectar cada uno de los Odroid a un puerto de un conmutador (Figura 2). Como los conmutadores son de 48 puertos, se han usado 32 en cada uno de ellos para conectar los Odroid y un cable adicional para conectar los conmutadores entre si. Por otra parte, se ha usado un Odroid adicional como nodo gestor para hacer de *frontend* del multicomputador. Este Odroid permitirá el lanzamiento de aplicaciones paralelas en los nodos de computación y ofrecerá acceso de red a los mismos. Este nodo tiene los mismos componentes que los otros, pero añadiendo teclado, ratón, conexión HDMI a pantalla y Gibagit Ethernet por USB para disponer de acceso al multicomputador desde el exterior.

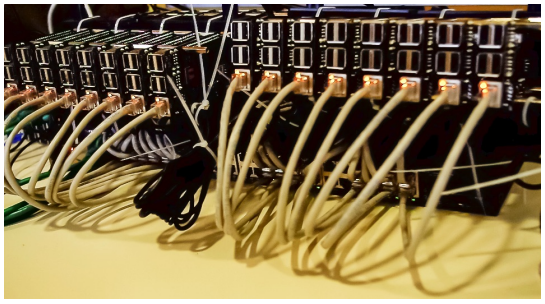


Figura 2: placas Odroid-C2 en funcionamiento

IV. Propuesta: Software

A. Sistema Operativo y servicios del nodo gestor

Para que el multicomputador propuesto ejecute aplicaciones paralelas, se instala el SO Linux proporcionado por el fabricante (Ubuntu 16.4) grabable directamente en tarjetas de memoria eMMC o MicroSD. Esto nos permite utilizar un gestor de paquetes y, de esta forma, mantener actualizado el software e instalar los paquetes necesarios de manera muy sencilla [23].

El sistema Linux instalado viene con el software estándar de Ubuntu para máquinas de sobremesa. Esto implica un gestor de arranque de servicios, un gestor de ventanas, etc. Se han añadido los siguientes servicios: servidor SSH (`openSSH`), servidor de nombres y servidor DHCP (`dnsmasq`), replicación de puertos, *tunneling* y *firewall* (`iptables`), servidor de ficheros en red (NFS).

Se instala también el paquete OpenMPI para así poder ejecutar programas paralelos. Simplemente se usan los comandos de OpenMPI: `mpicc` y `mpirun`, el primero para compilar y el segundo para ejecutar. Este último tiene una opción que permite indicar las direcciones de los nodos que formarán parte de la ejecución paralela. Así, se crea un fichero de texto con los 64 nombres DNS internos de los nodos.

B. Sistema Operativo y servicios de los nodos de computación

En los nodos también se ha instalado el SO Linux Ubuntu 16.04 proporcionado por el fabricante, pero en este caso se han eliminado la mayoría de paquetes que no aportan nada en la ejecución paralela de aplicaciones, quedando así una instalación mínima que consume pocos recursos del sistema. Los servicios que se han mantenido han sido: servicio de acceso remoto (SSH), cliente de acceso a disco remoto (NFS) y el paquete de ejecución MPI (OpenMPI). Cualquier paquete que suponga una ocupación considerable de espacio en disco se ha desinstalado, como por ejemplo el escritorio Gnome.

Para poder administrar los nodos se ha permitido que los usuarios con privilegios (*root* y *sudoers*) puedan ejecutar comandos en los nodos desde el *frontend* sin necesidad de introducir cada vez la contraseña del usuario privilegiado. Esto se ha realizado mediante la generación de una clave RSA (`ssh-keygen`) y la posterior distribución de la clave pública a los nodos para que se pueda acceder sin solicitar la contraseña (`ssh-copy-id`).

Por otra parte, se ha añadido a los nodos una entrada en el montaje del sistema de ficheros (`fstab`) para que, mediante el servicio NFS, puedan acceder al directorio de usuarios del nodo *frontend* (`/home`) y así compartirlo entre todos ellos. De este modo, en la ejecución paralela se ahorra el coste de transmisión de los códigos MPI binarios compilados. Esta transferencia se realiza por NFS y se aprovecha dicho protocolo para optimizar la distribución del código.

V. Propuesta: Verificación

A. Temperatura

Para verificar el correcto funcionamiento del sistema y que la refrigeración pasiva es suficiente, se ejecutaron pruebas de estrés con el paquete `stress`. Se comprobó que la temperatura de la CPU con los nodos inactivos estaba alrededor de 41 ºC y que después de un periodo de estrés de las CPUs, las temperaturas se estabilizaban alrededor de 60 ºC (algunos nodos hacia 56 ºC y otros hacia 62 ºC).

Dada la distribución física de las placas, se observa que si están apiladas horizontalmente, el calor de las inferiores perjudica a las superiores, llegando a temperaturas superiores a 75 ºC. Aunque este valor no supone ningún problema en el sistema (el límite del fabricante está por encima de 80 ºC) se observa que girando 90º la pila de Odroids y quedando en una distribución vertical (como en la Figura 2) se reducen drásticamente las temperaturas, estabilizándose alrededor de 60 ºC. Sin duda, se debe a que el calor generado se disipa por convección y no afecta tanto a las placas vecinas.

En la Figura 3 se observa una placa exterior, donde la temperatura máxima en estrés se mantiene en 55,7 ºC.

B. Consumo

Para verificar el consumo del nuevo sistema de Memoria Distribuida Odroid (MD-Odroid), se realiza una comparativa con el resto de sistemas hasta ahora existentes: el sistema de Memoria Distribuida Opteron (MD-Opteron) y el sistema de Memoria Compartida Xeon (MC-Xeon). De este modo, se miden los consumos de energía a partir del amperímetro que tiene una fuente de alimentación de un SAI (Sistema de Alimentación Ininterrumpida). Para ello se consideran dos modos de funcionamiento: *idle* y *full*. El primer caso sería el consumo de los 3 servidores en reposo, mientras que el segundo caso sería el consumo de los 3 servidores a pleno rendimiento al ejecutar *stress* al 100 % de CPU en cada uno de esos tres sistemas.

