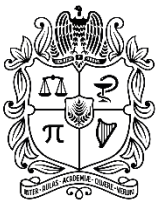


UNIVERSIDAD
NACIONAL
DE COLOMBIA

IMPLEMENTACIÓN DEL SUBSISTEMA DE VIRTUALIZACIÓN INALÁMBRICA DE RECURSOS DE PROCESAMIENTO PARA UNA RED AD-HOC

Juan Sebastián Triana Correa

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2021

IMPLEMENTACIÓN DEL SUBSISTEMA DE VIRTUALIZACIÓN INALÁMBRICA DE RECURSOS DE PROCESAMIENTO PARA UNA RED AD-HOC

Juan Sebastián Triana Correa

Tesis de investigación presentada como requisito parcial para optar al título de:

Magister en Ingeniería de Sistemas y Computación

Director:

PhD. Jorge Eduardo Ortiz

Línea de Investigación:

Sistemas inteligentes – Computación aplicada

Grupo de Investigación:

TLÖN

Universidad Nacional de Colombia

Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial

Ciudad, Colombia

2021

“El mero conocimiento no es sabiduría. La sabiduría sola tampoco basta, son necesarios el conocimiento, la sabiduría y la bondad para enseñar a otros hombres”

Héctor Abad Gómez

Agradecimientos

Un gran agradecimiento a mi director y co-director, sin su ayuda y paciencia no hubiese sido posible alcanzar las metas planteadas en este proyecto. A mi familia agradezco su apoyo incansable durante este reto.

Resumen

TLÖN es un sistema de cómputo propuesto por el grupo de investigación en redes de Telecomunicaciones Dinámicas y lenguajes de programación distribuidos de la Universidad Nacional de Colombia. El modelo que propone TLÖN es un sistema de cómputo distribuido que intenta resolver la necesidad de adaptarse a condiciones adversas en recursos, conectividad y existencia de nodos, abstraídos al plano de virtualización de una red inalámbrica tipo Ad-Hoc colaborativa de elementos móviles que comparten recursos sobre una capa física. El presente trabajo se encuentra en el marco de desarrollo de la capa de virtualización de recursos de procesamiento del sistema TLÖN. A continuación, se presenta el diseño, implementación y resultados del subsistema de virtualización de recursos de procesamiento, estableciendo los modelos y criterios de evaluación que se utilizaran para las fases de Resource Broadcasting, Matching y Scheduling teniendo en cuenta las condiciones inherentes de una red inalámbrica Ad Hoc.

Palabras clave: Computación heterogénea, Ah Hoc inalámbrica, Virtualización recursos, Computación ubicua.

Abstract

TLÖN is a computing system proposed by the research group in Dynamic Telecommunication networks and distributed programming languages of the Universidad Nacional de Colombia. The model proposed by TLÖN is a distributed computing system that has the aim to solve the need to adapt to adverse conditions in resources, connectivity and existence of nodes, abstracted from the virtualization plane of a collaborative Ad-Hoc wireless network of mobile elements that share resources on a physical layer. This work is framed under the development of the virtualization layer of the processing resources of the TLÖN system. In the following chapters you will find the design, implementation, and results of the processing resources virtualization subsystem, setting the models and evaluation criteria to be used for the Resource Broadcasting, Matching, and Scheduling phases taking into account the inherent conditions of an Ad Hoc wireless network.

Keywords: Heterogeneous computing, Wireless Ah Hoc, Resource Virtualization, Ubiquitous computing.

Esta tesis de maestría se sustentó el 28 de Junio de 2021 a las 2:00pm – 4:00pm,
y fue evaluada por los siguientes jurados:

Libia Denise Cangrejo Aljure (Ph.D.)
Universidad Nacional de Colombia

Ingrid Patricia Paez Parra (Ph.D.)
Universidad Nacional de Colombia

Contenido

Lista de figuras

Lista de tablas

1.Introducción

Las redes de comunicaciones son sistemas que han evolucionado desde los años 80 para convertirse en una herramienta fundamental para la evolución de la tecnología y el conocimiento. Con el desarrollo de nuevas estructuras y topologías heterogéneas, las redes han logrado cambiar totalmente la forma de comunicación entre individuos, en cuanto intercambio de contenidos y prestación de servicios. Mientras cada vez más gente tiene acceso a infraestructura global de comunicación e información, el siguiente paso se relaciona a el uso de internet como una plataforma global para permita a máquinas y "Smart Objects" comunicarse, dialogar, procesar y coordinar trabajo (Weiser, 1991). Bajo esta perspectiva, el concepto convencional de internet como una infraestructura fija del proveedor al usuario final desaparecerá, dejando lugar a una noción de dispositivos interconectados que forman entornos de computación ubicua y piscinas de recursos; que, a su vez, hace uso de internet como backbone para acceder a la red de difusión de información global. Es por esta razón que la investigación en el área de redes inalámbricas heterogéneas y modelos de computación distribuida está en auge (Miorandi, Sicari, & Pellegrini, 2012).

Adicionalmente, las aplicaciones de procesamiento de datos distribuidos que implican el procesamiento y la combinación de datos de fuentes heterogéneas para extraer valor, han cobrado un papel gigante en la era digital por el aumento exponencial del flujo de datos que domina la industria, por la invención de nuevas formas de medir la realidad y el acercamiento del consumidor a una mayor oferta de productos. Las aplicaciones emergentes, como los vehículos autónomos conectados, se basan en modelos complejos de aprendizaje automático que se aplican a los datos capturados en el perímetro, al tiempo que implican la colaboración con otros vehículos. Otras aplicaciones de ejemplo incluyen procesamiento de imágenes (Pengyao & Jianqin, 2018), análisis de mercado (Qiwan & Ruyin, 2020) e inteligencia de negocios (Zhenjie & Yuanming, 2020). Estas aplicaciones presentan un desafío a los enfoques computacionales existentes que tienen en cuenta solo

la nube o el procesamiento local en los mismos dispositivos del borde de la red. Los algoritmos de uso intensivo computacional ahora deben aplicarse a flujos intensivos de datos y la latencia debe minimizarse. En estas aplicaciones, los orígenes de datos transmiten flujos de información a través de una red para procesarse de forma remota, centrándose en el procesamiento continuo, a diferencia de otras aplicaciones que implican almacenamiento a gran escala y procesamiento retrasado. La latencia, el tiempo necesario para extraer información relevante de los flujos de datos, y el rendimiento, la velocidad a la que se pueden procesar estas secuencias, son métricas de rendimiento clave para dichas aplicaciones. La computación en la nube centralizada se utiliza a menudo en estos escenarios, ya que los orígenes de datos no suelen tener recursos informáticos adecuados para realizar cálculos complejos. Las aplicaciones también se basan en la conciliación de datos de múltiples fuentes, por lo que el procesamiento centralizado es útil. La nube también ofrece beneficios en escalabilidad, costo y se ha demostrado que proporciona beneficios en aplicaciones como el procesamiento de redes inteligentes (Simmhan, Cao, & Giakkoupis, 2011).

Sin embargo, muchas aplicaciones de streaming emergentes tienen restricciones estrictas de latencia y mover datos a la nube conlleva un retraso considerable. Si bien los datos generados por las fuentes pueden ser pequeños, un gran número de fuentes significa que, en conjunto, el volumen de datos que se transmitirán es alto. Por ejemplo, en 2011, la red inteligente de Los Ángeles requirió 2 TB de datos transmitidos de 1,4 millones de consumidores para ser procesados por día (Simmhan, Cao, & Giakkoupis, 2011). Algunas aplicaciones, como las que se ocupan de los datos de vídeo, también deben cumplir con los requisitos de datos de ancho de banda elevado.

Estas limitaciones han dado lugar a un mayor interés en la computación "borde" (Edge computing) o "niebla" (Fog Computing), un paradigma poco definido en el que el procesamiento se realiza en la fuente de datos o cerca a ella. También puede abarcar la realización del procesamiento dentro de la infraestructura de red, como en puertas de enlace inteligentes, conmutadores de red o enrutadores. CISCO ofrece un marco que permite que el código de aplicación se ejecute en recursos de computación de repuesto en algunos elementos de red, y los switches Ethernet de Juniper permiten que el proceso de aplicación se combine estrechamente con la estructura de conmutación.

La informática perimetral (Edge computing y Fog computing) es un área emergente. Los cloudlets y redes ad-hoc se han utilizado para diferentes tipos de aplicaciones, como procesamiento de imágenes (Nanne & Antheunis, 2020) y (Bilal & Khalid, 2017). Plataformas como edge Tensor Processing Unit de Google demuestran que existe una tendencia a acercar los cálculos complejos a la fuente de los datos. Para explorar las implicaciones de la distribución del cálculo de aplicaciones a través de una red de plataformas informáticas heterogéneas, se necesita un modelo adecuado. Esto permitiría evaluar diferentes estrategias de implementación mediante métricas como el rendimiento y la latencia de extremo a extremo.

La informática perimetral también puede incluir el concepto de 'cloudlets', que son recursos dedicados del servidor informático colocados a pocos saltos de los orígenes de datos. Estos pueden variar en escala, desde una sola caja colocada en una planta de una fábrica hasta un centro de datos a pequeña escala que comprende varias máquinas en red.

Esta creciente necesidad de procesamiento de datos a gran escala (big data) requiere un marco eficaz que pueda procesar conjuntos de datos de manera distribuida y paralela a una velocidad aceptable. Aunque es posible que el origen de los datos no tenga las capacidades informáticas necesarias, estos recursos pueden soportar aplicaciones complejas y son accesibles en latencias más cortas que una nube remota (Elazhary, 2018). En aplicaciones complejas, es probable que algunos procesos, como el filtrado y el preprocesamiento se puedan realizar en el perímetro, reduciendo en gran medida el volumen de datos transmitidos, y el procesamiento y la fusión adicionales de datos se pueden llevar a cabo en la nube. Las ventajas de este enfoque son que las partes sensibles a la latencia de la aplicación se pueden realizar localmente, mientras que las operaciones más intensivas computacionalmente que pueden requerir más potencia de procesamiento o datos adicionales se pueden realizar de forma centralizada. Las aplicaciones de procesamiento de secuencias son adecuadas para ser particionadas y distribuidas entre varias máquinas, como es común en marcos de procesamiento de flujos como Apache Storm e IBM Infosphere Streams. Además, los proveedores de servicios en la nube, como Microsoft Azure, tienen plataformas de análisis perimetrales que permiten dividir el procesamiento entre la nube y el perímetro.

El presente trabajo pretende establecer un modelo de procesamiento para ejecutar tareas distribuidas en una red ad-hoc de nodos con recursos heterogéneos y de conexión

espontánea dentro de la línea de investigación de redes de telecomunicaciones dinámicas y lenguajes de programación distribuidos, del grupo TLÖN de la Universidad Nacional de Colombia. Inicialmente en el capítulo 2 se da un estado del arte de sistemas distribuidos explicando tendencias e implementaciones usadas de inspiración en este trabajo; igualmente se explican conceptos y arquitecturas necesarias para entender la distribución de recursos de red en redes colaborativas. En el capítulo 3 se explican los principios de la virtualización de recursos, así como diferentes aproximaciones que buscan la abstracción lógica y simulación de componentes de hardware como memoria, almacenamiento y unidades de procesamiento. Luego en el capítulo 4 se da un vistazo a los modelos social inspirados y al marco de trabajo de este proyecto, el grupo TLÖN y sus objetivos.

El capitulo 5 se centra en la implementación del módulo de virtualización inalámbrica de recursos de procesamiento y una explicación detallada del modelo ya en un ámbito práctico. Los resultados obtenidos de la validación de este modelo se presentan en el capítulo 6, en el que se muestra la metodología de prueba y se hace un análisis del experimento elegido para validar el subsistema. El capítulo 7 expone las conclusiones finales del trabajo hecho y se proponen algunas opciones de trabajo futuro para continuar esta línea de investigación.

1.1 Objetivos

El presente trabajo esta enmarcado dentro del proyecto principal del grupo de investigación llamado TLÖN (Grupo de Investigación en Redes de Telecomunicaciones Dinámicas y Lenguajes de Programación Distribuidos), que pretende construir un modelo de cómputo distribuido con énfasis en el modelo social inspirado.

1.1.1 Objetivo general

- Implementar el subsistema de virtualización de recursos de procesamiento de una red Wireless Ad-Hoc que permita el mapeo y asignación de las tareas a los nodos disponibles.

1.1.2 Objetivos específicos

- Describir el estado del arte de técnicas de descubrimiento de recursos, *mapping* y *scheduling* en sistemas de procesamiento distribuido.
- Implementar un modelo de descubrimiento de recursos de procesamiento disponibles en una red Ad-Hoc.
- Diseñar y construir un prototipo del subsistema de *mapping* y *scheduling* de tareas distribuidas en una red inalámbrica Ad-Hoc.
- Validar en un entorno real la asignación de recursos de procesamiento sobre una red Ad Hoc estática de n=7 nodos.

1.2 Justificación

El problema que se abordará en esta investigación es la virtualización de recursos de cómputo sobre una red Ad-Hoc, específicamente la implementación del módulo que se encargará de asignar recursos a las subtareas que se corran sobre la red (matching) y tiempo de ejecución a aquellas que ya fueron asignadas (scheduling), con el fin de maximizar algunos criterios de rendimiento de un sistema heterogéneo. Este proceso de asociación (*matching)* y programación (*scheduling)* se llama mapeo de recursos (Fernandez-Baca, 1989) y en general es un problema *NP-completo* (Coffman, 1976). Es por eso por lo que actualmente sigue siendo un área activa de investigación con el desarrollo de diferentes heurísticas que buscan encontrar soluciones óptimas (Shivle, Siegel, Maciejewski, & Sugavanam, 2010). Adicionalmente, al ser un ambiente heterogéneo se debe tener en cuenta que cada procesador disponible puede variar en diferentes aspectos, incluyendo edad, set de instrucciones, velocidades de reloj, tamaño o estructura de cache, tamaño de memoria y anchos de banda, esto con el fin de optimizar el proceso de *matching*.

La asignación de recursos para ejecución de tareas en un sistema distribuido debe tener en cuenta la dependencia de tareas; dicha dependencia se puede clasificar en dos categorías: dependencias de control y dependencia de datos. En la dependencia de datos, las tareas intercambian datos entre ellas con el fin de resolver un problema; a su vez la

dependencia de datos se puede dividir en dos subcategorías, dependencias de precedencia y de ejecución en paralelo; en las primeras las tareas se ejecutan independientemente, pero requieren entradas generadas por otras tareas, mientras que en las dependencias de ejecución en paralelo las tareas intercambian datos periódicamente y la comunicación entre hilos puede ser en cualquier momento durante la ejecución. La dependencia de datos entre tareas implica una carga pesada para la comunicación entre los nodos que aporten recursos de procesamiento, es por eso por lo que se hace necesario un esquema de asignación de recursos robusto y efectivo (Gener & Syst., 2016).

1.3 Tipo de investigación

En la construcción de esta investigación se han identificado los siguientes aspectos metodológicos: el tipo de investigación que enmarcó este trabajo es de tipo exploratorio, esto se debe a que se realizó una exploración sobre los temas que encierran los tres ejes temáticos de la investigación: redes ad-hoc, procesamiento distribuido y resource allocation. En cuanto a la implementación se consideró una investigación de tipo experimental, ya que gracias a la naturaleza de los métodos que se utilizan para la comprobación y obtención de los resultados, se hacen sobre elementos computaciones en donde se debe realizar pruebas y plantear métodos para realizar dichas pruebas.

En el presente documento, inicialmente se hará una conceptualización de los aspectos teóricos que enmarcan los ejes fundamentales de la investigación que son: redes Ad-hoc, procesamiento distribuido y administración de recursos en redes heterogéneas. Luego se realizará el diseño del modelo de virtualización de recursos de computo en una red Ad-Hoc, definiendo las políticas de mapeo de recursos, los protocolos y el modelo de integración con el resto de los subsistemas.

Con el diseño propuesto se continuó con una fase de implementación, donde se codificó los scripts de los módulos de gestión, mapeo, integración y virtualización de recursos. Esta implementación fue evaluada en un entorno real, usando 7 dispositivos conectados en red Ad-Hoc y simulando un árbol de procesos distribuidos en los nodos que presten el recurso de computo.

Adicionalmente, para el proceso de investigación se hizo uso de la metodología DRM (Design Research Metodology) (Blessing & Chakrabarti, 2008). Esta metodología es

iterativa y retroalimentada en cada una de sus fases, lo cual permite ir enfocando la investigación cada vez que se realizan avances del modelo. Las fases fueron las siguientes cuatro: *Research Clarification*, en donde se aclara el entendimiento actual del objetivo de la investigación y se elabora un plan de investigación para tener un foco en las subsecuentes fases; una primera fase de *Estudio Descriptivo (DE-1)* en la cual se aumentó el entendimiento a nivel de diseño y los factores que influenciaron la investigación, para ello se determinó un modelo de referencia y una serie de criterios de éxito a partir del análisis empírico de la información recaudada en las anteriores fases; un fase de *Estudio Prescriptivo* donde se desarrolló un modelo de impacto que es una descripción visual de las situación final deseada mostrando los cambios esperados del modelo de referencia en base al *DE-1*; finalmente un segunda etapa de *Estudio Descriptivo* que se centra en la validación de la usabilidad y aplicabilidad de los productos de las anteriores fases.

1.4 Aportes al conocimiento

Como parte del desarrollo de este trabajo se mencionan a continuación los principales artefactos, resultado de los procesos de conceptualización, levantamiento de estado del arte y diseño de la solución propuesta.

- Estado del arte de métodos de difusión (*broadcasting*), mapeo (*mapping*) y asignación (*allocation*) de recursos de procesamiento sobre redes heterogéneas.
- Modelo de gestión de memoria y procesos distribuidos en ambientes estocásticos.
- Subsistema de virtualización de recursos de procesamiento para redes ad-hoc.
- Esquema de descubrimiento de recursos de procesamiento en redes ad-hoc.
- Esquema de solicitud de dependencias bajo demanda para procesos distribuidos.
- Validación del rendimiento del sistema en un entorno real restringido.
- Despliegue y validación del modelo en una topología de red con enrutador central.

Adicionalmente, este proyecto se encuentra enmarcado en el desarrollo del sistema de cómputo TLÖN; el subsistema de descubrimiento y distribución de recursos de procesamiento desarrollado en este trabajo hará parte de su capa de virtualización, elemento fundamental en la construcción del ecosistema propuesto en el grupo de investigación.

2. Sistemas distribuidos

La mayoría de los sistemas de procesamiento distribuidos se han diseñado tradicionalmente para entornos Cloud con granjas de servidores dedicados. Recientemente, han surgido modelos arquitectónicos para entornos más distribuidos que abarcan múltiples centros de datos o para explotar los bordes de Internet (computación de borde y niebla). El presente trabajo tiene como objetivo utilizar los recursos disponibles en los nodos al borde de la red para ejecutar tareas complejas de procesamiento, cerca al origen de los datos. En este capítulo se establecen conceptos claros relacionados con sistemas distribuidos, en cumplimiento del primer objetivo específico, el cual habla de establecer un estado del arte de técnicas de descubrimiento de recursos, *mapping* y *scheduling* en sistemas de procesamiento distribuido.

A pesar de muchos esfuerzos en la construcción de infraestructura, como la adaptación de OpenStack para ejecutarse en redes locales, gran parte del trabajo existente en el procesamiento de flujo continuo de datos en redes estocásticas permanece en un nivel conceptual o arquitectónico sin soluciones de software concretas o escalabilidad demostrada (Assunção & Veith, 2018).Esta sección proporciona una lista no exhaustiva de modelos de computación y su evolución hacia los sistemas distribuidos, así como tendencias e implementaciones usadas de inspiración en este trabajo. También se explican conceptos necesarios y arquitecturas propuestas para entender la distribución de recursos de red en redes colaborativas.

2.1.1 Arquitecturas tradicionales

La arquitectura Von Neumann (VNA) es un modelo de referencia para ordenador, según el cual se mantiene una memoria común tanto para las instrucciones del programa informático como para otros datos. Los sistemas Von Neumann incluyen la clasificación Flynnschen a la clase de arquitecturas SISD (Instrucción única, datos únicos), en contraste con el procesamiento paralelo. Esta arquitectura es la base para el funcionamiento de los

ordenadores más conocidos de hoy en día. Lleva el nombre del matemático austríaco-húngaro John Von Neumann, cuyo trabajo principal fue publicado alrededor de 1945. Otro nombre que suele utilizarse para referirse a esta arquitectura, es el de la universidad de los Estados Unidos - Princeton Architecture. En el año 1945, durante la construcción de la máquina de computación EDVAC, Von Neumann describió el concepto en su primer documento inédito "Primer borrador de un informe sobre EDVAC". Fue revolucionario en su época, ya que las computadoras previamente desarrolladas estaban conectadas a un programa sólido que estaba conectado ya sea a hardware o tarjetas perforadas tenían que ser leídos. Con la arquitectura Von Neumann, fue posible realizar los cambios en los programas muy rápidamente y sin realizar ningún cambio en el hardware o ejecutar diferentes programas en rápida sucesión.

En la arquitectura clásica de von Neumann (Figura 1), el ALU y la unidad de control están conectados a una sola memoria que almacena tanto los valores de datos como las instrucciones del programa. Durante la ejecución, se lee una instrucción de la memoria y se descodifica, las operaciones adecuadas se obtienen de la memoria y por último se ejecuta la instrucción. La principal desventaja es que el ancho de banda de memoria se convierte en el cuello de botella en una arquitectura de este tipo.

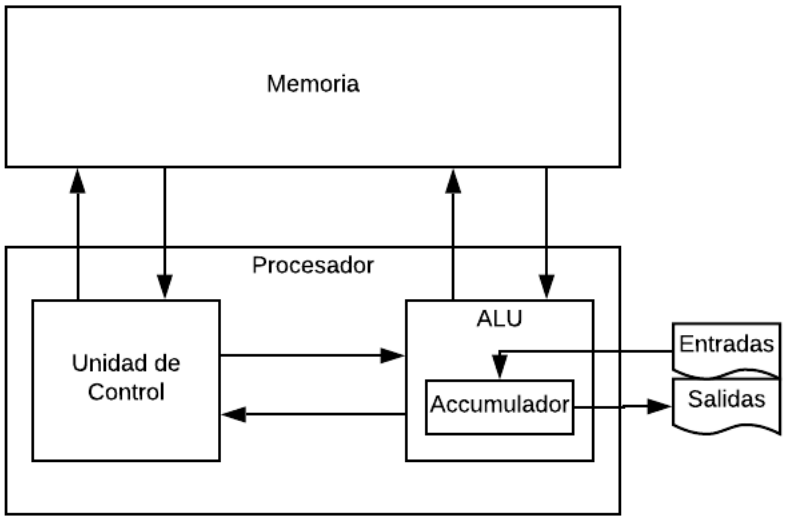


Figura 1. Arquitectura Von Neumann clásica

El cuello de botella de la arquitectura Von Neumann se refiere a los hechos arquitectónicos de que el bus del sistema de conexión (datos e instrucción) es el cuello de botella entre el procesador y la memoria. Procediendo del cuello de botella de Von Neumann, la única cosa a la vez. Dado que una arquitectura Von Neumann, a diferencia de la arquitectura de Harvard, solo tiene un bus común para los datos y las instrucciones que se utilizarán, la cantidad máxima transferible de datos que debe dividirse se limita. En los primeros ordenadores, la CPU era la unidad más lenta del equipo, es decir, el tiempo de entrega de datos era sólo una pequeña proporción del tiempo total de procesamiento para una operación aritmética. Durante algún tiempo, sin embargo, la velocidad de procesamiento de la CPU creció significativamente más rápido que las tasas de transferencia de datos de autobuses o memoria, lo que agravó el impacto del cuello de botella de Von Neumann.

La mayoría de los ordenadores en uso hoy en día se basan en el principio fundamental de la arquitectura Von Neumann, es decir, sus propiedades son similares a las de un VNA. Con el tiempo, muchas de las arquitecturas informáticas VNA simples originalmente concebidas, como la arquitectura x86, se diferenciaban más allá y se desarrollaban para ser más complejas. Esto se hizo con el fin de lograr ganancias de rendimiento, pero sin romper el modelo de VNA fácilmente manejable, es decir, términos de software compatible para poder quedarse a seguir utilizando esto en su beneficio.

