



ESQUEMAS DE DISTRIBUCIÓN DE TRABAJOS PARA UN SISTEMA ERP EN LA NUBE

110300083 CANUL CANUL CINDY GUADALUPE

110300083@UCARIBE.EDU.MX

110300097 KUMUL UC CRISTIAN ARIEL

110300097@UCARIBE.EDU.MX

110300079 PERAZA FELICIANO JONATHAN

110300079@UCARIBE.EDU.MX

Asesor: Joel Antonio Trejo Sánchez

Otoño 2015

Ingeniería en Telemática

CARTA DE ACEPTACIÓN DE PROYECTO TERMINAL

TÍTULO

Esquemas de distribución de trabajo para un sistemas ERP en la nube.

AUTORES

110300083

Cindy Guadalupe Canul Canul

110300083@ucaribe.edu.mx

110300097

Cristian Ariel Kumul Uc

110300097@ucaribe.edu.mx

110300079

Jonathan Peraza Feliciano

110300079@ucaribe.edu.mx

ASESOR



Joel Antonio Trejo Sánchez

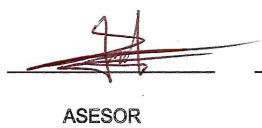
RESUMEN

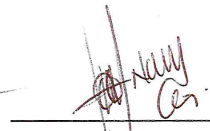
El cómputo en la nube es una tecnología prometedora como plataforma del futuro, permite el acceso a recursos de hardware de manera remota. Dentro del funcionamiento de los servicios es necesario optimizar los recursos en el centro de datos y para esto se utilizan los esquemas de calendarización. El problema incrementa con el número de tareas, pues es un problema NP-difícil, debido a las distintas características de cada host.

En este proyecto se estudiará y se hará una comparación de varios esquemas de calendarización para proponer una mejora en la optimización de recursos de acuerdo a las peticiones de un sistema ERP en la nube.

FIRMAS

  
AUTORES


ASESOR


TITULAR DE LA
ASIGNATURA

Resumen

El cómputo en la nube es una tecnología prometedora como plataforma del futuro, permite el acceso a recursos de *hardware* de manera remota. Dentro del funcionamiento de los servicios es necesario optimizar los recursos en el centro de datos y para esto se utilizan los esquemas de calendarización. La complejidad de la administración de los recursos y la calendarización incrementa con el número de tareas, pues es un problema NP-difícil debido a las distintas características de cada *host* y lo heterogéneas que son las tareas. En este proyecto se estudiaron y compararon cuatro esquemas de calendarización y una metaheurística para proponer una mejora en la optimización de recursos en un sistema *ERP* en la nube, para ello se utilizaron dos métricas; costo de procesamiento y tiempo de ejecución de las tareas.

Abstract

Cloud Computing is a promising technology as a platform of the future, allows access to hardware resources remotely. The main goal on a cloud environment is optimize the resources on the Data Center, for this scheduling algorithms are used.

The complexness of resources administration and scheduling increases with the number of tasks, is a NP-hard problem due to the heterogeneous nature of the tasks and resources. In this project they were studied and compared fourth scheduling schemes and a metaheuristic to propose an improvement in resources in an ERP system in the cloud. Two metrics were used; processing cost and time of execution of tasks.

Índice general

Resumen	3
Abstract	4
Introducción	10
Antecedentes y Planteamiento del problema	10
Propuesta de solución y Justificación	11
Objetivos	13
Objetivo general	13
Objetivos específicos	13
Hipótesis	14
1. Marco contextual	15
Introducción	15
1.1. Situación Actual	16
1.2. Estado del Arte	17
1.3. Marco teórico	18
Centro de Datos	19
Esquemas de Distribución de Trabajos	19
Balanceo de Cargas	20
Migración	21
Sistemas ERP	21
Tiempo de ejecución	21
Costo de procesamiento	21
Definición de Simulación	21
1.4. Marco Metodológico	23

2. Desarrollo	24
Introducción	25
2.1. Aplicación del marco metodológico y de actividades de experimentación	25
2.1.1. Simulación del Centro de Datos en la Nube	25
2.1.2. Implementación y evaluación de los Algoritmos	27
Configuración de Elementos del Datacenter y DatacenterBrokers para Algoritmos de Calendarización	32
DatacenterBrokers para Algoritmos de Calendarización	39
2.1.3. Mejorar el costo de procesamiento y el tiempo de ejecución	43
Algoritmo PSO y un ambiente en la nube	44
3. Resultados	47
Introducción	47
3.1. Tiempo de ejecución y costo de procesamiento	48
3.2. Simulación en <i>CloudSim</i>	51
Conclusiones	59
Glosario	63
A	63
C	63
E	63
F	63
H	64
P	64
S	64
Anexos	65
Simulación Básica en <i>CloudSim</i>	66
<i>Cloud Brokers</i>	70
<i>Metaheurística PSO</i>	77

Índice de figuras

1.	Propuesta de arquitectura en la nube, Fuente: Elaboración propia.	12
2.	Esquema general del Cómputo en la Nube, Fuente: Elaboración propia.	19
3.	Estilo de Trabajo de <i>CloudSim</i> , Fuente: Chatterjee et al. . . .	25
4.	Esquema de trabajo de algoritmos de calendarización, Fuente: Elaboración propia.	27
5.	Arquitectura FCFS para un entorno en la nube. Fuente: Elaboración propia.	28
6.	Arquitectura de <i>Min-Min</i> para un entorno en la nube. Fuente: Elaboración propia.	29
7.	Arquitectura de <i>Max-Min</i> para un entorno en la nube. Fuente: Elaboración propia.	30
8.	Arquitectura de <i>Round Robin</i> para un entorno en la nube. Fuente: Elaboración propia.	31
9.	Características del <i>datacenter</i> . Fuente: Elaboración propia. . .	32
10.	Características del <i>Host</i> . Fuente: Elaboración propia.	33
11.	Características de las máquinas virtuales. Fuente: Elaboración propia.	34
12.	Características de las Tareas (<i>cloudlets</i>). Fuente: Elaboración propia.	34
13.	Inicio de sesión <i>Odoo</i> . Fuente: Elaboración propia.	35
14.	Panel de módulos disponibles. Fuente: Elaboración propia. . .	36
15.	Módulo de contabilidad. Fuente: Elaboración propia.	36
16.	Factura generada en el módulo de contabilidad. Fuente: Elaboración propia.	37

17.	Peticiones realizadas en el servidor en orden decreciente por tiempo de respuesta. Fuente: Elaboración propia.	37
18.	Peticiones realizadas en el servidor en orden decreciente por tamaño de la tarea. Fuente: Elaboración propia.	38
19.	Cabeceras para impresión de Facturas. Fuente: Elaboración propia.	38
20.	<i>FCFS Broker</i> . Fuente: Elaboración propia.	39
21.	<i>Min-Min Broker</i> . Fuente: Elaboración propia.	40
22.	<i>Max-min Broker</i> . Fuente: Elaboración propia.	41
23.	<i>Algoritmos de ordenamientos</i> . Fuente: Elaboración propia. . .	42
24.	<i>Round Robin Broker</i> . Fuente: Elaboración propia.	43
25.	Promedio tiempo de ejecución con tareas 100-500, Fuente: Elaboración propia.	51
26.	Promedio costo de procesamiento con tareas 100-500, Fuente: Elaboración propia.	52
27.	Tiempo ejecución 50 muestras <i>FCFS</i> , Fuente: Elaboración propia.	53
28.	Tiempo ejecución 50 muestras <i>Max-Min</i> , Fuente: Elaboración propia.	54
29.	Tiempo ejecución 50 muestras <i>Min-Min</i> , Fuente: Elaboración propia.	55
30.	Tiempo ejecución 50 muestras <i>Round Robin</i> , Fuente: Elaboración propia.	56
31.	Tiempo ejecución 50 muestras de todos los algoritmos, Fuente: Elaboración propia.	56
32.	Tiempo ejecución con el algoritmo propuesto (izqda.) y el tiempo de ejecución sin procesamiento previo (dcha.), Fuente: Elaboración propia.	58
33.	Costo de procesamiento con el algoritmo propuesto (izqda.) y costo de procesamiento sin procesamiento previo (dcha.), Fuente: Elaboración propia.	58

Índice de cuadros

1.	Configuración <i>Datacenter</i> , Fuente: Elaboración propia.	48
2.	Configuración de <i>Host</i> , Fuente: Elaboración propia.	48
3.	<i>Virtual Machine</i> , Fuente: Elaboración propia.	49
4.	Configuración <i>Cloudlet</i> , Fuente: Elaboración propia.	49
5.	Configuración <i>Cloudlet</i> , Fuente: Elaboración propia.	49
6.	Desviación estándar del tiempo de ejecución, Fuente: Elaboración propia.	57

Introducción

Antecedentes y Planteamiento del problema

El cómputo en la nube es una tecnología que ha abarcado gran parte de los negocios para dar soporte a ellos. Ésta tecnología provee a los negocios una ventaja competitiva en los costos de recursos como se ve en los negocios tradicionales, además de la versatilidad que provee al ajustarse a la necesidad de las empresas (Srinivasan, 2014).

La tecnología en la nube ha desarrollado una infraestructura fuerte después del surgimiento del cómputo distribuido (Chen and Deng, 2009). Para obtener las ventajas de dicha tecnología los usuarios simplemente necesitarán conectarse a internet y de esta manera tendrán el acceso al procesamiento de manera remota (Aranganathan and Mehata, 2011). Sin embargo, para aprovechar el máximo potencial de éstos recursos, es necesario tener en consideración algunas variables, ya que en un entorno en la nube existe un comportamiento dinámico de los recursos a manera que se les provea a los usuarios el servicio (Shimpy and Sidhu, 2014). Una de las prácticas con mayor importancia en la nube es la calendarización, ya que tiene como objetivo administrar las tareas del centro de datos para optimizar los recursos del mismo. De esta manera la eficiencia de la carga de trabajo en la nube aumenta (Shimpy and Sidhu, 2014). En general, el objetivo de la calendarización en la nube es utilizar los recursos de manera apropiada, mientras la carga de trabajo es distribuida uniformemente para mejorar los tiempos de ejecución (Shimpy and Sidhu, 2014). Debido a la atención que se tiene en la tecnología en la nube, los centros de datos han tomado un papel muy importante para los servicios empresariales (Shimpy and Sidhu, 2014). Un centro de datos está compuesto por miles de servidores virtuales ejecutándose en una instancia de tiempo alojando muchas tareas, al mismo tiempo el centro de datos recibe miles de peticiones a esas tareas. Es aquí en donde la

programación de trabajos tiene un rol muy importante para el cómputo en la nube, ya que influye en el rendimiento del mismo (Srinivasan, 2014).

El problema de la calendarización pertenece a los algoritmos NP-Difícil, lo cual tiene un amplio rango de soluciones posibles y se toma mucho más tiempo de encontrar una respuesta óptima, ya que no existe un método para resolver estas incógnitas. Sin embargo, es posible estar cerca de la mejor solución contemplando algunos entornos (Shimpy and Sidhu, 2014).

Propuesta de solución y Justificación

Los centros de datos son una parte esencial en la tecnología en la nube, están conformados por varios servidores virtuales que alojan varios trabajos, ejecutándose por estancias de tiempo y a la vez recibiendo peticiones a esos trabajos (Shimpy and Sidhu, 2014). Para que el cómputo en la nube pueda rendir correctamente se necesita una buena programación de trabajos o calendarización. Este problema tiene un gran número de soluciones posibles, toma mucho tiempo en encontrar una respuesta óptima y por eso está dentro de los algoritmos NP-Difícil. No existe un método para resolver lo anterior, sin embargo es posible obtener una solución cercana contemplando algunos entornos (Shimpy and Sidhu, 2014). En esta investigación se propone una mejora a un algoritmo de calendarización para el caso de estudio de un sistema *ERP* en la nube, teniendo en cuenta sus posibles escenarios y peticiones heterogéneas. De esta manera se podrá contar con un esquema de distribución de trabajos que satisfaga las necesidades de un sistema *ERP*, mejorando el tiempo de respuesta y aprovechando los recursos. Con beneficios a los proveedores de servicios (*SaaS*) de sistemas *ERP*.

En la figura (1) se puede observar la arquitectura del centro de datos que se implementará en la investigación, el cual está compuesto de los siguientes elementos:



Figura 1: Propuesta de arquitectura en la nube, Fuente: Elaboración propia.

- **Servidor administrador:** Es el servidor que tendrá el rol de *front end* en el centro de datos, teniendo una interacción directa con las peticiones de los usuarios, que son especificados en éste documento como trabajos. El principal objetivo de éste elemento es delegar los trabajos a las máquinas virtuales en los distintos *hosts* del centro de datos.
- **Host:** es el recurso de *hardware* que se tiene en el centro de datos. Una de las características de éste elemento es que es finito.
- **Máquinas Virtuales:** son instancias dentro de los *host*, tienen como objetivo resolver los trabajos asignados.

Objetivos

Objetivo general

Proponer un esquema de distribución que resuelva de manera eficiente el problema de la calendarización en un centro de datos, tomando como caso de estudio un sistema ERP en la nube.

Objetivos específicos

- Definir una arquitectura para un centro de datos en la nube.
- Realizar la simulación en base a la arquitectura del centro de datos.
- Realizar e implementar diferentes esquemas de distribución de trabajos, rescatando sus características (costo de procesamiento y tiempo de ejecución).
- Reunir las características y realizar el nuevo esquema de distribución.
- Aplicar el nuevo esquema al centro de datos.
- Comparar los resultados obtenidos.

Hipótesis

Se puede mejorar el tiempo de ejecución y el costo de procesamiento de una sistema *ERP* en la nube, mediante un algoritmo de calendarización basado en una metaheurística.

Para realizar la verificación de validez de la hipótesis se usarán:

- **Pruebas de *hardware y software*:**
 - **Prueba de desempeño:** Validar el tiempo de respuesta para las tareas bajo condiciones normales y máximas.
 - **Prueba de carga:** Verificar el tiempo de respuesta del sistema, bajo diferentes condiciones de carga. Se mide el tiempo de respuesta y otros requisitos sensibles al tiempo.
- **Estadísticas:** en base a los resultados que se obtendrán en las pruebas de desempeño y de carga, se va realizar comparaciones y posteriormente visualizarlas por medio de gráficas y tablas.

Capítulo 1

Marco contextual

Introducción

En éste capítulo se describen algunos marcos conceptuales necesarios para fundamentar y analizar los requisitos de la investigación, las secciones son las siguientes: Situación actual, Estado del Arte, Marco Teórico y Marco Metodológico. En la primera sección se muestra cómo se encuentra la investigación en la actualidad, cómo se encuentra nacional e internacionalmente y cuál es el punto de partida del proyecto; en la siguiente sección se presenta la información sobre los diferentes esquemas de distribución que serán implementados durante la elaboración del proyecto. El Marco teórico nos describe los conceptos necesarios para fundamentar y comprender mejor el proyecto y su propósito, finalmente en la última sección se describe los pasos que se van a seguir para realizar y alcanzar los objetivos del proyecto.