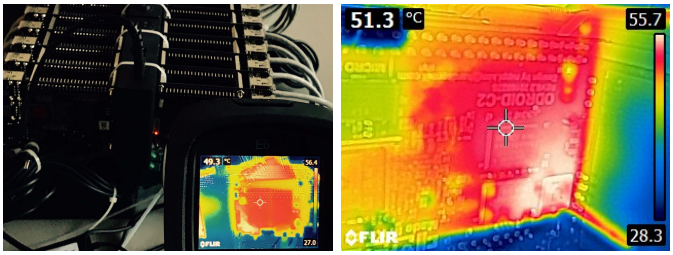


Figura 3: temperatura Odroid-C2 en pruebas de estrés

Cuadro II: comparativa consumo energía servidores

	1 nodo		*switch*	total sistema	
Servidor	***idle***	***full***		***idle***	***full***
MC-Xeon	-	-	-	110 W	418 W
MD-Opteron	144 W	180 W	20 W	1.172 W	1.460 W
MD-Odroid	2,5 W	4 W	40 W	201 W	303 W

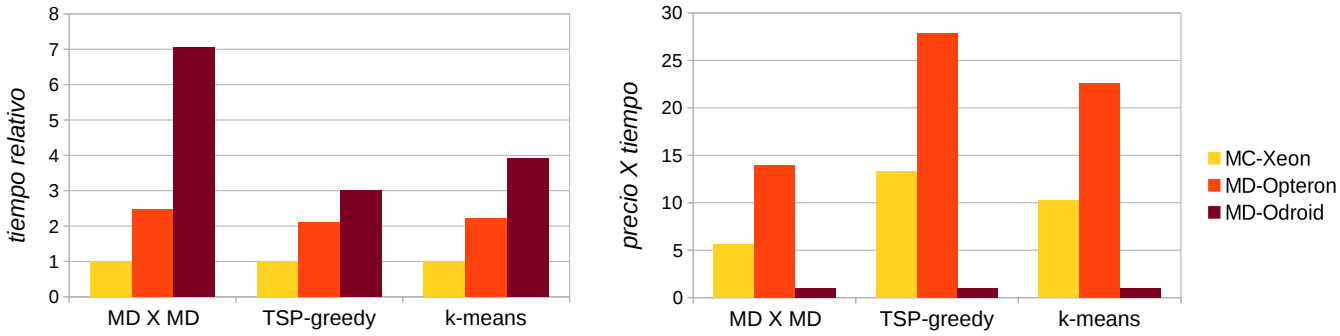
En el cuadro II se detalla el consumo de cada uno de los 3 servidores, considerando los dos modos de funcionamiento anteriormente descritos. Concretamente, se muestran los resultados individuales de uno de los nodos de cada sistema, y los resultados globales considerando todos los nodos del sistema. Además, y para ser más justos con la comparativa, se ha incluido el consumo que introducen los diferentes *switches* que necesitan cada servidor (2 *switches* el sistema MD-Odroid, 1 *switch* el sistema MD-Opteron y ninguno en el caso del sistema MC-Xeon)

Como se puede apreciar, el servidor MD-Odroid es el que tiene un consumo energético más bajo cuando se encuentra a pleno rendimiento, representado tan sólo el 72 % del consumo del servidor MC-Xeon y el 20 % del consumo total del servidor MD-Opteron. De hecho, como sólo tendrán un incremento de consumo aquellos nodos que se estén usando (el resto seguirán en *idle*), la opción de los sistemas MD-Odroid ofrecerá siempre una excelente relación consumo/rendimiento. Este análisis nos reafirma en la viabilidad de este nuevo sistema en términos de consumo.

C. Selección de códigos MPI

Para la verificación del sistema en términos de rendimiento, se seleccionan tres tipos diferenciados de códigos: (1) producto de matrices dispersas, (2) búsqueda de un camino óptimo del viajante de comercio y (3) cálculo de agrupamientos basado en el algoritmo *k-means*.

El primer problema calcula dos productos de matrices (Figura 4, MD×MD): (1) matriz densa por matriz dispersa y (2) el producto de dos matrices dispersas. El ejercicio no presenta muchos problemas de paralelismo, ya que basta con



(a) tiempo relativo

(b) coste: precio × tiempo

Figura 4: ejecución secuencial

repartir fragmentos de las matrices entre los procesos, para finalmente recoger los resultados y compactarlos en la matriz resultante. El tamaño del problema se ha dimensionado a matrices dispersas de 5.000×5.000 elementos.

El segundo problema es el "viajante de comercio" (Figura 4, *TSP-greedy*) que debe visitar una serie de puntos unidos por caminos, de manera que los visite todos una sola vez, con el mínimo recorrido posible. En este caso se obtiene una solución suficientemente buena, pero no la solución óptima. Se basa en una selección *greedy* de todos los posibles inicios. La solución paralela pasa por repartir los posibles inicios y al final escoger el camino mínimo que haya calculado uno de los procesos. El número de sitios a visitar para este ejemplo es de 3.000.

Por último, el algoritmo k-*means* (Figura 4, k-*means*) trata de crear agrupaciones de elementos generados aleatoriamente. Busca de forma iterativa k-centroides de los elementos a tratar, hasta que cada uno de ellos se estabilice y quede asignado a un centroide. Los centroides son calculados a partir de la media de los elementos que le pertenecen. En este caso hay una recurrencia en cada iteración, que obliga a sincronizar los procesos para actualizar correctamente los centroides. En este ejercicio se han buscado 200 centroides a partir de 400.000 elementos.

D. Rendimiento y coste secuencial

Para verificar el tiempo de ejecución secuencial, en la Figura 4a se muestra dicho tiempo para un mismo algoritmo, pero usando únicamente un procesador de cada sistema. La máquina con procesadores Xeon es la más rápida, siendo hasta 2,5 veces más rápida que la máquina con procesadores Opteron y hasta 7 veces más rápida que un procesador ARM de una placa Odroid-C2.

