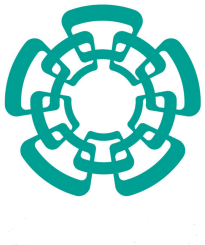


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Cinvestav Tamaulipas

Arquitectura para la construcción de sistemas de compartición de archivos en la nube basada en micro-aplicaciones

Tesis que presenta:

Alfredo Barrón Rodríguez

Para obtener el grado de:

**Maestro en Ciencias
en Ingeniería y Tecnologías
Computacionales**

Director de la Tesis:
Dr. Víctor Jesús Sosa Sosa

Cd. Victoria, Tamaulipas, México. Agosto, 2018

La tesis presentada por Alfredo Barrón Rodríguez fue aprobada por:

_____

Dr. Iván López Arévalo

Dr. José Luis González Compeán

Dr. Víctor Jesús Sosa Sosa, Director

Cd. Victoria, Tamaulipas, México., 10 de Agosto de 2018

A mi familia y amigos

Agradecimientos

- A Dios por sus grandes bendiciones que me ha brindado.
- A mis padres, hermanos y familia que siempre están para apoyarme y ofrecerme todo ese cariño, consejos y motivación.
- A mis amigos Gely, Tany, Rebeca, Ulises y Cristhian por su apoyo incondicional.
- A mi asesor Dr. Víctor J. Sosa Sosa por su conocimiento y dirección en este trabajo de tesis.
- A mis revisores Dr. José Luis González Compeán y Dr. Iván López Arévalo por sus enseñanzas, sugerencias, paciencia y apoyo para mejorar este trabajo de tesis.
- A mis compañeros, en especial a Miguel, Cristo, José y Emanuel por sus aventuras y experiencias vividas en la maestría.
- Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo financiero ofrecido durante los dos años de maestría.
- Al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional por brindarme una formación profesional durante mis estudios de maestría.

Índice General

Índice de Figuras

Índice de Tablas

Resumen

Arquitectura para la construcción de sistemas de compartición de archivos en la nube basada en micro-aplicaciones

por

Alfredo Barrón Rodríguez
Unidad Cinvestav Tamaulipas
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2018
Dr. Víctor Jesús Sosa Sosa, Director

Las tecnologías de almacenamiento y cómputo en la nube representan soluciones costo-beneficio que facilitan la creación de servicios para la compartición de grandes volúmenes de datos. Los servicios para la compartición de archivos (SCA) en la nube cuentan con componentes no-funcionales (p. ej. disponibilidad, balanceo de carga, etc.) que son comúnmente desarrollados para mejorar la calidad de la ejecución de estos servicios y facilitar su evolución. Los procesos de actualización, agregación o remoción de estos componentes se presentan de manera natural como parte de las necesidades cambiantes de una organización. La ejecución de estos procesos puede requerir que los servicios se detengan por un tiempo considerable, lo que podría afectar la continuidad del negocio de las organizaciones. En este trabajo de tesis se presenta una nueva arquitectura de software modular para la construcción de plataformas de compartición de archivos en la nube, encapsulando componentes funcionales y no-funcionales en contenedores virtuales dinámicamente acoplables, ofreciendo flexibilidad funcional a partir de un modelo basado en micro-aplicaciones. A partir del despliegue de la arquitectura, como parte de esta tesis se desarrollaron algoritmos como componentes no-funcionales, tales como disponibilidad, balanceo de carga y tolerancia a fallos que mejoran la experiencia del servicio a los usuarios de los SCA.

Abstract

Architecture for the construction of file sharing systems in the cloud based on micro-applications

by

Alfredo Barrón Rodríguez
Cinvestav Tamaulipas
Center for Research and Advanced Studies of the National Polytechnic Institute, 2018
Dr. Víctor Jesús Sosa Sosa, Advisor

Cloud storage and compute technologies represent cost-benefit solutions that facilitate the creation of services for the sharing of large volumes of data. Cloud file sharing services (FSS) have non-functional components (eg availability, load balancing, etc.) that are commonly developed to improve the quality of the execution of these services and facilitate their evolution. The processes of updating, aggregation or removal of these components are presented naturally as part of the changing needs of an organization. The execution of these processes may require that the services be stopped for a considerable time, which could affect the continuity of the organizations' business. In this work we present a new modular software architecture for the construction of cloud file sharing platforms, encapsulating functional and non-functional components in dynamically attachable virtual containers, offering functional flexibility based on the micro-applications model. From the deployment of the architecture and as part of the thesis, algorithms have been developed as non-functional components, such as load balancing, availability and fault tolerance, improving the service experience of the FSS users.

Nomenclatura

1

Introducción

En este capítulo se expone la motivación y antecedentes que inspiraron a este proyecto de tesis. Se presenta la problemática abordada, la hipótesis de investigación y los objetivos por alcanzar con esta investigación.

1.1 Antecedentes y motivación

El cómputo en la nube (*cloud computing*), de acuerdo con el NIST[1], "*es un modelo para permitir el acceso bajo demanda, a través de la red, a un conjunto compartido de recursos computacionales configurables (p. ej. redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente suministrados y liberados*" [36]. La tecnología de almacenamiento en la nube (*cloud storage*) posibilita el resguardo de datos, los cuales son alojados en espacios de almacenamiento virtualizados, por lo general aportados por terceros. Esto permite la construcción y despliegue de soluciones de costo-beneficio que facilitan la compartición de grandes volúmenes de datos.

[1]Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés)

Los sistemas de compartición de archivos (SCA) en la nube son herramientas que se han convertido en soluciones para ambientes colaborativos, ejemplos de este tipo de sistemas son *Phoenix* [22], *Google Drive* [13], *Dropbox* [14], *OneDrive* [39], etc. En este tipo de ambiente, el intercambio de información sigue un comportamiento similar al de una red social, lo cual significa que en el intercambio de archivos los usuarios desempeñan dos roles diferentes. El primer rol incluye un conjunto de usuarios llamados productores, los cuales comparten archivos. El otro tipo de rol es representado por un conjunto de usuarios llamados consumidores, los cuales descargan archivos de su interés de entre aquellos compartidos por productores. Los usuarios de estos sistemas pueden adquirir el doble rol de productor/consumidor.

El diseño de los SCA en la nube comúnmente toma como referencias arquitecturas de software que siguen modelos en niveles o capas (*tiers/layers*), en donde cada nivel se busca abstraer (o encapsular) cierta funcionalidad y complejidad. Una arquitectura de software está definida por la estructura comprendida por los elementos de sotfware, las propiedades visibles de esos elementos y las relaciones entre ellos, su objetivo general es asegurar que la estructura reunirá presentes y probables futuras demandas sobre el mismo [9]. Cuentan con dos tipos de módulos: funcionales y no-funcionales. Los componentes funcionales son los que describen la operatividad o los servicios que ofrece el sistema. Por otro lado, los componentes no-funcionales son aquellos que cumplen los requisitos que aportan atributos de calidad (p. ej. eficiencia, disponibilidad, seguridad, etc.), especifican los criterios utilizados para mejorar, juzgar y mantener el funcionamiento de un sistema, y por lo general son de naturaleza técnica y difíciles de adaptar [6, 48]. En el modelo de capas, el diseño de cada capa contempla una implementación rígida o poco flexible, es decir no se contempla una actualización dinámica de los aspectos no-funcionales de un mismo servicio.

Actualmente, estamos presenciando el surgimiento de los microservicios (o micro-aplicaciones), que representa una arquitectura que estructura un sistema de software como un conjunto de servicios débilmente acoplados, los cuales implementan diversas funcionalidades o capacidades de una organización. La arquitectura de microservicios permite desplegar aplicaciones complejas y de

tamaño considerable en forma modular, permitiendo a las organizaciones la evolución de su tecnología en fases bien estructuradas. Los microservicios son una nueva tendencia en el ámbito académico, un enfoque en la investigación y un potencial para la adopción industrial. Empresas como Netflix[2] y Amazon[3] han evolucionado sus aplicaciones hacia una arquitectura de microservicios [10].

La solidez técnica que respalda a los microservicios es la principal motivación de este trabajo de tesis, el cual propone el diseño e implementación de una nueva arquitectura de software modular para la construcción de SCA en la nube que sea capaz de encapsular componentes no-funcionales en contenedores dinámicamente acoplables, ofreciendo flexibilidad funcional. Asimismo se propone el encapsulamiento, la implementación y evaluación de componentes no-funcionales experimentales, a través de un prototipo de un SCA desplegado con la arquitectura propuesta para la mejora de la experiencia del servicio de los usuarios.

El grupo de investigación de tecnologías para la gestión de datos y redes de la Unidad Cinvestav Tamaulipas cuenta con un SCA en la nube que ofrece un servicio de almacenamiento. Este sistema al igual que otros SCAs cuentan con una arquitectura monolítica que lo hace poco flexible para la agregación y actualización de componentes no-funcionales, es por ello que se utilizó como sistema de referencia para probar nuestra arquitectura en un entorno real.

1.2 Planteamiento del problema

En la actualidad los SCA siguen un modelo de desarrollo en capas, donde cada capa es diseñada para cumplir una función específica que conlleva aspectos funcionales y no-funcionales que responden a la necesidades de una organización en un tiempo determinado. Este modelo de desarrollo de SCA presenta limitaciones en cuanto a su poca flexibilidad para adaptarse dinámicamente a los cambios y a las nuevas necesidades que surgen en las organizaciones. En este contexto, cuando se actualizan los componentes del SCA surgen desafíos de naturaleza técnica[6, 48] a medida de que se añaden

[2] www.netflix.com
[3] www.amazon.com

características o servicios al sistema, la base del código fuente será considerablemente más compleja, implicando lo siguiente:

1. Modificación del código fuente al actualizar aspectos no-funcionales.
2. Interrupción del servicio en caso de requerir mejoras o cambios en aspectos no-funcionales.
3. Limitación del uso de los servicios que provee la nube anfitriona.
4. Incremento en la complejidad en el proceso de despliegue y escalabilidad limitada.

Como se puede observar, la actualización dinámica de estos componentes es requerida para cubrir las necesidades cambiantes de los usuarios y las organizaciones, buscando evitar que los procesos de agregación, remoción y actualización de aspectos no-funcionales impliquen la interrupción del servicio, lo que afectaría la continuidad de la operación de las organizaciones.

1.3 Hipótesis

La hipótesis que se plantea a partir de la problemática y el marco teórico estudiado, es la siguiente:

El encapsulamiento de los componentes de un SCA en micro-aplicaciones y microservicios mediante contenedores virtuales, permitirá que se puedan construir y desplegar SCA en forma eficiente, flexible y adaptable.

1.4 Objetivos

En esta sección se describen los objetivos de investigación que definieron el rumbo del trabajo realizado en esta tesis.

1.4.1 Objetivo general

Contribuir al estado del arte con una nueva arquitectura de microservicios y micro-aplicaciones modularizada mediante contenedores virtuales que permita la construcción de sistemas adaptables de compartición de archivos en la nube en forma flexible y eficiente.

1.4.2 Objetivos particulares

- Crear un esquema de codificación que permita representar de manera abstracta a un SCA.
- Contar con un mecanismo de acoplamiento que permita la adición/remoción/integración de componentes no-funcionales en un SCA en la nube.
- Proponer nuevos algoritmos para su despliegue en la arquitectura propuesta que mejoren las prestaciones de los componentes no-funcionales en términos de tiempos de servicio/respuesta.

1.5 Solución propuesta

En la Figura 1.1 se presenta una representación de la propuesta de este trabajo de tesis donde se propone una arquitectura modular basada principalmente en tres componentes para la construcción de SCA en la nube, bajo un modelo de micro-aplicaciones:

1. *Unidad de construcción*: es una representación abstracta de un módulo que incluye una capa de control y una de procesamiento que se encontrará encapsulada en contenedores virtuales.

2. *Subservicio virtual*: este componente agrupa a un conjunto de unidades de construcción, permitiendo la formación de secuencias o estructuras de procesamiento, en función de un estilo arquitectónico (p. ej. maestro/esclavo, tubería o flujo de trabajo).

3. *Gestor global*: la función de este componente permite gestionar y configurar los elementos de la arquitectura (unidades de construcción y subservicios virtuales), para la construcción de un SCA en la nube.

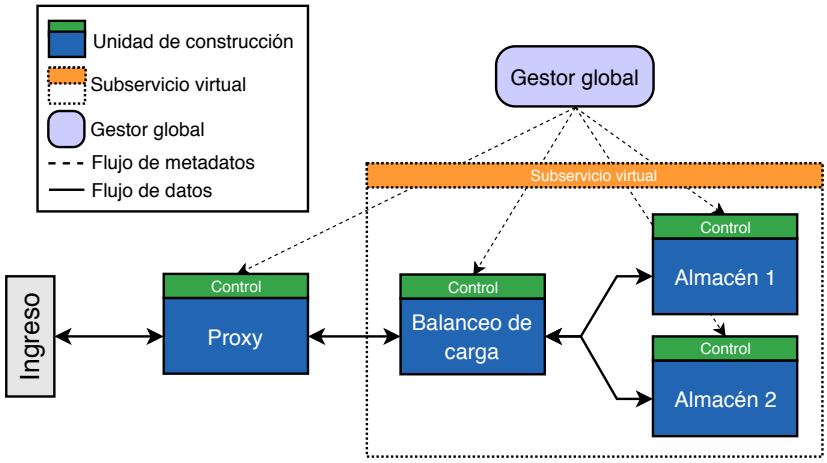


Figura 1.1: Esquema propuesto de la arquitectura modular.

Los componentes no-funcionales que formarán parte del despliegue del SCA en la nube son:

1. *Balanceo de carga*: es un método para distribuir la carga de trabajo entre varios equipos de cómputo para lograr una utilización óptima de los recursos, en el uso eficiente de almacenamiento para este caso.

2. *Disponibilidad*: es el proceso en el que se garantiza que se puede acceder a los datos en cualquier momento, de forma rápida y sencilla, este componente tendrá el enfoque basado en la replicación de archivos.

3. *Tolerancia a fallos*: permite a un sistema seguir funcionando correctamente en caso de alguna falla, esta propiedad aplicará a los nodos de almacenamiento de los archivos.

1.6 Metodología de trabajo

El desarrollo del trabajo de tesis se llevó a cabo en las siguientes cinco etapas:

- Etapa 1. Definición del alcance: En esta etapa se realizó una revisión profunda al estado del arte para una inspección detallada acerca de las posibles propuestas publicadas que abordan el problema, acotando las situaciones particulares, ambientes de despliegue y dominio específico que nuestra solución aborda. Para lo anterior, se revisó el estado del arte dando especial interés a los siguientes temas:

  - Arquitecturas de software
    - En capas o niveles.
    - Orientada a servicios y servicios web.
    - Microservicios y contenedores.
  - Sistemas de compartición de archivos en la nube.
  - Construcción de flujos de trabajo.
  - Construcción de sistemas de microservicios.

- Etapa 2. Diseño y desarrollo de arquitectura: Durante esta etapa se obtuvo el esquema de la arquitectura modular, para ello se plantearon una serie de actividades para el diseño y desarrollo de los componentes que forman parte de la arquitectura:

  - *Unidad de construcción*: es un componente de software esencial con características y/o propiedades definidas para adquirir la propiedad de modularidad.
  - *Subservicio virtual*: para cumplir la propiedad de acoplamiento y la definición de interacciones y comunicación entre los componentes modulares de software.

    - *Gestor global*: encargado de determinar la organización de las secuencias y estructuras con sus elementos, responsabilidades e interrelaciones para los componentes de software modulares.

- Etapa 3. Diseño y desarrollo de componentes no-funcionales para un SCA: Durante esta se diseñaron y desarrollaron algoritmos experimentales para los componentes no-funcionales, los cuales fueron encapsulados en las unidades de construcción, formando así la parte del despliegue modular que constituye la arquitectura de un SCA. Como parte del alcance de esta tesis, el esquema modular permite la agregación, remoción y actualización dinámica de componentes que brindan los siguientes aspectos no-funcionales:

    - Balanceo de carga.
    - Disponibilidad.
    - Tolerancia a fallos.

- Etapa 4. Evaluación experimental y mejoras: A partir de la arquitectura modular y los algoritmos de los componentes no-funcionales encapsulados se llevó acabo la implementación y despliegue de un prototipo de software, con la construcción de un SCA en la nube como prueba de concepto. Se realizó la definición, descripción de configuraciones, utilización de métricas, la elaboración de experimentos y mejoras del prototipo para la agregación, remoción o actualización de los componentes no-funcionales propuestos con el fin de evaluar la arquitectura modular.

- Etapa 5. Análisis y presentación de resultados: En esta etapa se analizaron los resultados de los componentes no-funcionales y la arquitectura modular de la evaluación experimental llevada a cabo en la etapa anterior, para la obtención de una discusión y conclusiones sobre la comprobación de la hipótesis del trabajo de tesis.

1.7 Organización de la tesis

Esta tesis está conformada de 6 capítulos, los cuales se encuentran organizados de la siguiente manera:

El Capítulo 1 presenta el contexto de este trabajo de tesis. En el Capítulo 2 se realiza una descripción del fundamento teórico requerido para el trabajo de tesis realizado. Además se presenta una revisión del estado del arte respecto a sistemas de compartición de archivos en la nube, construcción de flujos de trabajo e integración de microservicios, lo que permitirá al lector tener una mejor compresión del documento. En el Capítulo 3 se describe el diseño y desarrollo de la propuesta de solución para alcanzar los objetivos. En el Capítulo 4 se describe la metodología para seguir la evaluación experimental. Además, se discuten los resultados obtenidos de la experimentación realizada. En el Capítulo 5 se presenta un resumen de un estudio de caso del desarrollo y experimentación de un SCA con nuevos algoritmos para mejorar aspectos no-funcionales. Por último, el Capítulo 6 presenta las conclusiones, contribuciones y limitaciones del trabajo realizado, así como el trabajo futuro.

2

Marco teórico y estado del arte

En este capítulo se presentan distintos conceptos teóricos, arquitecturas y tecnologías involucradas en el desarrollo de este trabajo de tesis. Se presenta también una recopilación y análisis de las propuestas disponibles en la literatura sobre sistemas de compartición de archivos, construcción de flujos de trabajo e integración de microservicios.

2.1 Cómputo en la nube

El cómputo en la nube (*cloud computing*) se ha convertido en un tema de gran relevancia por su modelo de consumo de recursos. Este concepto es un modelo que permite el uso de un conjunto de recursos informáticos configurables de forma ubicua, conveniente y bajo demanda que pueden ser rápidamente aprovisionados y liberados con un esfuerzo mínimo de administración o interacción por parte de un proveedor de servicio [36]. Este modelo se componen de cinco características esenciales que define el NIST son: *1)* Auto-servicio bajo demanda, *2)* Amplio acceso a la red, *3)* Piscina de recursos, *4)* Rápida elasticidad y *5)* Servicio medido. Un servicio de nube provee tres

modelos de servicio que son: Software como servicio (SaaS), *2)* Plataforma como servicio (PaaS) y *3)* Infraestructura como servicio (IaaS). Existen cuatro modelos de despliegue que son: *1)* Pública, *2)* Privada, *3)* Híbrida y *4)* Comunitaria.

2.2 Virtualización para la nube

La virtualización es la simulación del sofware o hardware sobre el que se ejecuta otro software. Es una forma de particionar de forma lógica un equipo físico en distintos entornos simulados. Hay dos tipos de tecnologías para la virtualización de servidores que son comunes en entornos de virtualización de centros de datos: virtualización a nivel hardware y la virtualización a nivel software:

- **Virtualización a nivel hardware**.

  La virtualización de hardware implica virtualizar el hardware en un servidor y la creación de máquinas virtuales (MVs) que proporcionan la abstracción de una máquina física. La virtualización de hardware implica ejecutar un hipervisor, también llamado monitor de máquina virtual (VMM, por sus siglas en inglés), directamente en el servidor. El hipervisor emula el hardware virtual como CPU, memoria, dispositivos de E/S y red para cada MV. Cada MV se ejecuta un sistema operativo independiente y las aplicaciones sobre ese sistema operativo. El hipervisor también es responsable de multiplexar los recursos físicos subyacentes en las MVs residentes [47].

- **Virtualización a nivel de sistema operativo**.

  La virtualización del sistema operativo implica la virtualización del kernel del sistema operativo en lugar del hardware físico. A nivel del sistema operativo virtual las máquinas se conocen como contenedores virtuales (VC, por sus siglas en inglés). Cada VC encapsula un grupo de procesos que están aislados de otros contenedores o procesos en el sistema. El núcleo del sistema operativo es responsable de implementar la abstracción del VC. Se asignan recursos

compartidos de CPU, memoria, dispositivos de E/S y de red a cada VC y también puede proporcionar un sistema de archivos aislamiento [4, 47].