1.1. Situación Actual

Actualmente, son muchas las organizaciones que utilizan el cómputo en la nube, ya que trae beneficios en cuanto a gastos y recursos. Ya no hay necesidad de invertir tanto en la infraestructura de *TI (Information Technology)* y existe una mejor capacidad de almacenamiento para los recursos que el ámbito empresarial necesita. El cómputo en la nube utiliza los servicios *SaaS (Software as a Service)* y *HaaS (Hardware as a Service)* para hacer más eficiente y flexible el uso de la infraestructura de la nube (Mariscal and Gil-Garcia, 2013).

México utiliza, en cuanto a la infraestructura de almacenamiento virtual, los servicios ofrecidos por *Amazon, Google y Triada (TELMEX)*. Los cuales proporcionan almacenaje de información virtual, plataformas de hospedaje de aplicaciones y servicios en línea (Mariscal and Gil-Garcia, 2013).

Al proveer el servicio se tiene dos escenarios diferentes: cuando hay una mayor demanda de recursos y cuando no la hay, eso va dependiendo de las necesidades de las empresas. Para que se tenga una mejor eficacia durante los servicios, se tiene que mejorar el uso de los recursos en el centro de datos. Dentro del centro de datos hay muchos servidores virtuales que están recibiendo trabajos, mientras la nube las mantiene en los lotes de solicitud de trabajos. Es aquí, donde resaltamos la importancia de la distribución de trabajos dentro del centro de datos (Shimpy and Sidhu, 2014).

En este momento, la distribución de trabajos es un problema NP- difícil, pues no se ha encontrado una solución óptima. Sin embargo, existen diferentes propuestas para encontrar una mejor solución, utilizando diferentes esquemas de distribución o dicho de otra forma algoritmos (Shimpy and Sidhu, 2014).

Existen una gran variedad de algoritmos utilizados en el problema de distribución de trabajos. Los algoritmos más utilizados en las investigaciones son: *FCFS, Round Robin, Min-Min, Max-Min y Metaheurística*. Cabe mencionar que en las investigaciones solo hacen una selección de una serie de algoritmos para que, por medio de pruebas, determinen quién es el mejor de ellos.

1.2. Estado del Arte

Los siguientes algoritmos de calendarización actualmente están prevaleciendo en la nube:

- ***Resource-Aware-Scheduling Algorithm (RASA)***: Parsa, Entezari-Maleki (2009) proponen el algoritmo *RASA*, el cual utiliza las ventajas de dos algoritmos tradicionales (*Max-min* y *Min-min*) y cubre sus desventajas. Aunque el tiempo límite, la tasa de llegada, costo de ejecución y costo de comunicación no están considerados (Parsa and Entezari-Maleki, 2009).
- ***RSDC (Reliable Scheduling Distributed In Cloud Computing)***: Delevar, Javanmard, Shabestari y Talebi (2012) proponen un algoritmo confiable en un entorno en la nube, en este algoritmo los trabajos importantes son divididos en sub-trabajos, de tal manera que se puedan balancear las peticiones (Delavar et al., 2012).
- ***An Optimal Model for Priority based Service Scheduling Policy for Cloud Computing Environment***: Dakshayini, Gurupasad (2011), proponen un nuevo algoritmo de calendarización que se basa en la prioridad y un esquema de control de admisión. En este algoritmo, la prioridad se asigna a cada proceso admitido en la cola (Dakshayini and Guruprasad, 2011) .
- ***Pre-emptable Shortest Job Next Scheduling algorithm (PSJN)***: Este algoritmo se propone en una nube privada. Utiliza la técnica de suscripción preferente del algoritmo de *Round Robin* junto con el siguiente proceso más corto (*PSN*). Brinda beneficios de costos y mejora tiempo de respuesta y tiempo de ejecución (Nishant and R., 2015).
- ***User-priority Guided Min-min scheduling algorithm***: Se realiza una mejora para el algoritmo de balanceo de cargas, a través del algoritmo *Min-min* para la calendarización de trabajos con el fin de minimizar el tiempo de terminación del último trabajo (*makespan*) y maximizar la utilización de los recursos (Chen et al., 2013).

1.3. Marco teórico

El cómputo en la nube es una tecnología emergente, la cual está compuesta por un grupo de recursos heterogéneos que proveen servicios a través de internet (Agarwal et al., 2014). Esta tecnología permite a los consumidores y negocios utilizar aplicaciones sin necesidad de instalación y con acceso a sus archivos personales en cualquier computadora con acceso a internet (Chowdhury et al., 2012). El cómputo en la nube se puede clasificar de dos maneras:

- Por la ubicación:
 - **Nube Pública:** La infraestructura de cómputo se puede compartir entre cualquier organización (Chowdhury et al., 2012).
 - **Nube Privada:** La infraestructura de cómputo es dedicada a una organización en particular y no se comparte con otras organizaciones (Chowdhury et al., 2012).
 - **Nube Híbrida:** Las organizaciones pueden albergar sus aplicaciones críticas en nubes privadas y las aplicaciones con menos problemas de seguridad las puede albergar en nubes públicas (Chowdhury et al., 2012).
- Por el tipo de servicios ofrecidos:
 - ***Infrastructure as a Service (IaaS)*:** En éste nivel, la infraestructura se ofrece como servicio hacia los solicitantes en forma de Máquinas Virtuales (*VM*) (Agarwal et al., 2014).
 - ***Platform as a Service (PaaS)*:** Es una plataforma de desarrollo de aplicaciones que se provee como servicio hacia los desarrolladores para crear aplicaciones basadas en web (Agarwal et al., 2014).
 - ***Software as a Service (SaaS)*:** En éste nivel el proveedor en la nube provee las aplicaciones de *software* (Agarwal et al., 2014).

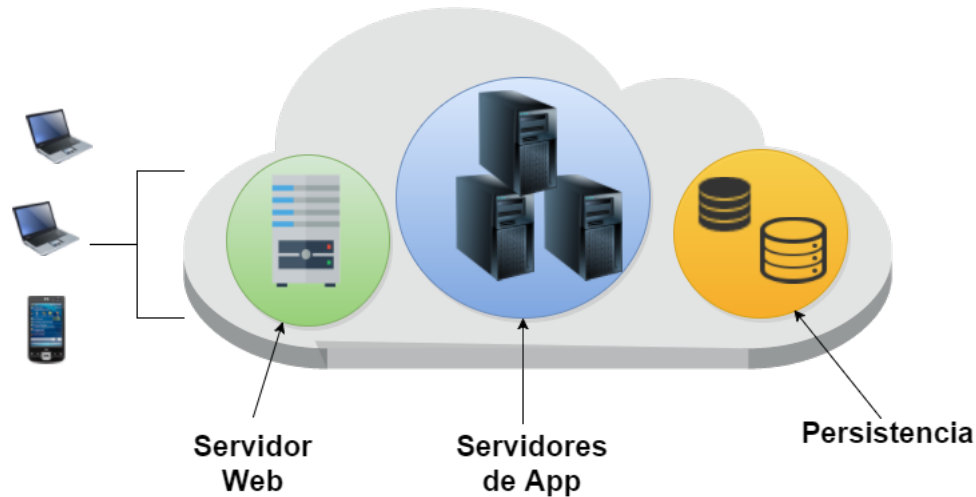


Figura 2: Esquema general del Cómputo en la Nube, Fuente: Elaboración propia.

Centro de Datos

Un centro de datos (*datacenter*) es un espacio dedicado donde las organizaciones almacenan y operan la mayoría de su información y la infraestructura de las tecnologías de comunicaciones que respaldan sus negocios (EPI, 8 31).

Esquemas de Distribución de Trabajos

La distribución de trabajo (calendarización) es uno de los puntos principales además de los más retadores problemas en el cómputo en la nube (Li et al., 2014). Esta consiste en un conjunto de políticas para controlar el orden en el cual los procesos serán ejecutados por el sistema (Agarwal et al., 2014).

Existen varios tipos de algoritmos de calendarización, su principal ventaja es obtener un alto rendimiento al cumplir con los trabajos. Los principales ejemplos de algoritmos de calendarización son (Salot, 2013):

- **FCFS (First Come First Serve) Algorithm:** significa que el trabajo que llegue primero será el primero en ser ejecutado.

- **Round Robin Algorithm:** consiste en agregar un tiempo de ejecución para cada trabajo, si dicho tiempo se agota y el trabajo actual no ha finalizado, pasa a un estado de espera y continúa con el siguiente trabajo, al finalizar el último trabajo regresa a revisar si existen trabajos en espera para proceder a ejecutarlas.
- **Min-Min Algorithm:** se va ejecutando una por una iniciando por los trabajos de menor tamaño.
- **Max-Min Algorithm:** se va ejecutando una por una iniciando por los trabajos de mayor tamaño.
- **Most fit task scheduling algorithm:** este algoritmo busca los elementos más aptos en la cola de trabajos para ejecutarlo primero. Tiene un alto radio de error.
- **Priority scheduling algorithm:** la idea básica es que a cada proceso se le asigna una prioridad, y dependiendo de las prioridades de los demás procesos es cómo se ejecutarán cada uno de ellos.

Balanceo de Cargas

Dentro de nuestro entorno en la nube cada *host* tiene recursos de *hardware* finitos y son susceptibles a fallos. Para mitigar contra las fallas y el exhaustivo uso de los recursos, los *host* son agrupados en *clusters*, los cuales son esencialmente un agrupamiento de recursos compartidos. El administrador (*manager*) es capaz de asegurarse que ninguno de los *host* dentro del *cluster* es responsable de todas las máquinas virtuales dentro de ese *cluster* (Dover et al., 2013).

Un entorno de virtualización responde a los cambios en la demanda para los recursos en cada *host* haciendo uso del balanceo de cargas, calendarización de trabajos y migración.

Una política de balanceo de cargas es configurada para un *cluster*, que a su vez contiene multiples *hosts*, cada uno de ellos con distintos recursos de *hardware* y disponibilidad de memoria (Dover et al., 2013).

Existen tres políticas para el balanceo de cargas:

- Sin Balanceo.
- Distribución Uniforme.

- Ahorro de Energía.

Migración

La migración se utiliza para hacer cumplir las políticas dentro del balanceo de cargas. La migración de una máquina virtual toma lugar de acuerdo a las políticas de balanceo de cargas para un *cluster* y la demanda actual sobre los *hosts* dentro de dicho *cluster*. La migración de igual manera puede ser configurada para ocurrir automáticamente cuando un *host* es bloqueado o movido a modo de mantenimiento (Dover et al., 2013).

Sistemas *ERP*

La sigla *ERP*, en inglés *Enterprise Resource Planning*, significa Planificación de los Recursos Empresariales. Un sistema *ERP* constituye un marco de trabajo que incluye aplicaciones comerciales, administrativas (finanzas, contabilidad), recursos humanos, planeamiento de manufactura y gestión de proyectos (S. and Horacio, 2002).

Tiempo de ejecución

Período en el que un programa es ejecutado por el sistema operativo. El período comienza cuando el programa es llevado a la memoria primaria y comienzan a ejecutarse sus instrucciones. El período finaliza cuando el programa envía la señal de término (normal o anormal) al sistema operativo (Dover et al. (2013)).

Costo de procesamiento

Costo o esfuerzo que utiliza una máquina virtual para ejecutar una tarea (Dover et al. (2013)).

Definición de Simulación

El tipo de implementación de éste proyecto será la simulación. Por ello no se puede avanzar sin haber dado previamente una definición de simulación. Antes de proseguir con la discusión del tema, se abordarán algunas definiciones de la palabra simulación.

Thomas H. Naylor define la simulación así :

”Técnica numérica para conducir experimentos en una computadora digital. Estos experimentos comprenden ciertos tipos de relaciones matemáticas y lógicas, las cuales son necesarias para describir el comportamiento y la estructura de sistemas complejos del mundo real a través de largos periodos de tiempo”.

(Naylor and Finger (1967))

Esta definición tiene una gran relación con el *cómo* se resolverá el objetivo del proyecto, ya que se describe el comportamiento de un sistema *ERP* y la estructura de un centro de datos en la nube, algo que sería demasiado complejo de lograr de manera física por cuestiones de recursos de *hardware* y el consumo de energía por largos periodos de tiempo.

Otros investigadores del tema tienen definiciones similares, como Robert E. Sahannon, que define la simulación como:

”Simulación es el proceso de diseñar y desarrollar un modelo computarizado de un sistema o proceso y conducir experimentos con este modelo con el propósito de entender el comportamiento del sistema o evaluar varias estrategias con las cuales puede operar el sistema”.

(Shannon and Johannes (1976))

Las definiciones anteriores especifican los elementos que se tratarán en la implementación de éste proyecto, tales como: el modelo del centro de datos y los modelos matemáticos (en este caso serán los algoritmos de calendarización), con el fin de entender y evaluar el comportamiento de un sistema *ERP* bajo un algoritmo propuesto.

1.4. Marco Metodológico

A continuación se propone la siguiente metodología para llevar a cabo el proyecto:

- **Simular:** Se implementará a manera de simulación un centro de datos con un entorno en la nube, las máquinas virtuales y el servidor inicializador que lo conforman.
- **Implementar:** En el centro de datos se desarrollarán los algoritmos de calendarización: *FCFS*, *Max-min*, *Min-min* y *Round Robin*.
- **Evaluar:** Se simulará el comportamiento de las peticiones de un sistema *ERP* consumiendo el servicio en la nube (*SaaS*), para conocer el rendimiento en tiempo de ejecución y costo de procesamiento de los algoritmos.
- **Mejorar:** Se propondrá un algoritmo, basado en una metaheurística, que mejorará el rendimiento en tiempo y costo, de acuerdo al comportamiento de un sistema ERP en la nube.
- **Comparar:** Se realizará una comparativa del tiempo de ejecución y el costo de procesamiento entre la versión mejorada y la original para determinar si hubo una mejora.

Capítulo 2

Desarrollo

Introducción

En éste capítulo se muestra la arquitectura de los distintos esquemas de calendarización de trabajos en un entorno en la nube, para la implementación de dichos esquemas se hace uso de un *framework* llamado *CloudSim*, el cual será definido de igual manera. Además se mostrarán algunos procesos de un sistema *ERP* para verificar el tamaño de las peticiones que realiza y poder acoplar las características en nuestro esquema de simulación. Para llevar a cabo la medición de las peticiones se hace uso de *Odoo* un sistema de ERP integrado de código abierto. Finalmente la implementación del algoritmo basado en una metaheurística (*PSO*).

2.1. Aplicación del marco metodológico y de actividades de experimentación

En base en los puntos descritos anteriormente en el marco metodológico se realizaron las siguientes actividades:

2.1.1. Simulación del Centro de Datos en la Nube

CloudSim es un nuevo, generalizado y extensible *framework* de simulación, que permite el modelaje, simulación y experimentación de infraestructuras emergentes de cómputo en la nube y servicios de aplicación (Calheiros, 2011, p. 2).