Sin embargo, cuando tenemos en cuenta el precio de un sistema con un procesador de cada tipo, el coste relativo de la ejecución de estos ejercicios cambia significativamente, siendo un Odroid-C2 unas 10 veces mejor que un sistema con procesador Xeon, y unas 20 veces mejor que un sistema con procesador Opteron. La Figura 4b muestra el coste, calculado como el tiempo de ejecución multiplicado por el precio[1] unitario con un sistema monoprocesador, que es equivalente al coste calculado como precio dividido por el rendimiento (tal como se verá en las figuras de la siguiente sección).

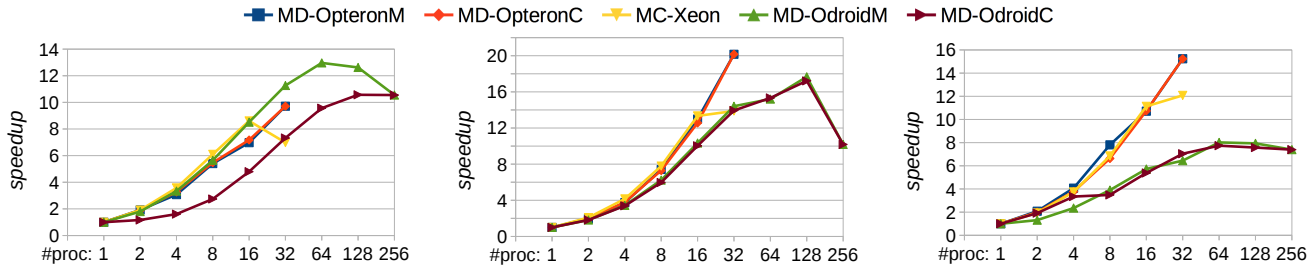
[1]Se asume un precio de 2.000 € para los sistemas con procesador Xeon y Opteron. Por contra, el precio de una placa Odroid-C2 es de unos 50 €.

E. Rendimiento y coste paralelo

Los códigos anteriores se han paralelizado con el estándar MPI y se han ejecutado mediante un sistema de colas Slurm (para garantizar exclusividad) en las diferentes máquinas disponibles para la asignatura. Mostramos únicamente aquellas que nos sirven para evaluar las diferencias con MPI. Una es un servidor de memoria compartida con dos procesadores Intel Xeon E5-2660 a 2,2 GHz y 32 GB de RAM (MC-Xeon). Otra son 8 servidores formando un *cluster* de memoria distribuida, con 2 procesadores Opteron 2210 a 1,8 GHz y 4 GB de RAM cada uno (MD-Opteron). Y, por último, el sistema objeto de este trabajo, que lo forman 64 placas Odroid-C2 con procesador ARM Cortex A-53 a 1,5 GHz y 2 GB de RAM cada una (MD-Odroid).

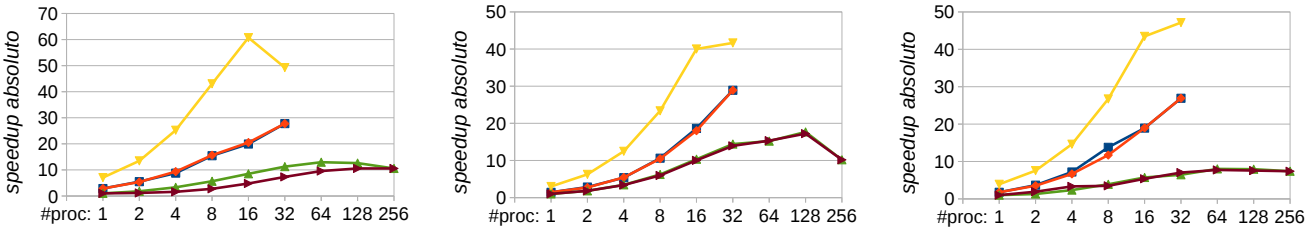
La Figura 5 muestra la evolución del *speedup* a medida que se crean más procesos en la ejecución paralela. Para MD-Opteron y MD-Odroid se presentan dos ejecuciones con mapeos de procesos diferentes. MD-OpteronM y MD-OdroidM equivale a un mapeo pensando en problemas que sean *Memory-Bound*, donde se producen muchos accesos a memoria por parte del algoritmo y se intenta que un proceso tenga toda la memoria de un nodo para él solo. Así, para MD-OpteronM una ejecución de 8 procesos asignará un proceso en cada nodo, ocupando de esta forma los 8 nodos. En MD-OdroidM cada placa ejecuta un único proceso, desde la ejecución de 2 procesos hasta 64 procesos (para 128 y 256 ya se asignan 2 y 4 procesos por placa). MD-OpteronC y MD-OdroidC equivale a un mapeo pensado para problemas *CPU-Bound*, donde no se hacen demasiados accesos a memoria y el cuello de botella se debe al uso de la CPU. En este caso se intenta asignar el máximo de procesos a cada nodo, de forma que la posible comunicación entre los procesos de un nodo se haga mediante memoria compartida, sin necesidad de utilizar la red de interconexión. Por ejemplo, para 4 procesos sólo se utilizará un nodo y, a medida que se incrementa el número de procesos, se van utilizando más nodos.

En MD×MD (Figura 5a) las máquinas tienen una evolución similar, pero sin obtener un *speedup* elevado. La



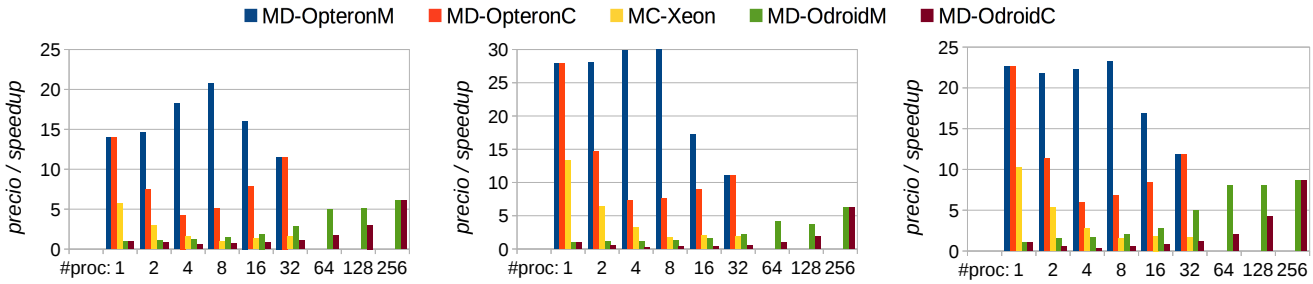
(a) MD × MD (b) TSP-*greedy* (c) *k-means*