Con la tendencia de aumentar el número de unidades de procesamiento paralelo (procesadores multinúcleo) y buses (por ejemplo, HyperTransport) se vuelve más compleja i su gestión. Por lo tanto, es de esperar este paradigma cambie a un modelo arquitectónico diferente que será necesario para lograr mejoras de rendimiento en las arquitecturas informáticas.

Una de las arquitecturas competitivas más importantes es la arquitectura de Harvard, la cual cuenta con una separación física de control y acceso a la memoria de datos a través de buses separados, es decir, de forma independiente. Esta arquitectura tiene dos espacios de memoria independientes dedicados al código de programa y a los datos respectivamente, dos buses de direcciones correspondientes y dos buses de datos para acceder a dos espacios de memoria. El procesador de Harvard ofrece “*fetching*” y ejecuciones en paralelo. La ventaja de esta arquitectura es que permite las instrucciones y la carga de datos al mismo tiempo evitando el cuello de botella presente en la Von

Neumann. La desventaja potencial en comparación con la arquitectura Von Neumann es el sistema paralelo que resulta en una ejecución de programas no deterministas.

2.1.2Microservicios

En los últimos años, las Arquitectura Orientada a Servicios (SOA) han recibido una atención cada vez mayor en línea con el avance hacia la lucha contra los desafíos asociados con la mejora y el mantenimiento de diversos entornos. Para modernizar el sistema de software de las organizaciones, la migración de un sistema heredado a un sistema basado en servicios se ha convertido en una tendencia dominante. Recientemente, muchos estudios han sobresaltado los beneficios de emplear SOA en el desarrollo de nuevas tecnologías líderes en el mundo, como el Internet de las cosas (IoT) y la computación en la nube (CC), y los microservicios. Esto se debe a que SOA ofrece integración flexible y reutilización de servicios debido a su arquitectura modular basada en servicios. SOA también ofrece transparencia porque encapsula varias aplicaciones y fuentes de datos en forma de caja negra. De este modo, un conjunto integrado de recursos de tecnología de la información (TI) podría seguir siendo accesible a pesar de la existencia de diversas tecnologías, códigos de lenguaje, funcionalidades y plataformas.

Es en este contexto en que el que se introduce el uso de microservicios para el diseño de sistemas distribuidos estocásticos. La arquitectura de microservicios (MSA) se basa en una filosofía share-nothing con larga experiencia en evolución en ingeniería de software y diseño de sistemas. Este estilo arquitectónico estructura un sistema como un conjunto de pequeños servicios de acoplamiento flexible que están aislados en pequeñas unidades coherentes y autónomas. Muchas organizaciones, como Amazon, Netflix y The Guardian, utilizan MSA para desarrollar su entrega continua de aplicaciones grandes y complejas, al tiempo que proporcionan flexibilidad y diversidad en el stack tecnológico usado. Además de estructurar el desarrollo de los sistemas, los principios de diseño de MSA se utilizan para migrar sistemas con estilo arquitectónico tradicional a MSA. Dado que MSA todavía es joven, aun hay muchos problemas abiertos, posibilidades de optimización y áreas que descubrir en este campo.

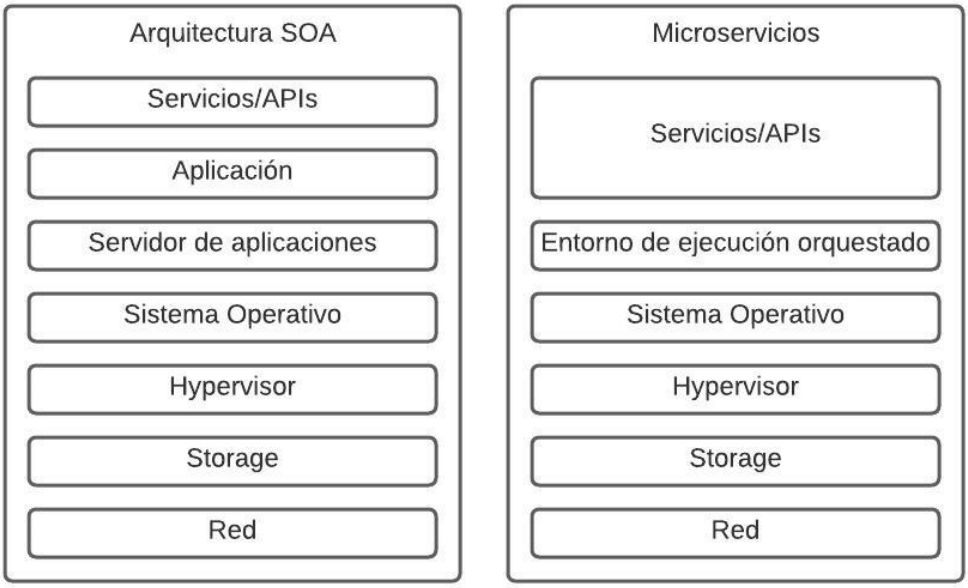


Figura 2.SOA vs Microservicios

Los microservicios es un enfoque de la arquitectura de sistemas que se basa en el concepto de software de modularización. Varias ventajas están asociadas con microservicios. Tres de los más importantes son entrega más rápida, escalabilidad mejorada y mayor autonomía. Las aplicaciones críticas en tiempo real como sistemas de alerta temprana ante desastres y la transmisión de eventos en vivo a menudo implican componentes distribuidos y una comunicación intensiva de datos; a menudo estos sistemas incluyen sensores y actuadores en varias ubicaciones geográficas sin posibilidad de medios físicos de transporte de datos. A pesar de la capacidad de los ecosistemas en la nube para admitir estas aplicaciones, es necesario implementar aplicaciones resistentes y reconfigurables mediante el entorno de tiempo de ejecución virtual. Hoy en día, los microservicios pueden ser empaquetados en contenedores y mecanismos de virtualización ligera como alternativa a pesadas y demandantes máquinas virtuales (VM). Los entornos de utilización de contenedores proporcionan métodos de escalado automático reactivos que resultan en mejoras del tiempo de respuesta de la aplicación, así como en términos de agrupación, asignación de recursos y escalabilidad (P. Jamshidi, 2018).

El término "escalabilidad" es algo ambiguo, podría hacer referencia a la escalabilidad en tiempo de ejecución del sistema, por ejemplo, su adaptabilidad (a un costo razonable) a

los cambios en el número de usuarios que acceden a él. O bien, podría referirse a la capacidad del proceso de desarrollo para dar cabida a muchos desarrolladores que trabajan en él en paralelo. Con los microservicios, la unidad de escalado es cada microservicio, por lo tanto, en tiempo de ejecución los servicios se pueden escalar de manera diferente de acuerdo con sus requisitos específicos. Pero el microservicio también es la unidad de desarrollo e implementación. Por lo tanto, cada servicio puede ser desarrollado, implementado y operado por un equipo diferente, lo que permite una introducción más paralela de nuevas características. Por último, en relación con el concepto de organización escalable, se espera que cada microservicio ofrezca una unidad limitada y autónoma de decisiones de desarrollo y de tiempo de ejecución. Esto permite a un equipo tomar decisiones localizadas para cada servicio, por ejemplo, en términos del lenguaje de programación, bibliotecas o marcos utilizados; la tecnología de base de datos (si la hubiera) empleada; o cualquier otro aspecto de su estrategia de implementación. Esto permite un enfoque del mejor de su clase, con cada equipo seleccionando la opción óptima para su área de responsabilidad (Kallergis D, 2020).

MSA está ganando impulso para el desarrollo e implementación de aplicaciones de software como un conjunto de pequeños servicios granulares que se pueden integrar a través de mecanismos de comunicación ligeros. Los microservicios son componentes pequeños y comprensibles que encapsulan funciones de negocio en torno a los servicios. Estos servicios se pueden escalar de forma independiente (ya que están acoplados libremente) mediante la implementación de diferentes stacks tecnológicos. Muchos investigadores y profesionales sostienen que MSA es una evolución de la arquitectura orientada a los servicios (SOA), como se ve en el contexto de la gestión independiente/autogestión de los servicios, y la naturaleza ligera (Zimmermann, 2017). Por otro lado, MSA se puede diferenciar de SOA en términos de uso compartido de componentes, comunicación de servicio, mediación de servicios y acceso remoto a servicios. SOA se basa en la idea de compartir tanto como sea posible, mientras que MSA se forma sobre la idea de compartir lo menos posible. MSA se centra en la idea de orquestación alrededor de la comunicación entre servicios, mientras que SOA emplea un estilo de orquestación en la gestión de los servicios. Para la mediación de servicios, MSA utiliza la capa de API que actúa como fachada de servicio, mientras que SOA adopta el

concepto de middleware de mensajería para la coordinación de servicios basado en tecnologías ligeras (p.e. REST y HTTP) (Muhammad W., 2020).

2.1.3Concurrencia de procesos

El procesamiento paralelo, el método de tener muchas tareas pequeñas para resolver un gran problema, ha surgido como una tecnología de habilitación clave en la computación moderna y fue el resultado de la demanda de mayor rendimiento, menor costo y productividad sostenida. Los últimos años han sido testigos de una aceptación y adopción cada vez mayores del procesamiento paralelo, tanto para la informática científica de alto rendimiento como para aplicaciones más "de uso general". La aceptación ha sido facilitada por dos grandes desarrollos: procesadores paralelos masivos (MPP) y el uso generalizado de la computación distribuida, los primeros siendo hasta hace pocos años los ordenadores más potentes del mundo. Estas máquinas combinan entre unos pocos cientos y unos pocos miles de CPU en un solo gabinete grande conectado a cientos de gigabytes de memoria. Los MPP ofrecen una enorme potencia computacional y se utilizan para resolver problemas computacionales de Grand Challenge, como el modelado climático global y el diseño de fármacos. A medida que las simulaciones se vuelven más realistas, la potencia computacional necesaria para producirlas crece rápidamente. Por lo tanto, los investigadores en el borde de corte recurren a los MPP y al procesamiento paralelo para obtener la mayor potencia computacional posible.

El segundo gran desarrollo que afecta a la resolución de problemas científicos es la computación distribuida. La informática distribuida es un proceso mediante el cual un conjunto de ordenadores conectados por una red se utilizan colectivamente para resolver un único problema grande. A medida que más y más organizaciones tienen redes de área local de alta velocidad que interconectan muchas estaciones de trabajo de uso general, los recursos computacionales combinados pueden exceder la potencia de un solo equipo de alto rendimiento. En algunos casos, se han combinado varios MPP utilizando computación distribuida para producir una potencia computacional sin igual. El factor más importante en la computación distribuida es el costo. Por el contrario, los usuarios ven muy poco costo en ejecutar sus problemas en un conjunto local de equipos existentes. Es poco común para los usuarios de computación distribuida darse cuenta de la potencia computacional sin procesar de un MPP grande, pero son capaces de resolver problemas varias veces más

grandes de lo que podrían usar uno de sus ordenadores locales. Común entre la informática distribuida y MPP es la noción de pasar mensajes.

En todo el tratamiento paralelo, los datos deben intercambiarse entre tareas de cooperación. Se han intentado varios paradigmas, como memoria compartida, compiladores de paralelización y esquemas *event-based*. El modelo de paso de mensajes se ha convertido en el paradigma de elección, desde la perspectiva del número y la variedad de multiprocesadores que lo soportan, así como en términos de aplicaciones, lenguajes y sistemas de software que lo utilizan.

En un MPP, cada procesador es exactamente igual que cualquier otro en capacidad, recursos, software y velocidad de comunicación. No así en una red. Los equipos disponibles en una red pueden ser creados por diferentes proveedores o tener compiladores diferentes. De hecho, cuando un programador desea explotar una colección de ordenadores en red, puede tener que lidiar con varios tipos diferentes de heterogeneidad; diferentes arquitecturas de procesador, formatos de payload, velocidad de computo, carga de los procesadores y en la misma red.

El conjunto de equipos disponibles puede incluir una amplia gama de tipos de arquitectura, como máquinas de clase PC 386/486, estaciones de trabajo de alto rendimiento, multiprocesadores de memoria compartida, superordenadores vectoriales e incluso MPP gigantes. Cada tipo de arquitectura tiene su propio método de programación óptimo. Además, un usuario puede enfrentarse a una jerarquía de decisiones de programación. La máquina virtual paralela puede estar compuesta por equipos paralelos. Incluso cuando las arquitecturas son sólo estaciones de trabajo en serie, todavía existe el problema de los formatos binarios incompatibles y la necesidad de compilar una tarea paralela en cada máquina diferente. Los formatos de datos en diferentes equipos son a menudo incompatibles. Esta incompatibilidad es un punto importante en la informática distribuida porque los datos enviados desde un equipo pueden ser ilegibles en el equipo receptor. Los paquetes usados para paso de mensajes desarrollados para entornos heterogéneos deben asegurarse de que todos los equipos entienden los datos intercambiados. La aproximación de una solución para una red heterogénea debe incluir formatos transversales y *light-weight* para disminuir los problemas de compatibilidad entre los nodos participantes.

2.2 Procesamiento distribuido

El procesamiento paralelo y distribuido se ha convertido en un área de gran desarrollo en las Ciencia de la Computación. Si bien, utilizar múltiples procesadores para resolver problemas, en general de complejidad creciente y para obtener resultados en menor tiempo, resulta un concepto intuitivo; los fundamentos subyacentes a los mecanismos y soportes de paralelización presentan numerosas variantes (Basney & Livny, 2012). Originalmente, los microprocesadores convencionales implicaban solo una CPU en un chip. A medida que evolucionó la ingeniería de microprocesadores, los fabricantes descubrieron que, para acelerar los procesos, se podía combinar más de un procesador en una sola unidad. Muchos procesadores modernos implican un diseño multinúcleo, como un diseño pionero de cuatro núcleos publicado por Intel, donde cuatro procesadores separados ofrecen velocidades extremadamente altas para la ejecución del programa y la lógica.

El procesamiento distribuido también se puede utilizar como sinónimo aproximado para el procesamiento paralelo, en el que los programas se hacen para ejecutarse más rápidamente con varios procesadores. Con la estrategia de incluir más de un procesador en un chip de microprocesador, los usuarios de hardware también pueden encadenar varios equipos para implementar el procesamiento paralelo con aplicaciones conocidas como software de procesamiento distribuido. El concepto de procesamiento distribuido va de acuerdo con la ley de Moore, que postula que el número de transistores en un circuito integrado individual (IC) se duplica cada dos años. Como esta teoría ha demostrado ser en gran medida correcta en las últimas cuatro décadas, estrategias de ingeniería como el procesamiento distribuido también han añadido a la velocidad de los dispositivos lógicos para algunos avances sorprendentes en la capacidad de las computadoras para realizar tareas funcionales. (Yue-Jiao Gong, 2015)

Las arquitecturas para procesamiento paralelo y distribuido han evolucionado, y en la actualidad las redes de computadoras constituyen una plataforma de cómputo paralelo muy utilizada por sus ventajas en términos de la relación costo/rendimiento (Anderson, Culler, & Patterson, 2005). El concepto de sistema distribuido es una generalización que permite que redes dedicadas a una aplicación paralela se interconecten y puedan cooperar en un algoritmo, compartiendo recursos e incrementando la potencia de cómputo (Abbas, 2013) lo cual un menor nivel de acoplamiento de las unidades de procesamiento.

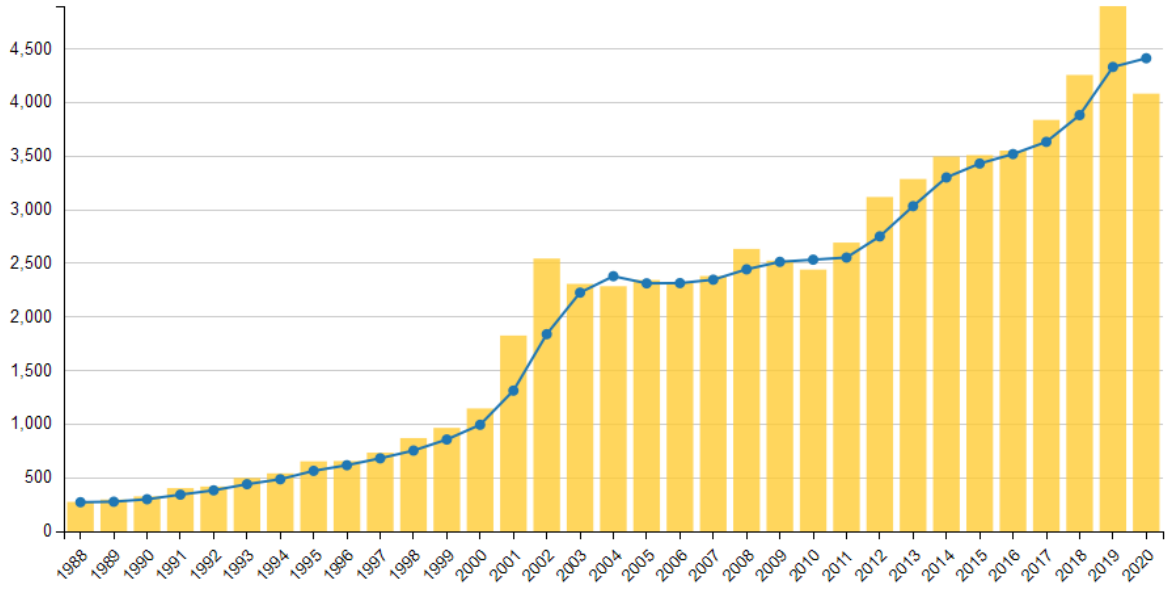


Figura 3. Actividad de patentes en "Ad-Hoc Networks" y "Distributed Systems"

Las redes inalámbricas se han vuelto muy populares en la industria en los últimos años (Figura 3), ya que permiten la movilidad entre los nodos. Existen dos variaciones de redes móviles, la primera es la red de infraestructura fija que posee puertas de enlace fijos e inalámbricos llamados bases. Una unidad móvil entro de la red se comunica únicamente con la base más cercana dentro de su radio de alcance. Una aplicación ampliamente conocida de este tipo de infraestructura son las redes WLAN que permiten la conectividad de nodos en redes locales con una salida a internet. El segundo tipo de redes móviles son llamadas Ad Hoc pues no tienen infraestructura fija y sus nodos están conectados de manera arbitraria. Este tipo de redes también llamadas MANET (Mobile Ad-Hoc Networks) son colecciones de nodos móviles, que forman una red temporal sin la ayuda de administración centralizada donde los dispositivos están conectados únicamente por medios no cableados (Kiess & Mauve, 2007). En las MANET cada dispositivo es libre de moverse de manera independiente y arbitraria en cualquier dirección; por lo que su topología puede cambiar rápidamente y de manera impredecible (Patel & Jhaveri, 2015). A falta de enrutadores dedicados, cada nodo de una red Ad-Hoc debe ser capaz de descubrir y mantener las rutas a otros nodos para comunicarse. Este tipo de topologías son aplicables en escenarios como extensión de cobertura, redes de interconexión local o área personal, computación ubicua, censado de zonas urbanas, carreteras inteligentes y redes de emergencia, entre otros (Boukerche & Turgut, 2011).

La caracterización y estudio de rendimiento del sistema de comunicaciones es de particular interés para la predicción y optimización de performance de los algoritmos, así como la homogeneidad o heterogeneidad de los procesadores que componen la arquitectura.

2.3 Memoria distribuida

Existen varias técnicas para particionar y asignar memoria entre máquinas virtuales en un sistema virtualizado. La asignación puede ser simple basada en partición fija continua o un esquema de asignación de nivel de página dinámico como la paginación a demanda. Además, los sistemas operativos virtuales emplean mecanismos de memoria virtual para admitir multitarea haciendo uso de Unidades de gestión de memoria (Memory Management Units, MMU) para permitir el acceso controlado a la memoria de los procesos. Por ende, el aislamiento de memoria a nivel de proceso requiere que las características de hardware de MMU sean accesibles desde la máquina virtual. Permitir el acceso directo al hardware sin control del hipervisor no es una característica de diseño sencilla, ya que presenta desafíos en relación al aislamiento y la eficiencia.

La mayoría de los sistemas operativos modernos administran la memoria física dividiéndola en regiones de tamaño fijo denominadas páginas. Un sistema operativo usa la entidad “Página” para administrar la asignación, los permisos y traducir la dirección virtual a la dirección física. Las soluciones de virtualización emplean la misma noción de memoria paginada para la administración de memoria en diferentes máquinas virtuales Figura 4.

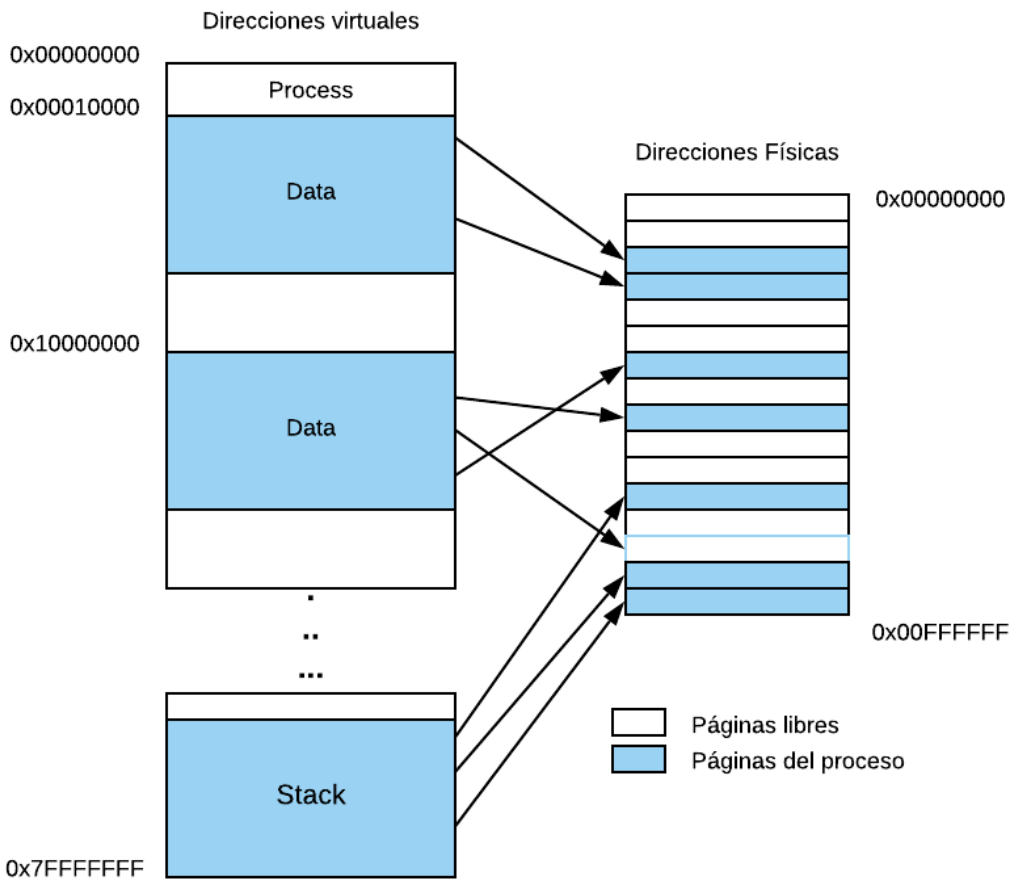


Figura 4. Paginación de memoria virtual

La memoria física linealmente direccionable a partir de la dirección física cero es la suposición subyacente de los sistemas operativos. Si el sistema operativo virtual no tiene conocimiento explícito del hipervisor, la asignación de memoria debe asegurarse de que el sistema operativo invitado no necesita un control especial para el diseño de memoria física. Para abordar este requisito fundamental, los hipervisores implementan la memoria pseudofísica como una abstracción de memoria física lógicamente continua para cada máquina virtual (Debadatta Mishra, 2018).