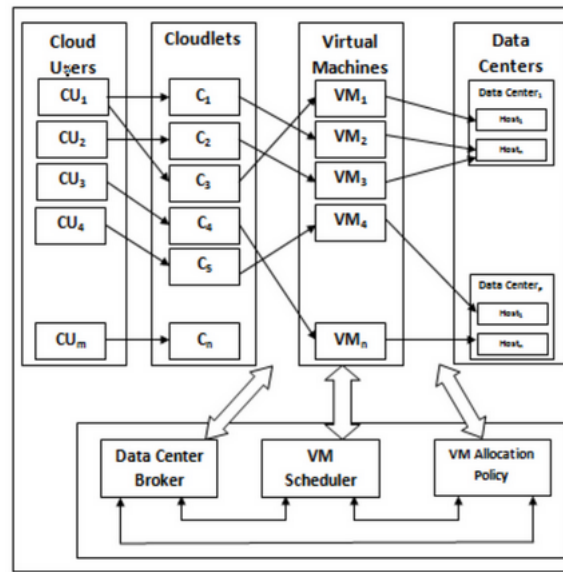


Figura 3: Estilo de Trabajo de *CloudSim*, Fuente: Chatterjee et al.

Entre los componentes que proporciona dicho *framework* se encuentran los siguientes:

- **Cloudlet:** Esta clase modela las aplicaciones de servicio basadas en la nube como pueden ser envío de contenido, redes sociales, y flujo de trabajo empresarial (Calheiros, 2011, cap. 4).

- **Datacenter:** Esta clase modela el núcleo de los servicios en un nivel de infraestructura (*hardware*) que son ofrecidos por *Cloud Providers* (*Amazon, Azure, App Engine*). Estos son encapsulados en un conjunto de *host* que pueden ser homogéneos o heterogéneos con respecto a sus configuraciones de *hardware* (memoria, núcleos, capacidad, y almacenamiento) (Calheiros, 2011, cap. 4).
- **DatacenterBroker:** Esta clase modela un *broker*, el cual es responsable de mediar las negociaciones entre el *SaaS* y los *Cloud providers*; y dichas negociaciones son manejadas por los requerimientos *QoS* (Calheiros, 2011, cap. 4).
- **Host:** Esta clase modela los recursos físicos como una computadora o un servidor de almacenamiento (Calheiros, 2011, cap. 4).
- **Vm:** Esta clase modela una Máquina Virtual (*VM*), la cual es administrada y hosteada por un componente *host* en la nube. Cada *VM* tiene acceso a un componente que almacena las siguientes características relacionadas a una *VM*: memoria accesible, procesador y tamaño de almacenamiento (Calheiros, 2011, cap. 4).

2.1.2. Implementación y evaluación de los Algoritmos

Existen varios algoritmos para calendarizar los trabajos en el cómputo en la nube. La mayor ventaja de estos algoritmos es obtener el mayor rendimiento. Los principales ejemplos de algoritmos de calendarización son: *FCFS*, *Round Robin*, *Min-Min*, *Max-Min* y *algoritmos de metaheurísticas* (Shimpy and Sidhu, 2014, p. 1).

De los algoritmos mencionados anteriormente, se presentan los siguientes:

- **FCFS:** Ejecuta las tareas en orden de llegada, es decir, el primero en llegar es el primero en ser atendido.
- **Min-Min:** Selecciona las tareas más pequeñas para ser ejecutadas primero.
- **Max-Min:** Selecciona las tareas más grandes para ser ejecutadas primero.
- **Round Robin:** Ejecuta las tareas en orden de llegada, si el número de tarea actual es mayor al número de máquinas virtuales reinicia el ciclo de asignación hacia la primera máquina virtual.

En un entorno de trabajo normal, la forma descrita anteriormente para los algoritmos de *FCFS*, *Min-Min*, *Max-Min* y *Round Robin* lo podemos apreciar de la siguiente manera, Figura (4):

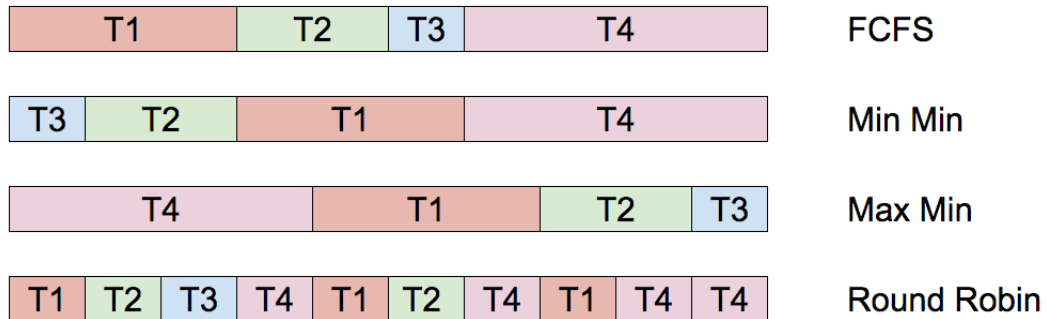


Figura 4: Esquema de trabajo de algoritmos de calendarización, Fuente: Elaboración propia.

Sin embargo en un entorno en la nube, al ser múltiples máquinas virtuales alojadas en distintos *hosts*, que a su vez pueden formar parte de uno o más

datacenters, dicho esquema tiene que ser modificado para poder adoptar un estilo de trabajo similar al proporcionado por el *framework*, por lo que los resultados quedan de la distinta manera:

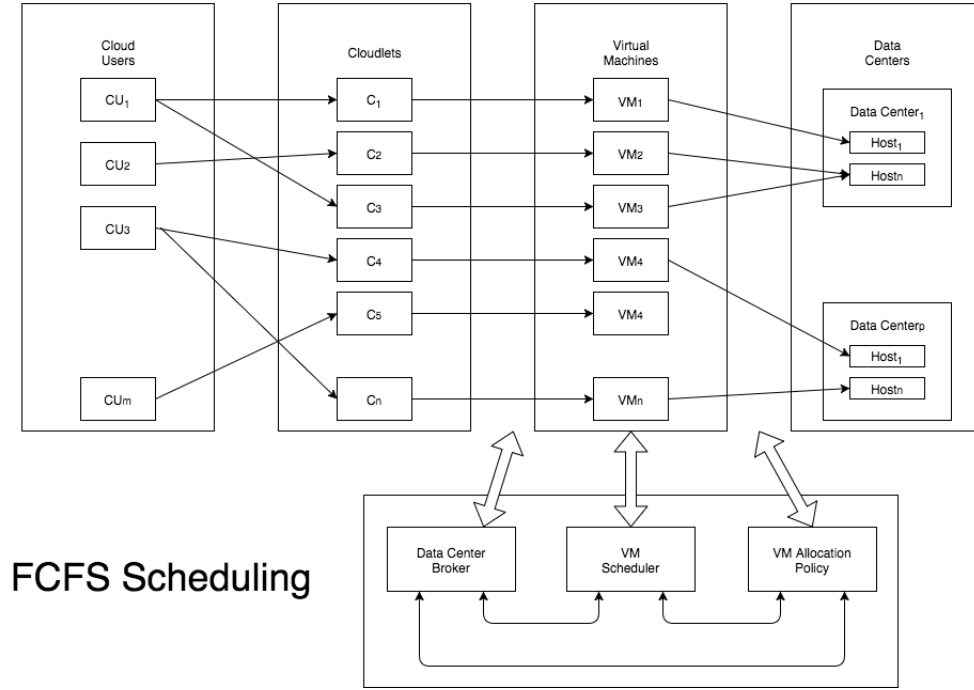


Figura 5: Arquitectura FCFS para un entorno en la nube. Fuente: Elaboración propia.

Como podemos apreciar en el diagrama (Figura 5), podemos tener m usuarios ejecutando n tareas, sin embargo la asignación de máquinas virtuales va dependiendo del orden de llegada de dichas tareas, y quien se encarga de repartir las tareas es el *datacenterBroker*.

De manera similar tenemos los siguientes algoritmos *Min-Min* (Figura 6) y *Max-Min* (Figura 7):

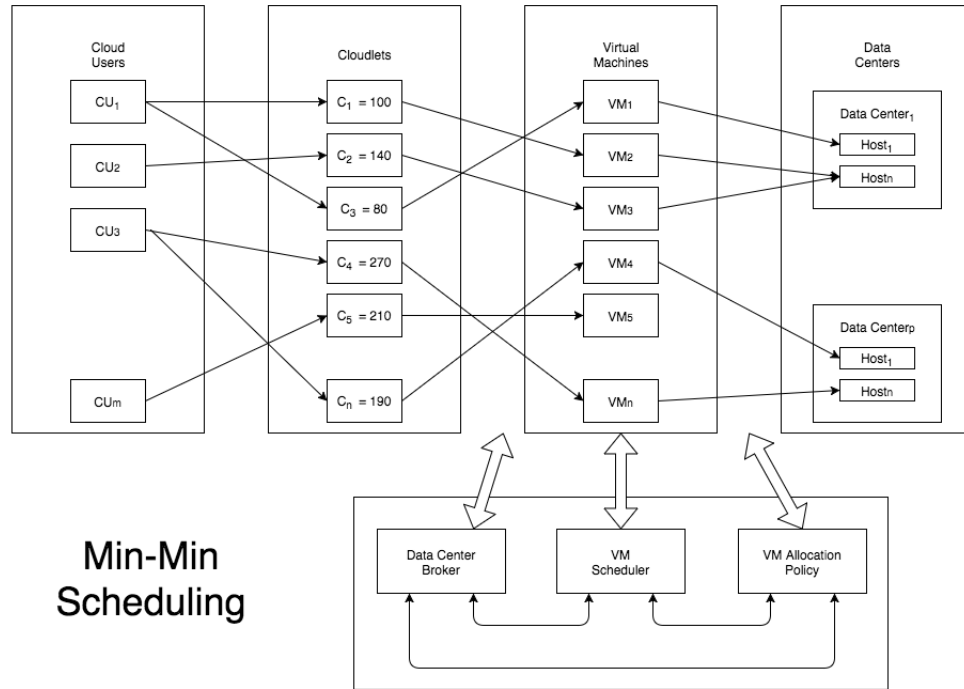


Figura 6: Arquitectura de *Min-Min* para un entorno en la nube. Fuente: Elaboración propia.

Para el algoritmo de *Min-Min* (Figura 6), obtenemos la lista de tareas a partir de la información de ellas, procedemos a ordenarlas de menor a mayor tamaño, asignando la de menor tamaño en la primera máquina virtual, la siguiente tarea, de acuerdo al tamaño, pasa a la segunda máquina y así sucesivamente.

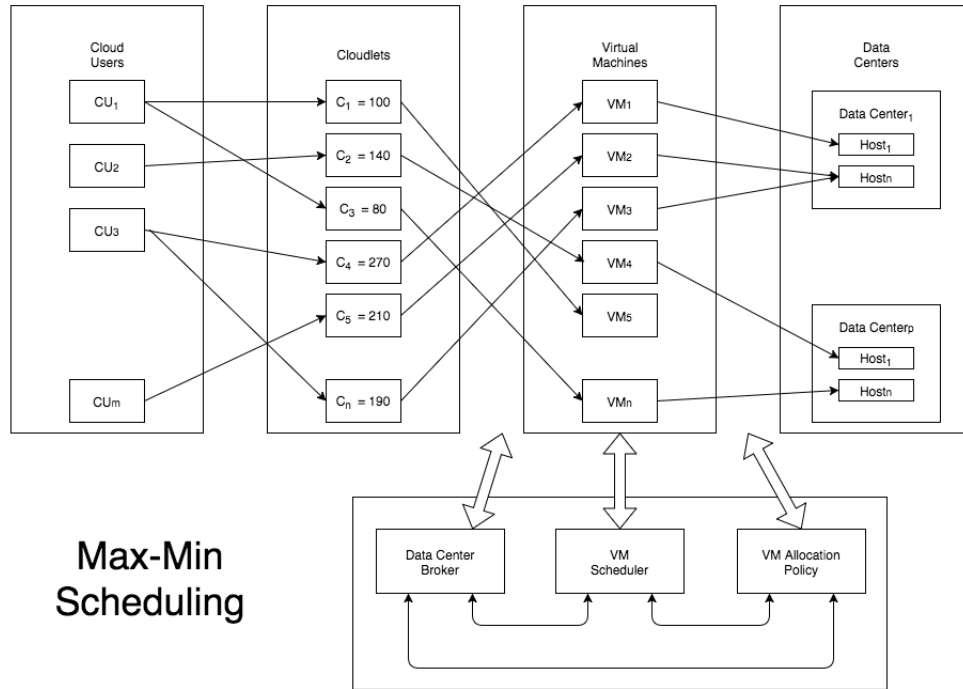


Figura 7: Arquitectura de *Max-Min* para un entorno en la nube. Fuente: Elaboración propia.

En la figura (7) tenemos el algoritmo de *Max-Min*, el cual ordena la lista de tareas de mayor a menor tamaño antes de asignarlas a las máquinas virtuales, una vez ordenada dicha lista procede a asignar la tarea más grande en la primera máquina, la siguiente en la segunda y así sucesivamente hasta terminar la asignación de las tareas.

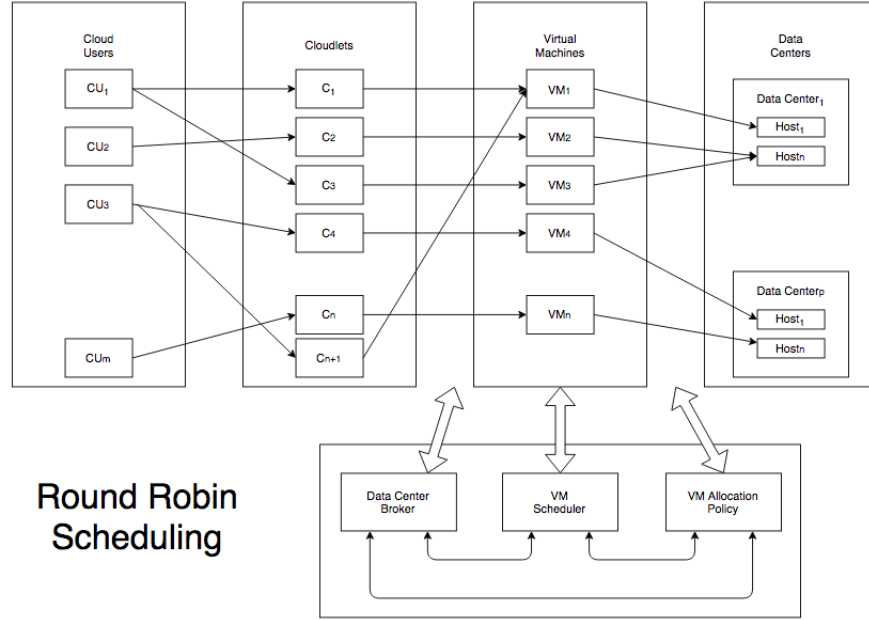


Figura 8: Arquitectura de *Round Robin* para un entorno en la nube. Fuente: Elaboración propia.