Figura 5: evolución del *speedup* y escalabilidad



(a) MD × MD (b) TSP-*greedy* (c) *k-means*

Figura 6: *speedup* absoluto



(a) MD × MD (b) TSP-*greedy* (c) *k-means*

Figura 7: precio / *speedup* absoluto

reducción de la eficiencia se debe sobretodo a que es un algoritmo *Memory-Bound* y a la necesidad de comunicar los resultados de la multiplicación al nodo recolector. Destaca la pérdida de rendimiento en 128 y 256 del sistema MD-OdroidM y el poco rendimiento obtenido en la versión MD-OdroidC, debido a la competición por la memoria que se produce en cada placa por el hecho de tener 4 procesos asignados.

Con el algoritmo TSP-*greedy* (Figura 5b) vemos una evolución similar pero con *speedup* más eficiente. Este algoritmo es de tipo *CPU-Bound* y tiene un mayor grado de paralelismo. Sólo es necesaria la comunicación al final de la ejecución, para decidir qué proceso ha encontrado el mejor camino y, entonces, transmitirlo al nodo recolector. Los dos tipos de mapeo obtienen resultados equivalentes.

Por último, observamos el algoritmo *k-means* (Figura 5c) que se comporta de una forma similar en MC-Xeon y MD-Opteron(MC), pero tiene una pérdida de rendimiento importante en los sistemas MD-Odroid(MC). En este caso se necesita una sincronización en cada iteración para recolectar y distribuir los nuevos centroides. Aquí se aprecia cómo el hecho de tener multiprocesadores en MD-OpteronC y MC-Xeon, donde algunos procesos comparten memoria, la comunicación entre ellos no utiliza la red. Sin embargo, los MD-Odroid(MC) tienen que usar más la red de interconexión para comunicar los centroides, resultando en una pérdida de rendimiento importante. Destaca el beneficio inicial (hasta 4 procesos) de MD-OdroidC sobre MD-OdroidM ya que la comunicación en este caso también es interna al nodo, sin necesidad de utilizar la red de interconexión.

En cualquier caso, se puede observar que, al trabajar con más nodos, salen a relucir de forma evidente los problemas y retardos producidos por la necesaria comunicación entre los procesos de un algoritmo paralelo. Este comportamiento es beneficioso, desde el punto de vista educativo, porque permite una experiencia más realista a los alumnos.

En la Figura 6 se puede observar el *speedup* absoluto de las tres máquinas (tomando como base la más lenta). Se hace evidente que la máquina de memoria compartida MC-Xeon es la que tiene mejor rendimiento, dada la potencia de cálculo de cada *core* del sistema, seguida por el sistema de memoria distribuida MC-Opteron y finalmente por el sistema MC-Odroid. En cualquier caso, cuando añadimos el precio a la función de coste (Figura 7), calculando el coste en dinero

relativo al rendimiento absoluto que ofrece cada sistema, los resultados cambian drásticamente.

Como se ha visto, el sistema MC-Xeon es el que tiene mejor rendimiento, debido a su potencia de cálculo y comunicación interna, presentando así una relación precio/rendimiento muy buena, sobretodo para ejecuciones de 8 a 16 procesos. Los sistemas MD-Opteron, en cualquier configuración y mapeo, suponen el mayor gasto por rendimiento de los tres. No hay ninguna combinación que indique, respecto a los demás, que sea una opción óptima cuando se tiene en cuenta la relación precio/rendimiento. Para mapeos *Memory-Bound* (MC-OpteronM) la situación es aún peor, porque el precio del sistema para ejecutar 8 procesos implica el coste de los 8 nodos.

Por último, cabe destacar que los sistemas MC-Odroid (MC) son los que ofrecen la mejor relación precio/rendimiento. El motivo es sobretodo por su bajo coste. El hecho de ir incrementando el coste, añadiendo placas cada vez que se quieren ejecutar más procesos, hace que en cualquier configuración y mapeo ganen al resto por una amplia diferencia. Sólo para mapeos de 4 procesos por nodo y ejecuciones de 64, 128 y 256 procesos, tienen una relación precio/rendimiento equivalente a los otros dos, pero en el resto son la mejor opción. De los diferentes tipos de mapeo, MC-OdroidC obtiene peores rendimientos para algoritmos *Memory-Bound* (Figura 7a) que la opción MC-OdroidM. Pero, a pesar de ello, MD-OdroidC sigue obteniendo mejor relación precio/rendimiento que la otra, ya que aprovecha al máximo los *cores* de cada placa.