Un subsistema de memoria virtual proporciona la ilusión de memoria ilimitada (tan grande como el espacio de direcciones) a los procesos. La asignación de direcciones virtuales a físicas se realiza en tiempo de ejecución mediante hardware especializado, la MMU. El

mapeo de memoria virtual a física se mantiene en un nivel de granularidad de página y se indexa en varios niveles (la tabla de páginas) para reducir los requisitos de espacio del almacenamiento de información de mapeo. Cuando se programa un proceso en una CPU, el puntero a la página física de la tabla de una página de nivel uno (también conocida como directorio de página o PGD) que contiene las asignaciones de segundo nivel se carga en un registro de CPU designado (el registro CR3 en arquitecturas x86). La MMU utiliza la ubicación de memoria almacenada en el registro CR3 para recorrer la jerarquía de la tabla de páginas para llegar a la página física correspondiente a una dirección virtual determinada.

En un cambio de contexto del sistema operativo (SO Context Switch), el registro CR3 se carga con la dirección de página física que contiene el PGD del proceso. Cuando el proceso accede a una dirección virtual de 32 bits, se utilizan 10 bits más significativos de la dirección virtual como desplazamiento en el marco de página PGD para localizar la dirección del marco de página que contiene las entradas de tabla de páginas (PTEs). Los siguientes 10 bits significativos se utilizan para desplazarse en el marco de página PTE para localizar el marco de página al que se accede por el usuario. Los 12 bits menos significativos se utilizan como desplazamiento en la página física para acceder a la dirección dentro de la página (INTEL, 2020).

La orquestación de la tabla de páginas se realiza completamente en hardware sin ninguna participación de software. El sistema operativo se involucra en el cambio de contexto entre procesos o para controlar las excepciones de hardware durante la gestión de la tabla. El page-fault es la excepción más común generada en un sistema de paginación a demanda. Uno de los escenarios más comunes de errores de página es cuando el puntero de la tabla de páginas a nivel de hardware se encuentra una asignación inexistente para una dirección virtual solicitada. El sistema operativo asigna un marco de página libre al proceso (intercambia alguna otra página si es necesario) en una excepción de error de página y actualiza la tabla de página en consecuencia antes de volver con controlador de excepciones.

En los sistemas virtualizados, las estructuras de tabla de páginas son administradas por el sistema operativo invitado que funciona con páginas pseudofísicas (GPFNs). El hardware MMU que gestiona la tabla de páginas requiere marcos de página de máquina (MFN) para la traducción de direcciones virtuales a direcciones físicas. Otro aspecto es que, para

garantizar el aislamiento y la corrección, el hipervisor no concede acceso al registro CR3 a los sistemas operativos invitados. Esto implica que las modificaciones de la tabla de páginas pueden necesitar la intervención del hipervisor.

Otras propuestas como (Barham P., 2003) proponen el concepto de paravirtualización donde un subconjunto de instrucciones en x86 Instruction Set Architecture (ISA) está disponible para el sistema operativo invitado. De igual forma en (Robin J.S., 2000) proporcionan un excelente análisis de los problemas y desafíos en el diseño de soluciones de virtualización en plataformas x86 de esa época. Uno de los principales problemas en los sistemas x86 fue que varias instrucciones se ignoraron silenciosamente sin causar ningún desvío cuando se ejecuta desde un nivel de privilegio inferior. Con el enfoque "para-virtualizado", se proponen modificaciones en el sistema operativo invitado para llevar a cabo estas operaciones sensibles a través del hipervisor. Las plataformas de paravirtualidad proporcionan VMCALL o API de hiperllamada para ejecutar operaciones con privilegios por parte del hipervisor, lo que garantiza el aislamiento y la corrección. La relajación lograda debido a los cambios en el sistema operativo invitado también se puede utilizar para la virtualización eficiente de la MMU.

A medida que las plataformas virtualizadas ganan terreno en el universo de la ubicuidad, los proveedores de hardware y los investigadores se esfuerzan continuamente por diseñar hardware compatible con la virtualización. Los avances de esta naturaleza que son aplicables a la virtualización de memoria son tablas de páginas extendidas (EPT) y tablas de páginas anidadas (NPT) (Debadatta Mishra, 2018). Estas técnicas proporcionan dos estados MMU por CPU: uno para la máquina virtual y otro para el hipervisor. El sistema operativo invitado y el hipervisor tienen acceso a sus registros PRIVADOS CR3. El registro del OS CR3 del invitado señala al invitado PFN del directorio de la página (PGD) y el hypervisor CR3 señala a las estructuras de la tabla de páginas mantenidas por el hipervisor para asociar las tramas físicas del invitado a las tramas de la máquina.

Estos dos niveles de traducción de memoria necesarios para la virtualización de memoria es la principal fuente de complejidad y sobrecarga en sistemas virtualizados. El diseño óptimo de la tabla de páginas en el sistema virtualizado depende de la carga de trabajo, más específicamente del uso de memoria y la huella de acceso de las aplicaciones. La dependencia de las características de la aplicación es el aspecto más difícil en el diseño de hardware MMU genéricos. Además, los proveedores de hardware proporcionan una

flexibilidad limitada en términos de configuración del hardware de la tabla de páginas. Por ejemplo, el hipervisor no puede personalizar dinámicamente el número de niveles en las traducciones de tablas de página debido a las restricciones de hardware. Las futuras arquitecturas de hardware deberán permiten más posibilidades de personalización de software en el hardware de MMU, para que las sobrecargas en la virtualización de memoria se puedan abordar de manera más eficiente.

2.4 Redes Ad-Hoc

Una Red Ad-Hoc Wireless colaborativa (AHG) es un sistema de comunicación y computación heterogénea, del cual todos sus elementos son móviles y se encuentran en una MANET para compartir recursos sobre una abstracción de la capa física. Las AHG permiten a un grupo nodos completar misiones que requieren procesamiento y comunicación a lo largo de la red de componentes. Las aplicaciones de esta tecnología encontradas en la literatura se pueden clasificar en dos categorías: Entramado de sensores no cableados y aplicaciones de colaboración transitiva. En la primera categoría se pueden encontrar escenarios médicos en situaciones de emergencia y desastre, donde se requiere información que pueda ayudar en la toma de decisiones rápidas, como son el estado clínico, los signos vitales y el transporte de pacientes (Marinescu D. , Marinescu, Ji, Boloni, & Siegel, 2013) (Gaynor, et al., 2004). La segunda categoría hace referencia a la naturaleza esporádica de las redes Ad-Hoc, que permiten grupos transitorios de usuarios colaboradores que a través de la red pueden compartir información y proveer de una piscina (*pool*) de recursos a los nodos integrantes a través de capas de virtualización Figura 5, por ejemplo Amin et al (Amin, Laszewski, & Mikler, 2014) propuso un caso donde un grupo de científicos lograron un colaboración Ad-Hoc temporal compartiendo sus recursos para evaluar diferentes simulaciones experimentales de una aplicación; cada involucrado contribuyo con diferentes servicios de simulación, visualización y almacenamiento de datos. Otras aplicaciones potenciales puede ser la implementación de redes locales con la llegada de IoT (Miorandi, Sicari, & Pellegrini, 2012) y modelos de ciudades inteligentes.

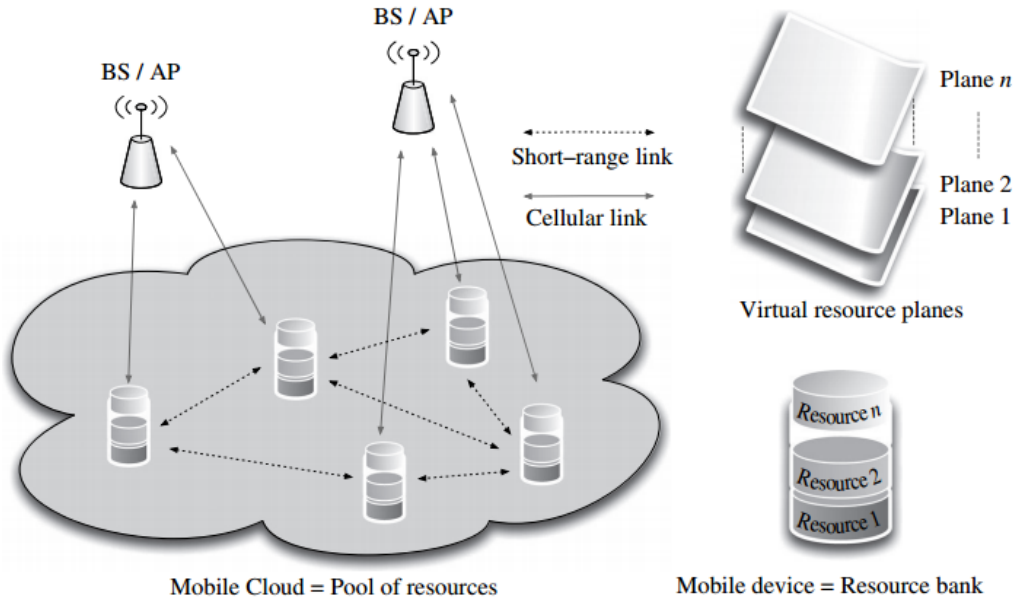


Figura 5. Red Ad-Hoc como piscina (pool) de recursos

Uno de los desafíos más grandes de las wireless AHG es el descubrimiento y descripción de recursos. Antes de compartir los recursos en un ambiente ad-hoc, los usuarios deben ponerse de acuerdo en cómo los describirán. En un ambiente ad-hoc heterogéneo, donde diferentes dispositivos pueden comunicarse esporádicamente para compartir información o servicios, el lenguaje de descripción debería ser suficientemente abierto y general para ser capaz de describir esa variedad de recursos (McKnight, Howison, & Bradne, 2004). Una vez hayan sido propiamente descritos, el descubrimiento de estos se vuelve una mayor preocupación.

2.4.1 Descubrimiento de recursos en redes ad-hoc

Tradicionalmente en redes cableadas el descubrimiento de recursos se aborda desde una perspectiva jerárquica, donde el servidor raíz o sus directorios agregados reúnen la información de sus nodos hijo (Czajkowski, Fitzgerald, Foster, & Kesselman, 2001). En redes Ad-Hoc sin esta organización predefinida este mecanismo jerárquico no es viable. Sin embargo, en este contexto se han propuesto soluciones basadas mecanismos peer-to-peer distribuidos (Koodli & Perkins, 2012) (Frank & Karl, 2014) (Helal, Desai, V. Verma, & Konark, 2003). Los modelos propuestos en la literatura se pueden ubicar en dos tipos de arquitecturas, Peer To Peer (P2P) y Directory Based (DB) (Figura 6).

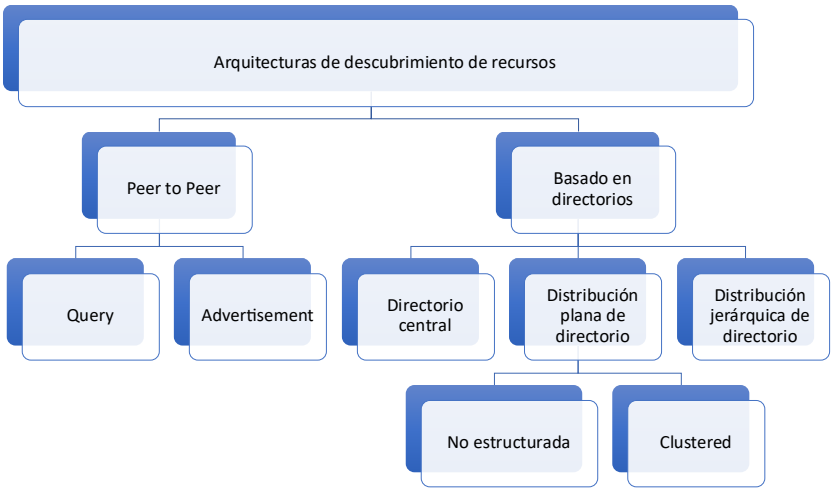


Figura 6. Descubrimientos de recursos

2.4.1.1 Arquitecturas Peer to Peer:

En este tipo de arquitecturas se basan en la negociación entre entidades una a una para descubrir los recursos disponibles en los nodos y determinar cual de ellos es apto para los requerimientos de un nodo que busca un servicio en específico. Se pueden definir dos mecanismos: Pull y Push.

Los mecanismos pull como el utilizado en BT-SDP (Bluetooth Interest Group, 2001) son de descubrimiento activo por demanda. En ellos cada nodo envía un mensaje de petición a la red sea por broadcast o multicast, solicitando servicios o recursos que cumplan con algún requerimiento o atributo en particular. Los proveedores responden a este mensaje con la descripción de sus recursos. Otro ejemplo de implementación de este tipo de mecanismo se encuentra en (Koodli & Perkins, 2012) que propone un protocolo de descubrimiento de servicios para redes ad-hoc con protocolos de enrutamiento on-demand (AODV, DSR, et). Su aproximación radica en adherir un header del servicio de descubrimiento a las tramas de enrutamiento. El protocolo usa dos tramas para el servicio solicitado/encontrado y para la ruta hacia el nodo que solicito/contesto una petición de recursos. Cada ves que un nodo descubre una nuevo servicio o proveedor guarda su información; cuando requiere algún servicio especifico envía un paquete con el encabezado del servicio que solicita y los nodos receptores, si no contienen información de algún nodo a su alcance que pueda contestar la petición, retransmite el mensaje, mientras que si puede cubrir las necesidades de la petición contesta con la ruta y servicio disponible de vuelta al cliente.

Otro mecanismo P2P es el PUSH, un modelo pasivo que se basa en la publicación periódica de atributos de servicios y recursos junto con su localización en la red, de manera que cada nodo pueda formar una vista local de la red. Un ejemplo de este tipo de descubrimiento es se usa en (Venanzi & Kantarci, 2018); este modelo permite una arquitectura descentralizada basada en controladores que recupera las descripciones de los servicios o dispositivos y envía/recibe variables de estado utilizando para el descubrimiento de recursos el Simple Service Discovery Protocol (SSDP) trasmitiendo un descriptor en formato XML (Khalila & Elgazzar, 2020).

2.4.1.2Arquitectura basada en directorios

La idea de esta arquitectura es utilizar un repositorio común como fuente de información de los servicios disponibles. Los proveedores pueden registrar sus recursos en esta base con el fin de que los consumidores accedan a ella cuando requieren algún servicio particular.

Bajo esta arquitectura se pueden tener dos aproximaciones para la persistencia del directorio central: Una centralizada y otra distribuida. Un directorio centralizado es una solución sencilla y fácil de administrar, pero en una red ad-hoc inalámbrica, este tipo de mecanismos representan un punto crítico de fallo y a su vez un cuello de botella para la capacidad de respuesta de la red.

El otro modelo de directorio es el distribuido en donde varios directorios cooperan bajo protocolos P2P para mantener la información de recursos y servicios esparcida en los nodos, ya sea mediante difusión amplia (Broadcast) de cada nodo para mantener copias locales de la base de datos, lo cual genera gran cantidad de tráfico o manteniendo una jerarquía en la red que permita cerrar la difusión a grupos específicos manteniendo relación padre-hijo para evitar saturar el medio.

Un ejemplo de esta arquitectura es el Intentional Naming System (INS/Twine) desarrollado por el MIT, que utiliza introduce dos componentes para el descubrimiento de recursos: el resolutor de nombres (INR) y el resolutor de dominios (DSR). Los nodos se registran a un INR enviándole publicaciones periódicas de sus servicios. Los INR a su vez hacen difusión de esta información únicamente con otros INR en la red conservando la estructura de árbol de dos niveles. Los resolutores de dominio son los que almacenan la lista de INRs activos y candidatos.

Otra falencia son las vulnerabilidades en seguridad en términos de (Moreno-Vozmediano, 2009) autenticación, encriptación y autorización (Qian Kang & Yao, 2016) (Subba, Biswas, & Karmakar, 2015). En los ambientes Ad-Hoc, las relaciones de colaboración no pueden ser anticipadas y son de naturaleza temporal, así que la estructura de seguridad debe ser adaptada a este contexto. Algunos trabajos como (Lorch & Kafura, 2002) proponen un acercamiento basado en manejo de privilegios y estrategias de refuerzo. Otra propuesta en este tema, es establecer un mecanismo de confianza entre los nodos ante la ausencia de una autoridad central usando Key management solution (KMS) (Liao & Manulis, 2007).

La calidad de servicio (QoS) en redes Ad Hoc es inherentemente inestable y poco fiable, debido a que no se pueden realizar predicciones acerca de la conectividad o disponibilidad de recursos. Es por esto por lo que para proveer criterios aceptables de QoS se deben implementar mecanismos de reserva de recursos y *Quality Negotiation* (Amin, Laszewski, & Mikler, 2014). Las restricciones de energía de las redes móviles es otro problema para considerar en el desarrollo de redes Ad Hoc. La política de *scheduling* debe ser capaz de examinar los dispositivos disponibles y seleccionar únicamente los recursos con las características de poder y rendimiento que mejor se acomoden a las tareas a ser ejecutadas (Marinescu & Marinescu, 2013).

2.4.2 Asignación de recursos en redes ad-hoc

La investigación en asignación de recursos en redes móviles ad hoc se encuentra madurando, como se observa en (Figura 7) desde 2004 el tema “Manejo de recursos locales” (*Local* Resource *Management)* ha estado en constante exploración según el número de patentes que se introducen año a año y solo algunos esquemas basados en arquitecturas descentralizadas han sido propuestos.

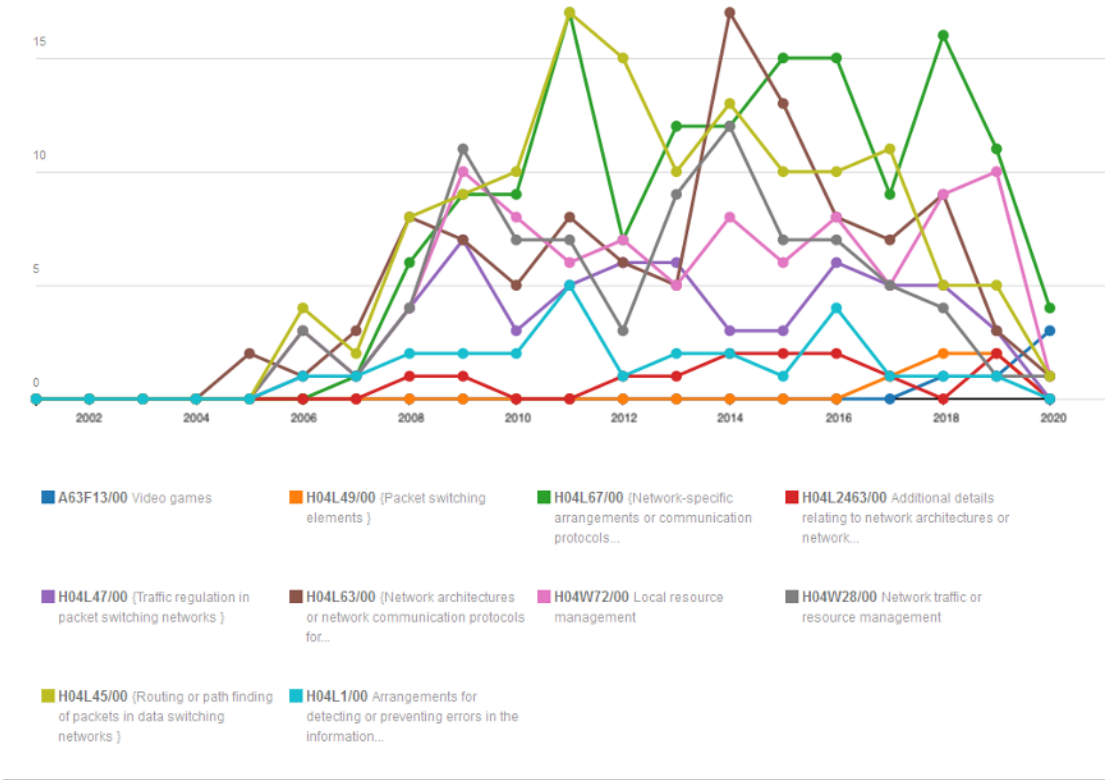


Figura 7. Tendencias en la actividad de patentes en "Ad Hoc Networks AND Distributed Systems"

En (Hummel & Jelleschitz, 2007) se propone una administración de recursos basado en colas FCFS (First-Come, First-Served) que permite a cada dispositivo hacer mapeo (*mapping)* basado en las características y condiciones de la tarea (*job)*. Además, emplea un mecanismo de tolerancia de errores y soporta ejecución redundante de tareas para mitigar fallos en sus dependencias. En (Li & Li, 2009) se propuso un esquema de scheduling para redes Ad-Hoc considerando las restricciones de energía de los nodos. En este modelo las tareas se caracterizan bajo ciertos criterios, como el tiempo máximo de espera, su presupuesto, el volumen de datos y sus requerimientos durante la ejecución; los nodos que están dispuestos a compartir recursos especifican su precio y los dispositivos que consumen servicios aceptan su prestación basados en ese precio y el tiempo de ejecución esperado. Otro modelo de asignación de recursos power-aware es el presentado en (Liu, Roeder, Walsh, Barr, & Sirer, 2015). Este modelo busca reducir la media de distancia de los caminos que trazan los paquetes de datos mediante la migración de tareas a nodos topológicamente más cercanos; sin embargo, la migración ocurre luego

del análisis de los patrones de la comunicación de datos a lo largo de la ejecución de la tarea.

Para el proceso de mapping, Gomes *et al* (Gomes, 2007) propone un esquema que utiliza el mecanismo de delay de respuesta según el cual un nodo con más recursos responde antes de un nodo con menos; (Gomes, 2007) también provee balanceo de carga y escalabilidad. Otro esquema basado en la topología es el presentado en (Selvi, Sharfraz, & Parthasarathi, 2007), el cual perfila la movilidad de los nodos en el tiempo; este perfil consiste en los lugares visitados por el nodo y la cantidad de tiempo gastado en cada uno. De este modo, se selecciona el nodo que haya estado previamente en el lugar por más tiempo para que ejecute la tarea en cola.

Finalmente, en la revisión de la literatura se encontró que la mayoría de los trabajos en asignación de recursos en redes de colaboración (*Grids*) computacional, se basa en sistemas de una gran infraestructura de comunicaciones a través de redes de alto rendimiento (McClatchey, 2017) (Villela, 2010) (Chang, 2009) (Hong & Prasanna, 2014), lo cual, no obedece a la naturaleza de una red ad hoc. Estos elementos nos llevan a evidenciar las brechas que hay en virtualización inalámbrica, implementaciones de redes Ad Hoc en sistemas embebidos y el problema de los servicios sobre dispositivos móviles inalámbricos heterogéneos. Con la presente propuesta se abordará el problema de la virtualización de recursos de procesamiento, la cual es el enlace entre la capa de hardware, en este caso la red Ad Hoc y la primera abstracción del sistema TLÖN, la capa lógica de virtualización.

2.5 Arquitectura de red dinámicas

Durante la última década, el IoT ha transformado el uso de Internet y se ha convertido en un término popular entre los investigadores. El IoT permite conectar personas y cosas (por ejemplo, sensores, actuadores y dispositivos inteligentes) en cualquier momento y en cualquier lugar, con cualquier persona y cualquier cosa (S.A. Alabady, 2018). Se espera que la mayoría de estos objetos y dispositivos tengan capacidades de detección. Pueden detectar y recopilar datos del entorno que nos rodea y luego compartir los datos a través de Internet donde se pueden procesar para diferentes propósitos. Conectar un gran número de objetos físicos con capacidades de detección (por ejemplo, sensores) a Internet introduce el concepto de "big data" que necesita un almacenamiento eficiente e inteligente

y su análisis puede ser la base para el diseño y la planificación de ciudades inteligentes sostenibles. Claramente, los objetos conectados requieren mecanismos eficientes para almacenar, procesar y recuperar datos. Sin embargo, con millones de dispositivos interconectados, el big data generado puede ser significativamente extenso en comparación con el tráfico de datos tradicional. Por lo tanto, no es posible utilizar los entornos de hardware disponibles y herramientas de software para administrar y procesar datos con un tiempo de respuesta aceptable (A. Al-Fuqaha, 2015). La tecnología en la nube permite a las organizaciones e individuos utilizar muchos recursos de forma remota y a un costo razonable. Hoy en día, la computación en la nube se utiliza ampliamente tanto en la industria como en el mundo académico. Sin embargo, todavía tiene algunas limitaciones siendo la mayor, la conectividad entre la nube y los dispositivos finales. Este enlace se establece a través de Internet lo que la hace inapropiada para un gran número de aplicaciones sensibles a la latencia, como vehículos conectados y grids inteligentes. Por lo tanto, la implementación independiente de componentes de aplicación en nubes separadas o cluster de nubes privadas se ha vuelto común. Sin embargo, esto hace que la latencia sea aún peor como resultado de la sobrecarga causada por las comunicaciones entre nubes. La computación de niebla es un paradigma introducido para hacer frente a estas limitaciones (C. Mouradian, 2017).