2.2.1 Plataformas y orquestadores de contenedores

Para la creación, ejecución de contenedores virtuales es necesario la instalación de una plataforma de despliegue y algunos frameworks de administración de recursos, como las que se listan a continuación:

Docker

Docker es una plataforma de contenerización que proporciona una forma de ejecutar aplicaciones aisladas de forma segura en un contenedor, empaquetado con todas sus dependencias y bibliotecas. Debido a que su aplicación siempre se puede ejecutar con el entorno que espera en la imagen de construcción, la prueba y el despliegue es más simple que nunca, ya que su compilación será completamente portátil y lista para ejecutarse de acuerdo a como se ha diseñado en un entorno particular. Debido a que los contenedores son ligeros y se ejecutan sin la carga adicional de un hipervisor, puede ejecutar varias aplicaciones que dependen de diferentes bibliotecas y entornos en un solo kernel, cada uno se encuentra aislado y no interfieren con los demás. Esto le permite sacar más provecho de su hardware cambiando la "unidad de escala" de su aplicación de una máquina virtual o física a una instancia de contenedor [11].

Docker proporciona una forma sistemática de automatizar el despliegue rápido de aplicaciones Linux dentro de VC. Básicamente, Docker extiende LXC[29] con una API a nivel de kernel y a nivel de aplicación que permiten la ejecución de procesos aislando: CPU, memoria, E/S, red, etc. Docker también utiliza espacios de nombres para aislar completamente la vista de una aplicación del entorno operativo subyacente, incluidos los árboles de proceso, la red, los ID de usuario y los sistemas de archivos[15, 31].

Algunas recomendaciones para la utilización de Docker para el desarrollo de investigación reproducible están disponibles en [5].

Kubernetes

Kubernetes es una plataforma de código abierto para automatizar el despliegue, la ampliación y las operaciones de los contenedores de aplicaciones entre clusters de hosts, proporcionando una infraestructura centrada en contenedores. Es un gestor de clúster de código abierto para contenedores Docker [1, 31].

Apache Mesos

Apache Mesos abstrae la CPU, la memoria, el almacenamiento y otros recursos de computación lejos de las máquinas (físicas o virtuales), permitiendo que los sistemas distribuidos elásticos y tolerantes a fallos sean fácilmente construidos y ejecutados con eficacia.

Apache Mesos proporciona varios mecanismos de aislamiento en los "esclavos" para realizar diferentes "tareas". La asignación de recursos a un framework tarea o usuario no debe tener ningún efecto no deseado en los trabajos en ejecución. Los contenedores actúan como máquinas virtuales ligeras, proporcionando el mecanismo de aislamiento necesario sin la sobrecarga de las máquinas virtuales. Utilizando contenedores, podemos limitar la cantidad de recursos a los que puede acceder el proceso y todos sus procesos secundarios [30].

2.3 Arquitecturas de software

Una arquitectura de software es la estructura de un sistema en términos de sus componentes especificados por separados y sus interrelaciones. Su objetivo general es asegurar que la estructura reune presentes y probables futuras demandas del sistema [9].

Las arquitecturas cuentan con dos tipos de módulos:

- **Funcionales**: Describen la operatividad o los servicios del sistema, depende del tipo de software, de los usuarios esperados y del tipo de sistema en el que se utilice el software [6], para este caso los sistemas de compartición de archivos son:
  - **Carga**: Sucede cuando un usuario transfiere un archivo desde su computadora o otro que está configurado para recibirla.
  - **Descarga**: Sucede cuando un usuario transfiere un archivo a su computadora de otra fuente.
  - **Compartición**: Esta operación se realiza mediante la URL[1] del archivo, que es la unidad de intercambio.
- **No-funcionales**: Son aquellos que cumplen los requisitos que aportan atributos de calidad de un sistema [48], algunos son:
  - **Rendimiento**: Es la capacidad para producir o realizar algo con la cantidad mínima de esfuerzo y/o recursos (efectividad), algunos métodos son *balanceo de carga*, *planificador*, *alta disponibilidad* y *escalabilidad*.
  - **Disponibilidad**: Es la capacidad de un sistema para ofrecer un servicio ininterrumpido, a pesar de que uno o más componentes fallen (tiempo sin interrupción, tiempo de inactivad), algunos métodos son *tolerancia a fallos*, *resiliencia* y *confiabilidad*.
  - **Seguridad**: Es la capacidad para proporcionar protección del sistema, sus componentes y datos contra amenazas o ataques, algunos métodos son *confidencialidad*, *integridad* y *privacidad*.

[1] Localizador Uniforme de Recursos (URL, por sus siglas en inglés)

- **Transparencia**: Es la capacidad de un sistema distribuido para ocultar su naturaleza distribuida de sus usuarios, pareciendo y funcionando como un sistema coherente unificado normal, algunos métodos son *monitoreo*, *adaptabilidad* y *flexibilidad*.

Existen diferentes formas de encapsular y desplegar aplicaciones y/o servicios en un entorno de nube, a continuación se mencionan las más releventes [15]:

- **Máquinas virtuales**: Es un entorno aislado que parece ser una computadora entera pero en realidad sólo tiene acceso a una parte de los recursos de la computadora [35].
- **Contenedores virtuales**: Son un concepto de virtualización similar a las máquinas virtuales pero más ligero, son menos recursos y tiempo de consumo [4, 41].

2.3.1 Arquitectura monolítica

El software monolítico está diseñado para ser autónomo; los componentes del programa están interconectados e interdependientes en lugar de estar débilmente acoplados, como es el caso de los programas de software modulares. En una arquitectura estrechamente acoplada, cada componente y sus componentes asociados deben estar presentes para que el código sea ejecutado o compilado.

- *Ventajas*
  - Fácil de implementar y optimizar el rendimiento.
  - Control estricto, sin dependencias de terceros, sin dependencias de código.
  - No tiene problemas de compatibilidad o de cambio de contexto.
- *Desventajas*
  - Punto único de falla.
  - Cuellos de botella.
  - Rendimiento, cada vez mas difícil de mantener.

2.3.2 Arquitectura en capas o niveles

Las arquitecturas tradicionales para construir sistemas de compartición de archivos como se muestra en la Figura 2.1, se componen de los siguientes elementos:

- **Módulos**: Software de propósito específico que cumpla con requisitos no-funcionales y/o funcionales.
- **Capas**: Representa una simplificación de un sistema mediante la abstracción de la complejidad de sus partes.
- **Niveles**: Representa un servicio o aplicación que integra módulos independientes que cubren aspectos funcionales y no-funcionales y que se encapsulan como una unidad de despliegue.
- **Inquilinos**: Usuarios concurrentes asistidos en cualquiera de los niveles, módulos o capas.

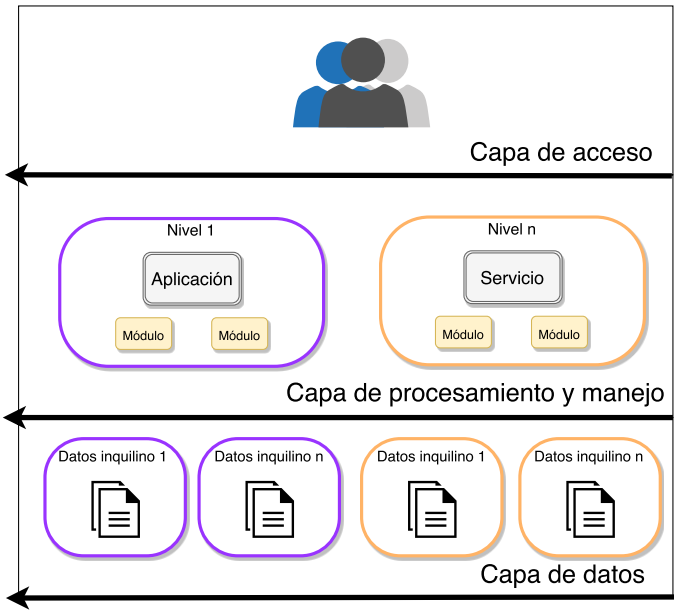


Figura 2.1: Arquitectura basada en niveles (*tiers/layers*) de un sistema de compartición de archivos.

Este tipo de arquitecturas generalmente están compuestas de las siguientes capas, con sus respectivas funciones [23]:

- **Acceso**: Se encarga de la autenticación, recibir las peticiones de carga, descarga y compartición de los usuarios del sistema. Es en este punto donde se inicia la interacción entre el usuario, la compartición de archivos y los recursos de almacenamiento. Dicha capa también es la encargada de autorizar las peticiones recibidas para determinar si éstas deben ser enviadas a la capa de procesamiento y manejo.
- **Procesamiento y manejo**: Se encarga de procesar y gestionar las peticiones que fueron autenticadas en la capa anterior, también determina el flujo que deben llevar las peticiones de entrada/salida de los usuarios.
- **Datos**: Se encarga de almacenar los archivos de los usuarios. La estrategia de almacenamiento es gestionado por la capa de procesamiento y manejo.

2.3.3 Arquitectura orientada a servicios y servicios web

La arquitectura orientada a servicios (SOA, por sus siglas en inglés) presenta un enfoque para crear sistemas distribuidos que ofrecen funcionalidad de aplicación como servicios a aplicaciones de usuario final u otros servicios. Este estilo arquitectónico permite la flexibilidad de negocios de una manera interoperable, tecnológica y agnóstica. SOA consiste en un conjunto compuesto de servicios alineados con los negocios que soportan una realización de procesos de negocios flexible y dinámicamente reconfigurable de extremo a extremo utilizando descripciones de servicios basados en interfaz. Esta arquitectura se componen de elementos que se pueden clasificar en funcionales tales como *transporte*, *servicio de protocolo de comunicación*, *descripción del servicio*, *servicio*, *registro de servicio* y *procesos de negocio*; y calidad de servicio como *política*, *seguridad*, *transacción* y *gestión*, como se observa en la pila de arquitectura [16] en la Figura 2.2.

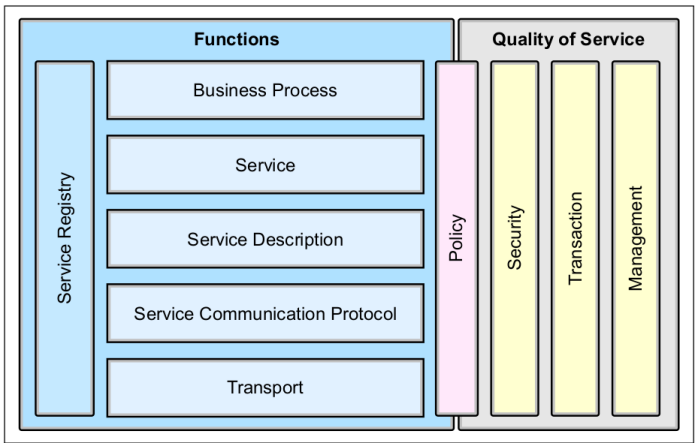


Figura 2.2: Elementos de SOA.

Sus principales características son:

- **Dinamismo**: Se pueden lanzar nuevas instancias del mismo servicio para dividir la carga del sistema.
- **Modularidad y reutilización**: Los servicios complejos se componen de servicios sencillos. Los mismo servicios pueden ser utilizados por diferentes sistemas.
- **Desarrollo distribuido**: Al acordar las interfaces del sistema distribuido, distintos equipos de desarrollo pueden desarrollar particiones de la misma en paralelo.
- **Integración de servicios heterogéneos y legados**: Los servicios simplemente tienen que implementar protocolos estándar de comunicación.

Los servicios web proporcionan un enfoque de computación distribuida para integrar aplicaciones extremadamente heterogéneas a través de Internet. Las especificaciones del servicio web son complementamente independientes del lenguaje de programación, el sistema operativo y el hardware para promover el acoplamiento suelto entre el consumidor del servicio y el proveedor, la tecnología se

basa en tecnologías abiertas tales como: HTTP[2], XML[3], SOAP[4], UDDI[5] y WSDL[6], que proporcionan una base sólida para implementar una arquitectura SOA [16].

2.3.4 Arquitectura de microservicios

Lewis and Fowler [33] definen el estilo arquitectónico del microservicio como un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose con mecanismos ligeros, a menudo una API de recursos HTTP.

Los microservicios representan un enfoque conceptual para el desarrollo de aplicaciones como conjuntos de servicios pequeños, poco compactos y compostables que trabajan en conjunto, donde cada servicio es un proceso autónomo completo que proporciona una pequeña cantidad de funcionalidad que se puede programar en cualquier lenguaje de programación y administrar por separado. La Figura 2.3 muestra el ejemplo de una arquitectura de microservicios que contiene componentes que conforman un sistema o servicio más complejo. Se suele considerar la arquitectura de microservercios como una forma especifica de realizar una arquitectura SOA [12, 37, 38, 45].

Sus características primordiales son [12]:

- **Flexibilidad**: Un sistema es capaz de mantenerse al día con el entorno empresarial en constante cambio y es capaz de apoyar todas las modificaciones que es necesario para una organización para mantenerse competitivo en el mercado.

- **Modularidad**: Un sistema está compuesto de componentes aislados donde cada componente contribuye al comportamiento general del sistema en lugar de tener un solo componente que ofrece funcionalidad completa.

[2]Protocolo de Transferencia de Hipertexto (HTTP, por sus siglas en inglés)
[3]Lenguaje de Marcado Extensible (XML, por sus siglas en inglés)
[4]Protocolo Simple de Acceso a Objetos (SOAP, por sus siglas en inglés)
[5]Descripción, Descubrimiento e Integración Universales (UDDI, por sus siglas en inglés)
[6]Lenguaje de Descripción de Servicios Web (WSDL, por sus siglas en inglés)

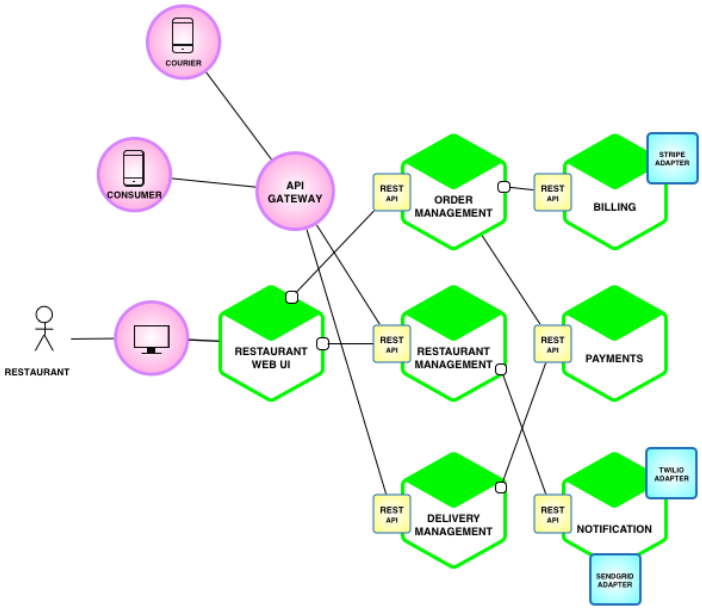


Figura 2.3: Ejemplo de arquitectura de microservicios.

- **Evolución**: Un sistema debe mantenerse mantenible mientras evoluciona constantemente y agrega nuevas características.

2.4 Trabajos relacionados

A continuación se presenta una comparativa del trabajo relacionado considerado los componentes principales de la investigación: sistemas de compartición de archivos en la nube, construcción de flujos de trabajo e integración de microservicios.

2.4.1 Sistemas de compartición de archivos en la nube

El uso compartido de archivos en la nube, también denominado compartición de archivos en la nube o el uso compartido de archivos en linea, es un sistema en el que se asigna espacio de almacenamiento a un usuario en un servidor y se realizan escrituras y lecturas a través de Internet.

Globus

Foster and Kesselman [19] aportaron el proyecto *Globus*[7] que es un conjunto de bibliotecas y programas que aborda los problemas comunes que se producen al crear servicios y aplicaciones en sistemas distribuidos, propone una variedad de componentes que abordan la gestión de recursos, la transferencia de datos, la publicación de datos, la compartición de datos y una plataforma como servicio. Contiene la arquitectura de servicios de Grid abierta [20, 28] (OGSA, por sus siglas en inglés) que está basada en la arquitectura SOA para computación Grid, que asegura la interoperabilidad en sistemas heterogéneos para diferentes tipos de recursos puedan comunicarse y compartir información mediante diferentes interfaces como se observa en la Figura 2.4, también se basa en otras tecnologías de servicios web, como WSDL y SOAP, pero pretende ser en gran medida independiente del manejo de datos a nivel transporte.

Principalmente fue propuesto en 2005 como *Globus Toolkit* [17] haciendo uso de cómputo Grid, en el 2011 cambio de Grid a la nube llamándose ahora como *Globus Services* [18]. Su característica esencial es la integración de recursos mediante diferentes interfaces para lograr que los servicios y aplicaciones distribuidas permitan el intercambio de información y utilizar la información intercambiada entre dos o mas sistemas o componentes.

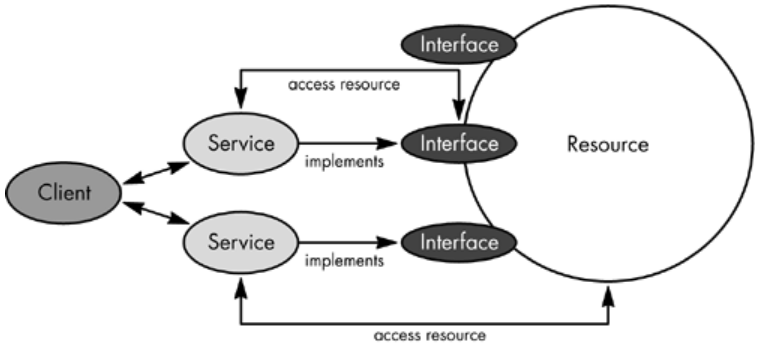


Figura 2.4: Arquitectura OGSA.

[7] www.globus.org

SkyCDS

En 2015 Gonzalez et al. [23] presentaron *SkyCDS*, el cual proporciona un servicio de distribución de contenido a usuarios. Su característica es la flexibilidad pues su arquitectura se basa en capas en una nube diversificada. Las capas de la arquitectura eran metadatos y de flujo de almacenamiento de contenido mediante el uso de servicios web, que permiten la comunicación entre las capas y componentes integrados, se puede observar en la Figura 2.5 que la capa superior, o de metadatos, se basa en patrones de publicación-suscripción para nutrir el contenido de los archivos por usuarios. La capa inferior, o de flujo de almacenamiento, se basa en la dispersión de la información en múltiples locaciones que externa el contenido de manera controlada.

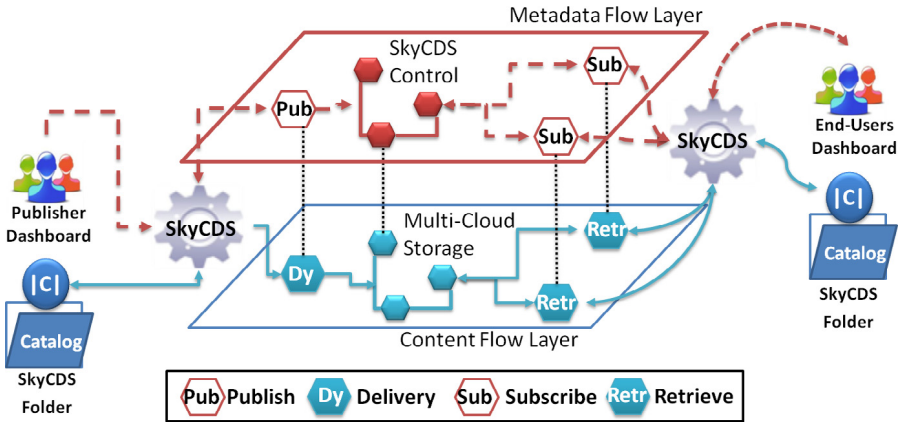


Figura 2.5: Arquitectura en capas de SkyCDS.

La ventaja de este enfoque se centra en distribuir la carga de trabajo con el aprovechamiento de múltiples núcleos en la capa de almacenamiento, así como la distribución de tareas de dispersión (por parte del publicador) así como las de recuperación (en el lado del suscriptor o usuario final) en la capa de metadatos, donde la utilización de capas fue el enfoque para la construcción de este sistema para la entrega de contenidos en la nube.

2.4.2 Flujos de trabajo

Los flujo de trabajo son estructuras configurables con aspectos operacionales de una actividad de trabajo a través de etapas encadenadas que realizan tareas específicas [27, 50]. Los aspectos operacionales refieren a la manera en cómo se estructuran las tareas, cuál es su orden correlativo, cómo se sincronizan, cómo fluye la información que soporta las tareas y cómo se hace el seguimiento del cumplimiento de las tareas.

Triana

Triana[8] presentado por Taylor et al. [49] es un entorno de flujo de trabajo que consiste en una GUI[9] y un sistema subyacente que permite la integración con múltiples servicios e interfaces. Esta herramienta pueden interconectar componentes que son bloques de construcción que contienen un algoritmo, proceso o servicio distribuido. La GUI contiene un conjunto de cajas de herramientas o servicios que permiten conectarlos para formar un flujo de datos y mediante puertos de entrada y salida dando como resultado una tubería de procesamiento como se muestra en la Figura 2.6.