Como podemos apreciar en la figura (8), las tareas se van ejecutando de manera que llegan, pero al momento de ser mayor el número de tarea la asigna en la máquina virtual principal, esta operación de asignación la podemos expresar de la siguiente manera:

$$C_n \rightarrow Vm_{n \% m}$$

donde C_n representa la n -ésima tarea y m es el número total de máquinas virtuales.

A continuación se muestran algunas capturas del código para la implementación de los primeros algoritmos en el centro de datos:

Configuración de Elementos del *Datacenter* y *DatacenterBrokers* para Algoritmos de Calendarización

```
// 5. Create a DatacenterCharacteristics object that stores the
// properties of a data center: architecture, OS, list of
// Machines, allocation policy: time- or space-shared, time zone
// and its price (G$/Pe time unit).
String arch = "x86";           // system architecture
String os = "Linux";           // operating system
String vmm = "Xen";
double time_zone = 10.0;       // time zone this resource located
double cost = 3.0;             // the cost of using processing in this resource
double costPerMem = 0.05;      // the cost of using memory in this resource
double costPerStorage = 0.1;   // the cost of using storage in this resource
double costPerBw = 0.1;        // the cost of using bw in this resource
LinkedList<Storage> storageList = new LinkedList<Storage>(); //we are not adding SAN devices by now
```

Figura 9: Características del *datacenter*. Fuente: Elaboración propia.

Como primer paso, debemos de configurar las características que tendrá nuestro *datacenter*, para eso tenemos que definir la arquitectura, el tipo de sistema operativo, costos de utilización de memoria, almacenamiento y ancho de banda, entre otros (Figura 9).


```
List<Host> hostList = new ArrayList<Host>();
// 2. Una máquina contiene uno o más PEs o CPUs/Cores.
List<Pe> peList = new ArrayList<Pe>();
int mips = 1000;
// 3. Crear los PEs y agregarlos a la lista.
peList.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating
//4. Create Host with its id and list of PEs and add them to the list of machines
int hostId = 0;
int ram = 2048; //host memory (MB)
long storage = 1000000; //host storage
int bw = 10000;

//Agregamos N máquinas a nuestro data center
for(hostId = 0; hostId < numHosts; hostId++){
    hostList.add(
        new Host(
            hostId,
            new RamProvisionerSimple(ram),
            new BwProvisionerSimple(bw),
            storage,
            peList,
            new VmSchedulerTimeShared(peList)
        )
    );
}
```

Figura 10: Características del *Host*. Fuente: Elaboración propia.

De igual manera tenemos que asignar las características que tendrá cada *host* dentro del *datacenter* (Figura 10), tales como el número de procesadores, número de instrucciones por segundo, memoria RAM, capacidad de almacenamiento y el ancho de banda.

```

public static List<Vm> createVm(int userId, int vms, int idShift){
    LinkedList<Vm> list = new LinkedList<Vm>();

    //parámetros de las máquinas virtuales
    long size = 10000; //MB
    int ram = 512;
    int mips = 250;
    long bw = 1000;
    int pesNumber = 1; //Num de procesadores
    String vmm = "Xen";

    //Creamos las máquinas virtuales
    Vm[] vm = new Vm[vms];

    for(int i=0; i<vms; i++){
        //Modificar new CloudletSchedulerTimeShared por FCFS
        if(i%2 == 0){
            vm[i] = new Vm(idShift + i, userId, mips, pesNumber * 2, ram * 2, bw, size, vmm, new CloudletSchedulerSpaceShared());
        } else {
            vm[i] = new Vm(idShift + i, userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerSpaceShared());
        }
        list.add(vm[i]);
    }

    Log.println("Máquinas virtuales creadas... Success");
    return list;
}

```

Figura 11: Características de las máquinas virtuales. Fuente: Elaboración propia.

Continuando con las configuraciones, en la Figura (11) se muestran las características utilizadas en las máquinas virtuales, las cuales son: tamaño de la imagen, memoria RAM virtual, número de instrucciones por segundo, ancho de banda y el número de procesadores.

```

public static List<Cloudlet> createTasks(int userId, int cloudlets, int idShift){
    LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

    Random rand = new Random();
    //parámetros de las tareas
    long length = 10000;
    long fileSize = 300;
    long outputSize = 300;
    int pesNumber = 1;
    UtilizationModel utilizationModel = new UtilizationModelFull();

    Cloudlet[] cloudlet = new Cloudlet[cloudlets];

    for(int i=0; i<cloudlets; i++){
        cloudlet[i] = new Cloudlet(idShift + i, length * getRandomInteger(100, 1),
            pesNumber, fileSize, outputSize, utilizationModel, utilizationModel,
            utilizationModel);
        cloudlet[i].setUserId(userId);
        list.add(cloudlet[i]);
    }

    return list;
}

```

Figura 12: Características de las Tareas (*cloudlets*). Fuente: Elaboración propia.

Además de los recursos, se debe de hacer la configuración de las tareas a simular (Figura 12), teniendo como parámetros el tamaño de la tarea, tamaño

de entrada y de salida, así como el número de procesadores que ocupará la tarea, esto último refleja la complejidad de la tarea.

***Odoo* - Sistema ERP para caso de prueba**

Para obtener una muestra representativa del tamaño de tareas y tiempo de ejecución de las mismas, en este trabajo se hace uso de *Odoo*, que es un sistema completo de gestión empresarial, de código abierto y sin costes de licencias que cubre las necesidades de las áreas de: Contabilidad y Finanzas, Ventas, RRHH, Compras, Proyectos, entre otras. (OpenERPSpain, 2015).

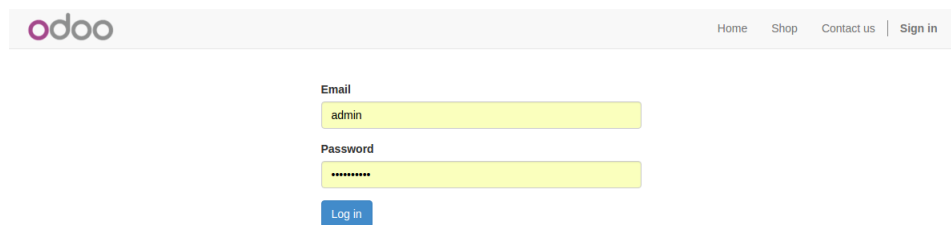


Figura 13: Inicio de sesión *Odoo*. Fuente: Elaboración propia.

Odoo ofrece una plataforma Web (Figura 13), en donde se pueden administrar aplicaciones sobre gestión de recursos empresariales, además de permitir instalar aún más módulos, los cuales se pueden encontrar de manera gratuita o con algunas licencias por parte de sus desarrolladores (Figura 14).

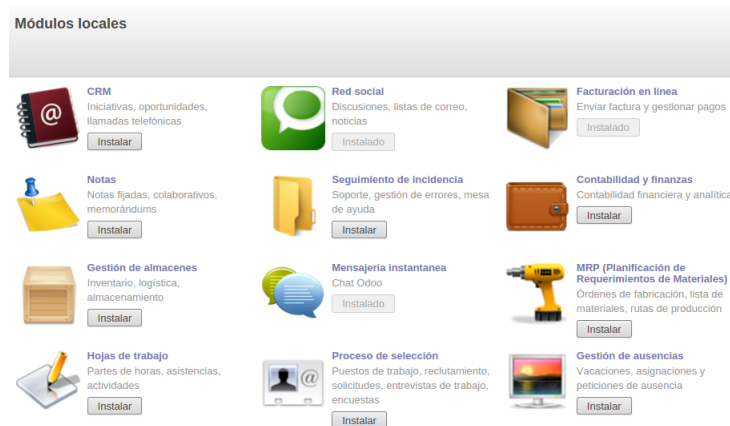


Figura 14: Panel de módulos disponibles. Fuente: Elaboración propia.

Para realizar la medición en tiempo de ejecución y tamaño de las tareas, se hizo uso de un módulo de contabilidad el cual permite generar e imprimir facturas hacia diversos clientes. Además de agregar módulos de RRHH para el levantamiento de usuarios de una empresa (Figura 15).

La imagen muestra una interfaz de usuario con una barra superior que incluye pestañas como 'Contabilidad', 'Recursos Humanos', 'Inventario', 'Seguros' y 'Configuración'. El título principal es 'Facturas de cliente'. Debajo hay un botón 'Clear' y un enlace '+ Importar'. El contenido principal es una tabla con las siguientes columnas: Cliente, Fecha factura, Número, Responsable, Equipo de ventas, Fecha de vencimiento, Documento origen, Saldo pendiente, Sub-Total, Total, Estado. La tabla contiene 7 filas de datos, con la última fila resaltada en azul.

Cliente	Fecha factura	Número	Responsable	Equipo de ventas	Fecha de vencimiento	Documento origen	Saldo pendiente	Sub-Total	Total	Estado
Agrolat, Thomas Passot	08/11/2015	SAJ2015/008	Administrator		08/12/2015		2549.00	2549.00	2549.00	Abierta
ASUSTEK	29/10/2015	SAJ2015/007	Administrator	Ventas indirectas	29/10/2015		0.00	500.00	500.00	Pagado
Best Designers	08/10/2015	SAJ2015/005	Administrator		08/10/2015		525.00	525.00	525.00	Abierta
Agrolat	15/10/2015	SAJ2015/004	Administrator		15/10/2015		500.00	500.00	500.00	Abierta
Agrolat	08/10/2015	SAJ2015/003	Administrator		08/10/2015		525.00	525.00	525.00	Abierta
Best Designers	01/10/2015	SAJ2015/002	Administrator		01/10/2015		650.00	650.00	650.00	Abierta
Best Designers	29/10/2015	SAJ2015/001	Administrator		29/10/2015		4610.00	4610.00	4610.00	Abierta
							9359.00	9359.00	9359.00	

Figura 15: Módulo de contabilidad. Fuente: Elaboración propia.

La figura (16) muestra una factura generada, que fue utilizada para medir el tamaño de las tareas para las peticiones de documentos en el sistema *ERP*, el tiempo de respuesta del servidor va dependiendo del tipo de petición que es generada, que puede ser desde agregar usuarios al sistema hasta generar los reportes para cada cliente o compañía que esté afiliada.

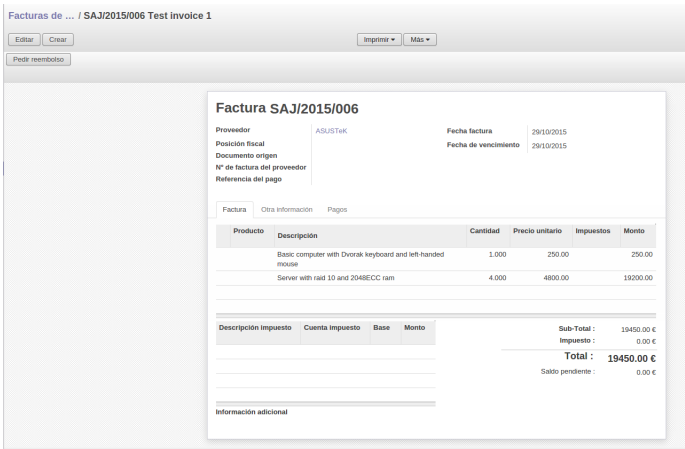


Figura 16: Factura generada en el módulo de contabilidad. Fuente: Elaboración propia.

Una vez realizadas las operaciones se procede a revisar el tiempo de respuesta del servidor así como el tamaño de las tareas para poder adecuar los parámetros en el entorno de simulación, esto a través de las herramientas para desarrolladores del navegador, entre las cuales se permiten listar las tareas y filtrarlas por tiempo de respuesta (Figura 17) y por tamaño de la petición (Figura 18).

<input type="checkbox"/>	4d8c69b8-7e03-11e5-8eb9-08...	301	text/html	f3ede3e550	261 B	1.48 s
<input type="checkbox"/>	991506c	200	script	web:16	2.2 MB	1.45 s
<input type="checkbox"/>	get_session_info	200	xhr	f3ede3e730	511 B	918 ms
<input type="checkbox"/>	4d8c69b8-7e03-11e5-8eb9-08...	200	stylesh...	https://service...	144 B	632 ms
<input type="checkbox"/>	991506c	200	stylesh...	web:15	391 KB	591 ms
<input type="checkbox"/>	message_read	200	xhr	f3ede3e730	82.6 KB	436 ms
<input type="checkbox"/>	load_needaction	200	xhr	f3ede3e730	3.6 KB	358 ms
<input checked="" type="checkbox"/>	web	200	docum...	:8069/web/log..	34.2 KB	318 ms
<input type="checkbox"/>	translations	200	xhr	f3ede3e730	101 KB	309 ms
<input type="checkbox"/>	f3ede3e	200	script	web:13	290 KB	281 ms
<input type="checkbox"/>	get_suggested_thread	200	xhr	f3ede3e730	11.1 KB	224 ms
<input type="checkbox"/>	f3ede3e	200	stylesh...	web:12	23.1 KB	219 ms
<input type="checkbox"/>	load	200	xhr	f3ede3e730	191 KB	208 ms
<input type="checkbox"/>	get_suggested_thread	200	xhr	f3ede3e730	323 B	187 ms
<input type="checkbox"/>	company_logo	200	png	web:138	13.0 KB	161 ms
<input type="checkbox"/>	fields_view_get	200	xhr	f3ede3e730	42.3 KB	160 ms
<input type="checkbox"/>	get_serialised_gamification_s...	200	xhr	f3ede3e730	1.3 KB	159 ms
<input type="checkbox"/>	modules	200	xhr	f3ede3e730	821 B	135 ms
<input type="checkbox"/>	search_read	200	xhr	f3ede3e730	4.0 KB	134 ms
<input type="checkbox"/>	fields_view_get	200	xhr	f3ede3e730	13.3 KB	125 ms
<input type="checkbox"/>	company_logo?db=test&comp...	200	png	Other	13.0 KB	116 ms
<input type="checkbox"/>	fields_view_get	200	xhr	f3ede3e730	13.3 KB	113 ms

Figura 17: Peticiones realizadas en el servidor en orden decreciente por tiempo de respuesta. Fuente: Elaboración propia.

991506c	200	script	web:16	2.2 MB
991506c	200	stylesh...	web:15	391 KB
f3ede3e	200	script	web:13	290 KB
load	200	xhr	f3ede3e:730	191 KB
translations	200	xhr	f3ede3e:730	101 KB
message_read	200	xhr	f3ede3e:730	82.6 KB
fontawesome-webfont.woff?v...	200	font	f3ede3e:636	64.2 KB
fields_view_get	200	xhr	f3ede3e:730	42.3 KB
mnmliconsv21-webfont.ttf	200	font	f3ede3e:645	39.2 KB
web	200	docum...	:8069/web/log..	34.2 KB
entypo-webfont.ttf	200	font	f3ede3e:645	28.8 KB
f3ede3e	200	stylesh...	web:12	23.1 KB
fields_get	200	xhr	f3ede3e:730	19.4 KB
fields_get	200	xhr	f3ede3e:730	19.4 KB
fields_view_get	200	xhr	f3ede3e:730	13.3 KB
fields_view_get	200	xhr	f3ede3e:730	13.3 KB
fields_view_get	200	xhr	f3ede3e:730	13.3 KB
fields_view_get	200	xhr	f3ede3e:730	13.3 KB
company_logo?db=test&comp...	200	png	Other	13.0 KB
company_logo	200	png	web:138	13.0 KB
message_read	200	xhr	f3ede3e:730	12.4 KB
get_suggested_thread	200	xhr	f3ede3e:730	11.1 KB
fields_get	200	xhr	f3ede3e:730	10.4 KB
es-MX.js	200	script	f3ede3e:1125	6.9 KB

Figura 18: Peticiones realizadas en el servidor en orden decreciente por tamaño de la tarea. Fuente: Elaboración propia.