VI. Uso del sistema en el contexto de la asignatura

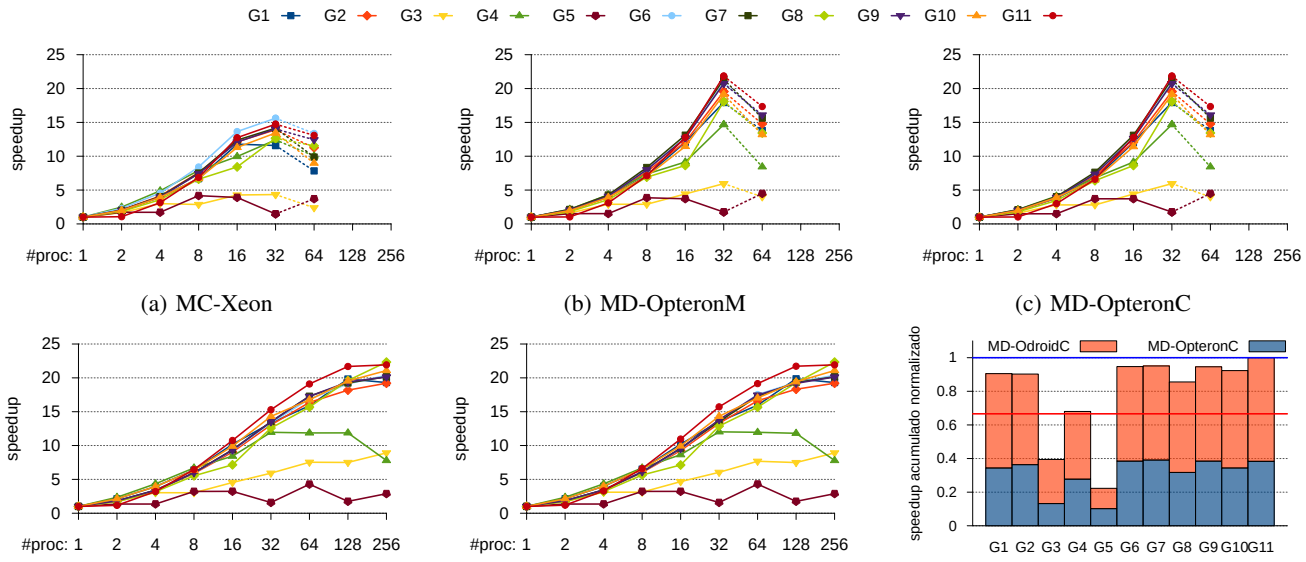
El sistema propuesto ha sido introducido, validado y analizado en el contexto de la asignatura *Computación Paralela y Masiva* del cuarto curso del Grado en Ingeniería Informática de la Universidad Rovira i Virgili. En esta asignatura se estudian las arquitecturas multiprocesador de memoria compartida y distribuida (MMC, MMD), profundizando en las técnicas específicas para su programación eficiente y paralela. En relación a la evaluación de la asignatura, se distinguen dos partes claramente diferenciadas, con el mismo peso sobre la nota final de la asignatura, y que además deben aprobarse por separado: parte teórica y parte práctica. La parte teórica consiste en un par de exámenes para evaluar los conocimientos teóricos adquiridos y, por otro lado, la parte práctica consiste en la paralelización de una serie de códigos secuenciales y su posterior ejecución para evaluar su *speedup*. Por tanto, la parte práctica tiene como objetivo consolidar los conceptos teóricos de programación paralela.

En relación a la parte práctica, en cada curso académico se seleccionan dos problemas distintos que planteen de manera creciente un grado relevante de dificultad, siendo el primer problema más asequible que el segundo. Cada uno de estos dos ejercicios deben solucionarse con los estándares OpenMP (memoria compartida) y MPI (memoria distribuida). La resolución de esos dos ejercicios con los dos modelos de memoria posibles implica cuatro entregas diferenciadas a lo largo del cuatrimestre en el que se imparte la asignatura. Estas entregas tienen el mismo peso sobre la nota final de la parte práctica (25 % cada una de ellas) pero tienen el requisito de obtener un *speedup* mínimo (marcado por el profesor de la asignatura). Por lo tanto, sólo una vez alcanzado este *speedup* mínimo se aceptarán como soluciones a ser evaluadas. En caso contrario, la parte práctica constará como no superada. Es también importante destacar que la resolución de los ejercicios prácticos se realiza en grupos de dos personas.

La evaluación de cada uno de los 4 ejercicios prácticos se divide en dos fases. En la primera fase se genera un *ranking* global obtenido a partir de la comparación de la práctica de cada grupo con el resto de grupos. En la segunda fase, se realiza una entrevista personal a todos los miembros del grupo, donde se discute la solución propuesta y el conocimiento adquirido a partir de la resolución del ejercicio. Cada una de estas dos fases tiene un peso del 50 % de la nota de esa entrega práctica.

Si nos centramos en la resolución de los ejercicios mediante un modelo de memoria distribuida, es importante comentar que, dado un ejercicio, éste se ejecuta en las máquinas de memoria distribuida Opteron y Odroid variando (a) el mapeo de los procesos y (b) el número de procesos creados. En cuanto al tipo de mapeo posible, se consideran dos enfoques: (a) un mapeo con orientación *Memory-Bound* y (b) un mapeo con orientación *CPU-Bound*. En el mapeo *Memory-Bound* se intenta ocupar el máximo número de nodos posibles con un único proceso, antes de asignar varios procesos a un mismo nodo (opción MD-OpteronM y MD-OdroidM en la Figura 8). Por otro lado, en el mapeo *CPU-Bound* se intenta mapear los procesos vecinos dentro de un mismo nodo. Así, la comunicación entre ellos será mas eficiente (opción MD-OpteronC y MD-OdroidC en la misma Figura 8). Finalmente, para cada una de las dos opciones de mapeo, se realizan diferentes ejecuciones variando el número de procesos posibles (se parte de un mínimo de 2 procesos y se va doblando su número hasta un máximo de 32 para la máquina Opteron y hasta un máximo de 256 para la máquina Odroid). Cabe destacar que estos valores máximos son el número de elementos de cálculo (*cores*) que tienen cada una de estas máquinas.

Tal como ya se ha comentado y para evaluar un ejercicio, en una primera fase se genera un *ranking* de las diferentes soluciones entregadas por los alumnos. De este modo, de las dos opciones de mapeo posibles (*Memory-Bound* y *CPU-Bound*) se selecciona aquella opción más favorable (con mejor *speedup*), tanto para la máquina Opteron como para la Odroid. Una vez seleccionado el mejor mapeo en ambas máquinas, se suman todos los *speedups* obtenidos al variar el número de nodos (de 2 a 32 en la máquina Opteron y de 2 a 256 en la máquina Odroid). El valor de las sumas de cada uno de los grupos de alumnos se normaliza a 1 y con ello se obtiene el *ranking* deseado. Es importante destacar que se utiliza la suma en lugar de la media harmónica con el objetivo de dar más peso a las ejecuciones con mayor número de procesos, ya que serán éstas las que