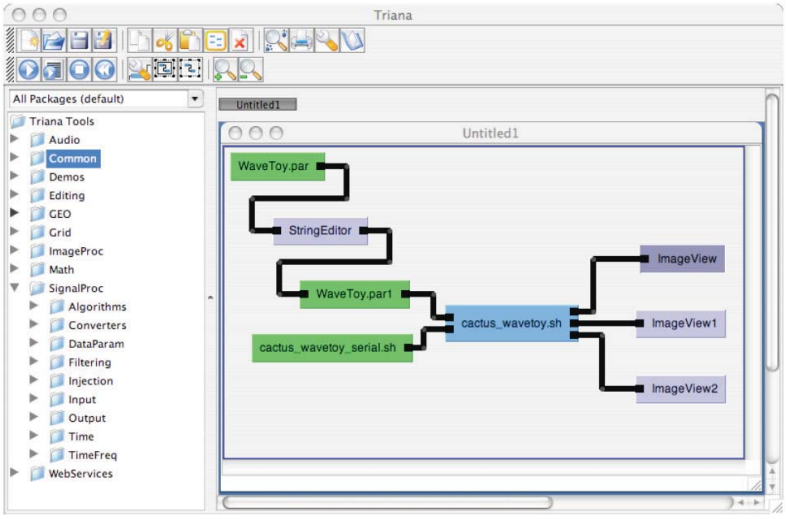


Figura 2.6: Interfaz gráfica de usuario en Triana.

[8] http://trianacode.org/
[9] Graphical User Interface

Su sistema subyacente consiste en una colección de interfaces que unen diferentes tipos de servicios GAT[10] o middleware incluidos que unen GRAM[11], GridGTP y GRMS[12]; y GAP[13] que integra servicios web, P2PS[14] y JXTA. Sus componentes distribuidos se dividen en dos categorías, componentes orientados a Grid que representan aplicaciones ejecutadas via Grid que son accedidos mediante GAT y componentes orientados a servicio que representan entidades que se pueden invocar vía interfaz de red que son accedidos mediante GAP para publicar y descubrir entidades dentro de las redes orientadas a servicios.

Su principal característica es la modularidad de sus componentes que son definidos en lenguaje XML que especifican el nombre, especificaciones de E/S y parámetros, que permite una granularidad pequeña de trabajo que aceptan, procesan y envían datos. La ventaja de este enfoque permite la creación de flujos de trabajo con sus módulos independientes para la configuración dinámica de ellos.

DevOps

Bass et al. [3] definieron *DevOps*[15] como un conjunto de prácticas que tienen como objetivo disminuir el tiempo entre cambiar un sistema y transferir ese cambio al entorno de producción. Las prácticas involucran procesos, arquitectura, herramientas y negocios que permite la construcción de aplicaciones de software.

En la arquitectura que ellos proponen utilizan los términos de módulo, componente, servicio y cliente como se pueden observar en la Figura 2.7, donde un módulo es una unidad de código con funcionalidad coherente, un componente es una unidad ejecutable, un servicio se refiere a un componente que proporciona una función y un cliente para referirse a un componente que solicita un servicio [2].

[10] Grid Application Toolkit
[11] Grid Resource Allocation Manager
[12] GridLab Resource Management System
[13] Grid Application Prototype
[14] Peer-to-Peer Simplified
[15] https://devops.com/

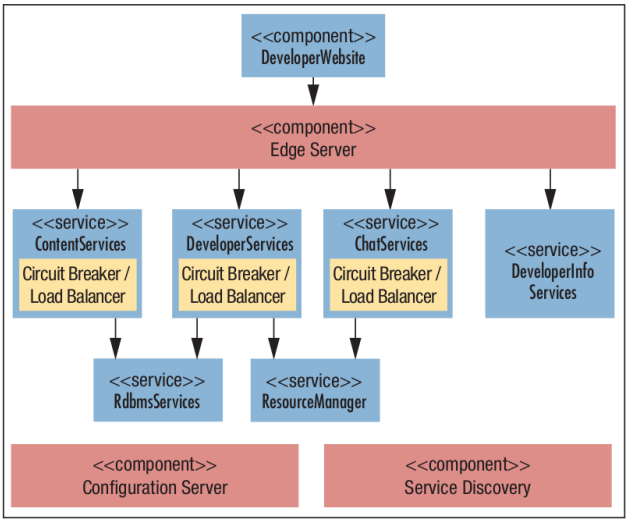


Figura 2.7: Arquitectura de DevOps.

El estilo de la arquitectura de microservicios es usado en la practica por las organizaciones que han adoptado o inspirado muchas prácticas de DevOps [2]. Su principal ventaja es que un componente no puede entrar en producción a menos que el componente es compatible con otros componentes con los que interactúa, también hace uso de su conjuntos de prácticas que son la construcción de software, desde la integración, las pruebas, la liberación hasta la implementación y administración de la infraestructura.

Sacbe

Gonzalez-Compean et al. [24] propusieron *Sacbe* que es una arquitectura de software modular que permite encapsular módulos de software en contenedores llamados *building blocks* (BBs) que están interconectados por las interfaces de comunicación de E/S (como la red, la memoria y el sistema de archivos), para el almacenamiento, compartición y seguridad de los datos [27]. Esta propuesta fue realizada para aplicaciones extremo a extremo de almacenamiento en la nube como se muestra en la Figura 2.8 donde el usuario construye las soluciones formando un flujo de trabajo, donde desarrolla procesamiento de los datos mediante una tubería para después enviarlo al servicio de almacenamiento en la nube.

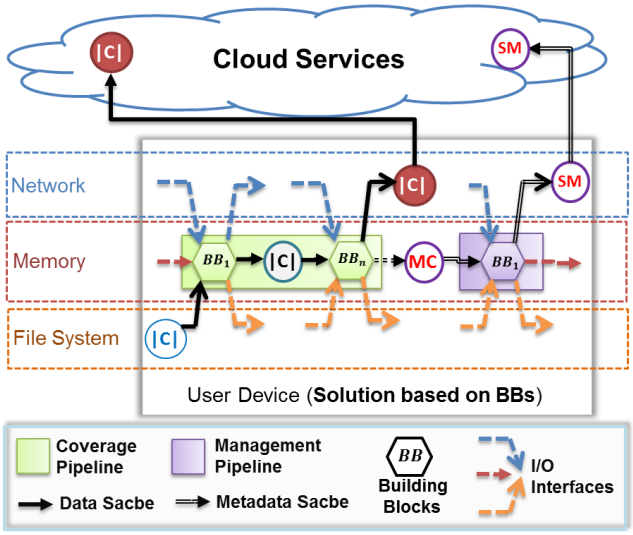


Figura 2.8: Representación conceptual de la arquitectura extremo a extremo de Sacbe.

En *Sacbe* sus módulos de cobertura y gestión son encapsulados en micro-aplicaciones independientes en los contenedores BBs. Donde estos BBs pueden realizar tareas como adquisición, entrega y procesamiento. Su esencial ventaja es el uso de micro-aplicaciones donde cada componente de lado del usuario es utilizado y configurado de manera modular, como el encapsulamiento de sus módulos dando una flexibilidad para generar un flujo de trabajo y un flujo de datos.

2.4.3 Microservicios

El software construido en base a microservicios se podrá descomponer en varias partes funcionales independientes para conformar un sistema global, la tendencia natural en cuanto a los microservicios es permitir el creciemiento del sistema tanto por nueva funcionalidad (escalado vertical) o por mejora en la eficiencia (escalado horizontal).

Istio

En el 2017, Google [25] proponen *Istio* es una solución que proporciona una forma uniformemente de integrar microservicios, gestionar el tráfico de flujo (balanceo de carga) entre los microservicios

en plataformas de la nube sin necesidad de cambios en el código de la aplicación [21]. Fue creado para abordar los desafíos inherentes que vienen con la integración de microservicios basados en aplicaciones en sistemas distribuidos.

Esta solución contiene un componente utilizado para gestionar el tráfico de peticiones y también se encarga de gestionar y configurar instancias de *proxy*, donde estas instancias representan una capa de software que mantiene todos los microservicios en un servicio de malla que es puesto en cada contenedor con su respectiva aplicación. Estos *proxies* son encargados son encargados de recibir y enviar todas las peticiones/respuesta entre el cliente y el servicio. En la Figura 2.9 se puede observar una aplicación distribuida haciendo uso de la solución *Istio*, donde son utilizados las instancias de *proxy* representadas por un rectángulo que son llamados *Envoy*.

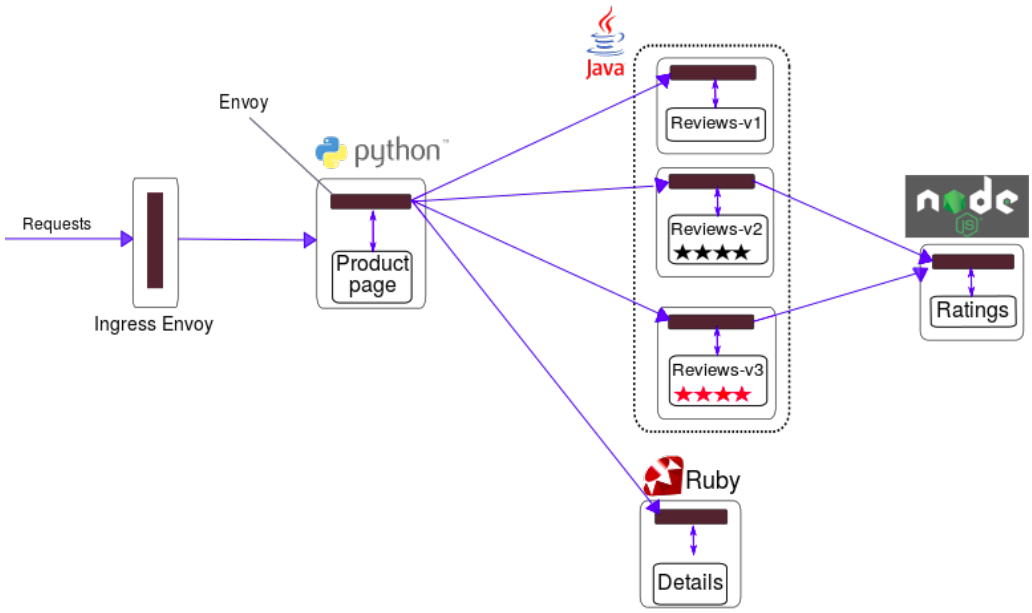


Figura 2.9: Istio en una aplicación distribuida.

Su principal ventaja es que permite al código de la aplicación desacoplarse de la evolución de sus servicios independientes, como también la manera de trabajar con diferentes lenguajes de programación de parte de la aplicación y el uso de contenedores, una desventaja es que es solo una solución y no cuenta con un modelo de su arquitectura para la construcción de sistemas de

compartición de archivos.

Spring Cloud

Spring Cloud[16] [7, 8] es un proyecto que proporciona herramientas para facilitar el desarrollo de aplicaciones distribuidas. Contiene componentes diseñados para ser utilizados para construir patrones comunes en sistemas distribuidos, la mayoría de los componentes de la arquitectura fueron desarrollados por Netflix.

Algunos patrones se muestran en la Figura 2.10, a continuación se presentan los siguientes: gestión de configuración (*Cloud-config*) que almacena las propiedades de configuración de los microservicios, descubrimiento de servicios (*Eureka*) que es un servidor para el registro y localización de microservicios, balanceo de carga y tolerancia a fallos, ruteo inteligente (*Zuul*) que permite un enrutado dinámico, balanceo de carga, monitorización y seguridad de peticiones, control de bus (*Hystrix*) que permite gestionar las interacciones entre servicios del sistema, balanceo de carga de lado del usuario (*Ribbon*) que se encarga de la comunicación del procesos en la nube que realiza balanceo de carga, etc.

La característica de este framework[17] es que permite la construcción de aplicaciones distribuidas de manera que el desarrollador tiene la opción de seleccionar algunos de los patrones disponibles de sus componentes de la arquitectura, una desventaja es que se desarrolla solamente en el lenguaje de programación de Java.

2.4.4 Resumen del estado del arte

En la Tabla 2.1 se muestra una comparación entre los mejores trabajos reportados en la literatura relacionados con el trabajo de tesis. Dicha tabla fue creada a partir de las siguientes características parecidas a la propuesta: función de la arquitectura ¿cuál es su objetivo?, escenario de despliegue y

[16] http://projects.spring.io/spring-cloud/
[17] Es un conjunto estandarizado de conceptos, prácticas y criterios

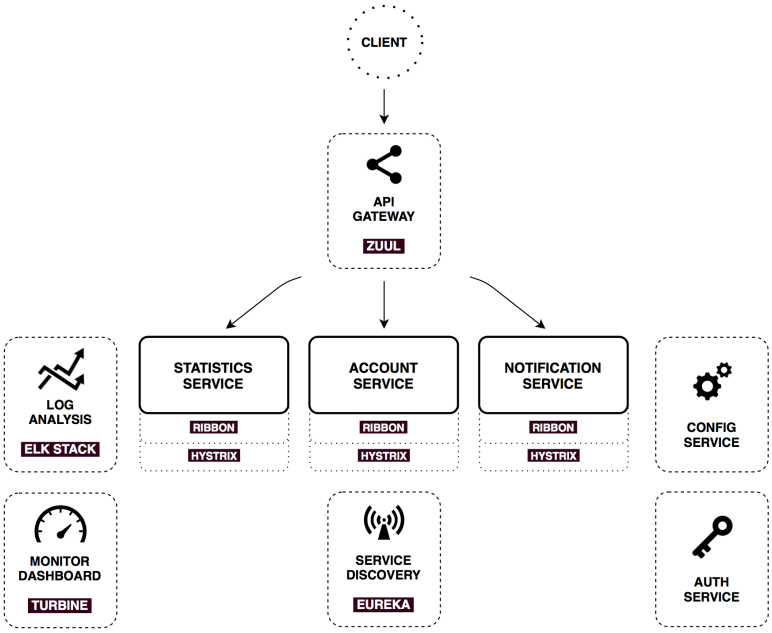


Figura 2.10: Arquitectura de Spring Cloud.

qué enfoque siguen para la construcción de dichas arquitecturas.

Analizando las características de los trabajos reportados en la literatura en la parte de sistemas de compartición de archivos en la nube resaltamos *Globus*, por ser un middleware ampliamente conocido para la construcción de sistemas distribuidos en un entorno de Grid, en el que se incluyen componentes para la compartición y publicación de datos en la nube. Su desventaja radica en la concentración monolítica del software para la gestión de los recursos y datos del usuario. Asimismo, para la compartición de archivos podemos encontrar *SkyCDS*, el cual se enfoca en la entrega de contenidos digitales, asemejando la compartición de archivos siguiendo un patrón de publicación/suscripción. La arquitectura de *SkyCDS* está desarrollada en capas con poca modularidad, lo que hace complicada la agregación y remoción de componentes no funcionales, a pesar de que permite un crecimiento funcional a través de la incorporación de capas.

En la parte de construcción de flujos de trabajo se encuentran los trabajos de *DevOps*, *Spring Cloud*, *Triana* y *Sacbe*. Éstos crecen de manera horizontal que necesitan dos componentes para

Referencia	Función	Despliegue	Enfoque orientado
Compartición de archivos			
Globus [18]	Servicios y aplicaciones distribuidas	La nube y Grid	Servicios
SkyCDS [23]	Entrega de contenidos	La nube	Servicios web
Flujos de trabajo			
Triana [49]	Flujos de trabajo	P2P y Grid	Servicios
DevOps [3]	Aplicaciones de software	La nube	Microservicios
Sacbe [24]	Procesamiento	La nube	Extremo a extremo
Microservicios			
Istio [25]	Integración de microservicios	La nube	Microservicios
Spring Cloud [8]	Aplicaciones distribuidas	La nube	Microservicios

Tabla 2.1: Comparativa del estado del arte

crear flujos de trabajo del inicio hasta la terminación (que indican la ejecución del proceso), donde su desventaja es que dependen de otros servicios y/o aplicaciones para lograr sus funciones de aplicaciones distribuidas y procesamiento.

Por último en la construcción de sistemas de microservicios se encuentra la solución de *Istio* para la integración de microservicios, que es muy similar a nuestra propuesta de solución por la gestión del escalado horizontal de los microservicios. Sin embargo, su desventaja radica en que no provee algún modelo para la construcción de sistemas de compartición de archivos en la nube, aunque podría ser una buena opción para una comparativa con el enfoque en general que se propone en este trabajo de tesis. Nuestra propuesta permite que un sistema escale de manera horizontal a través de componentes no-funcionales (mejorando la calidad del servicio) y de manera vertical, en forma de pila, añadiendo componentes funcionales (aumentando los servicios ofrecidos).

3

Arquitectura de un SCA basada en micro-aplicaciones

En este capítulo se presenta una arquitectura para la construcción de Sistemas de Compartición de Archivos (SCA) en la nube basada en micro-aplicaciones, ofreciendo una mayor flexibilidad a este tipo de sistemas en los procesos de inclusión, remoción y actualización de componentes no-funcionales de manera dinámica. La arquitectura de software se compone principalmente de tres entidades, Unidad de Construcción, Subservicio Virtual y Gestor Global, las cuales funcionan como microservicios encapsulados en contenedores virtuales.

En este capítulo se describe el diseño y desarrollo de la arquitectura modular que se propone, desglosando su contenido de la siguiente manera: en la Sección 3.1 se definen los componentes de la arquitectura modular, la Sección 3.2 muestra el diseño y desarrollo de los aspectos no-funcionales que son construidos con la arquitectura propuesta como parte esencial de un SCA. Por último, en la

Sección 3.3 se explican los pasos a seguir para construir un SCA desde su definición, representación, despliegue, acoplamiento y hasta su puesta en marcha.

3.1 Diseño conceptual

El diseño de la arquitectura modular se encuentra divida en tres entidades: *1)* unidad de construcción (UC), *2)* subservicio virtual (SV) y un *3)* gestor global. En la Figura 3.1 se muestra la integración de las tres entidades principales que conforman la arquitectura.

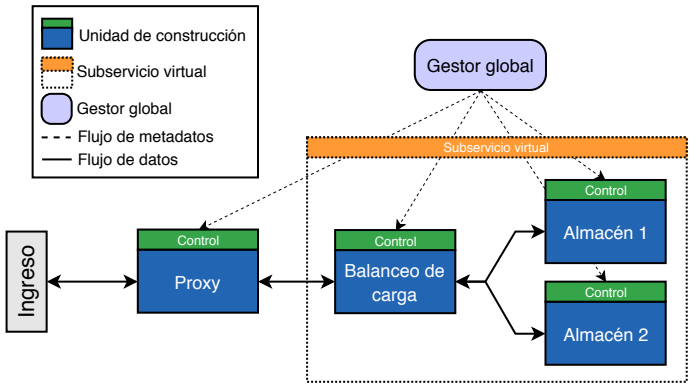


Figura 3.1: Esquema propuesto de la arquitectura modular.

Para entender el funcionamiento y los componentes de la arquitectura propuesta se debe conocer las definiciones de sus componentes:

3.1.1 Unidad de construcción

Es una representación abstracta de un módulo que incluye una capa de control y una de procesamiento, el cual es encapsulado en un contenedor virtual. En la Figura 3.2 se muestra el diseño de la Unidad de Construcción.

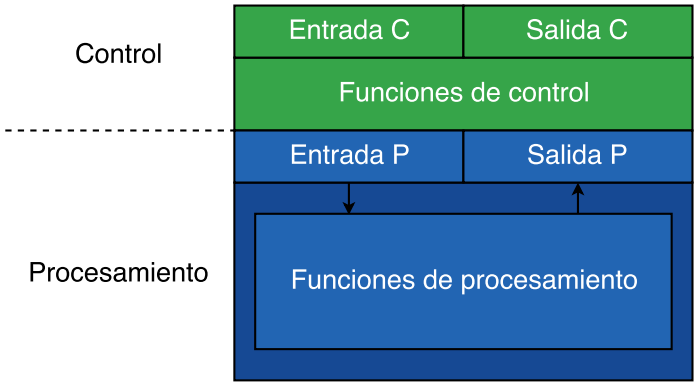


Figura 3.2: Unidad de construcción.

Las capas presentadas en el diseño de la unidad de construcción son descritas a continuación:

- **Control**: es la capa que ofrece las interfaces de E/S para la recepción y transmisión de datos relacionados con la gestión de la UC.
- **Procesamiento**: es la capa por donde se accede al servicio o aplicación encapsulado en la UC.

Las *funciones de control* que forman parte de la **capa de control** son los siguientes módulos:

- **Entrada C**: es la entrada de metadatos hacia la capa de control, para permitir la interacción con otras UC a través de la red.
- **Salida C**: es la salida de la capa de control, que permite el envío de metadatos hacia el *gestor global* a través de la red.
- **Funciones de control**: se encarga de la toma de decisiones con base en un archivo de configuración o en el envío de metadatos por parte del *gestor global* (p. ej. registro/descubrimiento de servicios) realizando las órdenes indicadas.

Las *funciones de procesamiento* forman parte de la **capa de procesamiento** se diseño con los siguientes módulos:

- **Entrada P**: permite la adquisición datos que son enviados a través de interfaces de E/S del servicio o aplicación.
- **Salida P**: se encarga de la entrega de los datos procesados por el servicio o aplicación a otros servicios o aplicaciones a través de interfaces de E/S.
- **Funciones de procesamiento**: permite el encapsulamiento del software del desarrollador donde puede estar uno o más servicios o aplicaciones independientes que forman parte del sistema a desplegar.