2.5.1 Fog Computing

Fog computing (FC) es una arquitectura que extiende la arquitectura de la computación en la nube hasta el borde de la red y distribuye el cálculo, el control y el almacenamiento de datos más cerca de los usuarios finales. La relevancia de la tecnología FC se puede identificar fácilmente cuando se consideran las limitaciones de la tecnología tradicional de la nube, así como las nuevas oportunidades introducidas a través de la aparición de las tecnologías relacionadas con el IoT y 5G. Con la informática FC, el procesamiento de aplicaciones basadas en la nube y sensibles a la latencia puede tener lugar en el borde de la red, mientras que otras aplicaciones tolerantes a retrasos se pueden manejar en la nube (C. Mouradian, 2017). Además, la tecnología de niebla proporciona baja latencia al permitir que las tareas de procesamiento se manejen cerca de los dispositivos finales, en el borde de la red. Además, a través de puntos de acceso, *proxies* y enrutadores ubicados en el borde de la red y cerca de las fuentes, la informática FC ofrece puntos muy distribuidos para recopilar datos generados por los dispositivos finales. El FC también puede garantizar

una mayor disponibilidad que sistemas basados en la nube ya que esta es menos fiable debido a varios problemas de conectividad (I. Stojmenovic, 2014).

El FC nos proporciona una arquitectura a nivel de sistema para una distribución óptima de las capacidades de redes, computacionales y de almacenamiento en una red. Esta arquitectura tiene como objetivo proporcionar un equilibrio adecuado entre las tres capacidades y asegurarse de que se encuentran óptimamente en la red. La computación de niebla extiende las capacidades básicas de la computación en la nube hasta el borde de la red, a veces hasta los sensores y actuadores. También extiende las diversas técnicas de nube, como la virtualización, los hipervisores y la seguridad hasta el borde con la ayuda de las capas de niebla. La arquitectura de niebla también ofrece ventajas de rendimiento, escalabilidad, eficiencia y seguridad para aplicaciones críticas de IoT. Fog Computing es el nuevo paradigma/arquitectura que proporciona servicios limitados de computación, almacenamiento y red en los dispositivos del usuario final en el perímetro del usuario Figura 8.

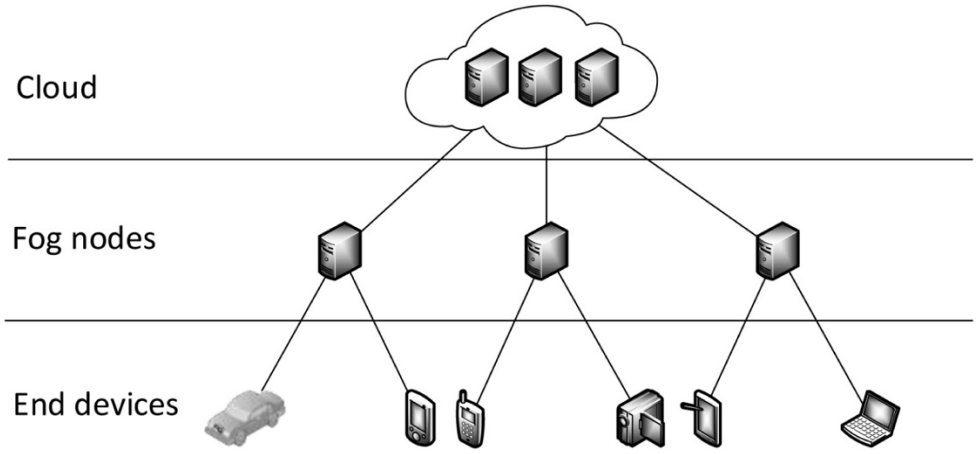


Figura 8. Fog Computing

Los nodos FC son el elemento fundamental de la arquitectura de “computación en la niebla”. Son elementos modulares de hardware y software que se configuran en orden para realizar funciones específicas. Los nodos FC tienen potencia de procesamiento y almacenamiento, lo que los convierte en un elemento esencial de la arquitectura de niebla al proporcionar un medio descentralizado para el almacenamiento de información. Estos nodos de niebla se distribuyen entre los diversos niveles jerárquicos de las capas de niebla. La configuración de estos nodos de niebla depende de su distancia desde el servidor de

nube central. Los nodos FC que están más cerca de los puntos finales tienen hardware y software modesto. Este tipo de nodos incluyen actuadores, sensores, hubs, etc. Mientras que los nodos FC que están cerca de la nube tienen hardware y software sofisticados con el poder de procesar una gran cantidad de datos, como por ejemplo enrutadores inteligentes, cámaras de vigilancia, etc.

El FC puede reducir la dependencia de la arquitectura de la nube y proporcionar un mecanismo para que los dispositivos perimetrales funcionen sin interrupción durante un período de tiempo razonable, incluso si la conexión a la nube ya no está disponible y se pierde. Además, es compatible con la agregación de datos de dispositivos heterogéneos, como múltiples sensores relacionados con el cuidado de la salud. proporcionando seguridad y protección de datos para información privada y datos confidenciales como la ubicación y los datos relacionados con la medicina (M. Aazam, 2018).

2.5.2 Edge computing

Mobile Edge Computing (MEC) proporciona entornos de servicios de Internet y funciones de computación en la nube en el borde de las redes móviles, cerca de redes de acceso radioeléctrico (RAN) y usuarios móviles. En realidad, el concepto principal de MEC y computación perimetral es el mismo: la ubicación de la informática está cerca de la fuente de datos y está más cerca de los usuarios. Desde la perspectiva de la biónica, es más fácil describir la definición de computación de borde: el pulpo hace pleno uso de sus tentáculos durante la caza, y cada tentáculo coopera sincrónicamente, nunca anudando. Las razones de este fenómeno son principalmente en que el sistema nervioso del pulpo es similar a un sistema distribuido simple: un cerebro en el centro es responsable de la programación, y múltiples unidades de procesamiento se encuentran en los tentáculos para la coordinación. La informática perimetral también es una especie de patrón de computación distribuida: los datos recopilados se pueden procesar en la puerta de enlace inteligente que está más cerca de los usuarios, sin cargar una gran cantidad de datos en el centro de la nube.

Los paradigmas de computación perimetral consisten en dispositivos conectados con recursos limitados como teléfonos inteligentes, gadgets portátiles, computadoras de placa única (SBC), dispositivos de red (interruptores, enrutadores) y recursos con capacidades moderadas como estaciones base (BS), LAN, Cloudlets, por nombrar algunos. Los dispositivos generan y recopilan una gran cantidad de datos que se procesan previamente

y se analizan inicialmente en el borde de la red. Estos datos se pueden transferir a una nube centralizada para extraer conocimiento profundo (deep knowledge) utilizando diferentes enfoques de Machine Learning (ML) y Optimización (convexos o bayesianos) (S. & M., 2019). Por lo tanto, la informática de borde es complementaria a la computación en la nube para aprovechar el poder de los recursos de vanguardia distribuidos para admitir aplicaciones de IoT como casas inteligentes, redes, tiendas minoristas, agricultura, automóviles autónomos conectados y atención sanitaria.

Al igual que la computación en la nube, la tecnología base de los paradigmas de computación perimetral es la virtualización que desacopla los recursos de hardware del software para ejecutar varios hosts en el mismo hardware. Sin embargo, la informática de borde tiene diferencias distintivas con la computación en la nube en términos de conciencia de ubicación, servicios basados en la movilidad, dispositivos con recursos limitados y heterogéneos y distribución amplia de dispositivos. Estas diferencias implican que la virtualización pesada (ejecutar el monitor de máquina virtual (VMM) directamente en hardware o sistema operativo) no puede ser aplicable a todos los casos de uso de aplicaciones de IoT que pueden aprovechar la informática perimetral. Esta restricción conduce a la implementación de técnicas de virtualización ligeras como contenedores y unikernels en los dispositivos perimetrales con capacidades de recursos limitadas y moderadas. Estas técnicas se pueden reducir a recursos de borde, ya que requieren menos ciclos de CPU, menor tamaño de huella de memoria, menos tiempo de creación de instancias y más agilidad en la movilidad. El uso puro de una técnica puede no ser eficaz en el rendimiento (Morabito, Cozzolino, & Ding, 2018) y una combinación de técnicas se puede utilizar en función de las características de virtualización, las capacidades de recursos y los requisitos de la aplicación. En el siguiente capítulo se abordará los diferentes modelos de virtualización encontrados en la literatura.

3. Virtualización

La virtualización de hardware es la virtualización de equipos como plataformas de hardware completas, ciertas abstracciones lógicas de sus componentes o solo la funcionalidad necesaria para ejecutar varios sistemas operativos. La virtualización oculta las características físicas de una plataforma informática a los usuarios, presentando en su lugar una plataforma informática abstracta. En sus orígenes, el software que controlaba la virtualización se llamaba un "programa de control", pero los términos "hipervisor" o "Máquina virtual" (VM) se volvieron mas relevantes con el tiempo.

Una máquina virtual es una implementación de software de un equipo que ejecuta programas como una máquina física. En (Gerald J. Popek, 1974), se definen condiciones que pueden probarse para determinar si la arquitectura puede admitir una máquina virtual, describiéndola como "un duplicado eficiente y aislado de una máquina real". El concepto fundamental de una máquina virtual gira en torno a una aplicación de software que se comporta como si fuera su propio equipo. Este es el mecanismo de intercambio de trabajos original prominente en los mainframes. La aplicación de máquina virtual ("invitado") ejecuta su propio sistema operativo autónomo en el equipo real ("host"). En pocas palabras, una máquina virtual es un equipo virtual que se ejecuta dentro de un equipo físico. El sistema operativo virtual puede variar desde un entorno Windows hasta un entorno Macintosh y no se limita a uno por máquina host. Por ejemplo, puede tener un equipo host de Windows 10 con máquinas virtuales Linux y Windows Server (Figura 9).

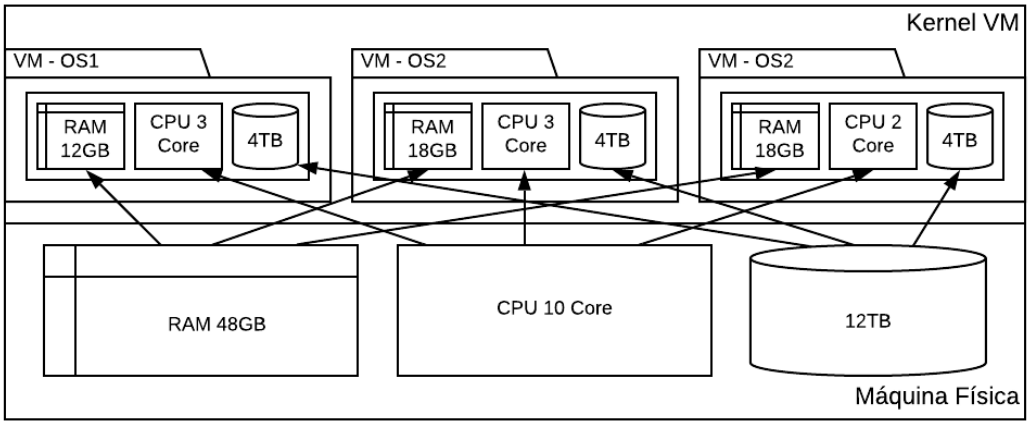


Figura 9. Máquinas virtuales

(Gerald J. Popek, 1974) explica la virtualización a través de la idea de un monitor de máquina virtual (VMM). Un VMM es una pieza de software que tiene tres características esenciales. En primer lugar, VMM proporciona un entorno para programas que es fundamentalmente idéntico al entorno de la máquina física; en segundo lugar, los programas que se ejecutan en este entorno tienen muy poca degradación de velocidad en comparación con la máquina física; y, por último, VMM tiene control total de los recursos del sistema. Sin embargo, la arquitectura x86 no logró la "virtualización clásica" como se define por los requisitos de virtualización de (Gerald J. Popek, 1974), y se utilizó en su lugar la traducción binaria del código del kernel virtual. Esto se debe a que los sistemas operativos x86 están diseñados para ejecutarse directamente en el hardware físico y suponen que controlan el hardware del equipo. La virtualización de la arquitectura x86 se ha logrado a través de la virtualización completa o la paravirtualización. Ambos crean la ilusión de hardware físico para lograr el objetivo de la independencia del sistema operativo del hardware, pero presentan algunas compensaciones en rendimiento y complejidad. La creación de esta ilusión se realiza colocando una capa de virtualización debajo del sistema operativo para crear y administrar la máquina virtual.

En un sistema con particiones físicas, se puede hospedar más de un sistema operativo en el mismo equipo. El ejemplo más común de este tipo de entorno es un sistema de arranque dual donde los sistemas operativos Microsoft y Linux coexisten en el mismo sistema. Cada una de estas particiones está siendo compatible con un único sistema operativo. los productos de software de máquina virtual permiten encapsular una pila completa de software en un contenedor denominado máquina virtual. La encapsulación comienza en el sistema operativo y se ejecuta hasta el nivel de aplicación. Este tipo de tecnología tiene la capacidad de ejecutar más de una máquina virtual en un único equipo físico siempre que el equipo tenga suficiente potencia de procesamiento, memoria y almacenamiento. Las máquinas virtuales individuales se aíslan entre sí para proporcionar un entorno compartimentado. Cada máquina virtual contiene su propio entorno al igual que un servidor físico e incluye un sistema operativo, aplicaciones y capacidades de red. Las máquinas virtuales se administran individualmente de forma similar a un entorno físico. A diferencia de una máquina con particiones físicas, los productos de máquina virtual permiten que existan varios sistemas operativos en una partición.

La virtualización a nivel de sistema desacopla el hardware físico del sistema operativo. Mediante el uso de un hipervisor, que es una capa de abstracción de software entre la máquina física y las máquinas virtuales, varios sistemas operativos pueden compartir el mismo hardware subyacente y ejecutarse simultáneamente en una máquina host. Esto permite un uso más eficiente de los recursos de hardware. Otras ventajas clave de la virtualización incluyen el aislamiento de la carga de trabajo y la asignación dinámica de recursos (Rosenblum M., 2005).

3.1 Modelos de virtualización

A continuación se hará énfasis en varias aproximaciones a la virtualización de sistemas y su evolución desde un enfoque clásico hacia sistemas distribuidos desacoplados como el modelo de contenerización de aplicaciones.

3.1.1 Virtualización de Recursos

La virtualización de hardware, se ejecuta en una plataforma de particular de hardware mediante software en el host que en esencia, oculta el hardware físico de la vista de las aplicaciones. El software host que es en realidad un programa de control, se denomina un hipervisor. El hipervisor crea un entorno informático simulado para el software guest que podría ser cualquier cosa, desde las aplicaciones de usuario hasta sistemas operativos completos. El software guest funciona como si se estuviera ejecutando directamente en el hardware físico. Sin embargo, el acceso a recursos físicos como el acceso a la red y los puertos físicos se administra normalmente en un nivel más restrictivo que el procesador y la memoria. A menudo se restringe a los huéspedes el acceso a dispositivos periféricos específicos.

3.1.1.1 Unidades de procesamiento

La virtualización de CPU implica una sola CPU que actúa como si fueran varias CPU separadas. La razón más común para hacer esto es ejecutar varios sistemas operativos diferentes en una máquina. La virtualización de CPU hace énfasis en el rendimiento y se ejecuta directamente en las CPU disponibles siempre que sea posible. Los recursos físicos subyacentes se usan siempre que sea posible y la capa de virtualización ejecuta

instrucciones solo según sea necesario para que las máquinas virtuales funcionen como si se estuvieran ejecutando directamente en una máquina física.

La virtualización de CPU también tiene la intención de ejecutar programas e instrucciones a través de la máquina virtual dando la sensación de que está trabajando en una estación de trabajo física. Todas las operaciones son manejadas por un emulador que controla el software para ejecutarse de acuerdo con él. Sin embargo, la virtualización de CPU no actúa como un emulador. El emulador funciona de la misma manera que lo hace un equipo de equipo normal. Replica la misma copia o datos y genera la misma salida al igual que una máquina física. La función de emulación ofrece una gran portabilidad y facilita el trabajo en una sola plataforma que actúa como trabajar en múltiples plataformas.

Con la virtualización de CPU, todas las máquinas virtuales actúan como máquinas físicas y distribuyen sus recursos de hospedaje al igual que tener varios procesadores virtuales. El uso compartido de recursos físicos tiene lugar en cada máquina virtual cuando todos los servicios de hospedaje reciben la solicitud. Por último, las máquinas virtuales obtienen una parte de la única CPU asignada, siendo un procesador único que actúa como procesador dual

3.1.1.2 Memoria primarias

La gestión de la memoria de un sistema informático en un sistema operativo multitarea es un área bien investigada (Herlihy & Shavit, 2020). Los sistemas operativos particionan la memoria en fragmentos administrados de forma independiente y los asignan a diferentes entidades de ejecución, como procesos y subprocesos en tiempo de ejecución. La segmentación y la paginación son instancias comunes de técnicas de administración de memoria basadas en particiones. El soporte de hardware como el hardware de segmentación, la unidad de administración de memoria (MMU) y el búfer de búsqueda de traducción (TLB) para la implementación eficiente de la segmentación y la paginación suelen estar disponibles en la mayoría de los procesadores de productos básicos modernos. La virtualización de memoria agrega una capa adicional de administración de memoria para la que los enfoques tradicionales basados en MMU y la administración de metadatos deben ser controlados (y si es necesario) propiedad del hipervisor para garantizar la corrección y el aislamiento.

3.1.1.3 Memorias secundarias

La administración de la memoria en la virtualización de sistemas es un reto debido a los desafíos generales de administración de recursos de los entornos virtualizados: oscuridad, dualidad, diversidad y dinamismo. Además, a diferencia de los recursos multiplexados por tiempo como los dispositivos de CPU y E/S, la memoria es multiplexada por espacio (particionada). Con recursos multiplexados en el tiempo, el hipervisor obtiene periódicamente el control del recurso para la toma de decisiones. Por ejemplo, el programador de CPU de nivel de hipervisor se puede invocar periódicamente para cambiar las asignaciones de CPU a máquinas virtuales. Con los recursos con particiones de espacio, esta toma de decisiones periódica en la ruta de acceso a los recursos o de asignación no es posible y aumenta el desafío del aprovisionamiento eficiente de recursos.

3.1.1.4 Almacenamiento

La virtualización de almacenamiento se utiliza normalmente en redes de área de almacenamiento (Storage Area Network, SANs). En un sistema SAN, todos los discos de almacenamiento independientes de la red se agrupan y luego se combinan en una matriz. A continuación, los servidores pueden acceder a la matriz como si fuera un dispositivo de almacenamiento local. Los SAN se componen de varios componentes, como matrices de discos y switches. El espacio de almacenamiento de cada unidad de una SAN se divide lógicamente en bloques de almacenamiento que se pueden asignar como un grupo más grande que abarca varias unidades físicas. Estos bloques de almacenamiento se administran como un único dispositivo de almacenamiento virtual denominado pool de almacenamiento.

Se pueden identificar cinco tipos de virtualización de almacenamiento (Diane Barrett, 2010):

- Virtualización de unidades de disco: tiene lugar cuando el firmware abstrae el cilindro físico, la cabeza y el sector en una única dirección de bloque lógico.
- El particionamiento de sistemas de almacenamiento: se produce cuando el sistema de almacenamiento crea varias particiones a partir de un único recurso de almacenamiento.
- La virtualización de bloques: se produce cuando el software abstrae varias matrices de discos en un único recurso de almacenamiento.

- La virtualización de cintas: se produce a medida que el software abstrae tanto las unidades de cinta como los medios de cinta en un único recurso de disco.
- La virtualización de archivos: es la creación de una capa de abstracción entre los servidores de archivos y los clientes que acceden a esos servidores de archivos.

La virtualización del almacenamiento de información ayuda a reducir la alta sobrecarga administrativa de la administración de SANs al proporcionar un entorno más fácil de administrar, que se puede utilizar para realizar copias de seguridad, archivar y recuperar datos en un período de tiempo más corto y hace que la complejidad de un SAN parezca menos formidable. La idea detrás de la virtualización de almacenamiento es agrupar y compartir recursos de almacenamiento. Este tipo de virtualización ofrece la capacidad para servicios de almacenamiento como el aprovisionamiento ligero, la replicación y la migración de datos.

3.1.2 Virtualización ligera

Con la creciente omnipresencia del paradigma de computación en la nube para todo tipo de aplicaciones, las técnicas de virtualización de baja sobrecarga se están volviendo indispensables. En particular, la arquitectura de microservicios, donde los pequeños servicios encapsulados son desarrollados, operados y mantenidos por equipos separados, requieren imágenes de máquinas y SO fáciles de usar y desechables. Idealmente, dicha infraestructura debería permitir un aprovisionamiento rápido y un funcionamiento eficiente. Los enfoques para la virtualización ligera se dividen aproximadamente en las categorías de virtualización de contenedores y unikernels (Figura 10, Fuente: (Goethals & Sebrechts, 2018)).

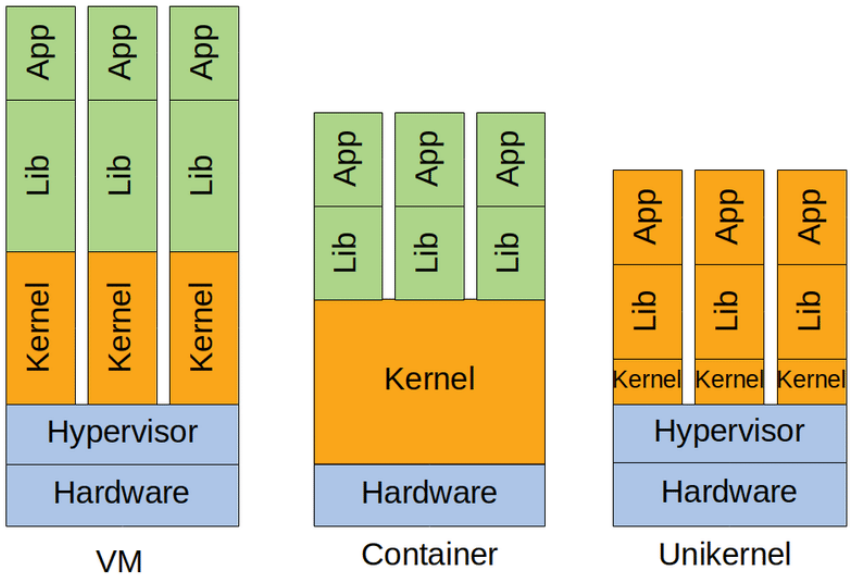


Figura 10. Comparación VM, Contenedor y unikernel. (Goethals & Sebrechts, 2018)

3.1.2.1 Contenedores

Los contenedores se introdujeron como un enfoque de oposición para la virtualización de todo el sistema. Se basaron en la observación de que todo el núcleo induce una sobrecarga excesiva de recursos por simplemente aislar y empaquetar pequeñas aplicaciones. Existen dos clases de enfoques de virtualización de contenedores: contenedores orientados a aplicaciones y a sistema operativo. Para los contenedores orientados a aplicaciones a aplicaciones, las aplicaciones individuales constituyen las unidades de implementación. Para los contenedores orientados al sistema operativo, se reproduce todo el espacio de usuario del sistema operativo. Este enfoque era particularmente popular entre las soluciones de servidor privado virtual (VPS), donde el ahorro de recursos era esencial. Actualmente, con LXD, este enfoque se está volviendo más prominente de nuevo, ya que permite la creación de un comportamiento similar a la máquina virtual (VM) sin la sobrecarga de un hipervisor.