(a) MC-Xeon (b) MD-OpteronM (c) MD-OpteronC

(d) MD-OdroidM (e) MD-OdroidC (f) Suma *speedup* normalizado

Figura 8: Práctica Sudoku

obtendrán un valor numérico de *speedup* superior, e influenciarán más en el orden del *ranking* de cada grupo de alumnos. Así se premia el comportamiento del ejercicio donde las ejecuciones tienen un mayor número de procesos. A partir del *ranking* se asigna una calificación de 10 al primero de los grupos y al resto una calificación proporcional a la distancia que tiene respecto al primero. De nuevo, es importante destacar que si no se consigue un valor mínimo en esta suma normalizada a 1, esa entrega práctica se considera no evaluable. Finalmente, en una segunda fase se realiza una entrevista presencial a cada uno de los grupos para validar el conocimiento que tienen de la solución presentada y los conocimientos que han adquirido con su resolución. Destacar aquí que la calificación de cada uno de los 2 miembros del grupo puede ser diferente, en función de cómo ha respondido a las cuestiones que se plantean en esa entrevista. En cualquier caso, hay una correlación muy alta de notas entre la fase 1 y la fase 2. Los grupos que consiguen un mejor *speedup* acostumbran a tener una buena calificación en la entrevista presencial.

La Figura 8 muestra los resultados obtenidos por los alumnos en el curso académico 2018-2019 del ejercicio que calcula, mediante búsqueda-con-poda, todas las soluciones de un *Sudoku* parcialmente inicializado. En este caso, la paralelización del código se realiza siguiendo un modelo de memoria distribuida y, por tanto, utilizando el estándar MPI en las máquinas Opteron y Odroid. Concretamente, se muestran los resultados obtenidos en la máquina Opteron con un mapeo *Memory-Bound* (Figura 8b) y con un mapeo *CPU-Bound* (Figura 8c), los resultados obtenidos en la máquina Odroid con un mapeo *Memory-Bound* (Figura 8d) y con un mapeo *CPU-Bound* (Figura 8e), y el *ranking* de los resultados de todos los alumnos, elaborado tal como se detalló anteriormente (Figura 8f). Adicionalmente, también se muestran los resultados obtenidos en la máquina Xeon de memoria compartida (Figura 8a).

Aunque los resultados de la máquina Xeon son irrelevantes para la evaluación del ejercicio de memoria distribuida, se incluyen para mostrar que la máquina Xeon de memoria compartida tiene un comportamiento parecido a la máquina Opteron de memoria distribuida (Figura 8b y Figura 8c). En este sentido, se puede apreciar que en ambos casos a partir de 32 nodos (número de *cores* de las máquinas) no parece tener sentido crear más procesos, ya que ambas máquinas se saturan. En cambio, en la máquina Odroid se puede llegar a ejecutar un mayor número de procesos obteniendo buenos rendimientos.

Respecto a la comparativa de las prácticas que han entregado los alumnos, se puede observar que el grupo G11 es el que obtiene mejor rendimiento en las máquinas distribuidas (MD-Opteron y MD-Odroid), sobretodo para un número alto de procesos. También cabe destacar, que los grupos G3 y G5 tienen una ejecución irregular y no mejoran el rendimiento por mucho que incrementen el número de procesos. Por último, son destacables los resultados del grupo G4, que tiene incrementos de rendimientos normales hasta la creación de 32 procesos (a partir de ese valor se estancan los resultados). Más centrados en el *ranking* obtenido por los diferentes grupos de alumnos (Figura 8f), se puede observar que el grupo G11 obtiene el máximo rendimiento, que el grupo G4 supera el ejercicio de manera muy ajustada y los grupos G3 y G5 no superan el ejercicio y deben mejorarlo en la siguiente convocatoria de prácticas. Tal como ya se ha comentado, el grupo G11 obtendría la máxima nota en esta parte, el grupo G4 la nota mínima y el resto una nota proporcional a la distancia respecto a la mejor, en el siguiente orden: G7, G6, G9, G10, G1, G2 y G8.

Finalmente, el análisis de la evolución de las calificaciones de los alumnos, así como la recolección de sus opiniones sobre el uso del sistema (de forma similar a la de Emiroglu y Sahin [24]), nos permitirá obtener el *feedback* necesario para ajustar el entorno y mejorar los objetivos académicos propuestos.

VII. Valor añadido del nuevo sistema

Los estudiantes de nuestra asignatura, se encuentran por primera vez con un entorno de programación basado en colas *batch* en servidores dedicados y disponibles 24/7. Esto representa una experiencia nueva, con un aprendizaje muy rápido, que les hace ver la necesidad de una ejecución exclusiva, ya que pueden repetir ejecuciones paralelas que sólo van a depender de su código. Al ser reproducibles las distintas ejecuciones, pueden comparar sus mejoras de código a partir de la mejora del tiempo de ejecución. La experiencia del docente siempre se ha considerado muy positiva, ya que los alumnos pueden acceder a sistemas que normalmente no tienen acceso y que veían relativamente lejanos y ajenos a su entorno.

Sin embargo, al introducir el nuevo servidor de memoria distribuida basado en placas Odroid, se ha podido comprobar un incremento del interés mostrado por los alumnos en ese servidor. Al estar basado en placas SoC que los alumnos ven como más cercanas y familiares (incluso algunos de ellos con experiencias docentes en bachillerato o a nivel personal) hace que perciban que podrían construir por su cuenta algún servidor parecido al de la asignatura. Este incremento de interés no se producía con el servidor anterior, ya que eran conscientes de su elevado precio para una economía familiar y que precisaba de unas instalaciones exigentes en cuanto a refrigeración y consumo de energía.

Para adquirir competencias de programación paralela, el reto es programar sistemas con un número de procesos que supere el centenar (en nuestro caso hasta 256 procesos). Si bien acceder a una máquina de memoria compartida con 8 o 16 *cores* es fácil, no lo es tanto acceder a máquinas con centenares de *cores*. Esta dificultad extra fuerza a los alumnos a programar sus ejercicios con soluciones suficientemente escalables. Esta experiencia docente no hubiera sido posible sin disponer de suficientes nodos/*cores*.