Las interfaces E/S de la capa de procesamiento disponibles son las siguientes:

- *Red*: Interfaz de comunicación usada para entrega y recepción de datos a través de una red de computadoras utilizando el modelo de servicio *REST* y *sockets*.
- *Memoria*: Interfaz de comunicación a través de memoria compartida usada para intercambiar datos entre los servicios o aplicaciones colocados en el interior de las funciones de procesamiento.
- *Sistema de archivos*: Interfaz de comunicación usada para leer y escribir datos en el sistema de archivos del sistema operativo anfitrión.

3.1.2 Subservicio virtual

Este componente agrupa a un conjunto de UC, permitiendo la formación de secuencias o estructuras de procesamiento, en función de un estilo arquitectónico que se especifique (p. ej. maestro/esclavo, tubería o flujo de trabajo). Para lograr su gestión mediante la capa de control de cada unidad de construcción se identificaron los siguientes elementos:

- *Componente*: identifica la UC que representa a alguno de los elementos acorde al diseño arquitectónico que se implemente, por ejemplo maestro, esclavo, filtro o etapa.

- *Estilo*: identifica una configuración de una estructura o secuencia (diseño arquitectónico) que forman, un conjunto de UC.

De esta manera, se pretende acoplar UC de manera horizontal, acorde con la secuencia o estructura de procesamiento que se desee, tal y como se muestra en la Figura 3.3. Las estructuras se crean a partir de un archivo de configuración que se incluye en cada UC. En este trabajo se definieron los siguientes tres SV, en el que cada uno representa una estructura diferente para el procesamiento de datos (estilos arquitectónicos):

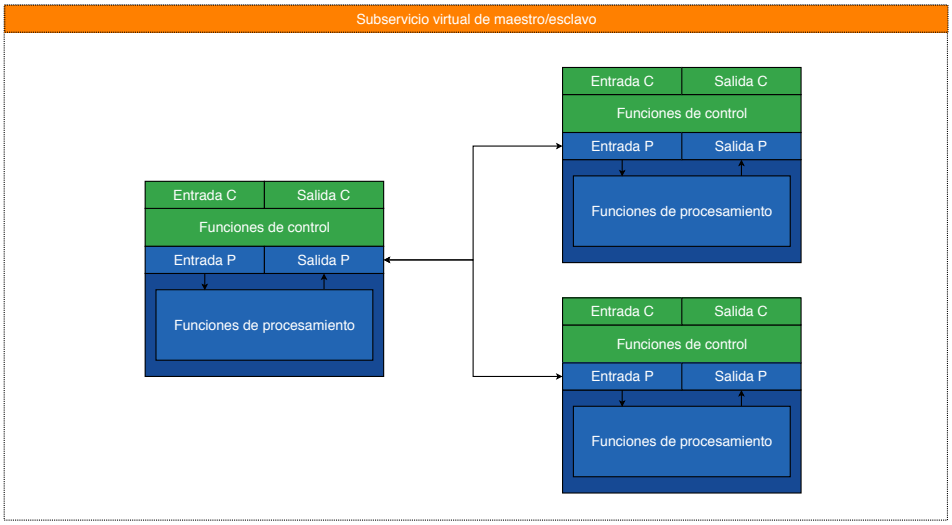
- *Maestro/esclavo*: esta configuración dispone de un componente (maestro) que distribuye trabajo a otros componentes (esclavos) (ver Figura 3.3(a)).

- *Tubería*: en esta configuración se conectan componentes (filtros) a través de conectores que se ejecutan a manera de un flujo para el procesamiento de datos (ver Figura 3.3(b)).

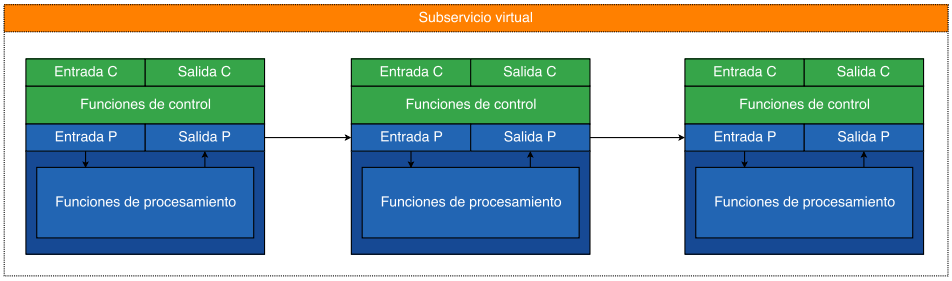
- *Flujo de trabajo*: son estructuras configurables de una actividad de trabajo a través de componentes (etapas) encadenados que realizan tareas específicas donde estas tareas pueden ser una función o servicio (ver Figura 3.3(c)).

3.1.3 Gestor global

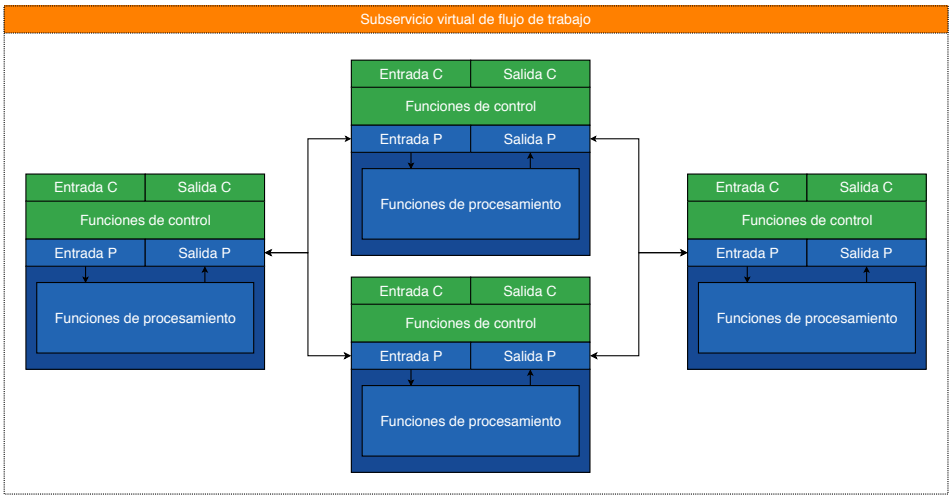
La función de este componente permite construir, gestionar y configurar los elementos de la arquitectura (UC y SV) para la construcción de un SCA en la nube. El gestor global cuenta con los siguientes módulos se muestran en la Figura 3.4 y se describen a continuación:



(a) Maestro/esclavo.



(b) Tubería.



(c) Flujo de trabajo.

Figura 3.3: Subservicio virtual.

- **Cliente**: Este módulo es el encargado de interactuar con el usuario o administrador para llevar a cabo la construcción de un SCA. Sus componentes son:

  - *Front End*: Interfaz que representa la capa de acceso por la que el administrador podrá construir un SCA.
  - *API a Docker Swarm*[1]: permite el lanzamiento de las UC.

- **Gestor**: Módulo que es encargado de llevar el control del SCA, proporcionando un API para que pueda ser utilizado por los demás módulos descritos. Su componente permite la administración de las unidades de construcción, como por ejemplo el envío de metadatos para las configuraciones, actualizaciones y prestaciones agregadas al SCA.

- **Registro**: Su función es llevar el registro de cada UC al momento de ser desplegada. Contiene un componente que es el encargado de llevar el registro y descubrimiento de los servicios que ofrece cada UC.

- **Directorio**: Módulo encargado de almacenar los registros en una base de datos, las UC y SV desplegadas del SCA. Contiene las configuraciones y estados de los UC y SV.

- **Silo**: Este módulo almacena las imágenes de contenedores de las UC, que son:

  - *Microservicios* (*MS*): es un proceso cohesivo e independiente que interactúa a través de mensajes (normalmente una API de recursos HTTP).
  - *Micro-aplicaciones* (*MA*): es un proceso independiente y autónomo que garantiza los datos en tres fases: adquisición, procesamiento y entrega, a través de un conjunto de interfaces de E/S (cualquiera de red, memoria o sistemas de archivos).

[1] Orquestador de contenedores virtuales.

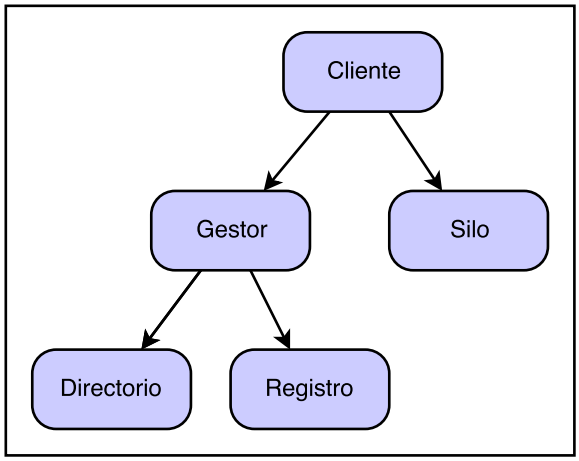


Figura 3.4: Gestor global.

3.2 Componentes no-funcionales para un SCA en la nube

Son aquellos que cumplen los requisitos que aportan atributos de calidad de un sistema [48], algunos son:

- **Rendimiento**: es la capacidad para producir o realizar algo con la cantidad mínima de esfuerzo y/o recursos (efectividad), algunos métodos son *balanceo de carga*, *planificador*, *alta disponibilidad* y *escalabilidad*.

- **Disponibilidad**: es la capacidad de un sistema para ofrecer un servicio ininterrumpido, a pesar de que uno o más componentes fallen (tiempo sin interrupción, tiempo de inactivad), algunos métodos son *tolerancia a fallos*, *resiliencia* y *confiabilidad*.

En este trabajo se desarrollaron tres componentes no-funcionales para un SCA (balanceo de carga, disponibilidad y tolerancia a fallos) con base en la arquitectura propuesta. Estos componentes se integraron de manera experimental en un SCA en la nube, a continuación se describe cada uno de ellos.

3.2.1 Balanceo de carga

Es un método para distribuir la carga de trabajo entre varios equipos de cómputo para lograr una mejora en la utilización de los recursos, para este caso es el uso eficiente del almacenamiento. Se propone construir un SV con un estilo arquitectónico maestro/esclavo, utilizando una UC como maestro y cinco UC como esclavos, la Figura 3.5 ilustra este subservicio. A continuación se describen las UC que componen el SV de balanceo de carga:

- *Balanceo de carga* (Maestro): en esta unidad de construcción se encapsularon balanceadores de carga experimentales como *UF*[2] [40], que es un algoritmo que puede ser aplicado a cargas y descargas de contenidos que utiliza el factor de utilización de los recursos (capacidad de almacenamiento y memoria) y el factor de impacto de los usuarios. También se consideró el uso del algoritmo *TwoChoices* [34] en el que se toma dos nodos al azar y se comparan sus factores de utilización quedándose con el que posea menor.
- *Almacenes* (Esclavos): estas UC se encargan de almacenar los contenidos otorgados por la UC de balanceo de carga.

3.2.2 Disponibilidad

Es el proceso en el que se garantiza que se puede acceder a los datos en cualquier momento, de forma rápida y sencilla, este componente tendrá el enfoque basado en la replicación de archivos. Este componente se construyó siguiendo un estilo arquitectónico de flujo de trabajo (es una automatización de proceso que incluye tareas o etapas), como se observa en la Figura 3.6 con cuatro UC, las UC que los componen son:

- *Proxy* (Etapa inicial): esta UC encapsula un algoritmo que se encarga de redireccionar los contenidos hacia la segunda (*KeyGraph*) y tercera etapa (Clasificador).

[2]Factor de utilización

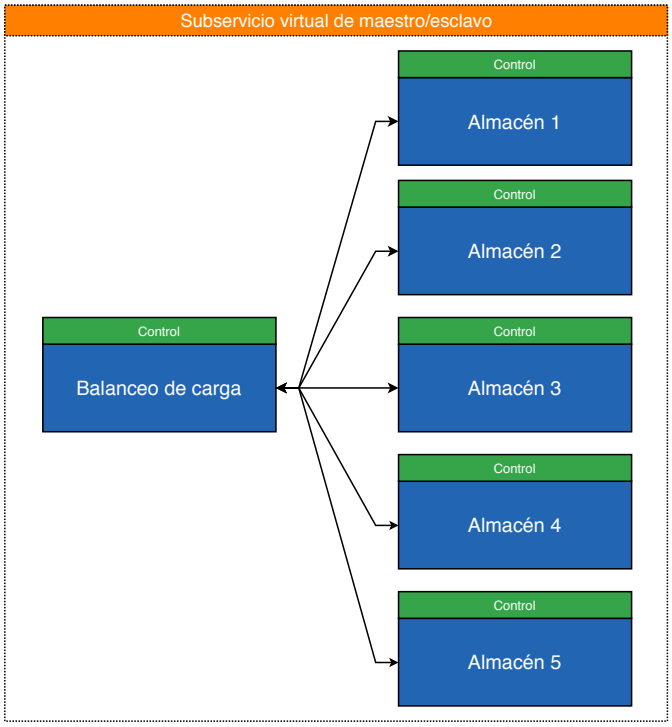


Figura 3.5: SV de balanceo de carga.

- *KeyGraph* [46] (Segunda etapa): la función de esta UC es encargarse de analizar los contenidos (solo texto, resúmenes de PDF o metadatos de archivos) a almacenar, detectar el tema al que se asocian y etiquetarlo.

- *Clasificador* (Tercera etapa): este algoritmo se encarga de clasificar los temas encontrados en función de dos métricas, las cuales son volumen y densidad (ver Subsección 5.2.2) de los contenidos almacenados.

- *Replicación* (Etapa final): la función de este algoritmo se encarga de hacer un número de copias con base a las decisiones dadas por las etapas anteriores (*KeyGraph* y Clasificador) por las etapas anteriores.

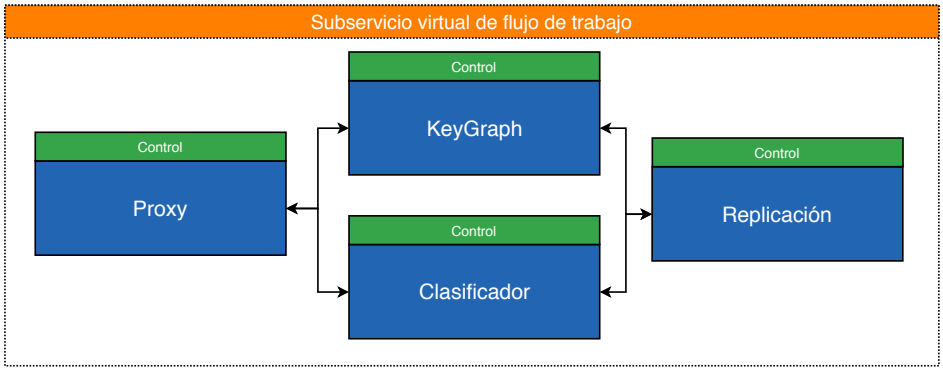


Figura 3.6: SV de disponibilidad.

3.2.3 Tolerancia a fallos

Su objetivo es permitir que un sistema siga dando servicio aun cuando se presente una falla. En este trabajo esta propiedad se aplicó al servicio de almacenamiento de archivos. Para este componente se utilizó el SV de tubería como se muestra en la Figura 3.7. Para este trabajo se encapsularon dos filtros de procesamiento en UC, los cuales son los siguientes:

- *Compresión*: filtro de procesamiento que realiza comprensión de datos utilizando la biblioteca de Zip[3]. En este trabajo utilizaremos el filtro de compresión implementado en Sacbe [24].

- *IDA*[4] [43]: es un algoritmo basado en técnicas de corrección de errores. Que se utiliza para separar paquetes de datos evitando que estos últimos puedan ser entendidos de manera independiente por un lector no deseado. Este algoritmo permite generar n piezas de un archivo y reconstruirlo utilizando cualesquiera de m piezas, donde $m < n$.

[3]Formato de compresión sin pérdida
[4]Algoritmo de Dispersión de Información

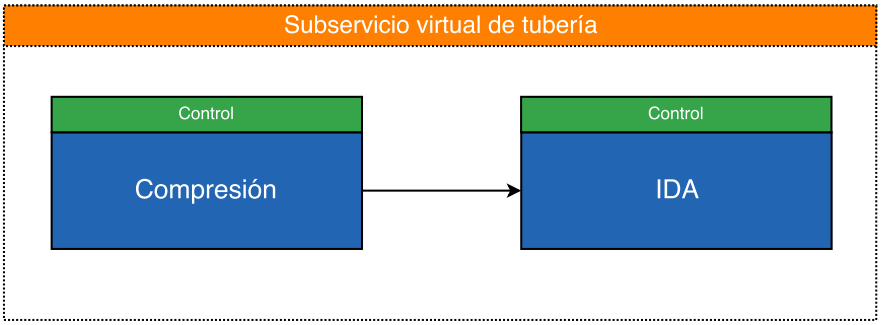


Figura 3.7: SV para la compresión de datos.

3.3 Construcción de un SCA en la nube basado en micro-aplicaciones

Para la construcción, implementación y despliegue de un SCA basado en micro-aplicaciones utilizando la arquitectura propuesta, se proponen las siguientes fases: *1)* Definición, *2)* Representación, *3)* Despliegue, *4)* Acoplamiento y *5)* Operación. Estas fases se describen a continuación.

3.3.1 Fase de definición

En la fase de definición se propuso una nomenclatura que permitirá representar de manera abstracta a los componentes de la arquitectura propuesta, con el fin de generar configuraciones abstractas que faciliten la construcción y despliegue de los componentes que formarán un SCA. A continuación se ofrecen las definiciones de los elementos de la nomenclatura (ver notación en la Tabla 3.1) que se propone:

Tabla 3.1: Notaciones de la nomenclatura

Notación	Descripción
UC	Unidad de construcción.
SV	Subservicio virtual.
SV_{M_E}	SV de procesamiento de maestro/esclavo.
M	Componente maestro de un SV_{M_E}.
E	Componentes esclavos de un SV_{M_E}.
SV_T	SV de procesamiento de tubería.
$F_{inicial}$	Componente de filtro inicial de un SV_T.
F	Componentes de filtros del SV_T
F_{final}	Componente de filtro final de un SV_T.
SV_{FT}	SV de procesamiento de flujo de trabajo.
$S_{inicial}$	Componente de etapa inicial de un SV_{FT}.
S	Componentes de etapas del SV_{FT}
S_{final}	Componente de etapa final de un SV_{FT}.
FM	Flujo de metadatos.
FD	Flujo de datos.

Una unidad de construcción (UC) es una pieza elemental que encapsula un microservicio o micro-aplicación almacenada en un silo que permite la construcción de diferentes estructuras que pueden ser identificadas. Se denota de la siguiente manera:

$$ID = UC_{MS|MA} < replicas > \tag{3.1}$$

Donde ID es el identificador de una unidad de construcción, $UC_{MS|MA}$ representa un microservicio (*MS*) o micro-aplicación (*MA*) encapsulado y $< replicas >$ es el número de unidades de construcción a desplegar.

A continuación se muestran algunos ejemplos de estas UC:

$$A = UC_{registro} < 1 > \tag{3.2}$$

Con esta nomenclatura se hace referencia a una unidad de construcción (UC) del tipo *registro* que contiene un microservicio para la gestión de metadatos (información de los archivos) de un SCA

y que se encuentra almacenada en un silo de imágenes de contenedores. En este ejemplo esta UC se identifica con el nombre de A.

$$B = UC_{publicacion\_subscripcion} < 1 > \tag{3.3}$$

Con esta nomenclatura se hace referencia a una unidad de construcción (UC), identificada con el nombre de B, que implementa un modelo de intercambio de datos basado en el patrón de *publicación/suscripción*. Este patrón permite la interacción entre usuarios que publican archivos en el SCA y usuarios interesados en accederlos (suscripción).

$$C = UC_{carga\_descarga} < 1 > \tag{3.4}$$

La unidad de construcción (UC) identificada con el nombre C, hace referencia a la UC tipo *carga y descarga* que contiene un microservicio encargado de llevar a cabo la carga y descarga de los contenidos que son publicados y accedidos por los usuarios del SCA.

$$D = UC_{proxy} < 1 > \tag{3.5}$$

Con la nomenclatura mencionada se hace referencia a una unidad de construcción (UC) del tipo *proxy*, identificada con el nombre de D, que despliega una micro-aplicación encargado de redireccionar los contenidos hacia otras UC disponibles.

$$E = UC_{almacen} < 3 > \tag{3.6}$$

La unidad de construccion (UC) tipo *almacén* identificada con el nombre de E, que contiene una micro-aplicación que almacena todos los contenidos cargados al SCA. Para este caso, se desplegarán tres réplicas identificadas en el grupo E con los nombres de E_1, E_2 y E_3.

3.3.2 Fase de representación

En la fase de representación se obtienen las interacciones entre las UC donde se definen los SV que permitirán el intercambio de datos. Asimismo, en esta fase se representan los flujo de datos y metadatos que no siguen una estructura específica, si no que se representan a partir de un grafo dirigido, donde los nodos representan UC y las aristas la comunicación o interacción entre ellos, como se muestra en la Figura 3.8.