Otra de las características de las herramientas de desarrolladores es que se pueden analizar cada una de las peticiones realizadas hacia el servidor, y obtener la información necesaria para asignar el tamaño de las tareas (Figura 19), en donde el apartado de **Content-Length** muestra el tamaño de la petición para la impresión de una factura en el sistema.

```

▼ Response Headers view source
Cache-Control: must-revalidate, max-age=604800
Content-Length: 2330355
Content-Type: application/javascript
Date: Wed, 04 Nov 2015 23:30:20 GMT
Server: Werkzeug/0.9.4 Python/2.7.6
Set-Cookie: session_id=d51e67e1ad7aff8499abea3e6f67c47ce124e87d; Expires=Tue, 02-Feb-2016 23:30:20 GMT; Max-Age=7776000; Path=/

```

Figura 19: Cabeceras para impresión de Facturas. Fuente: Elaboración propia.

En base a los resultados obtenidos dentro del sistema *Odoo*, el tamaño de las tareas para la simulación quedan establecidas en un intervalo de **10 kb** a **2 mb** como se muestra en la figura (12).

***DatacenterBrokers* para Algoritmos de Calendarización**

Partiendo del Esquema que se muestra en la figura (3), el trabajo presentado se lleva a cabo en la parte de Tareas (*cloudlets*) y Máquinas Virtuales (*Virtual Machines*) ya que se pretende mejorar un algoritmo de calendarización de tareas. Para ello según muestra el diagrama, el encargado de asignar cada tarea en una máquina virtual es el *datacenter broker*, el cual toma la lista de tareas así como la lista de máquinas virtuales, y de acuerdo a algún algoritmo realiza la asignación de cada una de las tareas en las distintas máquinas virtuales disponibles.

```
public class FcfsBroker extends DatacenterBroker {  
    public FcfsBroker(String name) throws Exception {  
        super(name);  
    }  
  
    //scheduling function  
    public void scheduleTaskstoVms(){  
        int reqTasks=cloudletList.size();  
        int reqVms=vmList.size();  
  
        System.out.println("\n\tFCFS Broker Schedules\n");  
        for(int i=0;i<reqTasks;i++){  
            bindCloudletToVm(i, (i%reqVms));  
            System.out.println("Task"+cloudletList.get(i).getCloudletId()+" is bound with VM"  
                +vmList.get(i%reqVms).getId() + "tam: " + cloudletList.get(i).getCloudletFileSize());  
        }  
  
        System.out.println("\n");  
    }  
}
```

Figura 20: *FCFS Broker*. Fuente: Elaboración propia.

La Figura (20) muestra el algoritmo de *FCFS*. Recorre la lista de tareas y la asigna en la máquina virtual correspondiente, calculando la tarea actual i en la máquina virtual $i \% reqVms$ donde $reqVms$ es el número de máquinas virtuales.

```
public class MinminBroker extends DatacenterBroker {  
    public MinminBroker(String name) throws Exception {  
        super(name);  
    }  
  
    //scheduling function  
    public void scheduleTaskstoVms(){  
        int numTareas = cloudletList.size();  
        int numVirtualM = vmList.size();  
  
        System.out.println("\n\tMinmin Broker Schedules\n");  
  
        CloudletList.sort(cloudletList);  
  
        //Asignamos las tareas a cada máquina virtual de manera que se atiendan primero las de menor tamaño  
        for(int i=0;i<numTareas;i++){  
            bindCloudletToVm(i, (i%numVirtualM));  
            System.out.println("Task"+cloudletList.get(i).getCloudletId()+" is bound with VM"  
                +vmList.get(i%numVirtualM).getId() + " tam: " + cloudletList.get(i).getCloudletLength());  
        }  
  
        System.out.println("\n");  
    }  
}
```

Figura 21: *Min-Min Broker*. Fuente: Elaboración propia.

En la Figura (21) se tiene la implementación del algoritmo *Min-Min* en el cual se aprecia el ordenamiento de la lista antes de pasar a la asignación de las máquinas virtuales.


```
public class MaxminBroker extends DatacenterBroker {  
    public MaxminBroker(String name) throws Exception {  
        super(name);  
    }  
  
    //scheduling function  
    public void scheduleTaskstoVms(){  
        int reqTasks=cloudletList.size();  
        int reqVms=vmList.size();  
  
        System.out.println("\n\tMaxmin Broker Schedules\n");  
  
        CloudletList.revert(cloudletList);  
  
        for(int i=0;i<reqTasks;i++){  
            bindCloudletToVm(i, (i%reqVms));  
            System.out.println("Task"+cloudletList.get(i).getCloudletId()+" is bound with VM"  
                +vmList.get(i%reqVms).getId() + "tam: " + cloudletList.get(i).getCloudletLength());  
        }  
        System.out.println("\n");  
    }  
}
```

Figura 22: *Max-min Broker*. Fuente: Elaboración propia.

En la Figura (22) se observa la implementación del algoritmo de *Max-Min* el cual hace un ordenamiento de mayor a menor, véase la función *revert* utilizada, la cual se explicará más adelante.

```
public static <T extends Cloudlet> void sort(List<T> cloudletList) {
    Collections.sort(cloudletList, new Comparator<T>() {

        @Override
        public int compare(T a, T b) throws ClassCastException {
            Double cla = Double.valueOf(a.getCloudletTotalLength());
            Double clb = Double.valueOf(b.getCloudletTotalLength());
            return cla.compareTo(clb);
        }
    });
}

public static <T extends Cloudlet> void revert(List<T> cloudletList) {
    Collections.sort(cloudletList, new Comparator<T>() {

        @Override
        public int compare(T o1, T o2) {
            Double cla = Double.valueOf(o1.getCloudletTotalLength());
            Double clb = Double.valueOf(o2.getCloudletTotalLength());
            return clb.compareTo(cla);
        }
    });
}
```

Figura 23: *Algoritmos de ordenamientos*. Fuente: Elaboración propia.

En la Figura (23) se tiene la implementación de los métodos de ordenamiento utilizados, el **sort** utilizado para ordenar las tareas de menor a mayor de acuerdo al tamaño y el **revert** de forma inversa.

```

public void scheduleTasksToVms(){
    int numTareas = cloudletList.size();
    int numVirtualM = vmList.size();

    Log.println("Round Robin Scheduler");
    int index = 0;
    int i = 0;
    while( index < numTareas ){
        long capacidad = vmList.get(i).getBw();
        long costo_actual = 0;

        while(costo_actual + cloudletList.get(index).getCloudletLength() <= capacidad && index < numTareas){
            costo_actual += cloudletList.get(index).getCloudletLength();
            bindCloudletToVm(index, i);
            System.out.println("Task" + cloudletList.get(index).getCloudletId() +
                " is bound with VM" + vmList.get(i).getId() + " tam: " +
                cloudletList.get(index).getCloudletLength());
            index++;

            if(index >= numTareas) {
                //Si terminamos de asignar todas las tareas
                break;
            }
        }

        i++;

        if(i >= numVirtualM) {
            //Reiniciamos la ronda de asignación a la máquina 0
            i = 0;
        }
    }
}

```

Figura 24: *Round Robin Broker*. Fuente: Elaboración propia.

En la Figura (24) se muestra la implementación del *Broker* para el algoritmo de *Round Robin*, en el cual se hace una variación con respecto al *Quantum* de tiempo hacia disponibilidad de ancho de banda de la máquina virtual.

2.1.3. Mejorar el costo de procesamiento y el tiempo de ejecución

El objetivo de éste proyecto es minimizar el tiempo de ejecución y el costo de procesamiento de acuerdo a los recursos heterogéneos del centro de datos. Para ello se hará la implementación de una *metaheurística* con un método llamado Optimización por Enjambre de Partículas *PSO* (*Particle Swarm Optimization*).

Optimización por Enjambre de Partículas (*PSO*) es una técnica de optimización basada en la búsqueda de un óptimo global, introducida por Kennedy and Eberhart Pandey et al. (2010).

Se eligió *PSO* debido al concepto simple que tiene Poli et al. (2007), la

implementación requiere de simples operaciones matemáticas que en términos de computación no consume memoria y la velocidad de convergencia es relativamente rápida dependiendo del número de partículas Eberhart and Kennedy (1995).

Está claro que como toda *metaheurística*, *PSO* no garantiza la obtención de una solución óptima en todos los casos (Osman and Kelly (2012)).

Algoritmo *PSO* y un ambiente en la nube

PSO permite optimizar un problema en base a una población de soluciones candidatas, que llevan el término “*partículas*”, dichas partículas se mueven en un espacio de búsqueda (tienen en cuenta dos aspectos; la posición y la velocidad). Cada una realiza un movimiento influido por su mejor posición local hasta ése momento y por las mejores posiciones globales encontradas en todo el enjambre mientras recorren el espacio de búsqueda (Poli et al. (2007)).

Una solución potencial es representada en *PSO* por un vector (partícula). Cada partícula i tiene una velocidad y una posición, denotados por las siguientes ecuaciones Pandey et al. (2010):

$$v_i^{k+1} = wv_i^k + c_1rand_1(pbest_i - x_i^k) + c_2rand_2(gbest - x_i^k) \quad (2.1)$$

$$x_i^k = x_i^k + v_i^{k+1} \quad (2.2)$$

Donde :

1. w inercia
2. c_j coeficiente de aceleración; $j = 1, 2$
3. $rand_i$ número aleatorio entre 0 y 1
4. $pbest_i$ La mejor posición de la partícula i
5. $gbest$ La mejor posición de partícula en el enjambre

Para la simulación en *CloudSim* éstas partículas serán las tareas en el centro de datos, $= \{T_1, T_2, \dots T_n\}$.

En la ecuación 2.1 se describe cómo será calculado la velocidad de la partícula i en la iteración $k + 1$. El cual depende de la mejor posición local

hasta ese momento y la mejor posición en el enjambre hasta ese momento, se tiene un parámetro de inercia que para éste proyecto será uno, el coeficiente de aceleración será entre uno y dos, quiere decir que nos moveremos como máximo dos posiciones en nuestro espacio de recursos.

Para la implementación del algoritmo en *CloudSim* dicha posición será dado por el conjunto de máquinas virtuales del centro de datos $VM = \{1, \dots, j\}$, es decir el espacio de búsqueda para las partículas.

La ecuación 2.2 se muestra como se actualiza la posición, que es la suma de la posición de la partícula i en la iteración k y la velocidad de la partícula i en la iteración $k + 1$.

Otro concepto importante de la técnica PSO, es cómo se implementa la función *fitness*, que en el desarrollo de ésta simulación, será dado por el costo de procesamiento de la partícula i en la máquina virtual VM_j durante la iteración k .

$$Cost = (MIPSVirtualMachine * MICloudlet) * costpersecond \quad (2.3)$$

Para el cálculo de ésta valor *fitness* es necesario realizar el estimado de cuánto costará ejecutar la tarea i en la máquina virtual j , antes de enviarla a la máquina virtual para ello fue necesario establecer dicho valor con ayuda de la ecuación 2.3.

En donde se pone la relación de los *MIPS* de la máquina virtual con los *MI* que posee la tarea.

Algorithm 1 Algoritmo PSO en la simulación

- 1: El número de partículas será igual al número de tareas en $\{t_i\} \in T$
 - 2: Se inicializa la posición de las partículas aleatoriamente de $VM = 1, \dots, j$ y con velocidad aleatoria v_i
 - 3: Para cada partícula, calcular *fitnessvalue*.
 - 4: Si *fitnessvalue* es mejor que el anterior *pbest*, igualar *fitnessvalue* como el nuevo *pbest*.
 - 5: Seleccionar la mejor partícula como *gbest*.
 - 6: Para todas las partículas, calcular la velocidad y actualizar su posición.
 - 7: Si no se alcanza el máximo número de iteraciones, repetir el paso 3.
-

En el algoritmo (1) se muestran los pasos del PSO. Inicia con una inicialización aleatoria de la posición y velocidad de las partículas. Para la simulación, las partículas serán las tareas en el centro de datos. El valor asignado a la dimensión de las partículas son los índices de los recursos (Máquinas Virtuales). Por lo tanto, cada tarea representa un mapeo de una tarea en todas las máquinas virtuales. Entonces la dimensión de una partícula depende del número de máquinas virtuales. Para cada partícula se calcula un *fitnessvalue* que es el óptimo local al nivel de partícula que se evalúa y se actualiza. Después se evalúa la mejor partícula como *gbest* que sería el óptimo del enjambre. Se delimita un número m , mientras no se llegue al límite la iteración continua; esto para no entrar a un ciclo que nunca termina, ya que PSO no garantiza la obtención de una solución óptima en todos los casos (Osman and Kelly (2012)).

Algorithm 2 Algoritmo heurístico propuesto

- 1: Se crean las tareas para el centro de datos $\{t_i\} \in T$
 - 2: El número de partículas será igual al número de tareas en $\{p_i\} \in T$
 - 3: Se ordenan las partículas de menor a mayor tamaño
 - 4: Se ejecuta *PSO* con enjambre actual $P\{p_1..p_n\}$
 - 5: La partícula candidata, resultado de PSO, se asigna a la máquina virtual seleccionada.
 - 6: Se elimina dicha partícula del enjambre.
 - 7: Se ejecuta PSO con el enjambre actualizado
 - 8: Mientras queden partículas por calendarizar, repetir el paso 5.
-

En el algoritmo 2 se aprecia el esquema propuesto, en donde inicialmente las partículas son un mapeo de todas las tareas en el centro de datos, es decir la clase *cloudlet* se extiende y se agregan los componentes de una partícula. Después las partículas son ordenadas por tamaño para así ejecutar PSO y calendarizar el resultado hasta que todas las tareas están asignadas a su respectiva máquina virtual candidata.

Capítulo 3

Resultados

Introducción

En éste capítulo se describe los resultados obtenidos en modelado y simulación del centro de datos en *CloudSim*. Se ilustran por medio de gráficas lineales el tiempo de ejecución de los algoritmos de calendarización. Se representa en gráficas de barras el promedio del costo de procesamiento y tiempo de ejecución y una interpretación de la desviación estándar obtenida. La experimentación fue desarrollada en una computadora con un procesador *Intel Core i5*, teniendo la siguiente configuración: *2.5 GHz*, *3 MB de caché*, *4 GB de RAM* y *JDK 8.6*.