A continuación se presentan las soluciones mas relevantes en el mercado para contenerización de aplicaciones.

Docker: Entre los contenedores orientados a aplicaciones, el proyecto de código abierto Docker es actualmente el enfoque más popular. Se basa en características del kernel de

Linux, como espacios de nombres y grupos de control, para aislar contenedores independientes que se ejecutan en la misma instancia del sistema operativo. Un contenedor de Docker encapsula una aplicación, así como sus dependencias de software; se puede ejecutar en diferentes máquinas Linux con el motor Docker. Además de proporcionar aislamiento básico y un rendimiento más cercano a la nativa que la virtualización de todo el sistema, la contenerización de Docker tiene las ventajas de que los contenedores Docker precompilados se pueden compartir fácilmente y que la tecnología se puede integrar en varias soluciones populares de infraestructura como servicio (IaaS), como los servicios web de Amazon (AWS), Azure, entre otros.

LXD: La solución de contenedor basada en Linux LXD se basa en la interfaz LXC (contenedor Linux) para las características de contenedor de Linux. LXD utiliza la biblioteca LXC para proporcionar contenedores de sistema operativo de baja sobrecarga. Además de las funciones avanzadas de creación y gestión de contenedores, LXD ofrece integración en el componente informático OpenStack Nova.

Lmctfy (Let Me Contain That For You): es un proyecto de código abierto de Google que proporciona contenedores de aplicaciones Linux. Se basa internamente en cgroups de Linux y proporciona más características de supervisión y administración en modo de usuario.

3.1.2.2 Unikernel

Unikernels son una reaparición del concepto de sistema operativo de biblioteca, que solo proporciona una capa delgada de protección e instalaciones de multiplexación para los recursos de hardware, mientras que la compatibilidad con hardware se deja a las bibliotecas empleadas y a la propia aplicación. Mientras que los sistemas operativos de bibliotecas tuvieron que tener problemas para tener que admitir hardware real, unikernels evitan esta carga al admitir sólo interfaces de hardware virtuales proporcionadas por hipervisores o VMMs

3.2 Orquestadores

Desde un punto de partida general, la orquestación es el uso y la selección de recursos por un orquestador para satisfacer las demandas del cliente según el nivel de servicio. Las principales funciones de la orquestación son dos. En primer lugar, la orquestación implica dividir las solicitudes de servicio de carga pesada en componentes de servicio. Además, distribuye los componentes antes mencionados entre las plataformas soportadas, creando una solución integrada de extremo a extremo en múltiples dominios

En el contexto de las redes Ad-Hoc se propone desacoplar el plano de control del plano de datos, con esta arquitectura propuesta los enrutadores y switches hacen envíos a elementos lógicos de la red proporcionados por una entidad externa, denominada Sistema de operación de Red (NOS en inglés), este controlador permite generar abstracciones que desplieguen servicios orquestados en los niveles lógicos y virtuales, permitiendo de esta forma la automatización.

Ahora bien el termino orquestación esta presente en varias disciplinas como la música, multimedia, computación en la nube, Redes definidas por Software (SDN) entre otras, pero la analogía es el conjunto de instrumentos que tocan juntos una sinfonía, el trabajo de cada uno de los instrumentos toma sentido cuando los otros se unen en completa sincronía para obtener la melodía deseada, se puede indicar que un orquestador puede coordinar tareas dentro de un sistema en diferentes capas , manejar flujos de tráfico y definir las acciones dependiendo de las interacciones entre las aplicaciones y el usuario, es decir, hay una diferencia entre automatizar y orquestar. En el campo de aplicaciones web por ejemplo, el proceso de orquestar puede ser definido como la ejecución de un proceso que puede interactuar con los procesos y/o servicios internos y externos, de forma dinámica, flexible y adaptables a los cambios para mantener la continuidad de negocio, la lógica de ejecución de actividades y la automatización (Ceballos, 2018).

Actualmente, el ámbito de orquestación se ha vuelto más amplio y abarca la automatización del ciclo de vida del servicio de red de extremo a extremo. En el entorno de SDN, la orquestación hace referencia a una función general para administrar y automatizar el comportamiento de red (Nathan F.Saraiva de Sousa, 2019). Según (Nathan F.Saraiva de Sousa, 2019), la orquestación del servicio se refiere al control programático de la infraestructura subyacente, incluidas las redes existentes y las tecnologías de habilitación, como SDN y NFV (Network Function Virtualization) (Mansouri & Babar, 2021).

En general, la orquestación es un mecanismo de automatización de tareas y control de los recursos, elementos y servicios que soportan una función o solución dentro de un sistema distribuido, en el campo de funciones de red es aún más complejo debido a la arquitectura necesaria para desplegar Funciones de Red en un ambiente virtualizado (Ceballos, 2018).

4. Sistemas social inspirados

Los métodos de optimización desempeñan un papel vital en la solución de problemas de ingeniería. Debido a que los métodos de optimización deterministas han mostrado no ser eficientes computacionalmente en la resolución de problemas complejos no lineales y multimodales que existen en la mayoría de las aplicaciones del mundo real, se han creado una serie de metodologías inspiradas en sistemas biológicos y naturales para resolver problemas complejos de optimización. En gran medida, la mayoría de los algoritmos inspirados en aspectos de la naturaleza se basan en ciertas características de sistemas biológicos y se han agrupado bajo el nombre de algoritmos bio-inspirados. Más cerca de los algoritmos bio-inspirados se encuentran los algoritmos basados en enjambres, que intentan simular comportamientos colectivos exhibido por ciertas especies de animales, que resuelven problemas complejos con reglas individuales relativamente sencillas. Estos algoritmos de enjambre plantean un gran número de agentes homogéneos en un medio ambiente funcionan a través de la cooperación mutua para lograr el objetivo deseado. Otra categoría popular son los Algoritmos Evolutivos (AE) que pueden ser categorizados bajo algoritmos bio-inspirados y buscan inspiración en los procesos de evolución biológica y selección natural. Algunos ejemplos incluyen algoritmos genéticos, colonia de hormigas, cardúmenes, etc. Muchos algoritmos también encuentran motivación de los sistemas físicos y químicos (Biswas A., 2013). Algunos ejemplos incluyen, entre otros, el templado simulado y búsqueda harmónica.

Una rama muy próxima y emergente de EA es la clase de algoritmos socio-inspirados, que toman la motivación de las interacciones sociales y culturales que se ven en el comportamiento humano. Una revisión detallada de esta clase de algoritmos se presenta en la sección posterior. Por lo tanto, los algoritmos inspirados en la naturaleza existentes pueden clasificarse brevemente en las siguientes categorías generales o principales: basadas en inteligencia de enjambre, basadas en la inteligencia biológica, inspiradas en la física /química y otras (como algoritmos sociales, culturales). El intento adicional de los

autores de subclasificar los algoritmos sociales en la Figura 11 agrupados en las ideas en que se inspiran. Todos estos métodos han ganado popularidad debido a su uso de reglas simples para la búsqueda de soluciones óptimas a problemas computacionales complejos y del mundo real. Metaheurísticas es un término utilizado para referirse a este marco algorítmico general que buscan inspiración de diversos fenómenos observados en la naturaleza. Las metaheurísticas combinan reglas y un cierto grado de aleatoriedad para encontrar soluciones óptimas (o casi óptimas) a los problemas y por lo tanto se pueden aplicar a una variedad de problemas de optimización. Las metaheurísticas son estrategias de aproximación que se pueden adaptar para resolver una amplia variedad de problemas de optimización con pequeños cambios en su marco algorítmico general; La popularidad de este tipo de estrategias en la resolución de problemas de optimización del mundo real se puede atribuir a sus siguientes características: diseño simple debido a su cercanía a los conceptos naturales, menos específico de problemas, capacidad de resolución de problemas más rápida y capacidad para escalar problemas de gran dimensión. El desarrollo de estas metaheurísticas ha sido un área de estudio en el campo de la inteligencia computacional desde hace décadas, con la nueva clase de algoritmos de rápida evolución, algoritmos genéticos (GA), optimización de enjambres de partículas (PSO), algoritmos de colonias de hormigas y algoritmos socio-inspirados.

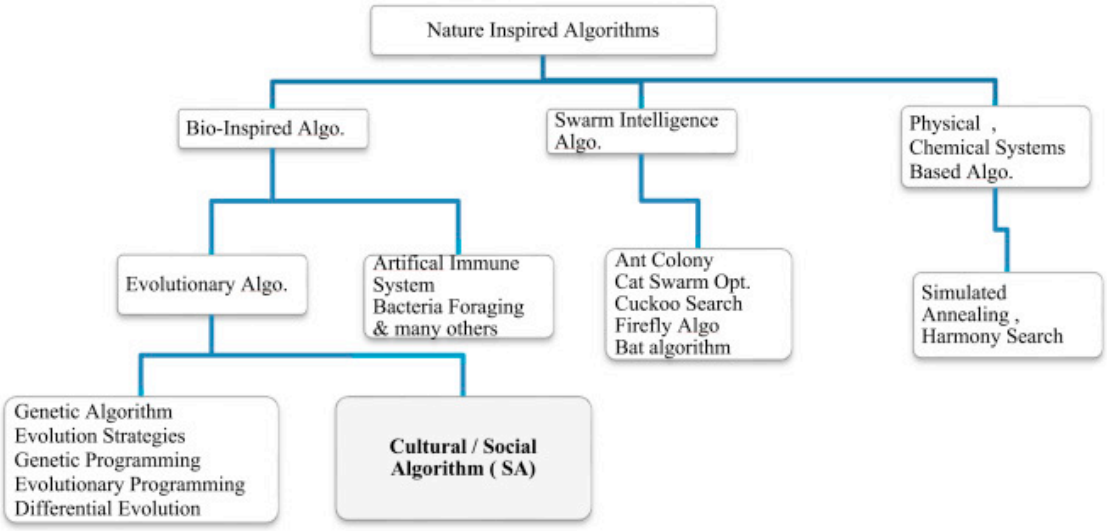


Figura 11. Nature inspired algorithms (Biswas A., 2013)

Ocasionalmente se introducen nuevas metaheurísticas que utilizan una metáfora novedosa como guía para resolver problemas de optimización. Los algoritmos social

inspirados son una de propuestas recientes y futuras de algoritmos de optimización, que utilizan la idea de simular e imitar el aprendizaje social de los seres humanos (o la evolución social). La noción de algoritmos social inspirados fue introducida por primera vez por Reynolds ya en 1994 que establece que los individuos evolucionan mucho más rápido a través de la evolución cultural que sólo a través de la evolución biológica o genética. Los seres humanos se adaptan a los manierismos y comportamientos observando/imitando a otros individuos, lo que les ayuda a mejorar su inteligencia rápidamente y alcanzar objetivos compartidos. La tendencia a cooperar y funcionar juntos como un grupo cohesionado se suma a su inteligencia colectiva. Esta idea ha formado el terreno para que muchos investigadores formalicen algunos algoritmos socio-inspirados recientes como el algoritmo de optimización de la sociedad y la civilización (SCO), el algoritmo competitivo imperialista (ICA), el algoritmo de campeonato (LCA), el algoritmo de optimización emocional social (SEOA), el algoritmo de optimización de la campaña electoral (ECO), el algoritmo de optimización de la sociedad anárquica (ASO), la optimización basada en la enseñanza y el aprendizaje (TLBO), el algoritmo de evolución cultural (CEA), la Optimización del aprendizaje social (SLO), optimización de grupos Sociales (SGO), etc.

No existe un algoritmo/método universal para todos los problemas de optimización y que en promedio funcionará igual de bien. Normalmente para un problema definido con funciones objetivas específicas habrá una clase de algoritmos de optimización que superarán a algunos otros. La tarea principal en cuestión es sólo identificar estos algoritmos de mejor rendimiento que están especializados en la estructura del problema de optimización específico que se está considerando. Por lo tanto, se puede afirmar que el desarrollo de nuevos algoritmos de optimización siempre será esencial y significativo y da a los autores un terreno de que siempre habrá un alcance para desarrollar algoritmos prospectivos más nuevos que podrían ser adecuados para alguna clase específica de tareas de optimización y también puede superar algunos otros algoritmos ya existentes para resolver algunos problemas de optimización específicos.

En la investigación actual, los autores proponen una novedosa metodología de optimización social-inspirada conocida como algoritmo de optimización de la evolución social y el aprendizaje. El metaheurístico está motivado y se inspira en el comportamiento social de los individuos en una familia, parte de una configuración social humana. Una familia representa un grupo social elemental en una sociedad que normalmente consiste

en padres y sus hijos y una sociedad se puede visualizar como una configuración multiagente de diferentes familias que coexisten juntas. Según Goldsmith [58]«una familia en sus diversas formas es la base de todas las sociedades humanas y estructuras sociales». Cada miembro de la familia puede ser considerado como un agente individual en una familia, tomando sus propias decisiones de comportamiento inspiradas en observar y aprender de los demás. La evolución de los padres y los hijos de una familia se basa en aprender unos de otros, así como de otras familias. Este aprendizaje descentralizado puede dar lugar a la mejora general del comportamiento de cada agente y de los objetivos asociados y, en última instancia, a todo el sistema social. El algoritmo de optimización propuesto modela la razón anterior del "comportamiento de aprendizaje descentralizado" de las personas de la familia que evolucionan colectivamente su entorno.

4.1 TLÖN

TLÖN es un sistema de cómputo propuesto por el grupo de investigación en redes de Telecomunicaciones Dinámicas y lenguajes de programación distribuidos de la Universidad Nacional de Colombia. El modelo que propone TLÖN es un esquema social inspirado que plantea una analogía entre los conceptos de Justicia de Rawls, Inmanencia de Spinoza, Estado de Hobbes, Paradigma de Kuhn y existencia y esencia de Sartre, abstraídos al plano de virtualización de una red inalámbrica. Cada capa del sistema TLÖN (Figura 12) representa un nivel en la analogía del Estado; inicialmente una capa física del modelo (Territorio) que será una red AdHoc inalámbrica que planteará las condiciones de prestación de los servicios de las capas superiores. La capa de virtualización se puede entender como la función de control que deben ejercer las instituciones de un estado. El concepto de sociedad vendría implícito en la capa del sistema Multiagente, como un conjunto de individuos que interactúan entre sí y con las otras capas del sistema con el fin de prestar y consumir servicios.

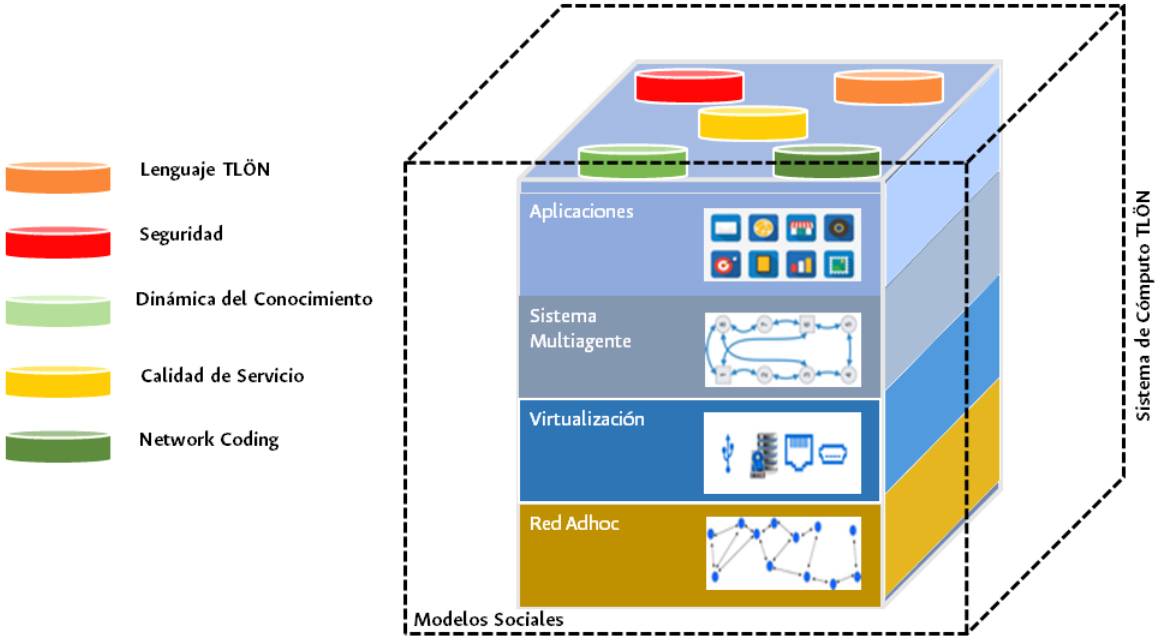


Figura 12. Capas del sistema de cómputo TLÖN (Ceballos, 2018)

Este modelo social inspirado es una abstracción de técnicas biológicas para la administración de sistemas complejos. En los últimos años ha crecido el uso de este tipo de métodos de simulación, por ejemplo en (Ferreira, 2013), (Vanhèe, 2013), (Degens, 2014) (Hofstede, 2015) se ha tomado el concepto de sociedad y cultura para el diseño de agentes que permitan realizar acciones determinadas por las interacciones entre los miembros de una comunidad. El objetivo es crear algoritmos y herramientas computaciones que se basen en el comportamiento social de individuos, en donde se simulen características sociales como altruismo, egoísmo, cooperación, coordinación entre otros. Estos elementos o niveles de abstracción conforman el sistema TLÖN, un sistema de cómputo distribuido que permite interactuar como un solo elemento conformado por diversos nodos o elementos, con la gran particularidad de estar inmersos en una red inalámbrica, móvil, auto configurable y con la necesidad de adaptarse a las condiciones adversas en recursos, conectividad y existencia. Este tipo de redes conocidas como redes ad hoc, no son la única estructura donde este sistema puede desplegarse, son estas condiciones las que lo ponen a prueba y se asemejan a las condiciones actuales de los usuarios, sistemas móviles, dinámicos, con acceso a Internet y a un sin fin de aplicaciones,

es el ingreso al dominio del usuario y de su componente social, lo que impulsa a la aparición de este sistema solución (Ceballos, 2018).

El presente trabajo se encuentra en el marco de desarrollo de la capa de virtualización del sistema TLÖN, que tiene como objetivo prestar servicios a los usuarios con el mismo grado de calidad sin importar los recursos que pueden aportar a la red.

4.2 Modelo de comunicación

Este esquema opera sobre redes inalámbricas en el modo ad hoc de las tarjetas de red, allí se activa el modulo batman-adv del kernel de linux para usar el protocolo B.A.T.M.A.N., este modelo permite la autoconfiguración de IP mediante el servicio avahi, de esta forma se hace una asignación dinámica de direcciones que es diseminada por el servicio de mensajería ALFRED, el orquestador tiene un mensaje etiquetado para que los nodos identifiquen ese rol y realicen los envíos de la información al nodo o nodos con esta etiqueta (Ceballos, 2018).

Este protocolo B.A.T.M.A.N fue desarrollado en Alemania por la comunidad Freifunk (http://freifunk.net/en/), la cual es una iniciativa no comercial para redes inalámbricas de uso libre. B.A.T.M.A.N. es un protocolo de ruteo proactivo para redes de malla inalámbricas tipo ad-hoc y redes ad hoc móviles (MANT). El protocolo mantiene proactivamente la información sobre la existencia de todos los nodos en la malla que son accesibles vía los links de comunicación de salto único o multisalto. La estrategia de el protocolo es determinar para cada destino en la malla un vecino de un solo salto, que se puede utilizar como mejor gateway para comunicar con el nodo de destino. Para realizar el ruteo basado en IP del salto múltiple, la tabla de ruteo de un nodo debe contener un gateway local de link para cada host o ruta de red. No hay necesidad de averiguar o calcular la ruta completa, lo que hace posible una implementación muy rápida y eficiente.

En B.A.T.M.A.N después de un intervalo de tiempo dado cada nodo debe emitir para todos los demás un mensaje originador (OGM), que básicamente establece la existencia del nodo. Para que los nodos que están fuera del alcance del nodo originador sepan de su existencia, los mensajes OGM son re-emitidos por los nodos receptores. El nodo que recibe el mensaje memoriza el vecino directo a través del cual recibió el OGM. Usando las estadísticas de la llegada de mensajes exitosos un nodo puede concluir cuál de sus

vecinos directos es el mejor para reenviar los paquetes hacia su destino. Las rutas y la topología de la red sólo son conocidas por los nodos que están dentro del alcance directo (Ceballos, 2018).

5. Implementación del subsistema de virtualización inalámbrica

A continuación, se explicará la implementación realizada del subsistema de virtualización de recursos en una red ad-hoc inalámbrica en cumplimiento del segundo y tercer objetivos específicos de este trabajo, para ello se abordarán las soluciones planteadas para el descubrimiento de recursos (broadcasting), la asignación (allocation/mapping) de los mismos y finalmente la orquestación (schedulling) de procesos.

El módulo creado se codificó como una librería en Python 2.7 con nombre **MultiPManager**, esta librería cuenta con el submódulo **distProc** que hace referencia a "*Distributed Processing*", los detalles de esta librería y su funcionamiento conceptual serán discutidos a continuación. La topología de despliegue usada para la validación del subsistema se muestra en la Figura 13 y consiste en 6 nodos de procesamiento más un nodo originador interconectados por una red mesh ad-hoc que interactuarán entre si para la ejecución de una tarea distribuida, virtualizando los recursos de procesamiento disponibles.

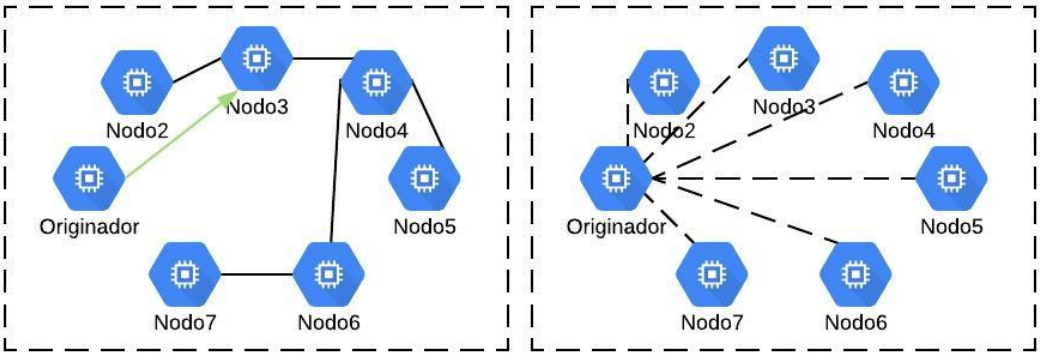


Figura 13. Topología de despliegue del subsistema

5.1 Red Ad-Hoc TLON

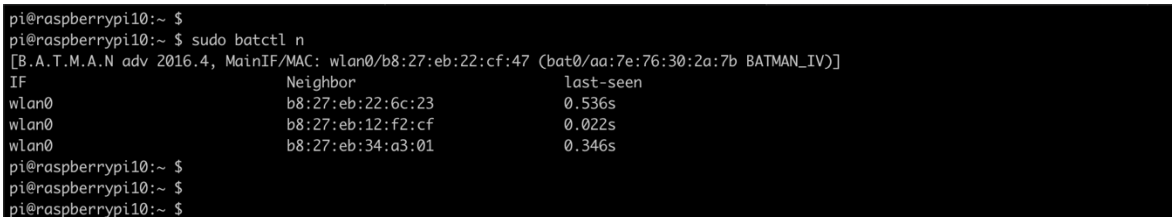
Como se expuso en la sección 4.1 y 4.2, para establecer la red ad-hoc de TLÖN hacemos uso del protocolo de enrutamiento B.A.T.M.A.N acompañado de Avahi para la asignación de IPs. Batman-adv inserta un encabezado adicional de 32 bytes en cada paquete enviado

sobre la red, por esta razón se debe incrementar el tamaño máximo del paquete (mtu) a 1532. A continuación, se adjunta los pasos a seguir para registrar el nodo en la red:

```
#Primero se carga el módulo batmna-adv al kernel con el comando
sudo modprobe batman-adv
#Se configura en modo ad-hoc la interfaz que utilizará batman
sudo ifconfig wlan0 down
sudo iwconfig wlan0 mode ad-hoc
sudo ifconfig wlan0 mtu 1532
sudo iwconfig wlan0 mode ad-hoc essid TLONadhoc ap 02:1B:55:AD:0C:02
channel 1
sudo sleep 1
sudo ip link set wlan0 up
sudo sleep 1
#Registramos la interfaz en batman-adv e iniciamos bat0
sudo batctl if add wlan0
sudo ifconfig bat0 up
#Usamos avahi para la asignación automática de IP
sudo avahi-autoipd -D bat0
```

Cada nodo mantiene una lista de los nodos vecinos que detecta a un salto de distancia. Que un nodo vecino este a un salto o múltiples (mediante otros nodos vecinos) depende de la calidad del enlace. La tabla de nodos vecinos se puede imprimir con el comando:

```
sudo batctl n
```



5.2 Descubrimiento de recursos

En el apartado 2.4.1 se da una clasificación detallada de varios métodos propuestos en la literatura para el problema de descubrimiento de recursos en diferentes contextos. Para el caso de una red ad-hoc inalámbrica con nodos y recursos heterogéneos, se decidió partir de un sistema de distribución de mensajes basado en tópicos como transporte de mensajería que describe las características de cada nodo inspirado en (Venanzi & Kantarci, 2018). Para ello cada nodo que ofrece recursos a la red envía un mensaje serializado con la descripción de los recursos que está aportando, este mensaje es generado al momento de recibir órdenes de trabajo con la información del nodo solicitante. Este esquema de publicación/suscripción proporciona una baja sobrecarga de comunicación y requiere menos potencia computacional. Por lo tanto, es adecuado para la detección de recursos en dispositivos restringidos en el ecosistema heterogéneo de la red-adhoc de TLÖN.