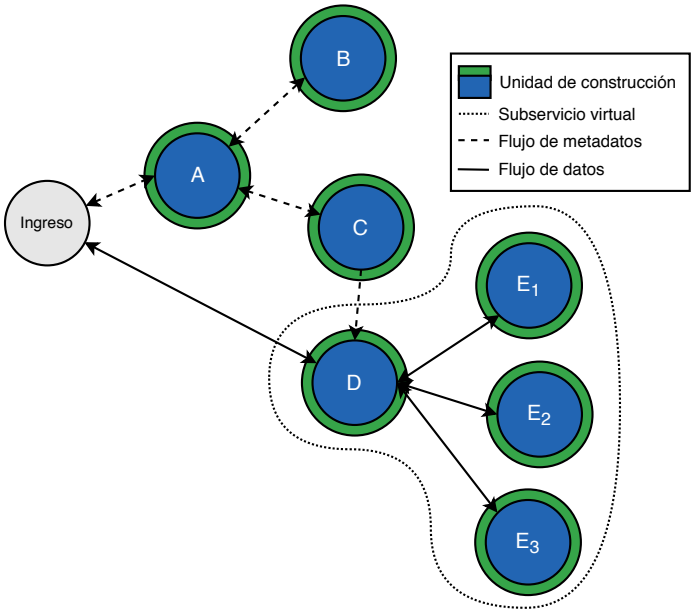


Figura 3.8: Representación abstracta de un SCA.

Un subservicio virtual (SV) es un agrupamiento de unidades de construcción que forman una secuencia o estructura, como ejemplos tenemos:

$$SV_{M_E} = \{M \leftrightarrow (E_1, E_2, ..., E_n)\} \tag{3.7}$$

Donde M_E representa el estilo arquitectónico de maestro/esclavo, M es un componente maestro, $(E_1, E_2, ..., E_n)$ son los componentes esclavos, n es el número de componentes esclavos y $\leftrightarrow$ es la comunicación bidireccional entre los componentes.

$$SV_T = \{F_{inicial} \rightarrow F_n \rightarrow F_{final}\} \tag{3.8}$$

Donde T es el estilo arquitectónico de tubería de procesamiento, $F_{inicial}$ es el componente de filtro inicial, F_n es el componente de filtro, n es el número de componentes filtros, F_{final} es el componente de filtro final y $\rightarrow$ es la comunicación unidireccional entre los componentes.

$$SV_{FT} = \{S_{inicial} \leftrightarrow S_{n-1} \leftrightarrow S_n \leftrightarrow S_{final}\} \tag{3.9}$$

Donde FT refleja el estilo arquitectónico de un flujo de trabajo, $E_{inicial}$ es el componente de etapa inicial, $E_{n-1,n}$ son los componentes de las etapas del flujo, n es el número de componentes etapas, E_{final} es el componente de etapa final y $\leftrightarrow$ es la comunicación bidireccional entre los componentes.

$$FM = \{UC_{n-2} \leftrightarrow UC_{n-1} \leftrightarrow UC_n\} \tag{3.10}$$

Donde FM es el intercambio de mensajes (metadatos) a través de las $UC_{n-2,n-1,n}$.

$$FD = \{UC_n \leftrightarrow SV\} \tag{3.11}$$

Donde FD es el intercambio de datos entre una UC_n que es una unidad de construcción y un SV subservicio virtual.

A continuación muestra el subservicio virtual de procesamiento maestro/esclavo que se usará como caso de estudio.

$$SV_{M_E} = \{D \leftrightarrow (E_1, E_2, E_3)\} \tag{3.12}$$

Se muestran a continuación los flujos de metadatos que formarán nuestro SCA de estudio:

$$FM_1 = \{Ingreso \leftrightarrow A \leftrightarrow B\} \tag{3.13}$$

$$FM_2 = \{Ingreso \leftrightarrow A \leftrightarrow C \rightarrow D\} \tag{3.14}$$

donde FM_1 es el primer flujo de metadatos, $Ingreso$ es una UC tipo *gateway*[5] que inicia el flujo, A y B son las UC tipo *registro* y *publicación y suscripción* y $\leftrightarrow$ es las comunicación bidireccional entre los UC. El FM_2 es el segundo flujo de metadatos entre $Ingreso$, A y C que es tipo *carga y descarga* que se comunican de manera bidireccional ($\leftrightarrow$), D es una UC tipo *proxy* que se comunican de manera unidireccional ($\rightarrow$).

Por último se muestra el flujo de datos que se implementará en nuestro SCA:

$$D = \{Ingreso \leftrightarrow SV_{M_E}\} \tag{3.15}$$

donde FD es el intercambio de datos encargado de transportar los contenidos de $Ingreso$ al SV_{M_E} que es el subservicio virtual de procesamiento de maestro/esclavo.

3.3.3 Fase de despliegue

El despliegue de UCs se realizó a partir de las descripciones de cada uno de los elementos que formarán el SCA que se desea crear. En la arquitectura propuesta se cuenta con un almacén de UC (silo) del que se podrán obtener los diferentes tipos de UC que se requieran para construir el SCA. Cada UC puede ser configurado según las necesidades de las organizaciones en la que se vaya a construir el SCA, pudiendo lanzar más de una UC dependiendo de la configuración dada. A continuación se muestran las UC definidos:

[5] Puerta de enlace

$$A = UC_{registro} < 1 >$$

$$B = UC_{publicacion\_subscripcion} < 1 >$$

$$C = UC_{carga\_descarga} < 1 >$$

$$D = UC_{proxy} < 1 >$$

$$E = UC_{almacen} < 3 >$$

Los identificadores asignados a cada UC son manejados en las siguientes fases para construir el SCA, en la Figura 3.9 se muestra el despliegue de las UC de la etapa de definición.

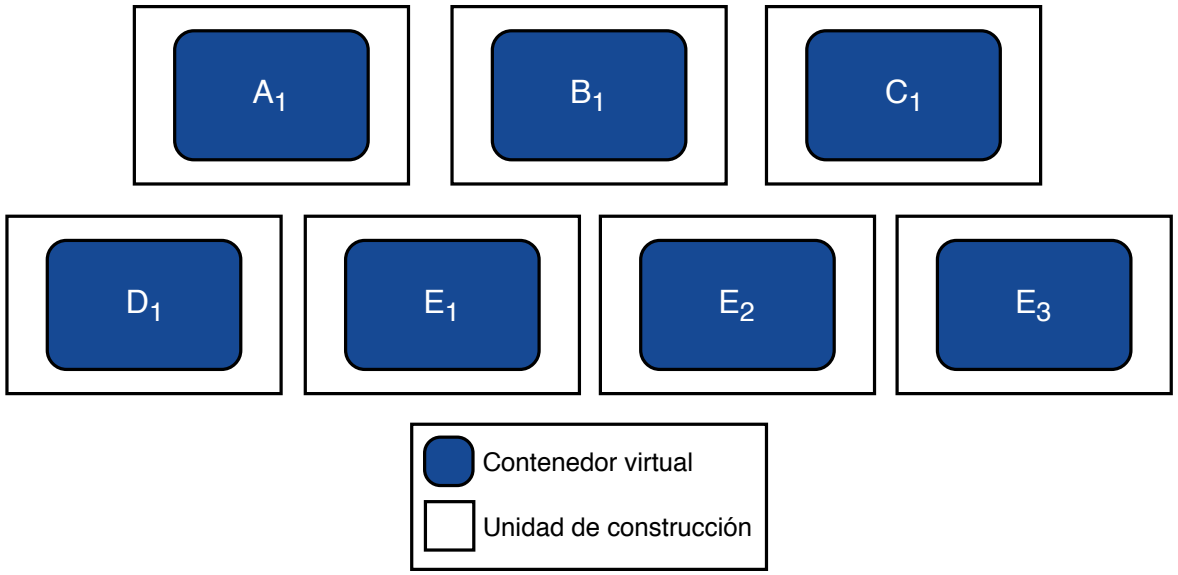


Figura 3.9: Despliegue de UC.

3.3.4 Fase de acoplamiento

La fase de acoplamiento de UC se organiza a partir de la fase de representación. Las estructuras definidas en las fases anteriores son para activar un mecanismo de acoplamiento que se encuentra en la capa de control de cada UC, permitiendo su interacción mediante el *gestor global* a través de

sus configuraciones y registro. En la Figura 3.10 se muestra el acoplamiento de las unidades que son los SV, FM y FD.

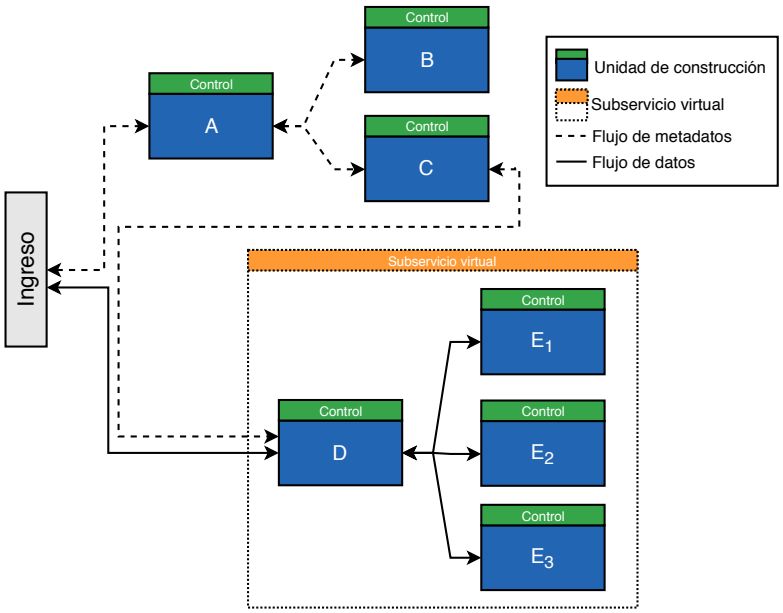


Figura 3.10: Acoplamiento de componentes.

3.3.5 Fase de operación

En la etapa de operación se lleva a cabo la ejecución de los componentes de la arquitectura que fueron definidos, mostrados en la Figura 3.11, que forman parte de un SCA. Se pueden ver las UC asignadas en la fase de definición, y la interacción de los componentes (SV, FM y FD) que fueron seleccionadas en la fase de representación, se observa también el *gestor global* que se encarga de registrar y configurar las UC y SV que fueron definidos en la definición y representación.

Existe una división que conocemos como la plataforma y la infraestructura, que son la capa de gestión y capa de construcción, que a continuación se explican:

- **Gestión**: se presenta una plataforma para definir y gestionar los módulos UC y SV para armar un SCA en la nube. Esta capa sirve para la agregación, remoción o actualización de los componente no-funcionales a un SCA.

- **Construcción**: se presenta la infraestructura para la construcción de un SCA en función de la plataforma, donde se despliega e implementa y opera un SCA en la nube.

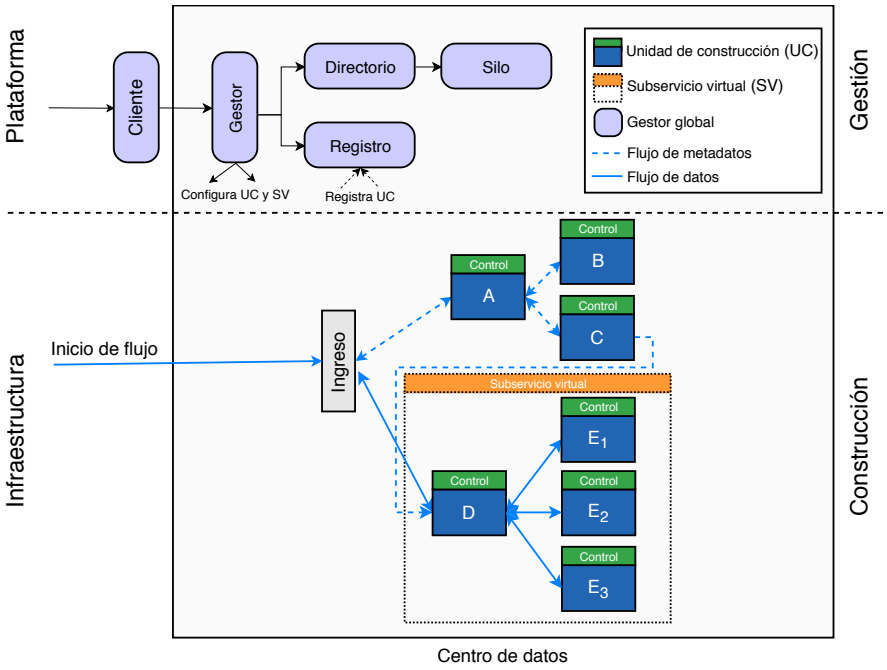


Figura 3.11: Construcción del SCA con la arquitectura propuesta.

4

Evaluación experimental de la arquitectura

En este capítulo se describen los procedimientos llevados a cabo durante la etapa de evaluación de la arquitectura propuesta con el fin de evaluar su desempeño. De igual manera, se describen los elementos necesarios para la creación de los escenarios de prueba planteados, así como las configuraciones dadas a cada escenario. Además, se presentan los resultados obtenidos de cada uno de los experimentos realizados.

Los aspectos considerados en este capítulo se desglosan de la siguiente manera: en la Sección 4.1 se define el proceso de evaluación y el prototipo utilizado, la Sección 4.2 muestra la simulación de carga de trabajo para ser atendidas por un SCA. En la Sección 4.3 se explica un escenario para evaluar el despliegue de los componentes no-funcionales y la Sección 4.4 muestra un escenario para agregar y actualizar aspectos no-funcionales a un SCA.

4.1 Proceso de evaluación

El proceso a seguir para realizar la evaluación experimental de la arquitectura propuesta está dividido en dos etapas. En la primera etapa se realiza una prueba de concepto de los componentes no-funcionales construidos a partir de los componentes de la arquitectura evaluando el desempeño de cada uno de ellos. En la segunda etapa se evalúa el rendimiento del sistema de compartición de archivos en la nube construido a partir del modelo de micro-aplicaciones como un caso de estudio.

4.1.1 Prototipo

El desarrollo del prototipo de la arquitectura fue realizado sobre una infraestructura de nube privada. La Tabla 4.1 muestra la infraestructura para la construcción y despliegue del SCA, se pueden apreciar las características del hardware, mencionando los nombres de las máquinas que componen la infraestructura, la cantidad de núcleos de procesamiento con las que cuenta cada máquina, el tamaño de RAM, el espacio de almacenaje, el software instalado en cada equipo y su respectivo sistema operativo. La plataforma para la definición y gestión de un SCA basado en micro-aplicaciones se describe en la Tabla 4.2, donde se muestra la cantidad de contenedores desplegados, la cantidad de núcleos, RAM y almacenaje resarvado para cada uno y la imagen semilla de contenedor. Por último en la Tabla 4.3 se describe la configuración de un SCA inicial llamado SkyCDS [23] donde se indica el número de contenedores, la cantidad de núcleos, RAM y almacenamiento; y el software instalado en cada uno.

Tabla 4.1: Infraestructura utilizada.

Máquina	CPU	RAM	HDD	Software	SO
disys0	6	12 GB	500 GB	Docker Swarm Leader	CentOS 7
disys1				Docker Swarm	
disys2					
disys3					
disys4					

Tabla 4.2: Contenedores de la plataforma de la arquitectura.

#	Contenedor	CPU	RAM	HDD	Imagen
1	Gestor global	2	2 GB	40 GB	Python
	Directorio	2	2 GB	40 GB	SQLite
	Silo	2	2 GB	40 GB	Registry
	Cliente	2	2 GB	40 GB	Ubuntu 14.04

Tabla 4.3: Prototipo SCA inicial.

#	Contenedor	CPU	RAM	HDD	Software
5	Almacenamiento	2	2 GB	40 GB	Apache2. Php5
1	Metadatos	2	2 GB	40 GB	Apache2, Php5
1	Cliente	2	2 GB	40 GB	Java8

4.1.2 Implementación del prototipo

Los componentes de la plataforma de la arquitectura fueron creados en contenedores virtuales individuales en la infraestructura de nube privada gestionada por un orquestador de contenedores

llamado Docker Swarm. El componente *Gestor global* se encarga de la definición y representación que permite la construcción y configuración de un SCA. El componente de *Directorio* muestra el registro y configuraciones de las Unidades de Construcción y Subservicio Virtuales que formarán parte de un SCA. El componente *Silo* almacena las imágenes de contenedores de las Unidades de Construcción sean microservicios o micro-aplicaciones para obtener las imágenes del despliegue. El *cliente* de la plataforma consiste en un *front end* del *gestor global* y una aplicación de escritorio para llevar a cabo el lanzamiento y despliegue del SCA para que pueda ser utilizado por un administrador. La construcción y despliegue del prototipo inicial del SCA se describe en la Tabla 4.3.

4.2 Simulación de cargas de trabajo sintéticas

La generación de carga de trabajo se realiza mediante un proceso denominado *Lanzador*, que se encarga de producir solicitudes de carga y descarga que emulan el comportamiento de clientes reales. El *Lanzador* se instaló en una instancia del cliente como parte de la infraestructura del proveedor de SkyCDS (Prototipo SCA inicial). El SCA recibe la carga de trabajo del *Lanzador* y que proviene de usuarios reales y válidos. Una carga sintética incluye dos componentes: el primero es una traza que describe las características de las operaciones que deberán ser atendidas por el SCA y el segundo son los archivos involucrados en esas operaciones.

4.2.1 Traza

Una traza es un archivo que incluye un conjunto de tuplas que describen una carga de trabajo en operaciones de entrada y salida, cada tupla incluye la siguiente información que se muestra en la Tabla 4.4.

Tabla 4.4: Características de la traza.

Campos	Descripción
ID_USER	Usuario que envía una operación de E/S (carga/descarga).
TYPE	Tipo de operación (carga o descarga). Este campo se genera a través de una distribución de Pareto [42, 44], donde el 20 % de los usuarios cargan archivos que son consumidos por el 80 % de usuarios restantes.
ID_FILE	Identificador del archivo.
ARRIVAL	Hora de llegada. Sigue una distribución de Poisson [32].
FILE_SIZE	Tamaño del archivo para cargar/descargar. Sigue una distribución de Poisson [32].
ID_USER2	En las operaciones de descarga, este campo identifica al usuario que ha cargado el contenido solicitado.

4.2.2 Generador de trazas

Las cargas de trabajo con las que se trabajó en la evaluación del SCA, construido con la arquitectura propuesta, fueron creadas a partir de un generador de trazas sintéticas desarrollado en C++, el cual recibe como entrada el total de las operaciones de la traza (tuplas), el tamaño promedio de los archivos (5 MB) y el tiempo promedio de arribo de las peticiones, por ejemplo 5 peticiones por cada segundo. El generador toma los parámetros mencionados, clasifica la traza siguiendo las distribuciones asignadas para generar un archivo de la traza de interacciones. El generador permite crear cargas de trabajo que se adapten a los escenarios de prueba, de manera que el prototipo del SCA de este trabajo resulte genérico para simular la carga y descarga de archivos de usuarios de reales.

4.2.3 Generador de archivos

El generador de archivos fue desarrollado en Java y se utilizó para crear archivos ficticios con los cuales se realizó la evaluación experimental del SCA. El contenido de cada archivo generado incluye una combinación de caracteres, letras y números elegidos en forma aleatoria. Este generador recibe un archivo que contiene una lista de contenidos a crear los cuales cumplen con tres parámetros que son el nombre y tamaño del archivo así como la ruta en la cual debe colocarlos (una carpeta, directorio, etc.).

4.3 Evaluación de los componentes no-funcionales de un SCA en la nube

En esta sección se describen el escenario de experimentación, las métricas y las configuraciones de los componentes no-funcionales experimentales. También se realiza un análisis de los resultados obtenidos para identificar el rendimiento de los componentes no-funcionales en cada uno de sus configuraciones.

4.3.1 Componentes no-funcionales

En el prototipo desarrollado se integran componentes de software por cada uno de los aspectos no-funcionales considerados como alcance de esta tesis. Estos componentes se abstraen en Unidades de Construcción encapsuladas en contenedores virtuales. Como se explican y definen en la Sección 3.2, los aspectos no-funcionales que se consideran como parte del alcance de esta tesis son:

- *Balanceo de carga*.
- *Disponibilidad*.
- *Tolerancia a fallos*.

A continuación se describe brevemente la implementación de cada uno de estos aspectos no-funcionales.

Balanceo de carga

Para la implementación de este aspecto no funcional se consideró el uso de los algoritmos *UF* [40] y *TwoChoices* [34] encapsulados en las UC. A continuación se describen las fases de definición y representación en la parte de la plataforma de la arquitectura para construir un SCA que contenga el componente de balanceo de carga, para llevar a cabo un intercambio de balanceadores de carga.

Fase de definición

En esta fase se utiliza una nomenclatura para representar las Unidades de Construcción que están almacenadas en el componente del *Silo* y con las que se llevará a cabo el despliegue. A continuación se describen las Unidades de Construcción a utilizar para construir el SCA. Para representar el prototipo inicial del SCA se utilizaron dos UC, una UC para el microservicio de *metadatos* y otra UC para integrar una micro-aplicación de *almacén* donde se decide utilizar cinco copias de esta UC. Para agregar e intercambiar el componente de balanceo de carga se utilizaron dos balanceadores de carga en sus respectivas UC.

$$A = UC_{metadatos} < 1 >$$

$$B = UC_{almacen} < 5 >$$

$$C = UC_{uf} < 1 >$$

$$D = UC_{twochoices} < 1 >$$

Fase de representación

En esta fase se construyen las secuencias de las UC para formar estructuras llamadas Subservicio Virtual, también los flujos de metadatos y los flujos de datos que serán parte del SCA a construir.