Otro reto que deben afrontar los alumnos es la programación de nodos con pocos recursos. El principal recorte en recursos es la capacidad de computación y la memoria RAM de cada nodo. En este caso la RAM es de sólo 2 GB, que si bien es muy reducida para los estándares actuales, es proporcional al tamaño de los ejercicios a resolver en la asignatura. Con esto queremos decir que en una situación real de programación paralela, donde los problemas deben tratar con cantidades inmensas de datos, siempre toca lidiar con la cantidad de RAM y la cantidad de datos a procesar. En estos casos es necesario encontrar un compromiso que permita resolver los problemas planteados dimensionándolos a la RAM disponible. En cualquier sistema de simulación, por norma general, más datos implica más precisión, pero no siempre es posible incrementar los datos por limitaciones de RAM y potencia de cálculo. Con este nuevo servidor los alumnos deben ser conscientes del hecho de que la RAM es de sólo 2 GB y que si asignan un proceso a un nodo, éste puede disponer para resolver la parte del ejercicio que le corresponda de estos 2 GB (restando la RAM necesaria para el sistema operativo). Ahora bien, si en ese mismo nodo se asignan tantos procesos como *cores* (por ejemplo 4), ahora cada proceso sólo podría disponer de 1/4 parte de la RAM, o sea sólo 512 KB, y esto supone un nuevo esfuerzo de programación para ajustar el volumen de datos que cada proceso debe tratar.

En resumen, constatamos que el nuevo servidor ofrece y extiende nuevos paradigmas, que permiten a los alumnos una programación paralela más realista, basada en el incremento de *cores* y en la restricción de recursos como la memoria RAM.

VIII. Conclusiones

En este trabajo se detalla la construcción de un sistema multicomputador de bajo coste, tanto a nivel hardware como software, para utilizarlo en prácticas de programación paralela. Está basado en Odroid-C2, con un coste equivalente a un servidor pequeño con memoria compartida y permite acceder a más procesadores. Por otra parte, tratar de igualar el número de procesadores en sistemas con servidores de memoria compartida interconectados por red, tendría un coste mucho mayor. En cualquier caso, el rendimiento obtenido en relación al precio y al consumo es excelente.

Se muestra también el uso de esta propuesta en el contexto de una asignatura de últimos cursos del Grado de Ingeniería Informática y cómo se ha integrado en el modelo evaluativo de los ejercicios prácticos. Los resultados obtenidos han sido plenamente satisfactorios, al demostrarse su viabilidad y el valor añadido que introduce este nuevo sistema de bajo coste en el contexto de una asignatura de programación paralela. Así, se ha podido observar que esta propuesta acerca a los alumnos a un sistema con un gran número de procesadores y más opciones de intercomunicación, permite hacer estudios de escalabilidad más realistas, y todo ello manteniendo un buen compromiso entre prestaciones, precio y consumo.

Finalmente, tenemos la firme convicción que este sistema es perfectamente aplicable en otros ámbitos educativos, donde se precisen de múltiples nodos para realizar ejercicios prácticos. De este modo, podría ser utilizado en asignaturas de Sistemas Distribuidos (*cloud computing*...), de Administración de Sistemas (`kubernetes`...) y de Administración de Redes (conmutadores...). Además, creemos que también puede ser interesante aplicarlo en asignaturas universitarias optativas, e incluso en asignaturas de módulos superiores de secundaria, ya que ha quedado patente que la inversión necesaria para poder tener el sistema funcionando desde cero parece bastante asumible y sin que suponga a priori una debacle para el presupuesto docente.

Agradecimientos

Este trabajo ha sido financiado por el Gobierno de España bajo los contratos TIN2016-77836-C2-1-R, TIN2016-77836-C2-2-R, TIN2016-75344-R y DPI2016-77415-R, y por la Generalitat de Catalunya como Grupos de Investigación Consolidados 2017-SGR-688 y 2017-SGR-990.

Referencias

[1] C. Aliagas, P. Millán, C. Molina, and R. Meseguer, "MiniMultiComputador de Bajo Coste," in *Actas de las XXV JENUI*, vol. 4, 2019, pp. 31–38.

[2] A. Chhabra and G. Singh, "A cluster based parallel computing framework for performance evaluation of parallel applications," *Int. journal of computer theory and engineering*, vol. 2, no. 2, p. 226, 2010.

[3] A. Apon, R. Buyya, H. Jin, and J. Mache, "Cluster computing in the classroom: Topics, guidelines, and experiences," in *Proc. 1st IEEE/ACM Int. Symp. Cluster Computing & the Grid*. IEEE, 2001, pp. 476–483.

[4] M. Xu and Q. Su, "The realization of small cluster parallel computing environment for college education," in *9th International Conference on Computer Science & Education*. IEEE, 2014, pp. 861–863.

[5] D. Giménez, "Un curso práctico de programación paralela basado en problemas de concurso español de programación paralela," in *Actas de las XXII JENUI*. Universidad de Almería, 2016, pp. 19–26.

[6] J. Santamaría, M. Espinilla, A. Rivera, and S. Romero, "Potenciando el aprendizaje proactivo con ilias&webquest: aprendiendo a paralelizar algoritmos con gpus," in *Actas de las XVI JENUI*, 2010, pp. 503–506.

[7] C. Ivica, J. T. Riley, and C. Shubert, "StarHPC - Teaching parallel programming within elastic compute cloud," in *Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces*. IEEE, jun 2009, pp. 353–356.

[8] Massachusetts Institute of Technology, "StarCluster home page," http://star.mit.edu/cluster/index.html, accessed: 2020-01-13.

[9] F. Gomez-Folgar, R. Valin, A. Garcia-Loureiro, T. F. Pena, and I. Zablah, "Cloud computing for teaching and learning MPI with improved network communications," in *Workshop on Cloud Education Environments (WCLOUD)*, vol. 945, 2012, pp. 22–27.

[10] G. A. Prieto and J. P. Mendoza, "Plataforma Robótica Didáctica de Bajo Coste Basada en la Arquitectura Software Player/Stage y en el Hardware de La Fonera," *IEEE RITA*, vol. 7, pp. 239–245, 2012.