3.1. Tiempo de ejecución y costo de procesamiento

Para evaluar los algoritmos de calendarización seleccionados se ha implementado un entorno de cómputo en la nube que consiste en un *datacenter*, un *broker*, máquinas virtuales y *host*. Con el objetivo de evaluar el tiempo de ejecución y el costo de procesamiento en el *datacenter* tras ejecutar cierto número de tareas (*cloudlets*).

Para la configuración del centro de datos en *CloudSim*, se utilizaron cincuenta *host*, quince máquinas virtuales por *host* y las tareas fueron establecidas en un intervalo de 100 hasta 500 (Cuadro 1). Como se puede apreciar en el (Cuadro 2), cada *host* tuvo 20480 Mb de memoria RAM, ocho núcleos de procesamiento y tienen un almacenamiento de 800 GB ó 1 TB, estas fueron elegidas de manera aleatoria y finalmente el ancho de banda fue de 10 GB/s.

Datacenter	
Host	50
VM	15
Cloudlet	100 - 500

Cuadro 1: Configuración *Datacenter*, Fuente: Elaboración propia.

Host	
RAM	20480 MB
CPU	8
Storage	800GB - 1TB
BW	10 GB/s

Cuadro 2: Configuración de *Host*, Fuente: Elaboración propia.

VirtualMachine	
RAM	512 MB — 2GB
vCPU	2
Storage	10 GB
BW	1 GB/s

Cuadro 3: *Virtual Machine*, Fuente: Elaboración propia.

Cloudlet	
length	100 - 1000 MI
fileSize	1KB - 2MB
outputSize	1KB - 2MB

Cuadro 4: Configuración *Cloudlet*, Fuente: Elaboración propia.

En el Cuadro (3) se muestran los parámetros que se consideró para las máquinas virtuales, donde cada *VM* tendrá 10 GB de almacenamiento, 512 MB o 2 GB seleccionado de manera aleatoria, además para la característica del procesador se tiene la propiedad *MIPS* (*million instructions per second*) con 250 ó 500 y un ancho de banda de 1 GB/s.

Para la configuración de las tareas, se tomó el tamaño con un intervalo de 100 MI a 1000 MI de acuerdo al tamaño mínimo y máximo en el ERP *Odoo*, el parámetro *fileSize*, que representa el tamaño del archivo de entrada, va de 1 kb a 2 mb así como el archivo de salida, como parámetro final se tiene los *outputSize* que irán de 1 kb a 2 mb (Cuadro 5).

Amazon EC2 t2.medium	
vCPU	2
RAM	2 GB
Costo por hora	\$0.052 USD

Cuadro 5: Configuración *Cloudlet*, Fuente: Elaboración propia.

Los *Cloudlets* en *CloudSim* tienen un parámetro configurable que permite establecer un costo de procesamiento de manera monetaria. Para ello se con-

sultó con el proveedor de servicios Amazon con el objetivo de encontrar una instancia equivalente a las máquinas virtuales establecidas en la simulación.

Se encontró que la *Amazon EC2 t2.medium* cumple con las características de la simulación (Amazon,2015).

El costo de la instancia en Amazon es \$0.052 USD, entonces se configuró la característica de los *Cloudlets* con una tarifa de \$ $1.5e^{-5}$ USD por segundo.

3.2. Simulación en *CloudSim*

Al tener la configuración completa del centro de datos en *CloudSim*, se procedió a realizar la simulación. Para la simulación, el programa fue ejecutado 30 veces y se ordenó ascendentemente la suma total del tiempo de ejecución de cada simulación. Se eliminaron las dos con menor tiempo y las dos con mayor tiempo, con el fin de evitar sesgos ya sea de manera positiva o negativa.

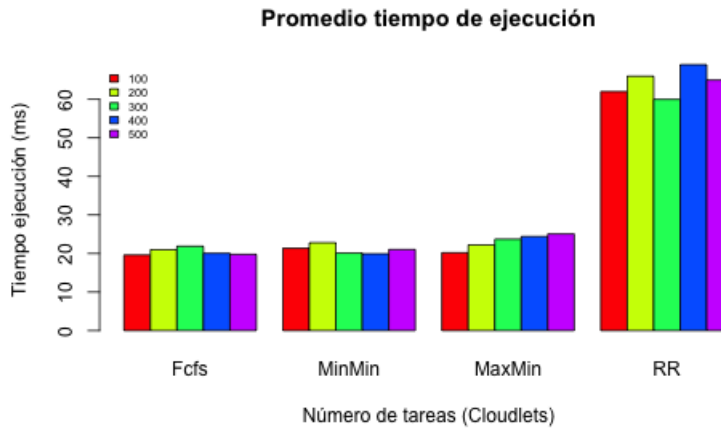


Figura 25: Promedio tiempo de ejecución con tareas 100-500, Fuente: Elaboración propia.

En la figura (25), se puede observar el promedio del tiempo de ejecución en *ms* para diferentes cantidades de tareas (de 100 a 500). A primera vista con el algoritmo *FCFS* y *Min-Min* se mantiene un tiempo de ejecución sin muchos cambios a pesar del aumento en la carga de tareas, a diferencia del algoritmo *Max-Min* que aumentó el tiempo de ejecución a medida que se incrementó el número de tareas. Sin embargo para el algoritmo *Round Robin* tiene el tiempo de ejecución a un poco más del doble de los anteriores. Esto es porque la tarea es procesada por diferentes máquinas virtuales.

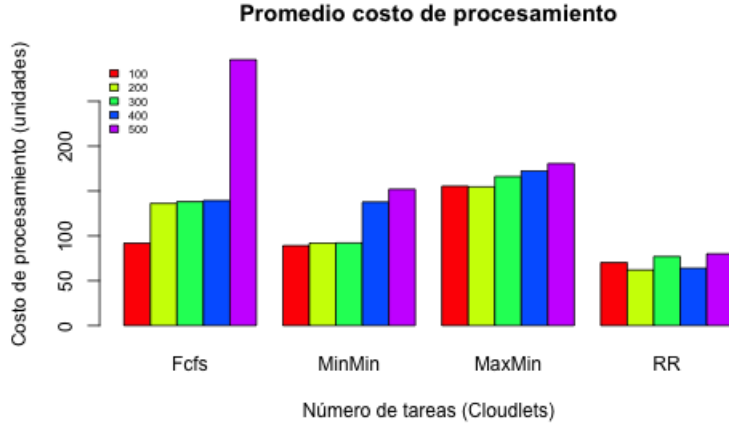


Figura 26: Promedio costo de procesamiento con tareas 100-500, Fuente: Elaboración propia.

El costo de procesamiento, de acuerdo a cada algoritmo, se puede observar en la figura (26), en donde el algoritmo *FCFS* tiene un incremento drástico al realizar la prueba con 500 tareas. El algoritmo *Max-Min* se conservó sin muchos cambios a pesar de la cantidad de tareas, mientras que *Min-Min* tiene un menor costo de procesamiento cuando las tareas son inferiores a 300. Pero el algoritmo *Round Robin* tiene un menor costo de procesamiento, ya que lo ejecutado en cada máquina virtual es la fracción de un tarea (*Cloudlet*), por ende los *MI* procesados son menores.

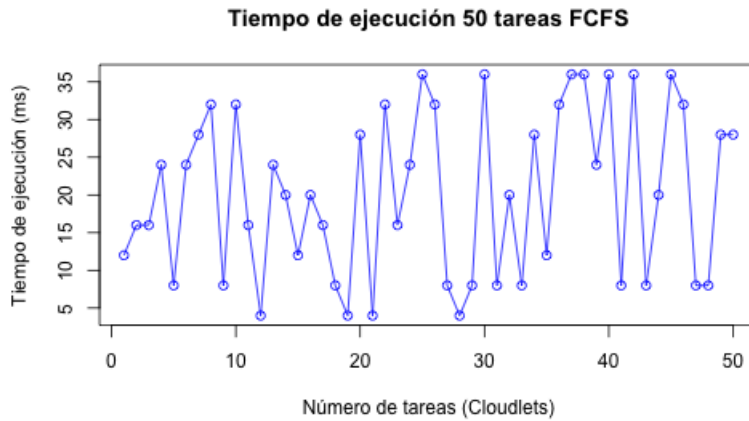


Figura 27: Tiempo ejecución 50 muestras *FCFS*, Fuente: Elaboración propia.

Para mostrar el comportamiento del tiempo de ejecución por cada algoritmo, se observó una ventana de 50 muestras de una simulación de 500 tareas. En la figura (27), se puede apreciar que el algoritmo *FCFS* tiene un comportamiento inestable ya que algunas tareas pueden tener menor complejidad o tamaño, lo que implica una respuesta rápida.

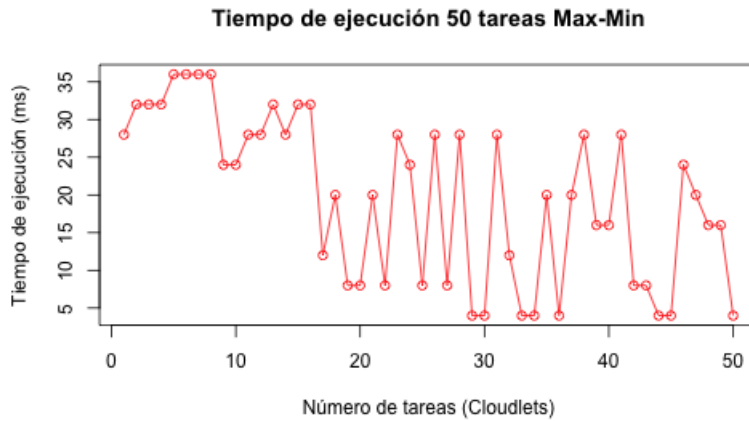


Figura 28: Tiempo ejecución 50 muestras *Max-Min*, Fuente: Elaboración propia.

En la figura (28) se tiene la misma simulación pero con el algoritmo *Max-Min*, de acuerdo a las características de este calendarizador, en las primeras tareas se toma un mayor tiempo en responder y va disminuyendo de manera gradual, sin embargo aún es inestable en las últimas muestras ya que no se contempla el grado de complejidad, es decir el parámetro *MI* de los *cloudlets*.

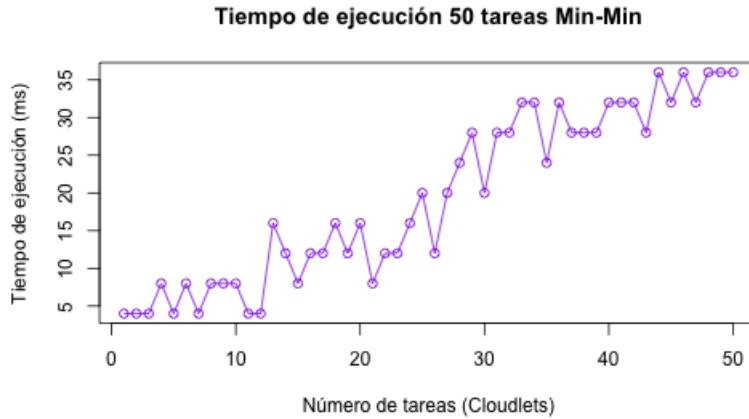


Figura 29: Tiempo ejecución 50 muestras *Min-Min*, Fuente: Elaboración propia.

En la parte de arriba se puede apreciar el algoritmo *Min-Min* en el que el tiempo de ejecución fue aumentando conforme se resolvían las tareas (figura 29). Por último, en la figura (30) podemos visualizar la gráfica correspondiente al algoritmo *Round Robin*. Para éste último se observan picos en el tiempo de ejecución, esto es porque la tarea es fraccionada y para pasar al estado de *COMPLETADA* cada fracción debe ser ejecutada por la máquina virtual correspondiente.

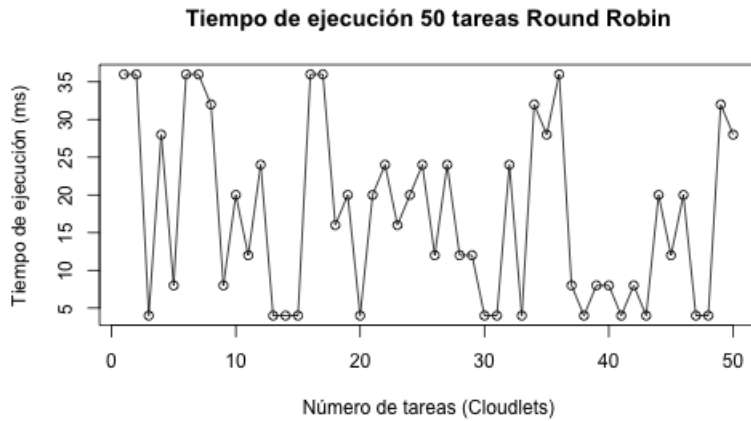


Figura 30: Tiempo ejecución 50 muestras *Round Robin*, Fuente: Elaboración propia.

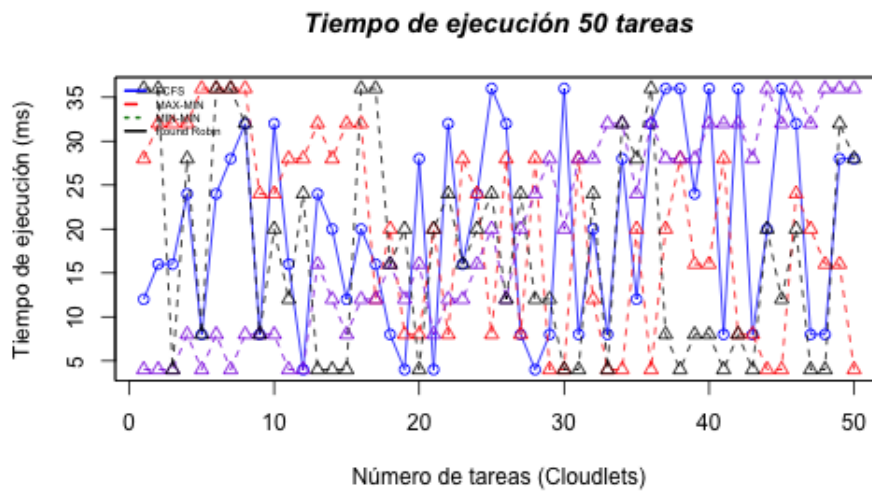


Figura 31: Tiempo ejecución 50 muestras de todos los algoritmos, Fuente: Elaboración propia.

La figura (31) muestra una comparativa de los cuatro algoritmos mostrados anteriormente. Durante la evaluación de los algoritmos se tomaron en cuenta dos datos estadísticos: la desviación estandar y el promedio que fue

descrito en las páginas 51 y 52.

Algoritmo	Desviación estándar
FCFS	11.00619
MAX-MIN	8.91444
MIN-MIN	11.25613
ROUND ROBIN	11.66722

Cuadro 6: Desviación estándar del tiempo de ejecución, Fuente: Elaboración propia.

Observando la desviación estándar de éstas muestras anteriores, los algoritmo *Min-Min*, *Round Robin* y *FCFS* tuvieron más variaciones en las muestras con respecto a la media, mientras que el *Max-Min* tuvo las variaciones por debajo de las dos anteriores (Cuadro 6).