Para este aspecto no-funcional se utilizó el SV_{M_E} para la agregación del balanceador de carga. Tomando en cuenta la notación que se explicó en las secciones anteriores, a continuación se indican los subservicios virtuales y flujos creados para nuestro componente de balanceo de carga. Subservicio de Maestro/Esclavo: $SV_{M_E} = \{C \leftrightarrow (B_1, B_2, B_3, B_4, B_5)\}$, Flujo de Metadatos: $FM = \{Ingreso \leftrightarrow A\}$ y Flujo de Datos: $FD = \{Ingreso \leftrightarrow SV_{M_E}\}$. De esta manera se crea un archivo de configuración para llevar a cabo el despliegue y la construcción del SCA en la infraestructura de nube privada. Asimismo se llevan a cabo las fases de despliegue y acoplamiento, con la interacción del gestor global, para finalmente pasar a la fase de operación del SCA.

Disponibilidad

Para este componente se implementaron técnicas de redundancia basadas en replicación y en dispersión de información. A continuación se describen las fases de definición y representación para este aspecto no-funcional.

Fase de definición

En esta fase se van a definir nuevas UC para hacer uso del componente de disponibilidad, utilizando el prototipo inicial del SCA declarado en el anterior componente. Se hace uso de las UC mencionadas en la Subsección 3.2.2 en el encapsulamiento de los microservicios y/o micro-aplicaciones que formarán dicho componente. Se declaran cuatro UC con una sola copia que es utilizada en la parte de despliegue.

$$E = UC_{proxy} < 1 >$$

$$F = UC_{keygraph} < 1 >$$

$$G = UC_{clasificador} < 1 >$$

$$H = UC_{replicacion} < 1 >$$

Fase de representación

En la fase de representación del componente de disponibilidad del SCA se utilizó el SV_{FT}, donde cada componente forma una etapa encargada de desarrollar una tarea que se encadena a otra. Esta representación se anexa a la configuración del SCA para el aspecto no-funcional del balanceo de carga. A continuación se denotan las representaciones usadas para el componente de Disponibilidad: Subservicio virtual de flujo de trabajo: $SV_{FT} = \{E \leftrightarrow F \leftrightarrow G \leftrightarrow H\}$, donde las $E = UC_{proxy}$, $F = UC_{keygraph}$, $G = UC_{clasificador}$, $H = UC_{replicacion}$ se comunican de manera bidireccional para cumplir el trabajo de su estructura. Flujo de metadatos: $FM = \{Ingreso \leftrightarrow A\}$, Flujo de datos: $FD = \{Ingreso \leftrightarrow SV_{FT} \leftrightarrow SV_{M_E}\}$.

Tolerancia a fallos

Para este componente se implementó un algoritmo de dispersión de información (IDA, por sus siglas en inglés), el cual fue encapsulado en una UC, también se utilizó una micro-aplicación de un compresor de archivos que fue encapsulada en otra UC con una sola instancia para su despliegue, para ser agregado al SCA y de esta manera obtener dicho aspecto no-funcional.

$$I = UC_{compresion} < 1 >$$

$$J = UC_{ida} < 1 >$$

Fase de representación

Para representar el aspecto no-funcional de tolerancia a fallos, se utilizó un subservicio virtual SV_T, donde cada componente es llamado filtros para ejecutar un flujo para el procesamiento de datos, con las UC seleccionadas en la fase de definición. Los subservicios y flujos que conforman este componente son los siguientes: Subservicio Virtual de Tubería: $SV_T = \{H \rightarrow I\}$, Flujo de metadatos: $FM = \{Ingreso \leftrightarrow A\}$, Flujo de Datos: $FD = \{Ingreso \leftrightarrow SV_T \leftrightarrow SV_{FT} \leftrightarrow SV_{M_E}\}$.

La definición y construcción del SCA se llevó de manera fuera de línea donde el administrador del SCA lo planifica, construye y despliega en una infraestructura de nube privada.

A continuación se muestra la métrica empleada para evaluar este escenario experimental:

- **Tiempo de despliegue**: esta métrica representa la cantidad de tiempo transcurrido desde que los elementos de la arquitectura (unidades de construcción y subservicios virtuales) son desplegados hasta su puesta en marcha para su ejecución.

4.3.2 Resultados

En esta sección se describen los resultados de la evaluación de los componentes no-funcionales construidos con la arquitectura propuesta en términos de tiempo de despliegue. Intuitivamente se espera que se produzcan tiempos de despliegue competitivos para mejorar el servicio a los usuarios en cada componente no-funcional evaluado. Fue evaluada la construcción del SCA con cada componente añadido, en sus diferentes configuraciones.

Tiempo de despliegue

La Figura 4.1 muestra en la vertical tiempo de despliegue en segundos de un SCA con cada componente no-funcional descrito anteriormente (eje horizontal). Los tiempos de despliegue aumentan en función del número de UC y SV elegidas al construir un SCA, las UC y SV fueron planificadas en las fases de definición y representación de cada componente no-funcional, donde al SCA se le fue agregando un componente no-funcional llevando a cabo el despliegue total del SCA en cada lanzamiento.

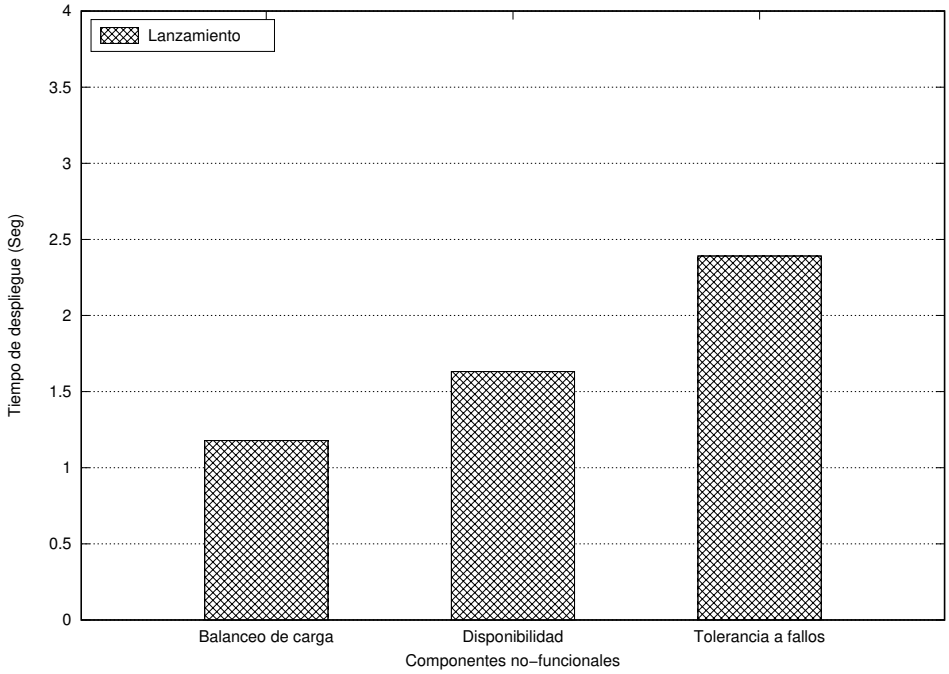


Figura 4.1: Tiempo de despliegue de cada componente no-funcional.

4.4 Evaluación de un SCA en la nube con la arquitectura propuesta

En esta sección se describen las configuraciones, el escenario de experimentación, las métricas, y las configuraciones de un SCA en la nube construido con la arquitectura propuesta. Para realizar la evaluación del SCA se llevó a cabo una experimentación basada en cargas de trabajo sintéticas para la simulación de usuarios reales y válidos para llevar a cabo a un caso de estudio.

4.4.1 Configuración

Servicio de Distribución y Almacenamiento de Archivos: Se implementó un servicio de distribución y almacenamiento de archivos basado en la arquitectura de *SkyCDS* [23]. El prototipo la arquitectura de *SkyCDS* se rediseña siguiendo el modelo basado en Unidades de Construcción

propuesto en esta tesis. De esta manera será posible que *SkyCDS* pueda agregar, actualizar y remover módulos de manera dinámica, logrando así integrar los aspectos no-funcionales de una manera flexible.

En el escenario anterior se definieron los aspectos no-funcionales de un SCA, en este escenario se rediseña la arquitectura de *SkyCDS* [23] con el modelo basado en Unidades de Construcción de manera flexible.

4.4.2 Métricas

A continuación se listan las métricas empleadas para este escenario:

- **Tiempo de acoplamiento**: esta métrica representa la cantidad de tiempo transcurrido entre el cambio o agregación de un componente no-funcional (subservicio virtual) hasta su puesta en marcha.
- **Tiempo de respuesta**: esta métrica representa la cantidad de tiempo transcurrido desde que un usuario realiza una petición hasta que se le entrega una respuesta.

4.4.3 Resultados

Como parte de la experimentación del segundo escenario se evaluó un SCA construido con la arquitectura propuesta y los resultados en términos de tiempo de acoplamiento y tiempo de respuesta en cada una de las etapas de actualización, agregación o remoción de los componentes no-funcionales evaluados anteriormente, mediante el seguimiento de una traza de interacción para la simulación de usuarios utilizando el SCA.

Tiempo de acoplamiento

La Figura 4.2 muestra como los componentes no-funcionales que fueron actualizados o agregados de manera dinámica al SCA construido con la arquitectura. El tiempo de acoplamiento en segundos

se utilizó (eje vertical) para medir el tiempo transcurrido entre las tareas definidas que son cambio de balanceo de carga, agregación de disponibilidad y agregación de tolerancia a fallos. Es de destacar que el tiempo entre de cada tarea es favorable y aumenta por el numero de UC que agrupa un SV.

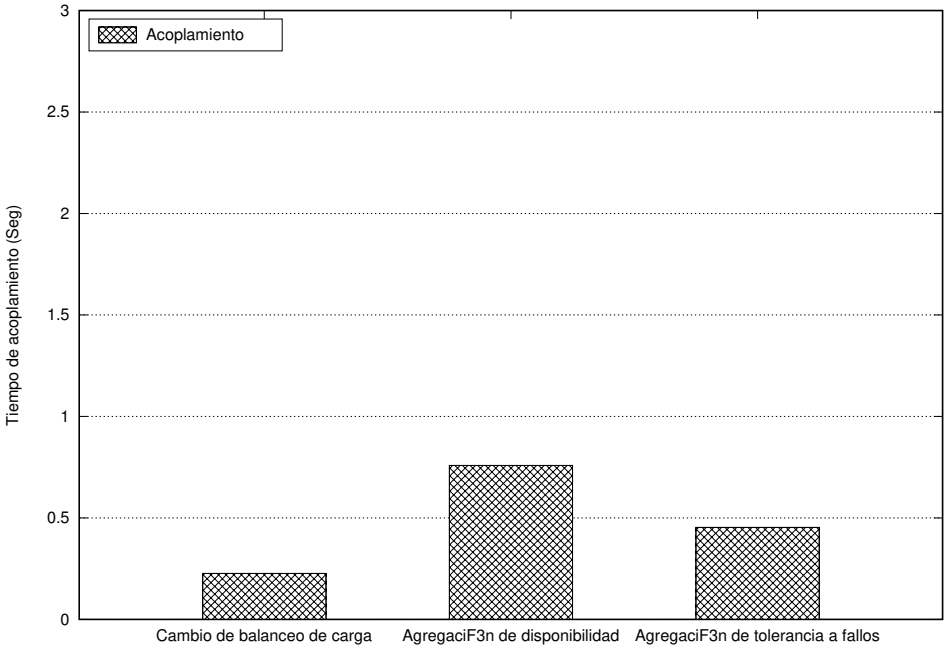


Figura 4.2: Tiempo de acoplamiento de los componentes no-funcionales al SCA.

Tiempo de respuesta

La Figura 4.3 muestra la media del tiempo de respuesta del SCA construido en las diferentes versiones que son *1)* prototipo inicial, *2)* cambio de balanceo de carga, *3)* agregación de disponibilidad y *4)* agregación de tolerancia a fallos. Considerando las operaciones de carga y descarga por separado y en conjunto (Global). Podemos ver que en cada versión mejora el SCA al actualizar o agregar un componente no-funcional. Esto demuestra que los componentes no-funcionales mejoran el servicio de un SCA que se ve reflejado en la experiencia de los usuarios.

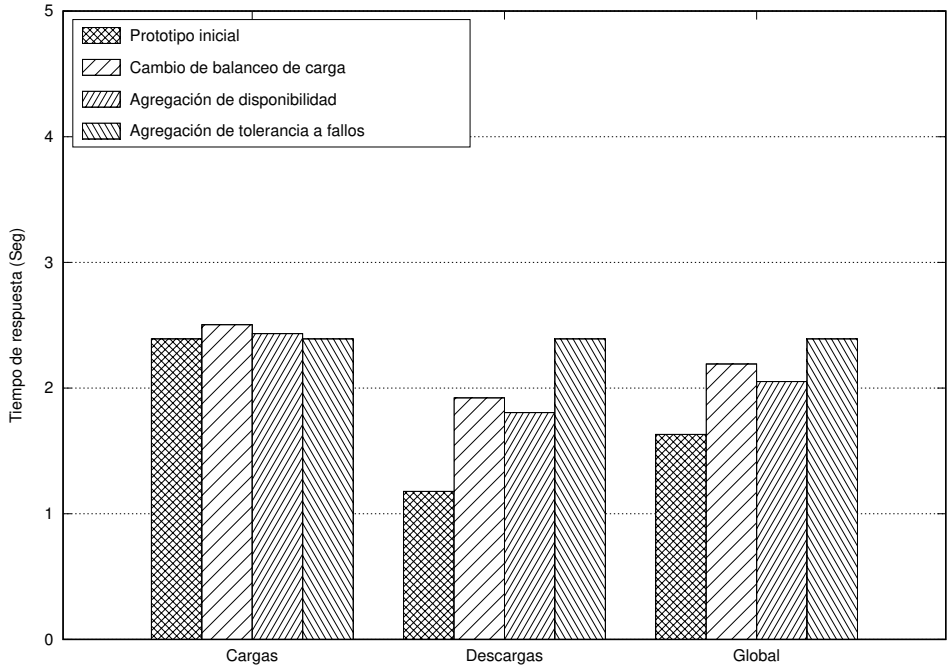


Figura 4.3: Tiempo de respuesta del SCA.

4.5 Resumen

En este capítulo presentamos una evaluación experimental de los componentes que conforman la arquitectura basada en Unidades de Construcción, en la que se encapsulan componentes de software para ofrecer aspectos no-funcionales a un Servicio de Compartición de Archivos (SCA). Se experimenta el funcionamiento de cada componente y los tiempos que se obtienen en cuestiones claves, como el acoplamiento de componentes, el tiempo de puesta en marcha y los tiempos de respuesta del componente. En el siguiente capítulo se presenta un caso de estudio utilizando esta arquitectura en el que se evalúan nuevos algoritmos para aspectos no funcionales que pueden ser integrados de manera dinámica a un SCA.

5

Caso de estudio: Implementación de un SCA aplicando nuevos algoritmos para mejorar sus aspectos no-funcionales

En este capítulo se presenta un resumen sobre el desarrollo y experimentación con un SCA en el que se integran nuevos algoritmos para mejorar algunos de sus aspectos no-funcionales. Se muestran nuevas estrategias de replicación y distribución de documentos en el que se toma en cuenta la popularidad del tema de los contenidos y la popularidad del proveedor de los mismos.

En este capítulo se describe un resumen sobre los algoritmos experimentales para la mejora de un SCA, desglosando su contenido de la siguiente manera: en la Sección 5.1 y 5.2 se muestra una estrategia de replicación y distribución de documentos con base en los proveedores y contenidos de un SCA, la Sección 5.3 muestra un escenario de un SCA con los aspectos no-funcionales para

mejorar su servicio a los usuarios. Por último, en la Sección 5.4 se muestran los resultados de la experimentación llevada a cabo.

5.1 Replicación de datos basado en proveedores de contenidos

Los Servicios de Compartición de Archivos (SCAs) tienen algunas similitudes con las redes sociales. En ambos tipos de sistemas los usuarios adoptan diferentes papeles, tales como el de Productor (usuarios que cargan contenidos en el sistema), Consumidor (usuarios que descargan los contenidos), y una combinación de ambos. Los contenidos de mayor atención para el sistema son aquellos cargados por un Proveedor que han sido al menos descargados una vez por algún Consumidor. A los proveedores de este tipo de contenidos los definimos como grupo S. Para este trabajo, los usuarios del sistema que pertenecen al grupo S se clasifican de acuerdo a la popularidad que tienen el contenido que producen. La siguiente sección describe brevemente un esquema de clasificación que se propone para estos proveedores.

5.1.1 Clases de proveedores de contenidos

La clasificación de los proveedores de contenido (S) en un SCA se hace a partir de dos métricas: Volumen y Densidad. En este trabajo, Volumen (V) representa la actividad que tiene el almacenamiento en el SCA cuando responde a peticiones de descarga durante un periodo de tiempo definido. Densidad (D) refiere al tiempo promedio de llegada de las peticiones de descargar durante una carga de trabajo.

El cálculo de estas métricas (Volumen y Densidad) nos permite clasificar a los proveedores (S) de acuerdo a la popularidad del contenido que proveen al SCA. En este trabajo hemos clasificado a los usuarios del conjunto S en tres tipos diferentes (ver notación en la Tabla 5.1):

1. **Proveedores de Contenido Gamma ($S\gamma$)**: Un proveedor S_i se convierte en proveedor Gamma si $V_{S_i} \leq AV_S$, es decir, el volumen de bytes servidos, en peticiones de descarga, hacia contenidos proporcionados por S_i es menor o igual al promedio del volumen de bytes servido durante la carga de trabajo. Los proveedores cuyo contenido no recibe alguna descarga también serán considerados de este tipo.

2. **Proveedores de Contenido Beta ($S\beta$)**: Un proveedor de contenido S_i se convierte en proveedor Beta si $V_{S_i} > AV_S$.

3. **Proveedor de Contenido Alfa ($P\alpha$)**: Un proveedor de contenido S_i se convierte en Alfa si S_i es un proveedor Beta y $D_{S_i} < D_S$, es decir, el tiempo promedio de llegada de las peticiones de descarga (Densidad) por contenidos cargados por el proveedor S_i es menor al tiempo promedio de llegada de las peticiones de descarga de cualquier contenido.

Esta clasificación es utilizada para ajustar el factor de replicación (RF) del contenido cargado por el proveedor S_i, buscando que el SCA reduzca el consumo innecesario de recursos de almacenamiento, promoviendo el balanceo de carga y mejorando el tiempo de respuesta al servir contenido considerado popular.

5.1.2 Cálculo de densidad y volumen

La clasificación de los proveedores de contenidos es un proceso que se ejecuta en ciertos periodos de tiempo en el SCA. Para simplificar la explicación de las fórmulas, usaremos la notación mostrada en la Tabla 5.1. Como mencionamos antes, Densidad (D_S) refiere el tiempo promedio de llegadas de peticiones de descarga para contenidos producidos por todos los proveedores (conjunto S). La Densidad de las peticiones de descarga por el contenido proporcionado por un proveedor específico (S_i) es denotado por D_{S_i}. A continuación, se describen las fórmulas para calcular el volumen.

Tabla 5.1: Notaciones

Notación	Descripción
U	Conjunto de usuarios registrados en el SCA.
$\|U\|$	Número de usuarios registrados en el SCA.
S	Conjunto de usuarios proveedores de contenido.
$\|S\|$	Número de proveedores.
S_i	Un proveedor del conjunto $S(S_i \in S, 1 <= i <= \|S\|)$.
F	Conjunto de contenidos o archivos almacenados en el SCA.
$\|F\|$	Número de contenidos almacenados en el SCA.
F_j	Un contenido del conjunto $F(F_j \in F, 1 <= j <= \|F\|)$.
$\|F_j\|$	Tamaño del contenido F_j.
F_{S_i}	Conjunto de contenidos que fueron cargador por el proveedor S_i.
$\|F_{S_i}\|$	Número de contenidos cargados por el proveedor S_i.
ND_{F_j}	Número de veces que F_j fue descargado.
F_{jS_i}	Contenido del conjunto (F_j) que fue cargado por el proveedor S_i.
$\|F_{jS_i}\|$	Tamaño del contenido (F_j) que fue cargado por S_i.
$ND_{F_jS_i}$	Número de veces que F_{jS_i}que fue descargado.
W	Carga de trabajo (trazas): Conjunto de peticiones de cargas/descargas.
WD	Conjunto de peticiones de descargas, incluido en W.
WD_{S_i}	Conjunto de peticiones de descargas en W para el contenido cargado por el proveedor S_i.
$\|WD_{S_i}\|$	Número de peticiones de descarga para el contenido cargado por S_i.
V_S	Volumen total de bytes servidos en peticiones de descarga.
AV_S	V_S promedio.
V_{S_i}	Volumen de bytes servido de contenidos que fueron cargados por S_i.
AV_{S_i}	Volumen promedio de bytes servido de contenidos de S_i.
D_S	Tiempo promedio de llegada de peticiones de descarga (Densidad).
D_{S_i}	Tiempo promedio de peticiones de descarga para contenidos del proveedor S_i.
CT	Conjunto de temas encontrado en el conjunto de datos.
$\|CT\|$	Número de temas encontrados en el conjunto de datos.
CT_i	Un tema del conjunto $CT(CT_i \in CT, 1 <= i <= \|CT\|)$.
F_jCT_i	Archivo j con el tema CT_i.
V_{CT}	Volumen total de bytes servido en peticiones de descarga.
AV_{CT}	Volumen promedio de bytes servidos en peticiones de descarga.