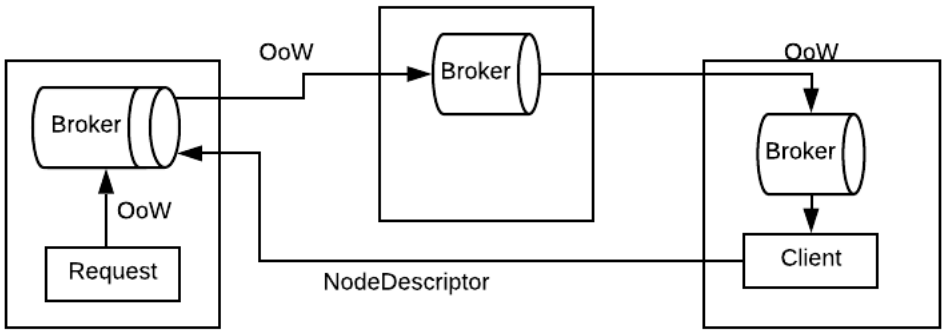


Figura 14. Esquema de descubrimiento de recursos PUSH

En la Figura 15 se imprime la estructura MultiPManager.resourcePool, un arreglo de descriptores de los nodos activos que están aportando recursos en la ejecución de un trabajo distribuido.

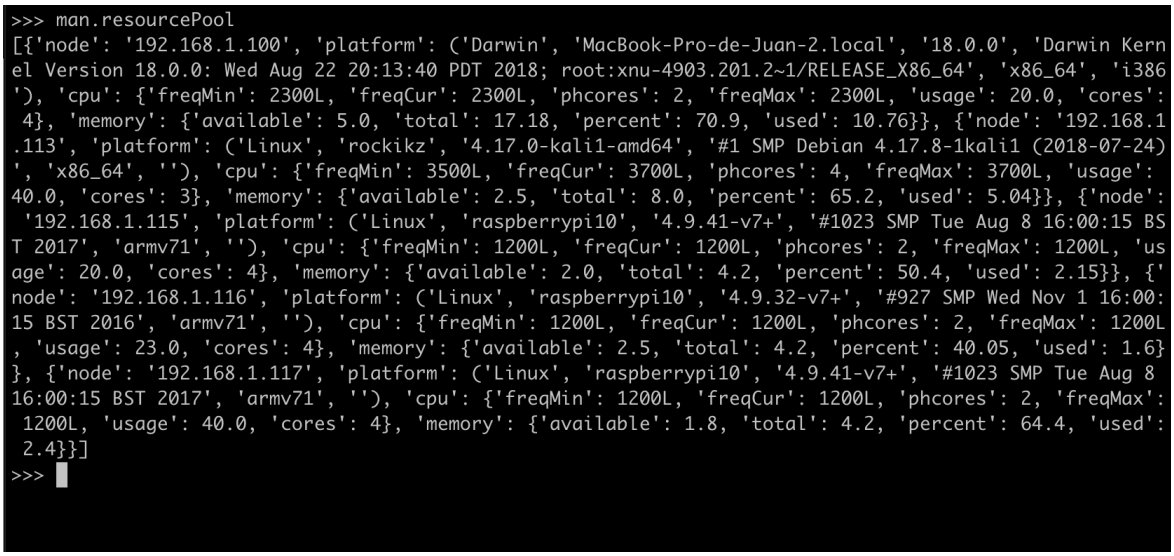


Figura 15. Arreglo de descriptores de recursos compartidos

A continuación se adjunta la salida del arreglo de recursos compartidos para un proceso haciendo uso de los recursos en la red.

man.resourcePool

```
[{'node': '192.168.1.100',
      'platform': ('Darwin', 'MacBook-Pro-de-Juan-2.local', '18.0.0', 'Darwin
Kernel Version 18.0.0: Wed Aug 22 20:13:40 PDT 2018; root:xnu-
4903.201.2~1/RELEASE_X86_64', 'x86_64', 'i386'),
      'cpu': {
            'freqMin': 2300L,
            'freqCur': 2300L,
            'phcores': 2,
            'freqMax': 2300L,
            'usage': 20.0,
            'cores': 4},
      'memory': {
            'available': 5.0,
            'total': 17.18,
            'percent': 70.9,
            'used': 10.76}},
 {'node': '192.168.1.113',
      'platform': ('Linux', 'rockikz', '4.17.0-kali1-amd64', '#1 SMP Debian
4.17.8-1kali1 (2018-07-24)', 'x86_64', ''),
      'cpu': {
            'freqMin': 3500L,
            'freqCur': 3700L,
            'phcores': 4,
            'freqMax': 3700L,
            'usage': 40.0,
            'cores': 3},
```

```
      'memory': {
            'available': 2.5,
            'total': 8.0,
            'percent': 65.2,
            'used': 5.04}},
 {'node': '192.168.1.115',
      'platform': ('Linux', 'raspberrypi10', '4.9.41-v7+', '#1023 SMP Tue Aug 8
16:00:15 BST 2017', 'armv71', ''),
      'cpu': {
            'freqMin': 1200L,
            'freqCur': 1200L,
            'phcores': 2,
            'freqMax': 1200L,
            'usage': 20.0,
            'cores': 4},
      'memory': {
            'available': 2.0,
            'total': 4.2,
            'percent': 50.4,
            'used': 2.15}},
 {'node': '192.168.1.116',
      'platform': ('Linux', 'raspberrypi10', '4.9.32-v7+', '#927 SMP Wed Nov 1
16:00:15 BST 2016', 'armv71', ''),
      'cpu': {
            'freqMin': 1200L,
            'freqCur': 1200L,
            'phcores': 2,
            'freqMax': 1200L,
            'usage': 23.0,
            'cores': 4},
      'memory': {
            'available': 2.5,
            'total': 4.2,
            'percent': 40.05,
            'used': 1.6}},
 {'node': '192.168.1.117',
      'platform': ('Linux', 'raspberrypi10', '4.9.41-v7+', '#1023 SMP Tue Aug 8
16:00:15 BST 2017', 'armv71', ''),
      'cpu': {
            'freqMin': 1200L,
            'freqCur': 1200L,
            'phcores': 2,
            'freqMax': 1200L,
            'usage': 40.0,
            'cores': 4},
      'memory': {
            'available': 1.8,
            'total': 4.2,
            'percent': 64.4,
            'used': 2.4}}
]
```

5.3 Asignación de recursos

En 2.4.2 se da un ejemplo de algunos métodos consultados en la literatura. Para el caso de TLÖN se optó por el modelo basado en un cluster de colas FCFS y tópicos de subscripción persistente inspirado en (Hummel & Jelleschitz, 2007); este módulo permite la distribución de la ejecución de una tarea a través de una red heterogénea utilizando bloques de datos como entrada. El transporte de la data de entrada (DATAIN[]) y de los bloques de resultados (DATAOUT[]), hace uso de las Queues para garantizar un único receptor en los nodos registrados en la red.

Cada instancia que ofrezca recursos al originador de la solicitud debe estar subscrita a los tópicos “/topic/TLONResources” y “/topic/TLONOrders” del cluster de brokers. El primero permite compartir objetos JSON de tipo rutinas de código entre los nodos que pertenecen a la red y que están registrados en el broker, las rutinas se almacenan localmente en el diccionario tlon_resources mientras se reciben las ordenes de trabajo, el segundo tópico distribuye ordenes de trabajo con la estructura OoW:

```
OoW =  {
        "resourceName": <NAME>,
        "ip": <IP>,
        "portnum": <MANAGER_PORTNUM>,
        "authkey": <AUTHKEY>,
        "descriptors": {["key":"value"]}
        }
```

Dependiendo de la caracterización de los recursos de procesamiento de la maquina y su ocupación, cada nodo subscribirá un número n de consumidores en el tópico de Ordenes, de esta manera se paraleliza el consumo de los bloques de data entrante.

El consumo de los bloques de DATAIN[] se hace en modo sesión transaccional para garantizar que en caso de que una excepción no controlada escale fuera del sistema TLÖN mientras se hace la resolución de las ordenes de trabajo, el bloque permanecerá en la queue y luego de una ROLLBACK, este será atendido por otro consumidor con recursos disponibles. Bajo este mismo concepto, los resultados en DATAOUT[i] no son entregados hasta que la totalidad del bloque DATAIN[i] haya sido atendido exitosamente. En la Figura 16 se muestra la secuencia de mensajes compartidos entre el nodo origen y los nodos que contribuyen recursos de procesamiento (Hosts).

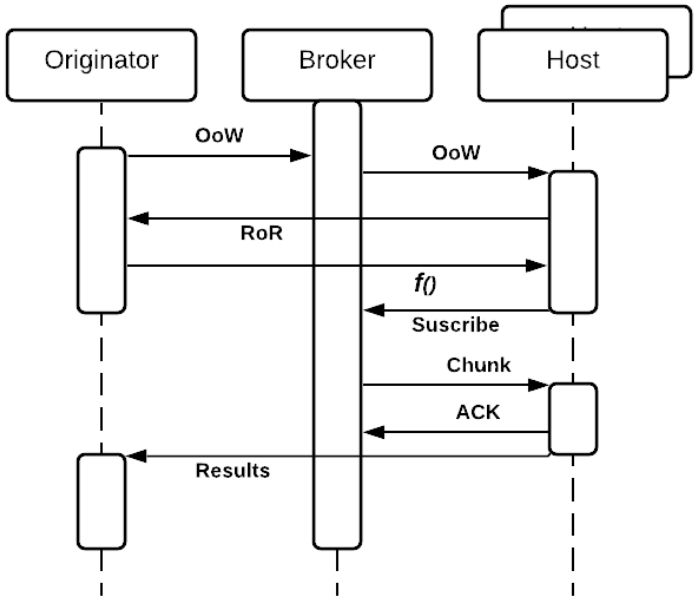


Figura 16. Diagrama de secuencia de OoW

La trasmisión de estos datagramas se diseñó sobre el protocolo STOMP. STOMP es un diseño simple interoperable para pasar mensajes asíncronos entre clientes a través de servidores de mediación. La comunicación entre el cliente y el servidor es a través de una "envoltura" que consiste en una serie de bloques de datos separados por saltos de línea. La primera línea contiene el comando, seguido de encabezados en la forma <clave>:<valor> (uno por línea), seguido por una línea en blanco y luego el contenido del cuerpo, que termina en un carácter nulo. La comunicación entre el servidor y el cliente es a través de una envoltura de MESSAGE, RECEIPT o ERROR con un formato similar de encabezados y contenido del cuerpo.

En la Figura 17 se muestra el diagrama de secuencia de una orden de trabajo en el sistema y el proceso de su recepción en un nodo. El originador llama el modulo MultiPManager.distProc.tlon_parallelize compartiendo la dirección de origen, el diccionario de recursos de software para ejecutar el trabajo distribuido y el arreglo de datos de entrada que se deben procesar.

```
tlon_parallelize ([Dirección] , [Recursos Software], [Arreglo de entrada])
```

El módulo **distProc** inicia la actualización en memoria de los recursos de software compartidos en los nodos suscritos a la red mediante el tópico de /topic/TLONResources

serializando y trasmitiendo el contenido del diccionario de recursos. La serialización se hace usando la librería pickle; el módulo pickle implementa protocolos binarios para serializar y des serializar una estructura de objetos de Python. "Pickling" es el proceso mediante el cual una jerarquía de objetos de Python se convierte en una secuencia de bytes, y "Unpickling" es la operación inversa, por la que una secuencia de bytes (de un archivo binario o un objeto similar a bytes) se convierte de nuevo en una jerarquía de objetos. Los consumidores que reciben el mensaje de actualización, reconstruyen el diccionario de objetos des serializando su contenido y comparándolo contra su diccionario actual, si hay registros de la rutina se actualiza el apuntador y si no los hay se crea un nuevo registro en el diccionario.

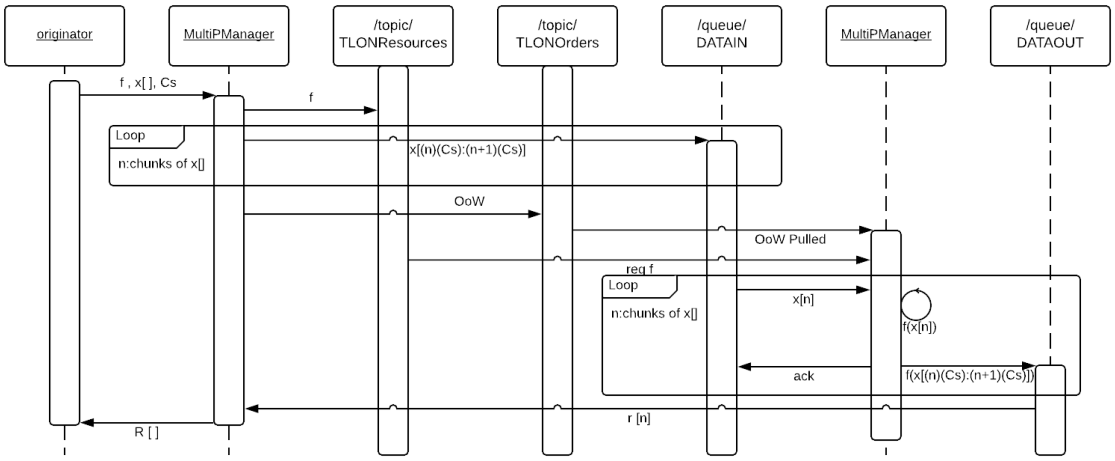


Figura 17. Secuencia de un proceso distribuido

Luego que los recursos son compartidos, el módulo distProc crea queues FIFO /queue/DATA_IN y /queue/DATA_OUT por donde se transportarán los datos de entrada y salida de los nodos atendiendo la orden de trabajo. El arreglo de entrada de divide en chunks de tamaño fijo y se procede a encolar estos chunks en la queue de trabajos pendientes (DATA_IN). Una vez todo el arreglo de chunks fue inyectado, se procede a crear el objeto OoW, una abstracción del trabajo distribuido con los componentes necesarios para la conexión al origen y un arreglo de los objetos necesarios para procesar un chunk de datos del trabajo distribuido. Esta OoW se distribuye a los nodos subscritos mediante el tópico /topic/TLONOrders.

Los nodos conectados a la red pueden aportar recursos de procesamiento por medio de la librera **MultiPManager.distProc** usando el método **runclient**. Como datos de entrada para este método se toma la ip local del broker y el número de threads con el cual se atenderá un consumidor; con estos datos el método runclient crea subscriptores a las queues de ordenes y actualización de recursos. Una vez los nodos cliente halan (*Pull*) la estructura OoW, tendrán los datos de conexión y descriptor del nodo origen de la solicitud y registran un consumidor en la queue de trabajos. Con esta subscripción inicia el desencolamiento de chunks del arreglo de datos de entrada para ejecutar el trabajo distribuido; los mensajes son multiplexados entre los hilos disponibles. Cuando la cola de trabajos queda vacía el proceso finaliza desconectando el cliente de la cola de trabajos.

```
class __ordersTopicListener__(stomp.ConnectionListener):
    def on_error(self, headers, message):
        print('Received an error {}'.format(message))

    def on_message(self, headers, message):
        global tlon_resources
        if sys.version_info[0]<3:
            tmp = pickle.loads(message)
        else:
            tmp = pickle.loads(message.encode())
        if tmp['resourceName'] in tlon_resources:
            manager = managerImp.make_client_manager(tmp['ip'], tmp['portnum'],
tmp['authkey'])
            job_q = manager.get_job_q()
            result_q = manager.get_result_q()
            multiProc.tlon_multiprocessing(job_q, result_q, tlon_resources[
tmp['resourceName']])
```

El nodo origen una vez encola la orden de trabajo crea un consumidor en la queue de respuestas (DATA_OUT) para desencolar los paquetes de respuestas enviadas por los nodos colaboradores. Una vez recibe la totalidad de respuestas esperadas devuelve el arreglo completo como salida del método **tlon_parallelize.**

5.4 Orquestación de procesos

En un ambiente distribuido heterogéneo como son las redes Ad Hoc cooperativas es importante asignar cada tarea al procesador disponible que sea mas apropiado para ejecutarla. Este problema ha sido categorizado en ambientes estáticos y dinámicos. Adicionalmente, se han propuesto modelos basados en programación en enteros, balanceo de cargas, método de dividir y conquistar y finalmente Grid Computing. El objetivo

general de un eficiente sistema de asignación y orquestamiento de tareas (N) en un ambiente de varios procesadores (M) distribuidos, es disminuir el costo de procesamiento sobre una red donde N>M.

La base del sistema de orquestamiento de tareas es el servicio orquestador que se encarga de recibir las tareas de diferentes nodos y selecciona recursos (ejecutores) probables en la red de acuerdo al INR local, para luego hacer el mapeo tarea-recurso en función a su idoneidad y desempeño. En (Yu & Ying, 2020) se propone un modelo descentralizado para sistemas distribuidos heterogéneos basado en colecciones de clusters. En (Carroll & Grosu, 2012) se presenta un modelo basado en árboles binarios que permite representar una arquitectura distribuida como un esquema jerárquico teniendo en cuenta un ambiente de recursos heterogéneos. (C.S.Xavier & L.Santos, 2020) propone un protocolo de asignación de recursos para garantizar calidad del servicio usando un árbol de probabilidad modelado como un árbol AND/OR y la ejecución de un proceso se lleva acabo mediante la búsqueda en el árbol de soluciones. Xu en (Xu & Li, 2015) introduce un algoritmo orientado a recursos usando técnicas bioinspiradas (ant colony) como estrategia de asignación de claves.

Para el caso puntual del proyecto TLON se escogió un algoritmo de orquestamiento que tenga en cuenta criterios como escalabilidad, heterogeneidad de recursos y los efectos por largos retrasos en entrega, por lo cual se desarrolló un sistema de colas bloqueante cuyo objetivo es garantizar que un mensaje no sea descartado del stack hasta que el consumidor haya terminado su procesamiento y envíe un ACK, el mensaje es bloqueado hasta entonces. Este bloqueo es llamado sesión negociada y actúa como un mecanismo de control de la confirmación de recepción de mensajes. El método de orquestación mediante colas de trabajo permite mantener aislados los procesos que corren en el SO y desacoplar la lógica interna de los trabajos distribuidos. Cada procesador que aporta un nodo entra competir por mensajes en la queue, por lo que descentraliza el control de qué procesador y qué nodo ejecuta una petición, adicionalmente el bloqueo por sesiones permite garantizar la no pérdida de mensajes dada la intermitencia de los nodos en una red Ad-Hoc heterogénea.

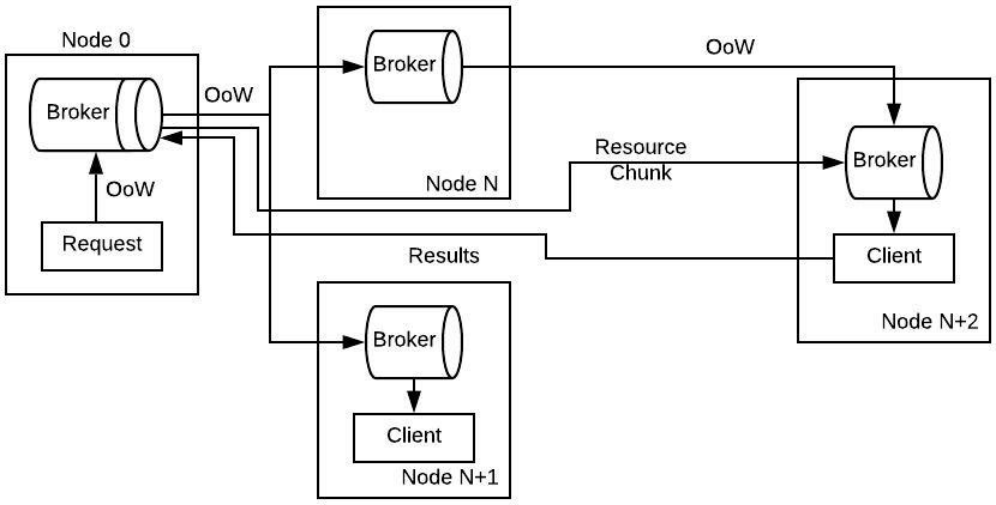


Figura 18. Transporte de ordenes de trabajo (OoW)

Adicionalmente, el mecanismo de tópicos permite la retrasmisión y distribución de órdenes de trabajo a través de toda la red. Esto se logra mediante un mecanismo de message-fordwarding en el que el mensaje es retrasmitido a todos los broker vecinos de cada nodo, identificado mediante un MessageID que impide su reproceso por un mismo cliente.

6. Validación de la implementación

Como parte del desarrollo del subsistema de *mapping* y *scheduling* de tareas distribuidas en una red inalámbrica Ad-Hoc, se planteó como cuarto objetivo específico de este trabajo la validación en un ambiente real con 7 nodos para observar el rendimiento de un algoritmo de prueba. No se plantearon modelos comparables ya que el objetivo de la tesis no es establecer u optimizar modelos existentes sino acoplar este modelo de procesamiento distribuido dentro del entorno de una red ad-hoc.

6.1 Algoritmo de prueba

Para la validación de la implementación se creó una rutina que por fuerza bruta con diccionario permitiese encontrar el valor X que genera la entrada Y en la función Hash(X)=Y.

Los tiempos serán tomados en base al número representativo de muestras inyectadas al diccionario origen, con ello se evaluarán la variación de la tasa de rendimiento en que se procesan muestras en un entorno real con nodos heterogéneos, conectados en una Red-AdHoc. El diccionario origen se compone de un conjunto palabras aleatorias que para el propósito de la evaluación de rendimiento puede o no tener el valor correcto de escape de la rutina.

6.2 Resultados

A continuación, se muestran las pruebas ejecutadas con un conjunto de nodos heterogéneos y se incluye la lista del stack de hardware utilizado (Tabla 1).