[11] R. A. Rodriguez, P. Cammarano, D. A. Giulianelli, P. M. Vera, A. Trigueros, and L. J. Albornoz, "Using Raspberry Pi to Create a Solution for Accessing Educative Questionnaires from Mobile Devices," *IEEE RITA*, vol. 13, pp. 144–151, 2018.

[12] C. Catalán Cantero and A. Blesa Gascón, "Enseñanza de sistemas empotrados: de Arduino a Raspberry Pi," in *Actas de las XXII JENUI*. Universidad de Almería, 2016, pp. 351–354.

[13] G. Ortega, J. M. G. Salmerón, C. Medina-López, J. L. Redondo, J. F. R. Herrera, N. C. Cruz, G. Barrionuevo, P. M. Ortigosa, V. González-Ruiz, and E. M. Garzón, "Procesadores de bajo coste y su aplicación en la docencia de Ingeniería de Computadores," in *Actas de las XXII JENUI*. Universidad de Almería, 2016, pp. 343–349.

[14] S. J. Johnston, P. J. Basford, C. S. Perkins, H. Herry, P. Tso Fung, D. Pezaros, R. D. Mullins, E. Yoneki, S. J. Cox, and J. Singer, "Commodity single board computer clusters and their applications," in *Future Generation Computer Systems Volume 89*, 2018, pp. 201–212.

[15] A. M. Pfalzgraf and J. A. Driscoll, "A low-cost computer cluster for high-performance computing education," in *IEEE Int. Conference on Electro Information Technology*. IEEE, 2014, pp. 362–366.

[16] I. E. Wina Rachmawan, N. Fahmi, E. W. Widodo, S. Huda, M. U. Pamenang, M. C. Roziqin, A. Permana W., S. Sukaridhoto, and D. Pramadihanto, "An Embedded System for applying High Performance Computing in Educational Learning Activity," *Int. Journal of Engineering Technology*, vol. 4, no. 1, pp. 46–64, 2016.

[17] K. Doucet and J. Zhang, "The creation of a low-cost raspberry Pi cluster for teaching," in *Proc. 24th Western Canadian Conf. on Computing Education*. New York, USA: ACM P., 2019, pp. 1–5.

[18] M. F. Cloutier, C. Paradis, and V. M. Weaver, "Design and analysis of a 32-bit embedded high-performance cluster optimized for energy and performance," in *2014 Hardware-Software Co-Design for High Performance Computing*. IEEE, 2014, pp. 1–8.

[19] R. V. Aroca and L. M. G. Gonçalves, "Towards green data centers: A comparison of x86 and arm architectures power efficiency," *Journal of Parallel and Distributed Computing*, vol. 72, no. 12, 2012.

[20] D. Göddeke, D. Komatitsch, M. Geveler, D. Ribbrock, N. Rajovic, N. Puzovic, and A. Ramirez, "Energy efficiency vs. performance of the numerical solution of pdes: An application study on a low-power arm-based cluster," *J. Comput. Phys.*, vol. 237, pp. 132–150, Mar. 2013.

[21] Raspberry Pi Foundation, "Raspberry home page, information and documentation," https://www.raspberrypi.org, accessed: 2019-02-01.

[22] Hardkernel co., Ltd., "HardKernel home page, odroid-c2 information," https://www.hardkernel.com/shop/odroid-c2, accessed: 2019-02-01.

[23] Hardkernel co. Ltd., "Odroid Wiki: Ubuntu," https://wiki.odroid.com/odroid-c2/os_images/ubuntu/ubuntu, accessed: 2019-02-01.

[24] B. G. Emiroglu and S. Sahin, "Analysis of students' performances during lab sessions of computer networks course," *Journal of Educational Technology & Society*, vol. 16, no. 3, pp. 329–346, 2013.

Carles Aliagas es Licenciado en Informática por la Universitat Politècnica de Catalunya (UPC, 1991) y Doctorando en Ingeniería Informática por la Universitat Rovira i Virgili (URV). En la actualidad es Profesor Titular de Escuela Universitaria a Tiempo Completo en el Departamento de Ingeniería Informática y Matemáticas de la URV, donde ejerce tareas docentes y de investigación. Sus intereses de investigación se centran en microarquitectura de procesadores, en la jerarquía de memoria y en la predicción en redes móviles.

Montse Garcia-Famoso es Licenciada en Informática por la Universidad de Deusto (1994). Ha sido docente en la Universidad de Deusto y en la UPC. En la actualidad es Profesora Titular de Escuela Universitaria a Tiempo Completo en el Departamento de Ingeniería Informática y Matemáticas de la Universitat Rovira i Virgili, donde ejerce tareas docentes y de investigación. Sus intereses de investigación incluyen las metodologías y herramientas para docencia universitaria en Ingeniería Informática.

Roc Meseguer es Ingeniero en Telecomunicaciones (1999) y Doctor en Ingeniería Informática (2012) por la UPC. Actualmente es Profesor Agregado a Tiempo Completo en el Departamento de Arquitectura de Computadores de la UPC, donde ejerce tareas docentes y de investigación. Sus intereses de investigación incluyen sistemas distribuidos, en particular la asignación de recursos para sistemas a gran escala (como redes *ad hoc* móviles y computación en la nube), sistemas descentralizados aplicados a la inteligencia ambiental y redes comunitarias basadas en iniciativas *bottom-up*.

Pere Millán es Licenciado en Informática por la Universitat Politècnica de Catalunya (1992) y Doctor en Ingeniería Informática por la Universitat Rovira i Virgili (URV, 2018). En la actualidad es Profesor Titular de Universidad a Tiempo Completo en el Departamento de Ingeniería Informática y Matemáticas de la URV, donde ejerce tareas docentes y de investigación. Sus intereses de investigación se centran en predicción de la calidad y mejora del rendimiento en redes móviles.

Carlos Molina es Licenciado en Informática (1996) y Doctor en Informática (2005) por la UPC. En la actualidad es Profesor Titular de Universidad a Tiempo Completo en el Departamento de Ingeniería Informática y Matemáticas de la URV, donde ejerce tareas docentes y de investigación. Sus intereses de investigación se centran en microarquitectura de computadores, predicción de la calidad y mejora del rendimiento en redes móviles, así como en computación *serverless*.