V_{CT_i}	Volumen de bytes descargados de contenido con tema CT_i.
AV_{CT_i}	Volumen promedio de bytes descargados con el tema CT_i.
D_{CT}	Tiempo promedio de llegada de peticiones de descarga (Densidad).
D_{CT_i}	Tiempo promedio de llegada de peticiones de descarga para el contenido con tema CT_i.

El volumen de un proveedor S_i (V_{S_i}) representa el consumo de almacenamiento del SCA al servir peticiones de descarga para el contenido producido por S_i y se define como:

$$V_{S_i} = \sum_{j=1}^{|F_{S_i}|} \sum_{1}^{ND_{F_j S_i}} |F_j S_i| \tag{5.1}$$

donde $|F_{S_i}|$ es el número de archivos cargados por el proveedor S_i, $|F_{j_{S_i}}|$ representa el tamaño de archivo específico, $F_j (1 \leq j \leq |F_{S_i}|)$, cargado por el proveedor S_i, y $ND_{F_j S_i}$ es el número de descargas hechas al archivo $F_{j_{S_i}}$. La transferencia de bytes o volumen (V_S) del SCA cuando sirve el conjunto completo de peticiones de descarga es dado por:

$$V_S = \sum_{i=1}^{|S|} VS_i \tag{5.2}$$

Donde $|S|$ es el número de proveedores cuyo contenido fue descargado al menos una vez.

La transferencia promedio requerida por las peticiones de descarga hechas al SCA es definida por:

$$AV_S = \frac{V_S}{|S|} \tag{5.3}$$

5.1.3 Proceso de clasificación

El proceso de clasificación de un conjunto de proveedores (S) inicia una vez que el Volumen y Densidad de sus contenidos han sido calculados. Los proveedores Gamma ($S\gamma$) son aquellos cuyo volumen es menor o igual al volumen promedio generado en el SCA. Estos proveedores se definen como: $S\gamma = \{x | x \in S \wedge Vx \leq AV_S\}$.

Un proveedor se convierte en Beta ($S\beta$) cuando la siguiente restricción se cumple: $S\beta = \{x|x \in S \wedge Vx > AV_S\}$. Finalmente, un proveedor de contenido es tipo Alfa si la siguiente condición se cumple: $S\alpha = \{x|x \in S\beta \wedge Dx < D_S\}$.

Los proveedores Alfa ($S\alpha$) se obtienen a partir de los proveedores Beta ($S\beta$), reduciendo el espacio de búsqueda y procesamiento. Para calcular el tiempo promedio de llegada de las peticiones de descarga (o Densidad D) es necesario obtener la Densidad de cada proveedor Beta ($D_{S\beta_i}$) como sigue:

$$D_{S\beta_i} = \frac{\sum_{j=2}^{|WDS\beta_i|}(DownloadArrivalTimeReq_j - DownloadArrivalTimeReq_{j-1})}{|WDS\beta_i|} \tag{5.4}$$

Donde $D_{S\beta_i}$ representa la densidad de un proveedor Beta ($S\beta_i$), $|WDS\beta_i|$ es el número de peticiones de descarga que el SCA ha recibido por contenido de proveedores $S\beta_i$, y $DownloadArrivalTimeReq_j$ es el tiempo de llegada de una petición de descarga j por contenido del proveedor $S\beta_i$. Una vez obtenida la Densidad de cada proveedor Beta, aplicamos la siguiente fórmula para calcular la densidad del conjunto de proveedores Beta ($D_{S\beta}$):

$$D_{S\beta} = \frac{\sum_{i=1}^{S\beta} DS\beta_i}{|S\beta|} \tag{5.5}$$

Donde $|S\beta|$ es el número de proveedores Beta.

Los valores de Densidad de cada proveedor Beta ($D_{S\beta_i}$) y del conjunto Beta completo (($D_{S\beta}$) son utilizados para detectar a los proveedores de tipo Alfa de la siguiente manera: $S\alpha = \{x|x \in S\beta \wedge D_{S\beta_x} < D_{S\beta}\}$. Recordemos que $D_{S\beta_x}$ y $D_{S\beta}$ representan el tiempo promedio de llegada de las peticiones de descarga de contenido de un proveedor Beta específico y para todo los contenidos pertenecientes a los proveedores Beta respectivamente.

5.1.4 Factor de replicación

La clasificación de los proveedores de contenido permite al SCA adaptar el factor de replicación (RF) con base en la popularidad del contenido (Densidad y Volumen).

FR indica la cantidad de copias que el sistema de almacenamiento debe generar para cada contenido cargado para distribuirlas en los n nodos disponibles en el grupo de almacenamiento. Como valor de RF inicial, hemos considerado la referencia de buenas prácticas con $FR = 3$ (tres copias) en una piscina de $n = 5$ nodos de almacenamiento. FR se ajustará según la clasificación de los proveedores de contenido. Los valores RF para los diferentes tipos de proveedores de contenido son: $RF\_S\gamma = 1$ (proveedores gamma), $2 \leq RF\_S\gamma \leq 3$ (proveedores beta) y $4 \leq RF\_S\alpha \leq n$ (provedores alfa). En los proveedores de alfa y beta, el valor final de RF se calcula mediante un ajuste de parámetros, llevando a cabo un conjunto de experimentos que comparan el impacto en el servicio del sistema de almacenamiento al usar valores de RF ajustados frente a usar buenas prácticas de valores RF.

Diferentes RFs producen diferentes impactos en los recursos de almacenamiento determinados por el tipo de proveedor de contenido (alfa, beta y gamma) y la cantidad de nodos en la piscina de almacenamiento. Para evaluar el impacto del proveedor de contenido (SI) en un servicio de almacenamiento, hemos asignado un valor a SI que está en el rango de $(0, 1]$, de acuerdo con la siguiente ecuación:

$$SI = \frac{RF}{n} \tag{5.6}$$

Donde RF es el factor de replicación para un proveedor de contenido y n es la cantidad de nodos en la piscina de almacenamiento. Por ejemplo, para configuraciones $RF\_S\gamma = 1$, $2 \leq RF\_S\beta \leq 3$ y $4 \leq RF\_S\alpha \leq n$, en un grupo de 5 nodos, proveedores de gamma tendría $SI = 0.2$, proveedores beta $SI = 0.4$ o $SI = 0.6$ y proveedores alfa $SI = 0.8$ o $SI = 1$. Estos valores representan la porción de recursos de almacenamiento (en términos de nodos de almacenamiento) utilizados para

ofrecer disponibilidad de contenido.

5.2 Replicación de datos basada en la popularidad del tema de contenido

Los SCA administran el contenido de diferentes temas con diferentes niveles de popularidad. Este comportamiento motiva a nuestro sistema de almacenamiento a clasificar los temas de acuerdo a su popularidad, ofreciendo una administración de almacenamiento diferenciada.

5.2.1 Clases de temas de contenido basadas en la popularidad

De forma similar a los proveedores de contenido, los temas de contenido (CT) en un SCA se pueden clasificar en diferentes clases según su popularidad o demanda. Es por eso que usamos una clasificación similar para los temas de contenido (ver notaciones en la Tabla 5.1):

1. **Tema de Contenido Gamma ($CT\gamma$)**: Un tema de contenido CT_i se considera Gamma si $V_{CT_i} \leq AV_{CT}$, es decir, el volumen de bytes descargados de los contenidos con el tema i (V_{CT_i}) es menor o igual que el volumen promedio de bytes atendidos en las solicitudes de descarga para el contenido de todos los temas de contenido (AV_{CT}). Los temas de contenido cuyos contenidos no reciban ninguna solicitud de descarga serán de este tipo.

2. **Tema de Contenido Beta ($CT\beta$)**: Un tema de contenido CT_i se convierte en Beta si $V_{CT_i} > AV_{CT}$.

3. **Tema de Contenido Alfa ($CT\alpha$)**: Un tema de contenido CT_i se convierte en Alfa si CT_i es un tema de contenido Beta y $D_{CT_i} < D_{CT}$.

Como ocurrió en la clasificación de proveedores de contenido, estas clases se usan para ajustar el factor de replicación (RF) en el servicio de almacenamiento, ofreciendo una administración de

almacenamiento diferenciada.

5.2.2 Cálculo de densidad y volumen

La clasificación de los contenidos por temas es una estrategia atractiva para las organizaciones que desean distribuir o compartir corpus de documentos (PDF, DOC, PPT, HTML, XLS, etc.) o contenidos que incluyen una porción de texto en ellos, por ejemplo, imágenes médicas como DICOM, que incluyen metadatos. En este contexto, es posible clasificar un conjunto de datos representativo que proporcione una descripción general del tipo de contenido que gestionarán los servicios de almacenamiento.

Utilizando una estrategia similar en la clasificación de proveedores de contenido, la densidad o D_{CT} se refiere al tiempo promedio entre llegadas de solicitudes de contenido de cualquier tema y D_{CT_i} denota el promedio de hora de llegada de solicitud de descarga para el contenido de un tema específico. El volumen de un tema de contenido CT_i (V_{CT_i}) representa el consumo de almacenamiento del SCA cuando se atienden solicitudes de descarga para el contenido del tema CT_i y se define como:

$$V_{CT_i} = \sum_{j=1}^{|F_{CT_i}|} \sum_{1}^{ND_{F_j CT_i}} |F_j CT_i| \tag{5.7}$$

Donde $|F_{CT_i}|$ es la cantidad de archivos descargados del tema de contenido i (CT_i), $|F_{j_{CT_i}}|$ representa el tamaño de un archivo específico, $Fj(1 \leq j \leq |F_{CT_i}|)$, con CT_i tema, y $ND_{F_j CT_i}$ es el número de descargas realizadas al archivo $F_{j_{CT_i}}$. La actividad o volumen de almacenamiento (V_{CT}) del SCA cuando se atienden solicitudes de descargas para cualquier tema de contenido está dada por:

$$V_{CT_i} = \sum_{i=1}^{|CT|} VCT_i \tag{5.8}$$

Donde $|CT|$ es la cantidad de temas que tienen contenidos/archivos que se descargaron al menos

una vez. El servicio de almacenamiento promedio requerido por las solicitudes de descarga al SCA está definido por:

$$AV_{CT} = \frac{V_{CT}}{|CT|} \tag{5.9}$$

5.2.3 Proceso de clasificación

Los temas de contenido también se clasifican utilizando una estrategia similar que en los proveedores de contenido. Se calculan los volúmenes y las densidades de los temas de los contenidos, y clasificamos los temas de contenido Gamma ($CT\gamma$) como aquellos cuyo volumen es menor o igual que el volumen promedio en el SCA. Este conjunto de temas de contenido se define como: $CT\gamma = \{c|c \in CT \wedge Vc \leq AV_{CT}\}$. Un tema de contenido se convierte en Beta ($CT\beta$) cuando se cumple la siguiente condición: $P\beta = \{c|c \in CT \wedge Vc > AV_{CT}\}$. Finalmente, los temas de contenido de la clase alfa son temas en el siguiente conjunto: $CT = \{c|c \in P\beta \wedge Dc < D_{CT}\}$.

Como los temas de contenido Alfa ($CT\alpha$) se clasifican utilizando solo el conjunto de temas de contenido Beta $CT\beta$), se reduce el espacio de búsqueda y procesamiento. La siguiente fórmula define la densidad del tema de contenido beta i ($D_{CT\beta_i}$):

$$D_{CT\beta_i} = \frac{\sum_{j=2}^{|WDCT\beta_i|}(DownloadArrivalTimeReq_j - DownloadArrivalTimeReq_{j-1})}{|WDCT\beta_i|} \tag{5.10}$$

Donde $D_{CT\beta_i}$ representa la densidad de un tema de contenido beta i (CT_i), $|WDCT\beta_i|$ es el número de solicitudes de descarga que el sistema de almacenamiento ha recibido contenido con el tema $CT\beta_i$, y $DownloadArrivalTimeReq_j$ es el tiempo entre llegadas de una solicitud de descarga j para el contenido de $CT\beta_i$. Una vez obtenidos los valores de densidad de cada tema de contenido beta, aplicamos la siguiente fórmula para calcular la densidad para el conjunto completo de temas de contenido beta ($D_{CT\beta}$):

$$D_{CT\beta} = \frac{\sum_{i=1}^{CT\beta} DS\beta_i}{|CT\beta|} \tag{5.11}$$

Donde $|CT\beta|$ es la cantidad de temas de contenido beta.

Los valores de densidad del tema de contenido beta individual ($D_{CT\beta_i}$) y la densidad de todos los temas de contenido beta ($D_{CT\beta}$) se utilizan para definir los temas de contenido alfa establecidos de la siguiente manera: $CT\alpha = \{c|c \in CT\beta \wedge D_{CT_c} < D_{CT}\}$.

5.2.4 Factor de replicación

La clasificación del tema de contenido es otra estrategia para definir el factor de replicación en un servicio de almacenamiento. La popularidad del tema llevará al sistema de almacenamiento a adaptar el factor de replicación (RF) según la popularidad del tema de contenido (Densidad y Volumen). De forma similar a la estrategia del proveedor de contenido, la configuración base será $FR = 3$ (tres copias) en un grupo de $n = 5$ nodos de almacenamiento. FR se ajustará según la clasificación del tema de contenido. Los valores RF para los diferentes temas de contenido son: $RF\_CT\gamma = 1$ (temas de contenido gamma), $2 \leq RF\_CT\beta \leq 3$ (temas de contenido beta) y $4 \leq RF\_CT\alpha \leq n$ (temas de contenido alfa). En los temas de contenido alfa y beta, el valor final de RF se calcula mediante ajuste de parámetros, llevando a cabo un conjunto de experimentos que comparan el impacto en el servicio del sistema de almacenamiento cuando se utilizan valores RF ajustados en comparación con las buenas prácticas para los valores de RF.

En la estrategia de tema de contenido, también definimos una métrica para medir el impacto en el sistema de almacenamiento de las diferentes clases de tema de contenido, de la siguiente manera:

$$CTI = \frac{RF}{n} \tag{5.12}$$

El impacto del tema de contenido (CTI) está en el rango de $(0, 1]$ y está definido por el factor de replicación del tema de contenido (RF) y la cantidad de nodos en la piscina de almacenamiento (n).

De forma similar a la estrategia de proveedor de contenido, CTI representa la porción de recursos de los nodos de almacenamiento utilizados para ofrecer disponibilidad de contenido.

5.3 Escenario del sistema de compartición de archivos en la nube

El SCA es una evolución respecto a la funcionalidad de SkyCDS [23], que gestiona proveedores de contenido y consumidores. También permite a los usuarios publicar y suscribirse a procesos de flujo de trabajo que crean contenido. *Lanzador* es un componente adicional que representa la parte cliente de la plataforma SCA. Dado que *Lanzador* emula clientes reales, su generación de carga de trabajo se basa en el comportamiento que se encuentra en los sistemas reales de intercambio de contenido [26]. El tiempo entre llegadas para las solicitudes en un SCA sigue una distribución de Poisson (el iniciador genera 10 solicitudes por segundo en la media). El tipo de solicitud sigue una distribución Zipf, donde el 80 % de las operaciones son descargas y el 20 % son cargas. Los tamaños de contenido también se definieron mediante una distribución de Poisson [32] con 5 MB de tamaño en promedio. Los tipos de usuarios (contenido Consumidor/Proveedor) también fueron definidos por una distribución Zipf, donde la mayoría de ellos son Consumidores.

5.3.1 Configuraciones

Se consideraron las siguientes configuraciones de prueba:

- **Norma o Estático**: En esta configuración, la colocación de datos se lleva a cabo utilizando un esquema aleatorio tradicional, como TwoChoices [34] (dos instancias de almacenamiento se seleccionan al azar y se elegirá la instancia con la menor carga de trabajo). La replicación se ejecuta de forma proactiva (una vez que una instancia de almacenamiento recibe el contenido, el control vuelve al cliente y la instancia comienza a enviar réplicas a otras instancias) con un

factor de replicación de 3 para las clases Alfa, Beta y Gamma.

- **Reactivo**: La colocación de datos se lleva a cabo utilizando un esquema aleatorio como TwoChoices. La replicación se ejecuta de forma reactiva, comportándose de manera similar a un servicio de caché (cuando una instancia de almacenamiento recibe una solicitud de descarga y no tiene una réplica del contenido solicitado, la instancia de almacenamiento obtendrá el contenido y conservará una réplica de este).

- **Proactivo**: Esta configuración representa nuestras propuestas de replicación basadas en el proveedor de contenido y la popularidad del tema de contenido. La colocación de datos se lleva a cabo utilizando también la estrategia TwoChoices.

Las configuraciones proactivas usan un factor de replicación flexible (RF) que puede variar según el interés del administrador del sistema de almacenamiento y los usuarios finales. Las siguientes son configuraciones proactivas utilizadas en nuestros experimentos implementados en un conjunto de servicios de almacenamiento de 5 nodos ($n = 5$):

- ***ProA3B1***: Configuración proactiva interesada en encontrar una solución de compromiso entre el consumo de almacenamiento y el tiempo de servicio de almacenamiento con prioridad de rendimiento (clase Alfa con $RF = 3$, clase Beta con $RF = 1$, otros con $RF = 1$).

- ***ProNorm***: Configuración que aplica buenas prácticas ($RF = 3$) para las clases Alfa y Beta (otras con $RF = 1$), con interés en el consumo de almacenamiento.

- ***ProFlood***: Aplica replicación por inundación en el grupo de almacenamiento de $n = 5$ nodos con $RF = 5$ para las clases Alfa y Beta (otros con $RF = 1$).

- ***ProA5B3***: Aplica inundaciones de ($RF = 5$) para la clase Alfa y buenas prácticas ($RF = 3$) para la clase Beta, las demás tendrán $RF = 1$.

Tabla 5.2: Configuraciones proactivas.

Proactivo Configuración	Método	$\mathbf{RF}(\alpha, \beta, \gamma)$, $\mathbf{n = 5}$
ProA3B1	Buenas prácticas (Norma) para la mayoría de los usuarios activos (α)	$RF(3, 1, 1)$
ProNorm	Buenas prácticas (Norma) para usuarios activos (α, β)	$RF(3, 3, 1)$
ProFlood	Inundaciones para usuarios activos (α, β)	$RF(5, 5, 1)$
ProA5B3	Inundación & Norma para usuarios activos (α, β)	$RF(5, 3, 1)$
ProA5B2	Inundación & Frugalidad	$RF(5, 2, 1)$

- ***ProA5B2***: Usa inundación de ($RF = 5$) para la clase Alfa y moderación (frugalidad) de almacenamiento para la clase Beta ($RF = 2$), los otros tendrán $RF = 1$.

La implementación de las configuraciones proactivas varía de acuerdo con la técnica de replicación utilizada, que podría basarse en la popularidad del proveedor de contenido o la popularidad del tema de contenido. Es importante notar que para estos experimentos las configuraciones proactivas en el SCA siempre implican el uso de $RF = 1$ en proveedores de contenido o temas de contenido que pertenecen a la clase Gamma.

5.3.2 Métricas

Durante el proceso de experimentación, se obtuvieron las siguientes métricas del SCA:

- **Tiempo de Respuesta (RT)**: Cubre desde el momento en que un cliente envía una solicitud (carga/descarga) al SCA hasta que el contenido se transfiere por completo. RT también incluye el tiempo empleado por el cliente para obtener, desde la instancia de metadatos, la dirección (URL) del nodo de almacenamiento que atenderá la solicitud del cliente.
- **Consumo de Almacenamiento (SC)**: Representa la cantidad de recursos de almacenamiento utilizados por el servicio de almacenamiento.

- **Throughput (Th)**: Volumen de datos (Mb/s) que fluye al sistema de almacenamiento.
- **Nivel de Balanceo de Carga (LBL)**: Esta métrica representa el nivel de distribución de la carga de trabajo entre los nodos de almacenamiento en un período de tiempo. El LBL de una instancia de almacenamiento x viene dado por:

$$LBLx = CWx - IW \tag{5.13}$$

Donde CWx es la carga de trabajo actual en la instancia de almacenamiento x y IW representa la distribución de carga de trabajo ideal, que viene dada por:

$$IW = TSC/n \tag{5.14}$$

Donde TSC representa el consumo total de almacenamiento, y n la cantidad total de instancias de almacenamiento. Un valor de LBL cercano a cero significaría que las instancias de almacenamiento están bien equilibradas.