Por último se realizó la simulación contemplando las características de las tareas, pero sin procesamiento previo antes de calendarizar.

De igual forma se hizo la simulación del algoritmo propuesto en el capítulo dos. Para ello se realizaron diez simulaciones, descartando la de mejor y peor resultado.

En la figura (32) se aprecia el tiempo de ejecución del algoritmo propuesto y de la simulación sin ningún procesamiento previo. Con la heurística el promedio del tiempo de ejecución se reduce de 80 *ms* a 60 *ms*, sin embargo es el tiempo que le lleva al centro de datos procesar esa cantidad de tareas, pero no se tiene en cuenta el tiempo previo para la búsqueda de partícula óptima. En promedio éste proceso se toma entre 20 *ms* a 30 *ms* por simulación.

En cuanto al costo de procesamiento, las simulaciones que plasmaron el resultado esperado; fue menor de 50 unidades como se muestra en la figura (33). Comparado con *Round Robin* que tenía un costo de entre 60 a 80 unidades, existe una mejoría.

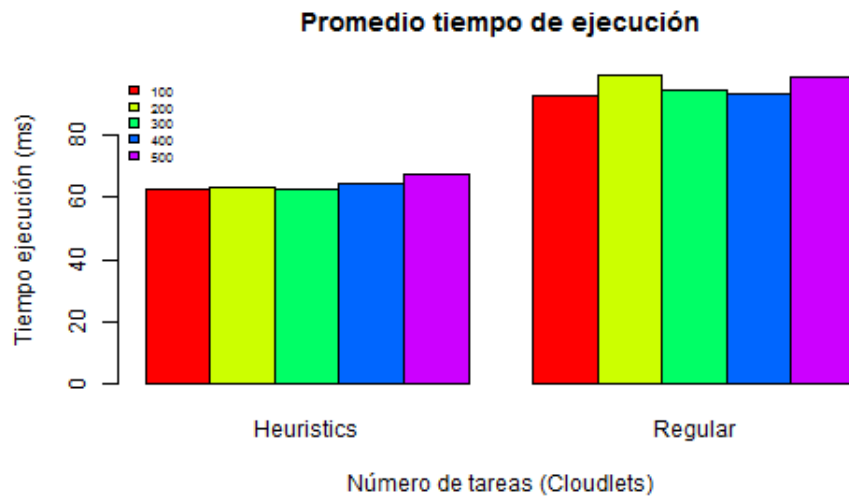


Figura 32: Tiempo ejecución con el algoritmo propuesto (izqda.) y el tiempo de ejecución sin procesamiento previo (dcha.), Fuente: Elaboración propia.

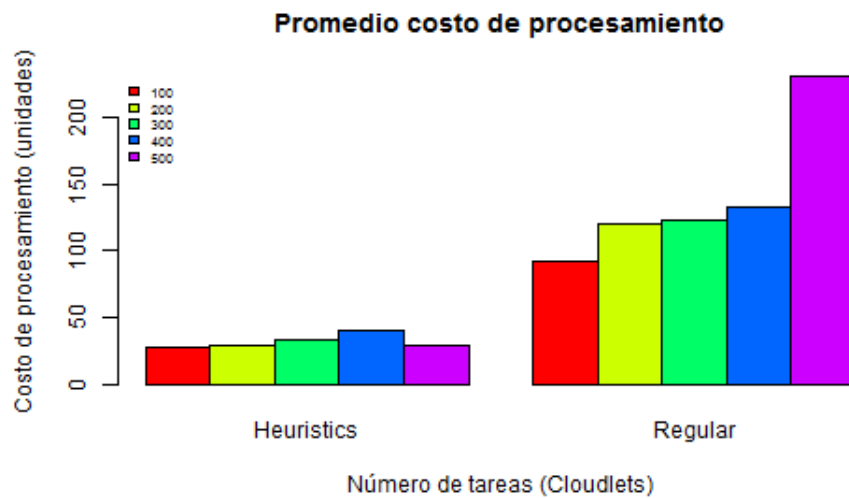


Figura 33: Costo de procesamiento con el algoritmo propuesto (izqda.) y costo de procesamiento sin procesamiento previo (dcha.), Fuente: Elaboración propia.

Conclusiones

En éste proyecto, se presentó una serie de esquemas de distribución con implementaciones sencillas para seleccionar *Min-Min* y complementarlo con una metaheurística llamada *Particle Swarm Optimization (PSO)*. Se usó esta técnica con el objetivo de minimizar el costo de procesamiento y el tiempo de ejecución en un centro de datos con un entorno en la nube. Se encontró que el esquema propuesto reduce el costo de procesamiento en 50 % en comparación con una calendarización sin procesamiento previo. Sin embargo, se determinó que el mapeo de cada partícula en los recursos del centro de datos tiene un costo en tiempo de ejecución, ya que no se observó mejoría. Por lo tanto, el costo de procesamiento de una tarea en un recurso del centro de datos (en éste caso las máquinas virtuales) es inversamente proporcional al tiempo que se toma en resolver la tarea. PSO encuentra los recursos del centro de datos en donde las tareas a procesar tendrán un menor costo. Ésta heurística tiene un carácter genérico es decir, puede ser usado para cualquier número de tareas en el centro de datos y los recursos pueden tener cualquier dimensión. Ya que simplemente se incrementa el número de partículas y el espacio de búsqueda sería mayor. Como trabajo a futuro, sería interesante analizar el comportamiento del costo de procesamiento con variaciones en las políticas de alojamiento de las máquinas virtuales.

Bibliografía

- Agarwal, S., Jain, S., et al. (2014). Efficient optimal algorithm of task scheduling in cloud computing environment. *arXiv preprint arXiv:1404.2076*. Recuperado 2015-08-21.
- Aranganathan, S. and Mehata, K. (2011). An aco algorithm for scheduling data intensive application with various qos requirements. *Int. J. Comput. Appl*, 27(10). Recuperado 2015-08-20.
- Calheiros, R. N. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.
- Chen, H., Wang, F., Helian, N., and Akanmu, G. (2013). User-priority guided min-min scheduling algorithm for load balancing in cloud computing. In *Parallel Computing Technologies (PARCOMPTECH), 2013 National Conference on*, pages 1–8. IEEE. Recuperado 2015-08-23.
- Chen, Q. and Deng, Q. (2009). Cloud computing and its key techniques. *Journal of Computer Applications*. Recuperado 2015-08-20.
- Chowdhury, A., Ahmed, M., Ahmed, M., and Rafee, M. M. H. (2012). An advanced survey on cloud computing and state-of-the-art research issues. *IJCSI International Journal of Computer Science Issues*, 9(1):1694–0814. Recuperado 2015-08-20.
- Dakshayini, M. and Guruprasad, H. (2011). An optimal model for priority based service scheduling policy for cloud computing environment. *International Journal of Computer Applications*, 32(9):23–29. Recuperado 2015-08-23.

- Delavar, A. G., Javanmard, M., Shabestari, M. B., and Talebi, M. K. (2012). Rsdsc (reliable scheduling distributed in cloud computing). *International Journal of Computer Science, Engineering and Applications (IJCSEA)* Vol, 2. Recuperado 2015-08-22.
- Dover, Z., Gordon, S., and Hildred, T. (2013). The technical architecture of red hat enterprise virtualization environments. Recuperado 2015-08-22.
- Eberhart, R. C. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY.
- EPI (2015-08-31). What is a data centre. http://www.epi-ap.com/content/28/48/What_is_a_Data_Centre.
- Li, J., Feng, L., and Fang, S. (2014). An greedy-based job scheduling algorithm in cloud computing. *Journal of Software*, 9(4):921–925. Recuperado 2015-08-21.
- Mariscal, J. and Gil-Garcia, J. (2013). El cómputo en la nube en méxico: Alcances y desafíos para los sectores público y privado. cide. Recuperado 2015-08-21.
- Naylor, T. H. and Finger, J. M. (1967). Verification of computer simulation models. *Management Science*, 14(2):B–92.
- Nishant, S. and R., K. (2015). Pre-emptable shortest job next scheduling in private cloud computing” in journal of information, knowledge and research computer engineering. In *Knowledge and research computer engineering*. Recuperado 2015-08-20.
- OpenERPSpain (2015-09-27). ¿qué es openerp? <http://openerpspain.com/openerp/que-es-openerp>.
- Osman, I. H. and Kelly, J. P. (2012). *Meta-heuristics: theory and applications*. Springer Science & Business Media.
- Pandey, S., Wu, L., Guru, S. M., and Buyya, R. (2010). A particle swarm optimization-based heuristic for scheduling workflow applications in cloud

- computing environments. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 400–407. IEEE.
- Parsa, S. and Entezari-Maleki, R. (2009). Rasa-a new grid task scheduling algorithm. *JDCTA*, 3(4):91–99. Recuperado 2015-08-22.
- Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. *Swarm intelligence*, 1(1):33–57.
- S., S. and Horacio, R. (2002). *Sistemas de información en la era digital*. Fundación OSDE. Recuperado 2015-08-22.
- Salot, P. (2013). A survey of various scheduling algorithm in cloud computing environment. *IJRET: International Journal of Research in Engineering and Technology, ISSN*. Recuperado 2015-08-21.
- Shannon, R. and Johannes, J. D. (1976). Systems simulation: the art and science. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(6):723–724.
- Shimpy, E. and Sidhu, M. J. (2014). Different scheduling algorithms in different cloud environment. *algorithms*, 3(9). Recuperado 2015-08-22.
- Srinivasan, S. (2014). Cloud computing evolution. In *Cloud Computing Basics*, pages 1–16. Springer. Recuperado 2015-08-31.

Glosario

A

- **Amazon:** Compañía de comercio electrónico y servicios de computación en la nube.

C

- **Cloud Computing:** cómputo en la nube, tiene como objetivo ofrecer servicios a través de Internet.
- **Cloudlet:** Es una representación de tareas en Cloudsim.
- **CloudSim:** Framework para el modelado y simulación de la infraestructura y los servicios de Cloud Computing.
- **Cluster:** Conjunto de computadoras unidas entre sí normalmente por una red de alta velocidad y que se comportan como si fuese una única computadora.

E

- **ERP:** Planificador de recursos empresariales, son sistemas informáticos destinados a la administración de recursos de una organización.

F

- **Framework:** Marco de trabajo, es un conjunto de clases que implementa todos los servicios comunes de un cierto tipo de aplicación.

H

- **Heurística:** Tipo de búsqueda en la cual se añade información, basándose en el espacio estudiado hasta determinado momento.

P

- **Pruebas de software/hardware:** técnicas cuyo objetivo es evaluar la calidad del software/hardware.

S

- **Simulación:** Representación matemática y/o computacional de un sistema a través de un modelo establecido para comprender o evaluar estrategias dentro de los límites impuestos por un criterio o conjunto de ellos.

Anexos

En ésta sección se presentan los anexos utilizados para el desarrollo del proyecto: código fuente de los algoritmos implementados y de otras herramientas complementarias.

Simulación básica en *CloudSim*

```
/*
 * Title:      CloudSim Toolkit
 * Description: CloudSim (Cloud Simulation) Toolkit for Modeling
 *              and Simulation
 *              of Clouds
 * Licence:    GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */
package FCFS;

import java.util.*;

import common.CloudletUtilities;
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;
import common.VirtualMachineCreator;
import common.CloudletCreator;
import common.DataCenterCreator;
/**
 * FCFS Task scheduling
 * @author
 * Canul, Cindy
 * Kumul, Cristian
 * Peraza, Jonathan
 */
public class FCFS {

    //Lista de tareas
    private static List<Cloudlet> cloudletList;

    // Lista de maquinas virtuales
    private static List<Vm> vmList;

    public static List<Cloudlet> main(String[] args) {
```

```
Log.println("Iniciando FCFS...");

try {
    // Primer paso: Se inicializa CloudSim

    int num_user = 1; // numero de usuarios
    Calendar calendar = Calendar.getInstance();
    boolean trace_flag = false; // característica de seguimiento de
        eventos desactivada

    // La libreria CloudSim es iniciada
    CloudSim.init(num_user, calendar, trace_flag);

    // Segundo paso: Crear el centro de datos
    // Los centros de datos son los proveedores de recursos en CloudSim
    // Necesitamos a la lista uno de ellos para ejecutar una
    // simulación en CloudSim

    Datacenter datacenter0 = new
        DataCenterCreator().createDatacenter("Datacenter_0");

    // Tercer paso: Crear un broker
    // DatacenterBroker broker = createBroker();
    FcfsBroker broker = createBroker();
    int brokerId = broker.getId();

    // Cuarto paso: Crear las maquinas virtuales
    // vmList = new VmsCreator().createVirtualMachines(brokerId);
    vmList = new VirtualMachineCreator().createVM(brokerId, 15, 0);
    // se crean las tareas
    cloudletList = new CloudletCreator().createTasks(brokerId, 500, 0);

    Log.println("Initial Cloudlet :"+cloudletList.size());

    // se envia la lista de maquinas virtuales al broker
    broker.submitVmList(vmList);

    // Quinto paso: Enviar las tareas al broker
```

```
broker.submitCloudletList(cloudletList);

//llamar a la funcion del broker encargada de calendarizar las
//tareas a las m\'aquinas virtuales
broker.scheduleTaskstoVms();

//Sexto paso: Iniciar la simulacion
CloudSim.startSimulation();

// Paso final: Obtener el listado de las tareas completadas
List<Cloudlet> newList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

CloudletUtilities.printCloudletList(newList);

return newList;
//Log.println("FCFS finished!");
}
catch (Exception e) {
e.printStackTrace();
Log.println("The simulation has been terminated due to an
unexpected error");
}
return null;
}

// funcion que crea un broker del tipo especifico para cada
// algoritmo
private static FcfsBroker createBroker(){

FcfsBroker broker = null;
try {
broker = new FcfsBroker("Broker");
```

```
} catch (Exception e) {  
    e.printStackTrace();  
    return null;  
}  
return broker;  
}  
  
}
```

Cloud Brokers

FCFS Broker

```

package FCFS;

import PSO.Particle;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.Host;
import PSO.FitnessFunction;

/**
 * FCFS Task scheduling
 * @author
 * Canul, Cindy
 * Kumul, Cristian
 * Peraza, Jonathan
 */
public class FcfsBroker extends DatacenterBroker {

    public FcfsBroker(String name) throws Exception {
        super(name);
        // TODO Auto-generated constructor stub
    }

    //Funcion de calendarizacion
    public void scheduleTaskstoVms(){
        int reqTasks=cloudletList.size();
        int reqVms=vmList.size();

        System.out.println("\n\tFCFS Broker Schedules\n");
        for(int i=0;i<reqTasks;i++){
            //Asignacion de las tareas a las maquinas virtuales
            bindCloudletToVm(i, (i%reqVms));
            System.out.println("Task"+cloudletList.get(i).getCloudletId()+" is
                bound with VM"+vmList.get(i%reqVms).getId() + "tam: " +
                cloudletList.get(i).getCloudletFileSize());
        }
    }