Tabla 1. Hardware usado en la validación

Nodo	**Procesador**	**Cores**	**Frecuencia**
1	Intel Core I5	4	2.3GHz

Nodo	Procesador	Cores	Frecuencia
2	IntelCore I5-9600 (Docker)	6(3*)	3,7GHz
3	ARMv8 Raspberry Pi 3	4	1.2GHz
4	ARMv8 Raspberry Pi 3	4	1.2GHz
5	ARMv8 Raspberry Pi 3	4	1.2GHz
6	BCM2835 Raspberry Pi Zero	1	1GHz
7	Exynos7904	2	1.8GHz

6.2.1 Relación Cores vs Rendimiento

Para la validación del modelo de procesamiento distribuido se ejecutó el algoritmo propuesto en un solo nodo (Nodo 1) restrinjiendo el número de cores habilitados para los trabajos programados. Para establecer unos resultados base de la prueba con 1k de muestras, se ejecuto el originador de la solicitud y subscriptor en el mismo nodo con el fin de descartar problemas de red. En la Figura 19 se muestra la cantidad de muestras procesadas (Eje Y) por el tiempo transcurrido (Eje X) por un nodo con 2 cores.

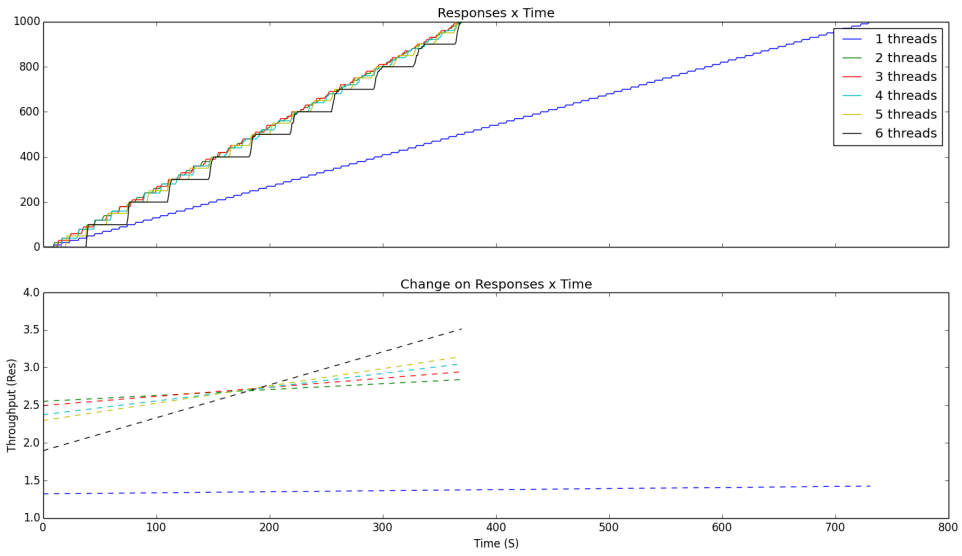


Figura 19. Cores vs Rendimiento (2 Cores)

Se puede ver que una vez el número de hilos alcanza el número de cores (nC) disponibles el rendimiento no mejora significativamente. En la Figura 19 (Abajo) se establece la tendencia del rendimiento (muestras procesadas por segundo) según el método de regresión de mínimos cuadrados con grado 1. Se identifica que el rendimiento de un solo hilo activo, sirviendo las solicitudes del originador, permanece estable alrededor de 1.36 mensajes por segundo. Cuando se habilita el segundo hilo el rendimiento de la aplicación se duplica alcanzando un rendimiento de 2.57 mensajes por segundo. Una vez se fueron adicionando mas hilos de procesamiento se observó que el rendimiento no mejora significativamente. Esto puede ser debido a que en el nodo que esta prestando recursos de procesamiento, los hilos activos tienen que competir individualmente por los cores disponibles.

Ese cambio de contexto que hace el apuntador sirviendo los nC+ hilos consume tiempo de máquina y hace que se presente una baja (despreciable para el caso de el algoritmo usado en las pruebas) de rendimiento individual del nodo.

Adicionalmente, a partir los nC+ hilos se ve un comportamiento escalonado incremental del rendimiento en respuestas por segundo, esto obedece a que el sistema operativo está distribuyendo el tiempo de procesador entre los procesos activos equitativamente, esto en un SO preemptive, lo que causa que los procesos inicien de manera sincronizada atendiendo paquetes de entrada y terminen en tiempos muy similares, lo que genera un delta en el rendimiento pues los hilos envían las respuestas al mismo tiempo. Luego de un tiempo los procesos se van desincronizando por acción de otros procesos del sistema operativo que pueden estar corriendo paralelamente, por esta razón se empiezan a desincronizar los hilos y vemos que se distribuyen los picos en picos de menor tamaño como se observa en la Figura 20.

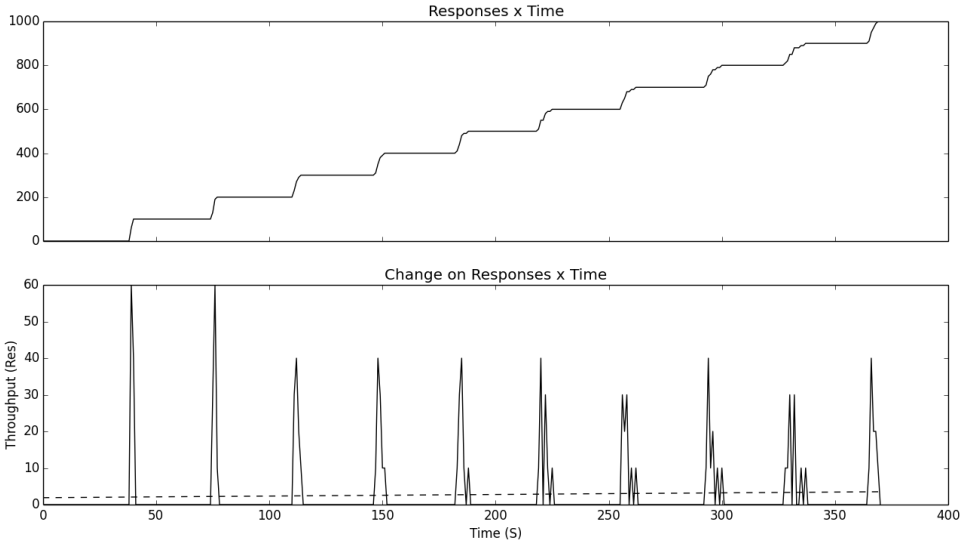


Figura 20. Rendimiento de 1 nodo con 6 hilos

Seguido a ello se realizó la prueba aumentando el número de cores y se observa que hay una relación directa entre el número de cores y el rendimiento (Figura 21). La tasa en que fluctúa el rendimiento por nodo varía según la velocidad de los cores, los procesos ajenos que se estén corriendo en el nodo y otras variables no previstas, por esta razón el alcance de estas pruebas se limitó a establecer una relación entre número de cores y rendimiento.

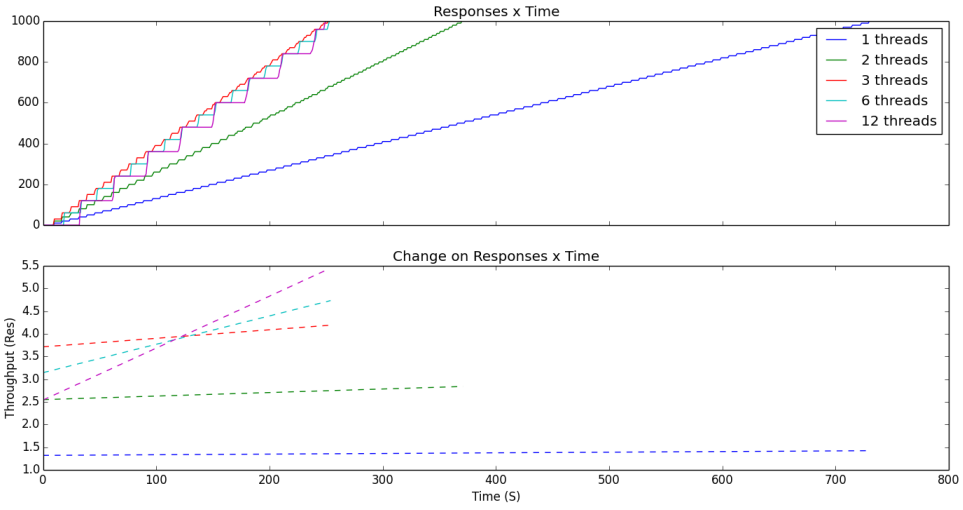


Figura 21. Cores vs Rendimiento (3 Cores)

6.2.2 Relación número de nodos vs Rendimiento

En segunda instancia se realizó la validación conectando 7 nodos en red AdHoc para realizar el procesamiento 1k de muestras de manera distribuida. En todos los nodos se limitó el número de hilos a 1 para observar el impacto individual de cada uno. En la Figura 22 se muestra una de las pruebas, donde se inicia con un tiempo base de 720 segundos para procesar la totalidad de las muestras con un nodo de manera local (azul) y se agregan nodos para verificar la mejora de rendimiento hasta llegar a 7 nodos (negro), con un tiempo de procesamiento de 78 segundos.

Esta prueba se ejecutó 20 veces para obtener un tiempo medio por cantidad de nodos, manteniendo siempre el orden de los nodos. En la Figura 23 se muestra el promedio del tiempo requerido para procesar las 1k muestras por cada arreglo en condiciones similares del experimento y el error aproximado.

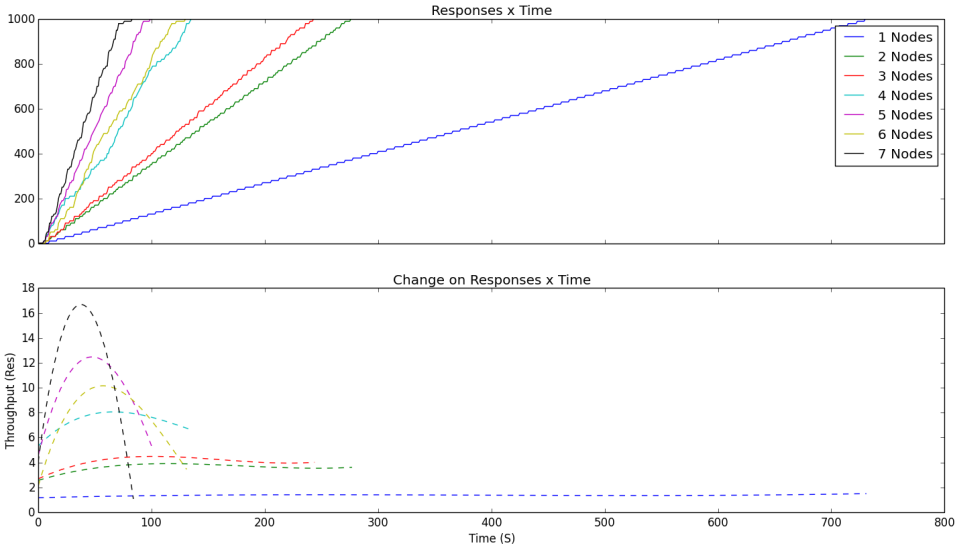


Figura 22. Procesamiento con 1-7 nodos

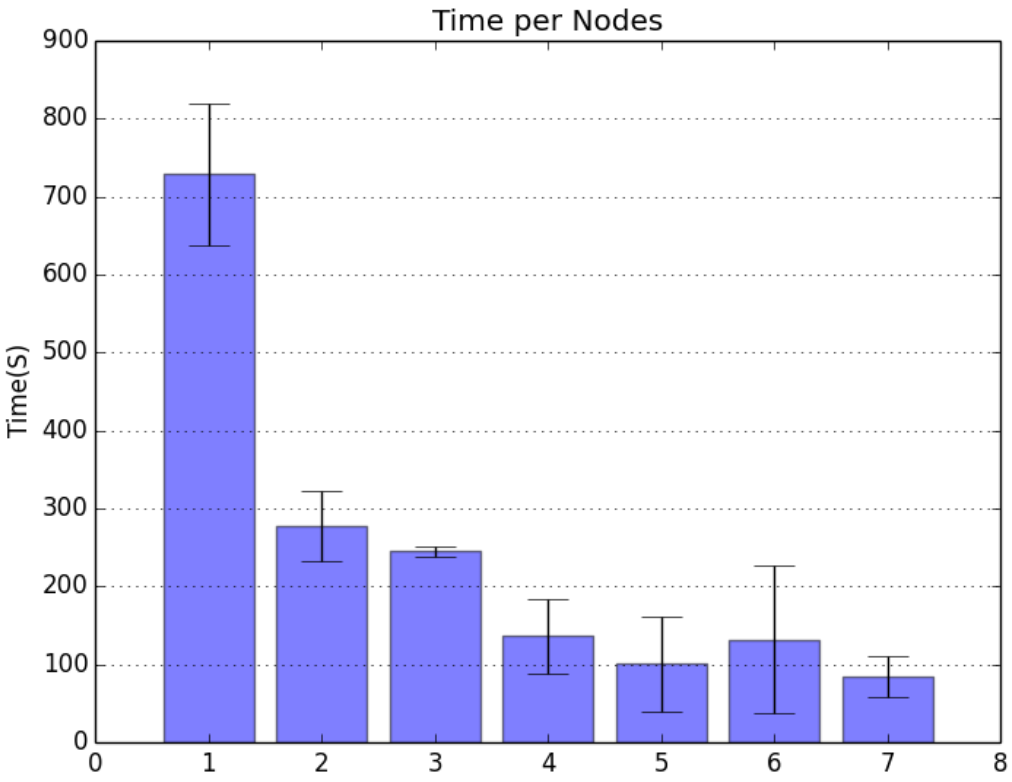


Figura 23. Tiempo de procesamiento de 1k de muestras

6.2.3 Nodo intermitente

Con el fin de verificar el comportamiento del sistema ante la intermitencia de nodos, se ejecutó una prueba procesando 10k muestras con 6 nodos iniciales. Para esta prueba, una vez iniciada la ejecución de las ordenes de trabajo, se procedió a desconectar/conectar uno nodos a fin de ver la respuesta en el rendimiento y garantizar la no pérdida de mensajes.

La primer prueba (Test 1) consistió en conectar un nodo en tiempo de ejecución en t+70 segundos. Se puede observar cómo se registra un incremento del rendimiento luego de unos segundos (mientras el nodo se registra en los tópicos y accede a los recursos de software para ejecutar la rutina de prueba). El nuevo nodo se deja 1 min y se desconecta aproximadamente en t+130, lo que causa que el rendimiento regrese a su estado anterior. No se registró pérdida ni duplicación de mensajes.

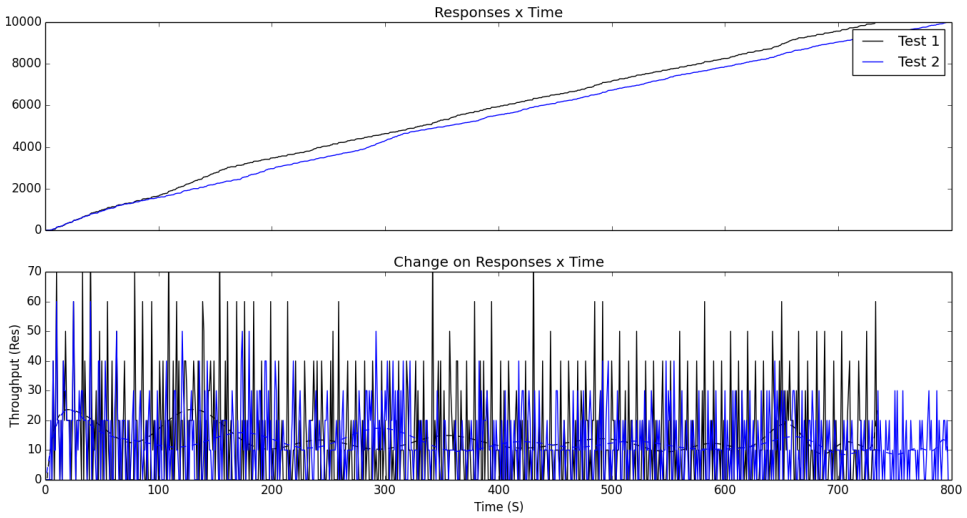


Figura 24. Pruebas de intermitencia

Para la segunda prueba (Test 2) se desconectó uno de los nodos en t+20, lo que causa la caída en el rendimiento. Luego en t+110 segundos ingresa un nuevo nodo a la red para reemplazarlo, una vez el nodo se conecta debe registrarse y obtener la orden de trabajo y recursos necesarios, por lo que toma un tiempo antes que empiece a tomar muestras para procesar. En la Figura 25 se muestra la tendencia estimada del rendimiento (mensajes por segundo) respecto al tiempo.

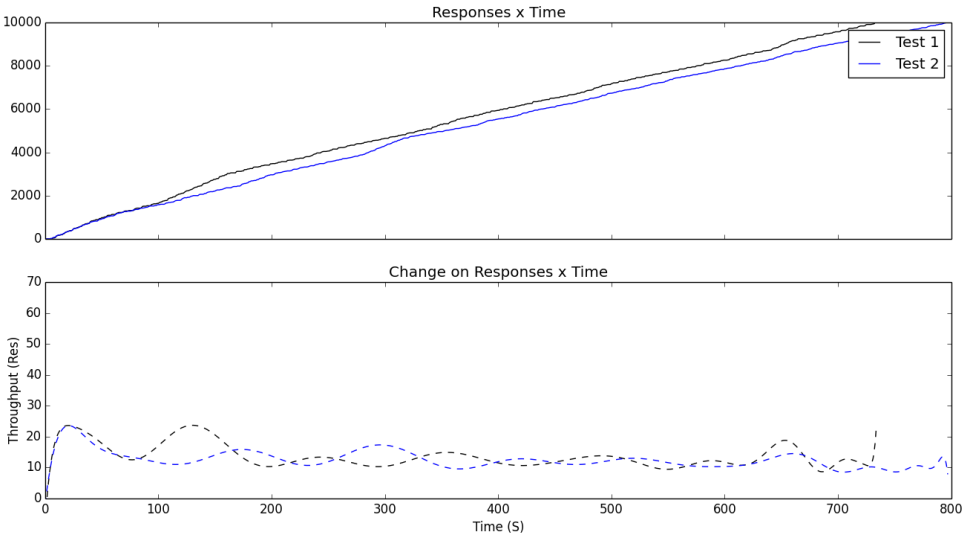


Figura 25. Prueba de intermitencia (Tendencia)

6.2.4 Memoria usada

En relación a la memoria usada por los procesos, a continuación se muestran los diagramas del tamaño de la memoria usado por la aplicación en el nodo origen (Node 0) versus los nodos que aportan recursos de procesamiento (Node 1-5) (Figura 26).

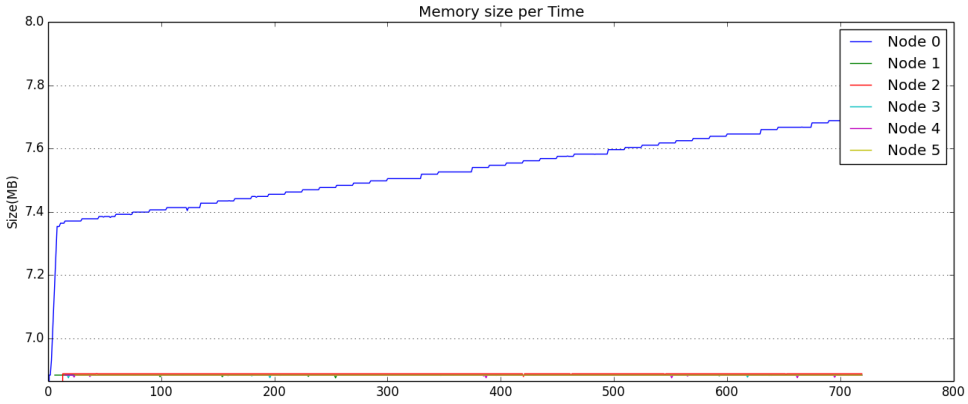


Figura 26. Memoria usada en origen

Se observa un comportamiento estable a través de toda la prueba con un incremento equivalente al peso de las respuestas recibidas en memoria, con saltos escalonados debido la naturaleza de bloque de los datos de entrada y salida.

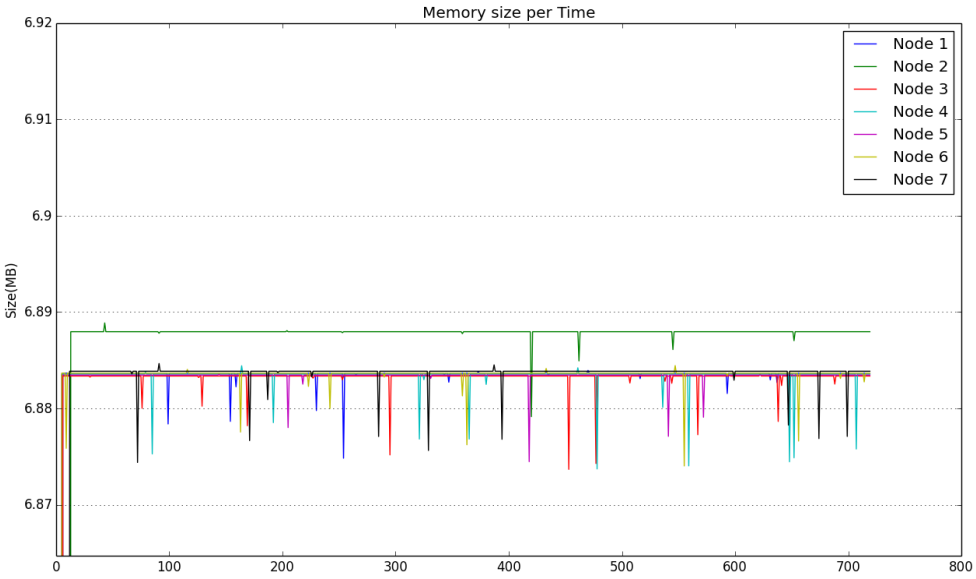


Figura 27. Memoria usada por nodo

En la Figura 27 se observa el comportamiento de la memoria de los nodos aportantes, evidenciando algunos instantes de tiempo en que el consumo de memoria baja producto de la limpieza de las entidades usadas en la ejecución de un chunk de datos una vez el trabajo es ejecutado.

6.2.5 Prueba de carga

Una última prueba consistió en aumentar el tamaño de la muestra utilizada para observar el comportamiento del modelo con una carga masiva. Se ejecutaron pruebas con 1k, 10k, 100k y 1M de datos de entrada. Los resultados expuestos en la Figura 28 muestran el tiempo total de cada una de las pruebas con diferentes tamaños de muestras procesadas, en la Figura 29 se presenta la misma información pero el eje Y en escala logarítmica para mejor visualización. Se puede concluir que existe una relación lineal entre el tiempo de procesamiento y la cantidad de muestras y no se percibe una disminución significativa en el rendimiento.

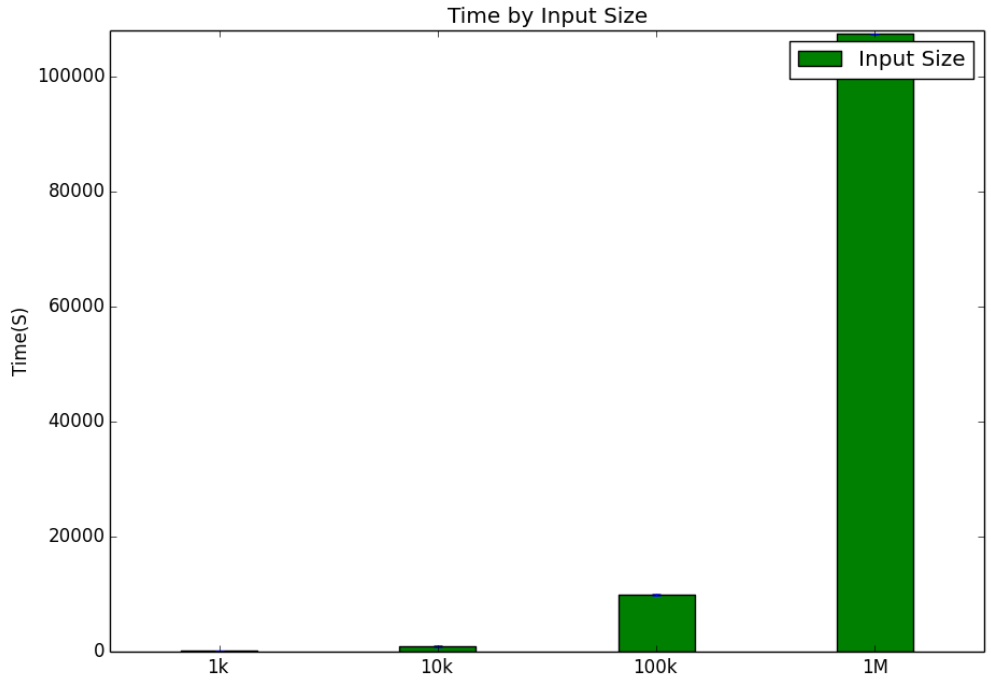


Figura 28. Tiempo por tamaño de muestra

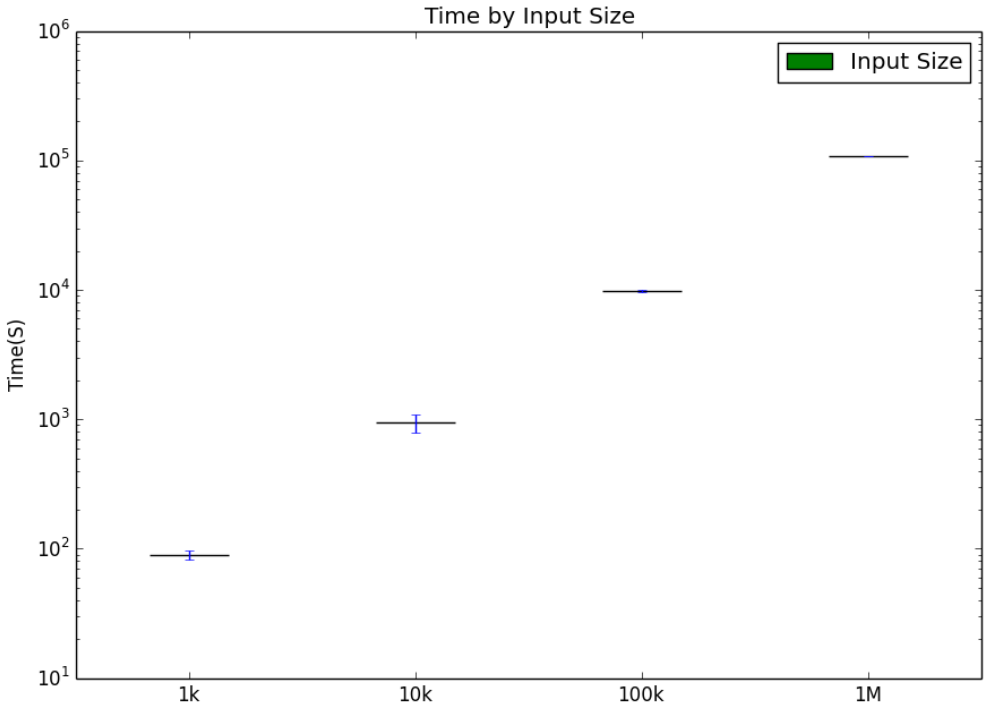


Figura 29. Tiempo por tamaño de muestra (escala Log)

6.3 TLÖN vs Wi-Fi

Como experimento final, se realizó una comparación del esquema de procesamiento distribuido propuesto en este trabajo en dos entornos diferentes de topología de red, una red Ad-Hoc (TLÖN) y una red Wi-Fi local con un enrutador central. Se ejecutó la misma prueba procesando 10k de muestras con los mismos 7 nodos conectados a la red, 10 veces en TLÖN y 10 veces en Wi-Fi (Figura 30).

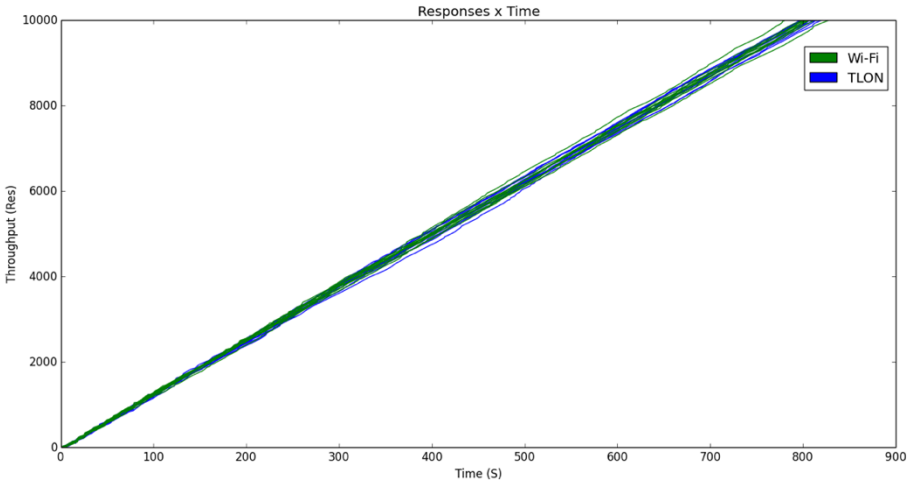


Figura 30. TLÖN vs Wi-Fi

Se evidencia que no hay diferencia visible en los tiempos de procesamiento entre una topología y la otra. Los tiempos se mantuvieron estables alrededor de los 800 segundos para las 10k muestras. Con el fin de establecer una mejor relación entre estos dos modos de red se ejecutó nuevamente la prueba escalando nodos de N=3 a N=7. Cada prueba tuvo 3 repeticiones y los resultados son expuestos en la Figura 31. Nuevamente no se observa ninguna diferencia significativa entre los dos modelos en el rango de número de nodos que se determinó para esta validación.

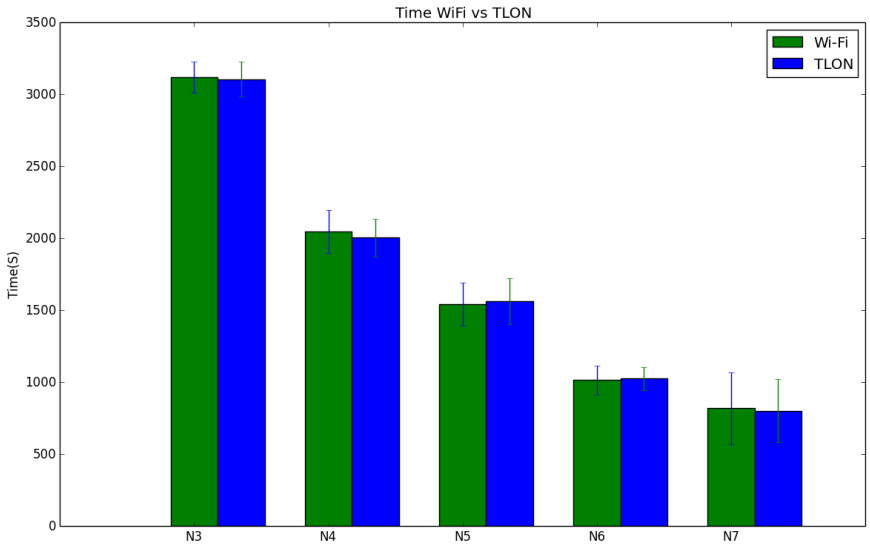


Figura 31. TLÖN vs Wi-Fi 3-7 Nodos

7. Conclusiones y recomendaciones

7.1 Conclusiones

El desarrollo del sistema TLÖN es un proceso largo y multidiciplinario que contiene diferentes frentes de trabajo y cuya base es la distribución de recursos en una topólogia de red descentralizada. El objetivo principal de este trabajo está enmarcado en la capa de virtualización de TLÖN que como fin tiene proporcionar modelo de cómputo distribuido con énfasis en modelos social inspirado. En este trabajo se logró aportar un modelo viable de distribución de tareas en los nodos aportantes como un esquema de disposición de recursos de procesamiento.

Las conclusiones mas relevantes de este trabajo son las siguientes:

1. El panorama de descubrimiento de recursos en redes ad-hoc aún está en desarrollo. Modelos mas complejos han sido propuestos en otras topologías que podrían servir de inspiración a TLÖN para resolver problemas de clusterización por vecindarios y mecanismos de confianza entre nodos ante la ausencia de autoridades centrales con estrategias de refuerzo positivo.
2. Con respecto a los esquemas de asignación de recursos se hizo evidente en la literatura que un aspecto fundamental de este es la caracterización de los nodos. Es necesario definir un modelo apto social inspirado para abstraer los nodos participantes de modo que se logre definir una política justa de asignación.
3. El modelo propuesto presenta la restricción inicial de que únicamente logra resolver algoritmos de programación funcional, cuyo proceso sea reductible a funciones desacopladas e independientes al igual que su matriz inicial de entrada de datos. Una familia de problemas aplicables en este esquema de procesamiento distribuido son las búsquedas exhaustivas (fuerza bruta) en entornos de recursos de procesamiento no centralizados.

4. Este modelo de procesamiento está desacoplado de la topología de red en el que se despliegue, aunque fue diseñado en base a principios de redes ad-hoc puede ser replicado en redes centralizadas o cableadas. Se demostró que en el rango acotado de número de nodos que se seleccionó para la validación de la implementación, no hay una diferencia significativa en el rendimiento con respecto al entorno de red. Con mayor cantidad de nodos esto podría variar.
5. Se observó que bajo este esquema de procesamiento distribuido existe una relación directa entre el número de cores y el rendimiento que puede aportar un nodo. Esa relación se mantiene lineal (excluyendo procesos ajenos paralelos) con el aumento de los hilos hasta llegar al número de cores disponibles. Posterior a esto, los hilos empiezan a competir por tiempo de procesamiento y se observa una desmejora en el rendimiento que obedece al aumento de cambios de contexto que debe ejecutar el sistema operativo.
6. Con las pruebas de nodo intermitente se comprobó que este modelo propuesto es estable en entornos de red de acceso espontáneo como las redes ad-hoc mesh. La adición y substracción de nodos en tiempo real no afecta la integridad de los datos ni la ejecución de una tarea distribuida.
7. Este modelo permite escalabilidad y elasticidad de recursos disponibles en una red ad-hoc con la interacción de arquitecturas y conceptos como IoT, fog computing y micro data centers.

7.2 Recomendaciones y trabajo futuro

A lo largo del desarrollo de este trabajo se han resuelto interrogantes, corroborado supuestos y deshecho modelos e ideas que no se ajustan o resultaron no ser necesarias en la topología propuesta en TLÖN. Así mismo se identificaron algunas aproximaciones de otros autores a problemas comunes que pueden servir como inspiración en trabajos futuros en la evolución continua de TLÖN. A continuación se exponen las mas relevantes:

1. Se puede escalar el experimento realizado con un número incremental para observar el comportamiento del modelo propuesto en entornos masivos de distribución de recursos.

2. Se debe trabajar en un esquema de caracterización de nodos que asocie conceptos social inspirados, que permita la toma de decisiones en los procesos de asignación de recursos y mapeo de tareas.
3. En el desarrollo de este trabajo se identificaron modelos de descubrimiento de recursos que pueden ser evaluados para en un trabajo futuro ser implementados en la red TLÖN para resolver problemas de agrupamiento de los nodos de red (*clustering*) y disminuir la cantidad de *overhead* inyectado en el canal de comunicación.
4. Se propone evaluar diferentes algoritmos de prueba bajo este modelo, como tratamiento de imágenes, video y texto en streaming.
5. Los nodos usados en los experimentos ejecutados en este trabajo tenían como base Linux cuyo orquestador de procesos funciona por defecto en modo preemptive. Se propone evaluar el modelo con diferentes esquemas de orquestación como Round Robin, FIFO, EDF (Earliest Deadline) y SJF (Shortest Job).

8. Bibliografía

Miorandi, D., Sicari, S., & Pellegrini, F. D. (2012). Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*.

Kiess, W., & Mauve, M. (2007). A survey on real-world implementations of mobile ad-hoc networks. *Ad Hoc Networks* .

Patel, K. N., & Jhaveri, R. h. (2015). A Survey on Emulation Testbeds for Mobile Ad-hoc Networks . *Procedia Computer Science*.

Boukerche, A., & Turgut, B. (2011). Routing protocols in ad hoc networks: A survey. *Computer Networks*.

Marinescu, D., Marinescu, G., Ji, Y., Boloni, L., & Siegel, H. (2013). Ad hoc grids: communication and computing in a power. *Workshop on Energy-Efficient Wireless*.

Gaynor, M., Welsh, M., Moulton, S., Rowan, A., LaCombe, E., & Wynne, J. (2004). Integrating wireless sensor networks with the grid. *IEEE Internet Computing*.

Fitzek, F. H., & Katz, M. D. (2014). *Mobile Clouds: Exploiting Distributed Resources in Wireless, Mobile and Social Networks.* Inglaterra: John Wiley & Sons, Ltd.

Basney, J., & Livny, M. (2012). Deploying a High Throughput Computing Cluster. *High Performance Cluster Computing: Architectures and Systems*.

Yue-Jiao Gong, W.-N. C. (2015). *Distributed evolutionary algorithms and their models: A survey of the state-of-the-art.* China: Applied Soft Computing.

Anderson, T., Culler, D., & Patterson, D. (2005). A Case for NOW (Networks of Workstations). *IEEE Micro*.

Abbas, A. (2013). *Grid Computing: Practical Guide To Technology & Applications (Programming Series).* Charles River Media.

McKnight, L., Howison, J., & Bradne, S. (2004). Wireless grids: Distributed resource sharing by mobile, nomadic, and fixed devices. *IEEE Internet Computing*.

Czajkowski, K., Fitzgerald, S., Foster, I., & Kesselman, C. (2001). Grid information services for distributed resource sharing. *Proc. of the 10th IEEE International Symposium on High-Performance Distributed Computing*.

Frank, C., & Karl, H. (2014). Consistency challenges of service discovery in mobile ad-hoc networks. *Proc. of the 7th ACM Int. symposium on modelling, analysis and simulation of wireless and mobile systems*.

Helal, S., Desai, N., V. Verma, C. L., & Konark. (2003). A Service Discovery and Delivery Protocol for Ad-hoc Networks. *Proc.of the Third IEEE Conference on Wireless Communication Networks (WCNC)*.

Bluetooth Interest Group. (2001). *Service discovery protocol Version 1.1 Parte E.*

Villela, D. (2010). Minimizing the average completion time for concurrent grid applications. *Grid Comput 8*.

Chang, R. (2009). An ant algorithm for balanced job scheduling in grids. *Journal of future generation computer system*.

Hummel, K., & Jelleschitz, G. (2007). Robust de-centralized job scheduling approach for mobile peers in ad hoc networks. *7th IEEE Int. Symp. on Cluster Computing and the grid*.

Li, C., & Li, L. (2009). Utility-based scheduling for grid computing under constraints of energy budget and deadline. *Comput. Stand. Inter*.

Liu, H., Roeder, T., Walsh, K., Barr, R., & Sirer, E. (2015). Design and implementation of a single system image operating system for ad hoc networksroc. *Proc 3rd Int. Conf. on Mobile Systems*.

Gomes, A. (2007). DICHOTOMY: a resource discovery and scheduling protocol for multihop ad hoc mobile grids. *7th IEEE Int. Symp. on Cluster Computing and the Grid*.

Selvi, V., Sharfraz, S., & Parthasarathi, R. (2007). Mobile ad hoc grid using trace based mobility model. *GPC*.

Chtepen, M., Flip, H., & Turck, F. D. (2008). Scheduling of dependant grid jobs in absence of exact job length information. *International Workshop NGNM*.

Shah, S. C., & Nizamani, Q.-U.-A. (2012). An effective and robust two-phase resource allocation scheme for interdependent tasks in mobile ad hoccomputational Grids. *Journal of Parallel and Distributed Computing*.

Amin, K., Laszewski, G., & Mikler, A. (2014). Toward an architecture for ad hoc. *Proc. of the IEEE 12th Int. Conf. on Advanced Computing and Communications*.

Koodli, R., & Perkins, C. (2012). Service discovery in on-demand ad hoc networks. *IETF Internet Draft*.

Moreno-Vozmediano, R. (2009). A hybrid mechanism for resource/service discovery in ad-hoc grids. *Future generation computer system*.

Qian Kang, X. L., & Yao, Y. (2016). Efficient authentication and access control of message dissemination over vehicular ad hoc network. *Neurocomputing*.

Subba, B., Biswas, S., & Karmakar, S. (2015). Intrusion detection in Mobile Ad-hoc Networks: Bayesian game formulation. *Engineering Science and Technology, an International Journal*.

Lorch, M., & Kafura, D. (2002). Supporting secure ad-hoc user collaboration in grid enviroments. *Proc. 3rd Int. Workshop on Grid Computing*.

Liao, L., & Manulis, M. (2007). Tree-based group key agreement framework for mobile ad-hoc networks. *Future Generation Computer Systems*.

Marinescu, D., & Marinescu, G. (2013). Ad hoc grids: Communication and computing in a power constrained enviroment. *Proc. 22nd Int. Performance, Computing*.

P. Jamshidi, C. P. (2018). *Microservices: the journey so far and challenges ahead.* IEEE Softw.

Kallergis D, G. Z. (2020). *CAPODAZ: a containerised authorisation and policy-driven architecture using microservices.* Ad Hoc Networks.

Zimmermann, O. (2017). *Microservices tenets.* Comput. Sci.-Res. Dev.

Muhammad W., P. L. (2020). *A Systematic Mapping Study on Microservices Architecture in DevOps.* Journal of Systems and Software Volume 170.

S.A. Alabady, M. S.-T. (2018). *LCPC error correction code for IoT applications.* Sustainable Cities and Society.

A. Al-Fuqaha, M. G. (2015). *Internet of things: A survey on enabling technologies, protocols, and applications.* IEEE Communications Surveys & Tutorials.

C. Mouradian, D. N. (2017). *A comprehensive survey on fog computing: State-ofthe- art and research challenges.* IEEE Communications Surveys & Tutorials.

I. Stojmenovic, S. W. (2014). *The fog computing paradigm: Scenarios and security issues.* Federated Conference on Computer Science and Information Systems.

M. Aazam, S. Z. (2018). *Fog computing architecture, evaluation, and future research directions.* IEEE Communications Magazine.

INTEL. (2020). *Intel 64 and ia-32 architectures developer's manual: Vol. 3a.*

Debadatta Mishra, P. K. (2018). *A survey of memory management techniques in virtualized systems.* Computer Science Review Volume 29.

Barham P., D. B. (2003). *Xen and the art of virtualization.* SIGOPS Oper. Syst. Rev. 37.

Robin J.S., I. C. (2000). *Analysis of the intel pentium's ability to support a secure virtual machine monitor.* Proceedings of the 9th conference on USENIX Security Symposium.

Gerald J. Popek, R. P. (1974). *Formal Requirements for Virtualizable Third Generation Architectures.* Communications of the ACM.

Rosenblum M., G. T. (2005). *Virtual machine monitors: Current technology and future trends.* Computer.

Daley R.C., D. J. (1966). *Virtual memory, processes, and sharing in multics.* Commun. ACM.

Diane Barrett, G. K. (2010). *How Virtualization Happens.* Virtualization and Forensics.

Nathan F.Saraiva de Sousa, D. A. (2019). *Network Service Orchestration: A survey.* Computer Communications.

Biswas A., M. K. (2013). *Los métodos de optimización desempeñan un papel vital en la solución de problemas de ingeniería. Debido a que los métodos de optimización deterministas han mostrado no ser eficientes computacionalmente en la resolución de problemas complejos no lineales y.* J. Optim.

Ferreira, N. a. (2013). *An agent model for the appraisal of normative events based in in-group and out-group relations.*

Vanhèe, L. (2013). *Artificial culture in artificial societies.* International Foundation for Autonomous Agents and Multiagent Systems.

Degens, N. a. (2014). Creating a world for socio-cultural agents. *Emotion Modeling*, 27--43.

Hofstede, G. J. (2015). Gender differences: the role of nature, nurture, social identity and self-organization. *Multi-Agent-Based Simulation XV*, 72--87.

Liang, C., & Yu, F. R. (2015). Wireless Network Virtualization: A Survey, Some Research Issues and Challenges. *IEEE Communications Surveys & Tutorials*.

Fernandez-Baca, D. (1989). Allocating modules to processors in a distributed system. *IEEE Trans. Software Eng*, 427.

Coffman, E. (1976). *Computer and Job-Shop Scheduling Theory.* New York: Wiley.

Shivle, S., Siegel, H., Maciejewski, A. A., & Sugavanam, P. (2010). Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment. *Journal of parallel and distributed computing*.

Pengyao, W., & Jianqin, W. (2018). *Rapid processing of remote sensing images based on cloud computing.* Future Generation Computer Systems.

Qiwan, W., & Ruyin, C. (2020). *E-commerce brand marketing based on FPGA and machine learning.* Microprocessors and Microsystems.

Zhenjie, Z., & Yuanming, Z. (2020). *A novel complex manufacturing business process decomposition approach in cloud manufacturing.* Computers & Industrial Engineering.

Simmhan, Y., Cao, B., & Giakkoupis, M. (2011). *Adaptive rate stream processing for smart grid applications on clouds.* International Workshop on Scientific Cloud Computing.

Elazhary, H. (2018). *Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions.* Journal of Network and Computer Applications.

Nanne, A. J., & Antheunis, M. L. (2020). *The Use of Computer Vision to Analyze Brand-Related User Generated Image Content.* Journal of Interactive Marketing.

Bilal, K., & Khalid, O. (2017). *Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers.* Computer Networks.

Gener, F., & Syst., C. (2016). *Integration of Cloud computing and Internet of Things: A survey.* Future Gener. Comput. Syst.

Assunção, M. d., & Veith, A. S. (2018). *Distributed data stream processing and edge computing: A survey on resource elasticity and future directions.* Journal of Network and Computer Applications.

Goethals, T., & Sebrechts, M. (2018). *Unikernels vs Containers: An In-Depth Benchmarking Study in the Context of Microservice Applications.* 2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2).

Ceballos, H. Z. (2018). *Diseño de un Sub-Sistema de Cómputo Distribuido que permita implementar virtualización inalámbrica para gestionar recursos (Procesamiento, memoria, almacenamiento y dispositivos E/S) distribuidos en una Red Ad Hoc, mediante el modelo de pseudo Estado.* Universidad Nacional de Colombia.

Yu, D., & Ying, Y. (2020). *Balanced scheduling of distributed workflow tasks based on clustering.* Knowledge-Based Systems.

Carroll, T. E., & Grosu, D. (2012). *An incentive-based distributed mechanism for scheduling divisible loads in tree networks.* Journal of Parallel and Distributed Computing.

C.S.Xavier, T., & L.Santos, I. (2020). *Collaborative resource allocation for Cloud of Things systems.* Journal of Network and Computer Applications.

Xu, L., & Li, Y.-p. (2015). *Proportional fair resource allocation based on hybrid ant colony optimization for slow adaptive OFDMA system.* Information Sciences.

Khalila, K., & Elgazzar, K. (2020). *Resource discovery techniques in the internet of things: A review.* Internet of Things.

Venanzi, R., & Kantarci, B. (2018). *MQTT-driven sustainable node discovery for internet of things-fog environments.* Proceedings of the IEEE International Conference on Communications.

S., M., & M., M. (2019). *Using machine learning for handover optimization in vehicular fog computing.* Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing.

Morabito, R., Cozzolino, V., & Ding, A. Y. (2018). *Consolidate IoT edge computing with lightweight virtualization.* IEEE Netw.

Mansouri, Y., & Babar, M. A. (2021). *A review of edge computing: Features and resource virtualization.* Journal of Parallel and Distributed Computing.

Herlihy, M., & Shavit, N. (2020). *Foundations of shared memory.* The Art of Multiprocessor Programming.

Hong, B., & Prasanna, V. (2014). Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. *Proc. 18th Int. Parallel and Distributed Processing Symposium*.

McClatchey, R. (2017). Dara intensive and network aware (DIANA) grid scheduling. *Grid Comput 5*.