```

```
System.out.println("\n");
}
}
```

Max-Min Broker

```
package Maxmin;

import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.lists.CloudletList;
/**
 * FCFS Task scheduling
 * @author
 * Canul, Cindy
 * Kumul, Cristian
 * Peraza, Jonathan
 */
public class MaxminBroker extends DatacenterBroker {

    public MaxminBroker(String name) throws Exception {
        super(name);
        // TODO Auto-generated constructor stub
    }

    //funcion de calendarizacion
    public void scheduleTaskstoVms(){
        int reqTasks=cloudletList.size();
        int reqVms=vmList.size();

        System.out.println("\n\tMaxmin Broker Schedules\n");

        CloudletList.revert(cloudletList);

        for(int i=0;i<reqTasks;i++){
            bindCloudletToVm(i, (i%reqVms));
        }
    }
}
```

```
System.out.println("Task"+cloudletList.get(i).getCloudletId()+" is  
    bound with VM"+vmList.get(i%reqVms).getId() + "tam: " +  
    cloudletList.get(i).getCloudletLength());  
}  
  
System.out.println("\n");  
}  
}
```

Min-Min Broker

```
package Minmin;  
  
import org.cloudbus.cloudsim.DatacenterBroker;  
import org.cloudbus.cloudsim.Log;  
import org.cloudbus.cloudsim.lists.CloudletList;  
  
/**  
 * FCFS Task scheduling  
 * @author  
 * Canul, Cindy  
 * Kumul, Cristian  
 * Peraza, Jonathan  
 */  
  
public class MinminBroker extends DatacenterBroker {  
  
    public MinminBroker(String name) throws Exception {  
        super(name);  
        // TODO Auto-generated constructor stub  
    }  
  
    //funcion de calendarizacion  
    public void scheduleTaskstoVms(){  
        int numTareas = cloudletList.size();  
        int numVirtualM = vmList.size();
```



```

System.out.println("\n\tMinmin Broker Schedules\n");

CloudletList.sort(cloudletList);

//Asignamos las tareas a cada maquina virtual de manera que se
//atiendan primero las de menor tama-o
for(int i=0;i<numTareas;i++){
bindCloudletToVm(i, (i%numVirtualM));
System.out.println("Task" + cloudletList.get(i).getCloudletId() +
    " is bound with VM" + vmList.get(i % numVirtualM).getId() + "
    tam: " + cloudletList.get(i).getCloudletLength());
}
}
}

```

Round Robin Broker

```

package RoundRobin;

import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.lists.CloudletList;

/**
 * FCFS Task scheduling
 * @author
 * Canul, Cindy
 * Kumul, Cristian
 * Peraza, Jonathan
 */
public class RoundRobinBroker extends DatacenterBroker{

    public RoundRobinBroker(String name) throws Exception {
        super(name);
    }

    public int get_quantum(){

```

```
int numTareas = cloudletList.size();
int resultado = 0;

for(int i = 0; i < numTareas; i++){
    resultado += cloudletList.get(i).getActualCPUTime();
}

return resultado /= numTareas;
}

public void scheduleTasksToVms(){
    int numTareas = cloudletList.size();
    int numVirtualM = vmList.size();

    Log.println("Round Robin Scheduler");
    int index = 0;
    int i = 0;
    while( index < numTareas ){
        long capacidad = vmList.get(i).getBw();
        long costo_actual = 0;
        long currentCost = (costo_actual +
            cloudletList.get(index).getCloudletLength()) / 100;
        while (currentCost <= capacidad && index < numTareas) {
            costo_actual += cloudletList.get(index).getCloudletLength();
            bindCloudletToVm(index, i);
            System.out.println("Task" +
                cloudletList.get(index).getCloudletId() + " is bound with VM" +
                vmList.get(i).getId() + " tam: " +
                cloudletList.get(index).getCloudletLength());
            index++;
        }
        if(index >= numTareas) {
            //Si terminamos de asignar todas las tareas
            break;
        }

        i++;
    }

    if(i >= numVirtualM) {
```

```
//Reiniciamos la ronda de asignacion a la maquina 0
i = 0;
}
}
}

}
```

Heuristics Broker

```
package Heuristics;

import PSO.PSOResult;
import PSO.PSOUtilities;
import PSO.Particle;
import PSO.Swarm;

import org.cloudbus.cloudsim.DatacenterBroker;

import java.util.List;

/**
 * FCFS Task scheduling
 * @author
 * Canul, Cindy
 * Kumul, Cristian
 * Peraza, Jonathan
 */
public class HeuristicBroker extends DatacenterBroker {

    public HeuristicBroker(String name) throws Exception {
        super(name);
    }

    public void scheduleTaskstoVms(List<Particle> particleList) {

        // Se crea el objeto Swarm
        Swarm swarm = new Swarm();
    }
}
```

```
// las tareas (particulas) son enviadas para evaluar
PSOResult result = swarm.init(particleList, vmList, 10, 2);

int cont = 0;
//Se realiza la iteracion hasta que todas las tareas esten
//calendariadas
while (cont < particleList.size()) {
    // se asigna la mejor particula a la maquina virtual encontrada
    // con PSO
    bindCloudletToVm(result.getParticle().getCloudletId(),
        result.getVmId());
    particleList.get(result.getParticle().getCloudletId()).scheduled =
        true;
    System.out.println("Task" + result.getParticle().getCloudletId() +
        " is bound with VM" + result.getVmId() + "tam: " +
        result.getParticle().getCloudletFileSize());
    cont++;
    //Se ejecuta PSO con la lista actualizada
    result = swarm.init(particleList, vmList, 10, 2);
}

}

}
```

Metaheurística PSO

Partícula

```

package PSO;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.UtilizationModel;

/**
 * Created by Jake The Dog on 28/10/15.
 */
public class Particle extends Cloudlet {

    /**
     * Allocates a new Cloudlet object. The Cloudlet length, input and
     * output file sizes should be
     * greater than or equal to 1. By default this constructor sets the
     * history of this object.
     *
     * @param cloudletId      the unique ID of this Cloudlet
     * @param cloudletLength  the length or size (in MI) of this
     *                        cloudlet to be executed in a
     * @param PowerDatacenter
     * @param pesNumber       the pes number
     * @param cloudletFileSize the file size (in byte) of this cloudlet
     *                        <tt>BEFORE</tt> submitting
     * to a PowerDatacenter
     * @param cloudletOutputSize the file size (in byte) of this
     *                        cloudlet <tt>AFTER</tt> finish
     * executing by a PowerDatacenter
     * @param utilizationModelCpu the utilization model cpu
     * @param utilizationModelRam the utilization model ram
     * @param utilizationModelBw the utilization model bw
     * @pre cloudletID >= 0
     * @pre cloudletLength >= 0.0
     * @pre cloudletFileSize >= 1
     * @pre cloudletOutputSize >= 1
     * @post $none

```

```

*/

public boolean scheduled = false;

public Particle(int cloudletId, long cloudletLength, int
    pesNumber, long cloudletFileSize, long cloudletOutputSize,
    UtilizationModel utilizationModelCpu, UtilizationModel
    utilizationModelRam, UtilizationModel utilizationModelBw) {
super(cloudletId, cloudletLength, pesNumber, cloudletFileSize,
    cloudletOutputSize, utilizationModelCpu, utilizationModelRam,
    utilizationModelBw);
}

public void initialise() {

}

/* Extends attributes and methods */

/**
 * the velocity of cloudlet particle is denoted by :
 *  $v[i][k+1] = (\text{inertia weight})v[i][k] + (\text{acceleration coefficient}$ 
 *   1)Rand(0,1) * (pbest -  $x[i][k]$ ) + (acceleration coefficient
 *   2)Rand(0,1) * (gbest -  $x[i][k]$ )
 * <p>
 * -----
 */
private int velocity;
private int position;
private double pbest;
private double gbest;

private int pbestPosition;
private int gbestPosition;

private int bestVm;

/**

```

```
* coefficients
* W = inertia weight
* C = acceleration coefficients
* rand = rand number between 0,1
*/
private int w = 1;
private int c1 = 1;
private int c2 = 1;

public static int getRandom() {
return ((int) (Math.random() * (1)));
}

public static int getRandomInteger(int maximun, int minimum) {
return ((int) (Math.random() * (maximun - minimum))) + minimum;
}

public void initParticle(int vms, int velocityLimit) {
this.position = getRandomInteger(vms, 0);
this.velocity = getRandomInteger(velocityLimit, 1);
this.pbest = Double.POSITIVE_INFINITY;
}

public int getVelocity() {
return velocity;
}

public void setVelocity(int gbest) {
c1 = getRandomInteger(2, 1);
c2 = getRandomInteger(2, 1);

velocity = (w * position) + ((c1 * getRandom()) *
    (this.pbestPosition - this.position)) + ((c2 * getRandom()) *
    (gbest - this.position));
}

public int getPosition() {
return this.position;
}
```

```
public void setPosition(int velocity) {
    this.position = this.position + velocity;
}

public void updatePbest(double fit) {
    //si el fit es menor que el pbest se actualiza
    if (fit < this.pbest) {
        this.pbest = fit;
    }
}

public void setPbest(double fit) {
    this.pbest = fit;
}

public double getPbest() {
    return this.pbest;
}

public void setBestVm(int x) {
    this.bestVm = x;
}

public int getBestVm() {
    return this.bestVm;
}

}
```

FitnessFunction

```
package PSO;

import PSO.Particle;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Vm;
```



```
import org.cloudbus.cloudsim.Cloudlet;
import common.CloudletUtilities;

import java.util.List;
/**
 * FCFS Task scheduling
 * @author
 * Canul, Cindy
 * Kumul, Cristian
 * Peraza, Jonathan
 */

public class FitnessFunction {

    public double evaluate(Cloudlet particle, Vm vm) {
        CloudletUtilities utilities = new CloudletUtilities();

        return utilities.getProcessingCostBefore(vm, particle);
    }

}
```

Swarm Class

```
package PSO;

import org.cloudbus.cloudsim.Vm;

import java.util.List;

/**
 * Created by Jake The Dog on 16/11/02.
 */
public class Swarm {

    public static int getRandomInteger(int maximun, int minimum) {
        return ((int) (Math.random() * (maximun - minimum))) + minimum;
    }

}
```

```
public static void initialize(List<Particle> list, int vms, int
    velocityLimit) {
    for (int i = 0; i < list.size(); i++) {
        list.get(i).initParticle(vms, velocityLimit);
    }
}

public static Particle getBestParticle(List<Particle> list) {
    int bIndex = 0;
    double best = Double.POSITIVE_INFINITY;
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i).getPbest() < best) {
            best = list.get(i).getPbest();
            bIndex = i;
        }
    }
    return list.get(bIndex);
}

public static PSOResult init(List<Particle> particles, List<?
    extends Vm> vms, int iteration, int velocityLimit) {
    double gbest = 0;
    int gBestPosition = 0;
    Particle gBestParticle;
    PSOResult result = new PSOResult();
    // Se inicia de manera aleatoria la posici'on y la velocidad de
    cada particula
    initialize(particles, vms.size(), velocityLimit);
    //Criterio de terminaci'on de PSO
    for (int x = 0; x < iteration; x++) {

        //Para cada part'icula, calcular su fitness value
        FitnessFunction fitness = new FitnessFunction();
        for (int i = 0; i < particles.size(); i++) {
            if (!particles.get(i).scheduled) {
                double fitnessValue = fitness.evaluate(particles.get(i),
                    vms.get(particles.get(i).getPosition() % vms.size()));
```

```
//Si el acutal fitnessValue es mejor que el previo pbest,
    colocamos el fitnesValue como nuevo pbest
if (fitnessValue < particles.get(i).getPbest()) {
particles.get(i).setPbest(fitnessValue);
particles.get(i).setBestVm(particles.get(i).getPosition() %
    vms.size());
}
}
}
// se selecciona la mejor particula como gbest
gBestParticle = getBestParticle(particles);
gbest = gBestParticle.getPbest();
gBestPosition = gBestParticle.getPosition();

//Actualizamos la velocidad y la posicion de cada particula
result.setParticle(gBestParticle);
result.setVmId(gBestParticle.getBestVm());
result.setCost(gbest);
for (int i = 0; i < particles.size(); i++) {

particles.get(i).setVelocity(gBestPosition);
int velocity = particles.get(i).getVelocity();
particles.get(i).setPosition(velocity);
}
}

return result;

}
}
```

PSOResult Class

```
package PSO;

import org.cloudbus.cloudsim.Vm;
```

```
/**
 * Canul, Cindy
 * Kumul, Cristian
 * Peraza, Jonathan
 */
public class PSOResult {

    private double cost;
    private int vmId;
    private Particle particle;

    public PSOResult() {
        this.cost = 0.0;
        this.vmId = 0;
        this.particle = null;
    }

    public void setCost(double x) {
        this.cost = x;
    }

    public double getCost() {
        return this.cost;
    }

    public void setVmId(int x) {
        this.vmId = x;
    }

    public int getVmId() {
        return this.vmId;
    }

    public void setParticle(Particle particle) {
        this.particle = particle;
    }

    public Particle getParticle() {
        return this.particle;
    }
}
```

```
}  
}
```

PSOUtilities Class

```
package PSO;  
  
import org.cloudbus.cloudsim.Cloudlet;  
  
import java.util.LinkedList;  
import java.util.List;  
  
/**  
 * @author  
 * Canul, Cindy  
 * Kumul, Cristian  
 * Peraza, Jonathan  
 */  
public class PSOUtilities {  
  
    public List<Particle> parseParticles(List<? extends Cloudlet>  
        listCloudlet) {  
  
        List<Particle> particles = new LinkedList<Particle>();  
  
        for (int i = 0; i < listCloudlet.size(); i++) {  
            Cloudlet cloudlet = listCloudlet.get(i);  
            Particle particle = new Particle(cloudlet.getCloudletId(),  
                cloudlet.getCloudletLength(), cloudlet.getNumberOfPes(),  
                cloudlet.getCloudletFileSize(),  
                cloudlet.getCloudletOutputSize(),  
                cloudlet.getUtilizationModelCpu(),  
                cloudlet.getUtilizationModelRam(),  
                cloudlet.getUtilizationModelBw());  
  
            particles.add(particle);  
        }  
    }  
}
```

```
return particles;
}

public List<Cloudlet> particles2Cloudlets(List<Particle> list) {

List<Cloudlet> cloudlets = new LinkedList<Cloudlet>();

for (int i = 0; i < list.size(); i++) {
Particle particle = list.get(i);
Cloudlet cloudlet = new Cloudlet(particle.getCloudletId(),
    particle.getCloudletLength(), particle.getNumberOfPes(),
    particle.getCloudletFileSize(),
    particle.getCloudletOutputSize(),
    particle.getUtilizationModelCpu(),
    particle.getUtilizationModelRam(),
    particle.getUtilizationModelBw());
cloudlet.setUserId(particle.getUserId());
cloudlets.add(cloudlet);
}
return cloudlets;
}
}
```