5.3.3 Cargas de trabajo para la clasificación de temas de contenido

La estrategia de replicación basada en la popularidad del tema del contenido requiere clasificar el contenido de texto. Esto significa que todos los archivos distribuidos y compartidos en el SCA incluyen texto como parte de sus contenidos, por ejemplo, archivos PDF, HTML, PPT o imágenes con metadatos. Como parte de nuestra experimentación, el SCA entregó y compartió contenidos tomados del conjunto de datos de Medline [51], que es un corpus de documentos relacionado con el área de medicina. El proceso iniciador usa su componente Generador de Tráfico (ver Subsección 4.2.2) para generar 5 cargas de trabajo para el SCA, cada una con 2530 operaciones de carga (20 %) y descarga (80 %). El componente Generador de Archivos (ver Subsección 4.2.3) genera archivos sintéticos de tamaños de 5 MB en promedio, más de 500 archivos por carga de trabajo. Cada archivo

sintético está asociado con el contenido textual de un archivo tomado del corpus Medline, que representa sus metadatos. Esta parte textual del contenido permite al clasificador etiquetarlo con el tema correspondiente.

Como se menciona en la Subsección 5.2.2, Volumen y Densidad son dos parámetros principales que se utilizan para determinar la popularidad de un tema de contenido y clasificarlo en una de las siguientes clases: Alfa, Beta y Gamma (de más a menos popularidad, respectivamente). La Figura 5.1 muestra el Volumen (eje vertical) de los diferentes temas extraídos de los contenidos en un conjunto de datos, que fueron etiquetados con números (eje horizontal) por el proceso de clasificación. Los temas cuyos volúmenes de contenido están por encima del valor de la mediana se clasifican como clase Beta, el resto de los contenidos se clasifican como clase Gamma, que es el valor predeterminado para cualquier contenido.

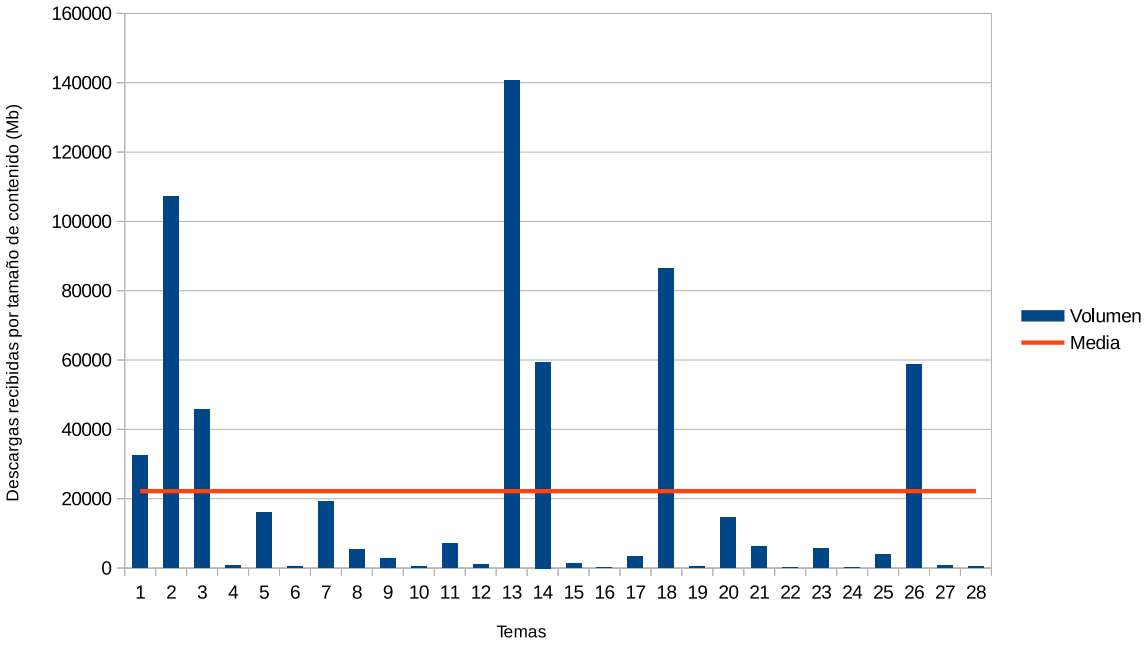


Figura 5.1: Volumen de 28 temas identificados en documentos cargados en el SCA.

Una vez que se identifican los temas Beta, se analiza la densidad de este conjunto de temas de contenido para determinar los temas Alfa, que son aquellos temas cuya densidad está por encima del valor de la mediana (más frecuentemente demandado). La Figura 5.2 muestra la densidad (eje vertical) de los contenidos cuyos temas se clasificaron como temas Beta. El valor medio de densidad se indica con una línea horizontal. Los temas etiquetados con los números 1, 3, 13 y 18 tienen una densidad superior al valor de la mediana, por lo que se considerarán temas de la clase Alpha.

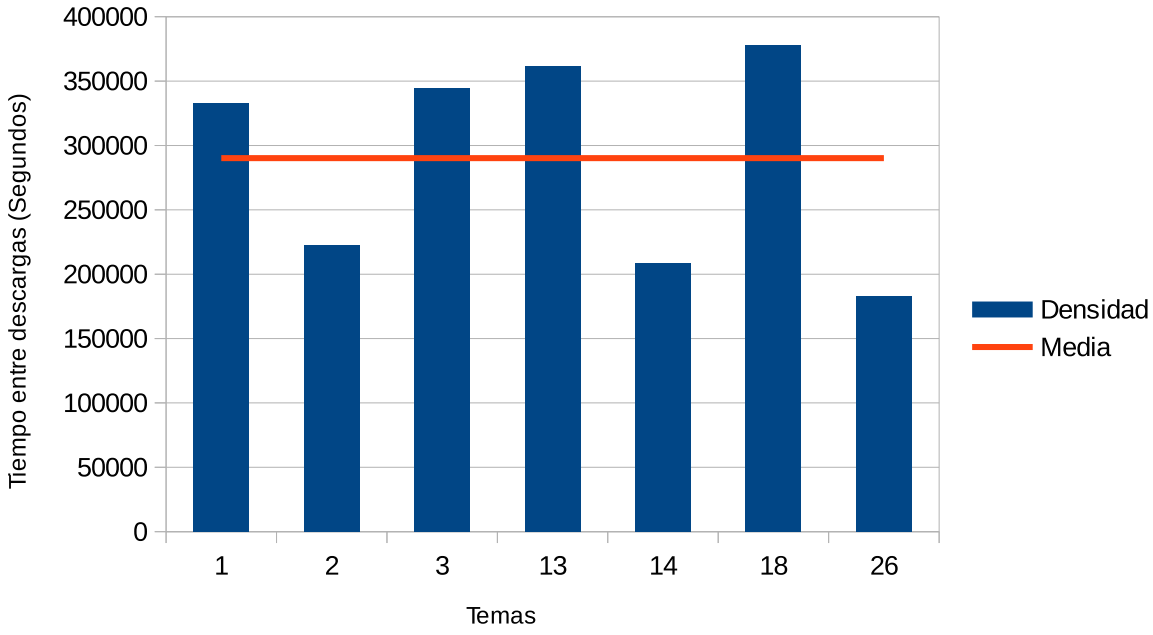


Figura 5.2: Densidad de contenidos en la clase Beta.

5.4 Resultados

Esta sección resume los resultados más relevantes obtenidos por el SCA cuando se implementan las técnicas de replicación y colocación de datos descritas en la Sección 5.1 y 5.2. Las configuraciones de prueba utilizadas en nuestra experimentación se describieron en la Subsección 5.3.1.

5.4.1 Enfoque de popularidad del proveedor

Comenzamos la experimentación configurando las técnicas de replicación en el SCA en función de la popularidad de los proveedores de contenido. Evaluamos el tiempo de respuesta que los usuarios del SCA percibieron cuando se aplicó una configuración de prueba diferente teniendo en cuenta la popularidad de los proveedores de contenido. Para este escenario, la configuración Estática/Norma representa nuestra línea base para la comparación, ya que implementa la configuración de replicación tradicional o de buenas prácticas con $RF = 3$ y no cambia durante la experimentación completa. El proceso de inicio (que representa la parte del cliente del SCA) reprodujo 12 cargas de trabajo con 5000 operaciones de carga/descarga, con un tiempo entre llegadas de 10 solicitudes por segundo. El tiempo entre llegadas sigue una distribución de Poisson [32], y el tipo de operación (carga/descarga) sigue una distribución Zipf con un 80 % de descargas y un 20 % de cargas aproximadamente.

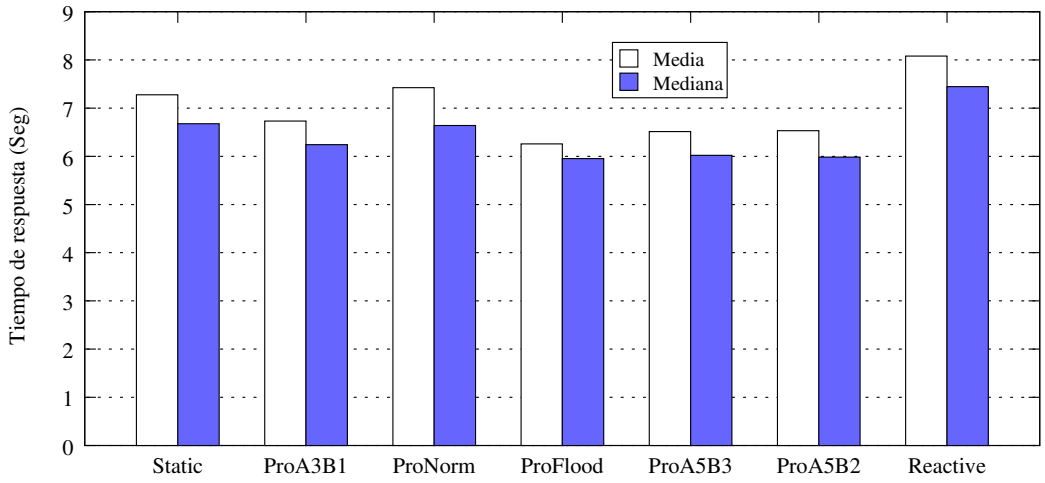


Figura 5.3: Tiempo de respuesta del SCA al implementar la replicación basada en la popularidad del proveedor de contenido.

La Figura 5.3 muestra en la vertical la mediana de los tiempos de respuesta en segundos del SCA para la carga de trabajo (operaciones de descarga/carga), implementando la replicación basada en la popularidad del proveedor de contenido siguiendo las configuraciones de prueba descritas anteriormente (eje horizontal). La mayoría de las configuraciones proactivas (ProA3B1, ProFlood, ProA5B3 y ProA5B2) mejoraron la configuración Estática/Norma (línea base), excepto la configuración ProNorm, que usa $RF = 3$ para los proveedores de contenido Alfa y Beta. Como la Figura 5.3 muestra los valores medios del tiempo de respuesta obtenidos de la carga de trabajo completa, considerando descargas (80 % de la carga de trabajo) y carga (20 % de la carga de trabajo), parece que la mayoría de las configuraciones de replicación tienen un comportamiento similar cuando se basan en la popularidad del proveedor de contenido. Sin embargo, es importante observar que algunas configuraciones de replicación, como ProFlood, requieren más de dos veces espacio de almacenamiento para proporcionar una experiencia de usuario similar.

La Figura 5.4 muestra la mejora del rendimiento (positiva) o el retroceso (negativa) en términos de porcentaje (eje vertical), teniendo en cuenta el tipo de operaciones (carga/descarga y global -todas juntas-). Como era de esperar, las configuraciones que aumentan el número de réplicas (ProFlood, ProA5B3, ProA5B2) tienen un efecto positivo en las operaciones de descarga y un efecto negativo en las operaciones de carga (algunas de ellas, ProFlood, disminuyen el rendimiento de carga en más del 60 %). Cuando las buenas prácticas ($RF = 3$) se aplican solo en proveedores de contenido alfa (ProA3B1), el rendimiento mejora tanto en las operaciones de carga como de descarga.

Podemos ver que la mejora significativa en el rendimiento de las operaciones de carga producidas por las configuraciones ProNorm y Reactive no produce una mejora del rendimiento en las operaciones de descarga, lo que se convierte en un impacto importante para la experiencia del usuario, considerando que las operaciones de descarga son mayoría. Otro aspecto importante en esta evaluación fue analizar cómo la mejora del rendimiento afecta el consumo de almacenamiento con respecto a la configuración de línea de base (Estático). La Figura 5.5 muestra una comparación entre diferentes políticas de replicación, que contrastan el rendimiento y el consumo de almacenamiento

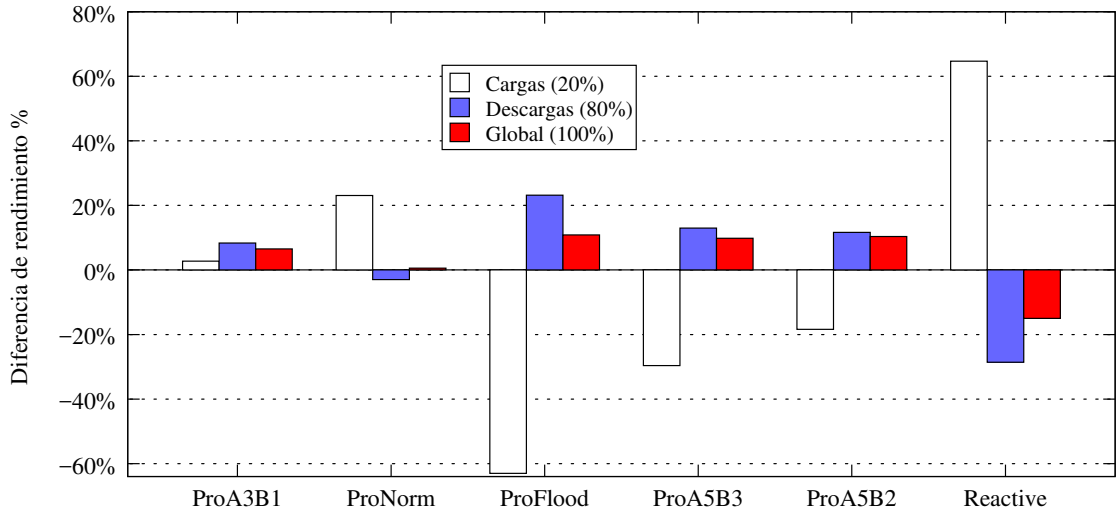


Figura 5.4: Diferencia de rendimiento considerando la popularidad del proveedor de contenido.

con la configuración de referencia (representada con un valor de cero en el eje vertical).

Podemos ver cómo la configuración reactiva, que produjo menos réplicas y se comporta como memoria caché (las réplicas se generan a pedido), mostró frugalidad en el consumo de almacenamiento, ahorrando más del 20 % de almacenamiento en comparación con la configuración de referencia ($RF = 3$). Sin embargo, el rendimiento mostró un impacto negativo de casi 15 %. Todas las configuraciones proactivas mejoraron el rendimiento en diferentes consumos de almacenamiento. Es interesante observar que cuando se aplican buenas prácticas ($RF = 3$) a los proveedores de contenido Alfa (ProA3B1) y proveedores de contenido Beta (ProNorm) se mejora el rendimiento del servicio, ahorrando recursos de almacenamiento con respecto a la configuración de línea de base. ProA3B1 se convierte en el mejor equilibrio entre el rendimiento y el consumo de almacenamiento, ProNorm ofrece un buen compromiso cuando la frugalidad es una prioridad, y ProA5B2 es una buena opción para mejorar el rendimiento con un consumo moderado de recursos.

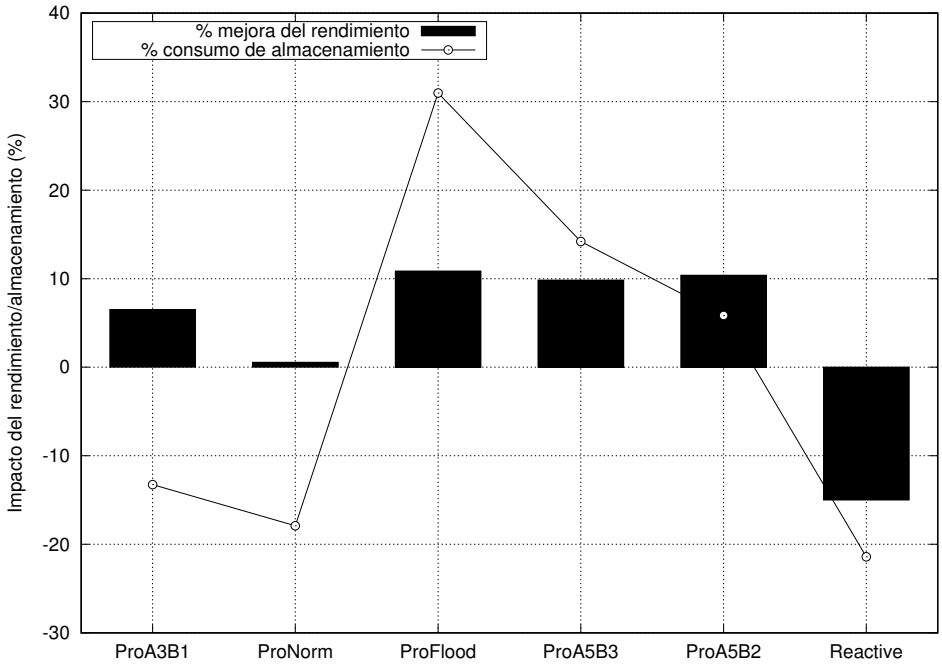


Figura 5.5: Impacto en el rendimiento y el almacenamiento con respecto a la línea de base teniendo en cuenta la popularidad del proveedor de contenido.

La Figura 5.6 muestra cómo las políticas de replicación basadas en la popularidad del proveedor de contenido afectaron el equilibrio de carga en los nodos de almacenamiento (en términos porcentuales). El equilibrio de carga ideal definido en la Subsección 5.3.2 se utilizó como referencia (valor de cero en el eje vertical). Es de destacar que las configuraciones Reactiva y ProActiva produjeron un equilibrio de carga muy cercano al ideal (es decir, muy cercano a cero), mejorando el equilibrio de carga obtenido por la configuración de buenas prácticas (Estático, $RF = 3$).

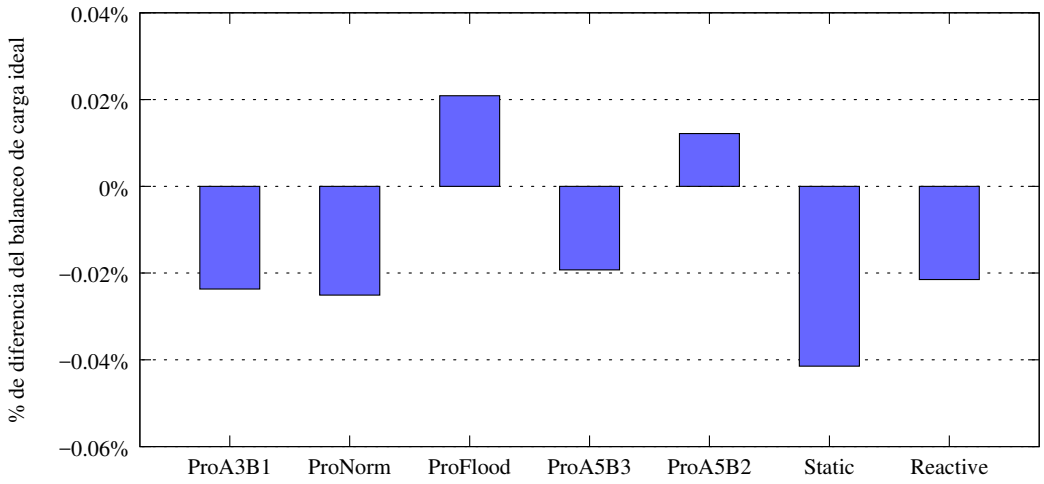


Figura 5.6: Diferencias de equilibrio de carga en comparación con ideal considerando la popularidad del proveedor de contenido.

5.4.2 Enfoque de popularidad del tema del contenido

Esta sección presenta los resultados obtenidos de la evaluación del prototipo del SCA que implementa la replicación de contenido basada en la popularidad del tema de contenido (explicado en la Sección 5.2). Similar a la evaluación anterior, en estos experimentos usamos las métricas descritas en la Subsección 5.3.2, las cargas de trabajo preparadas para la clasificación de temas (Subsección 5.3.3) y las configuraciones de prueba (Subsección 5.3.1): Estático/Norma, Reactivo y Proactivo, donde Estático/Norma o la configuración de buenas prácticas (con $RF = 3$ para temas Alfa, Beta y Gamma) representa nuestra línea base para la comparación.

Para esta prueba, el proceso iniciador reprodujo las cargas de trabajo descritas en la Subsección 5.3.3. La Figura 5.7 muestra la media del tiempo de respuesta del SCA implementando diferentes políticas de replicación basadas en la popularidad de los temas de contenido, considerando las operaciones de carga y descarga por separado y en conjunto (Global). Podemos ver que la mayoría de las configuraciones ProActive (excepto ProFlood) mejoraron la configuración de buenas prácticas (Estático). Esto demuestra que una aplicación selectiva (Temas de Contenidos Alfa, Beta y Gamma)

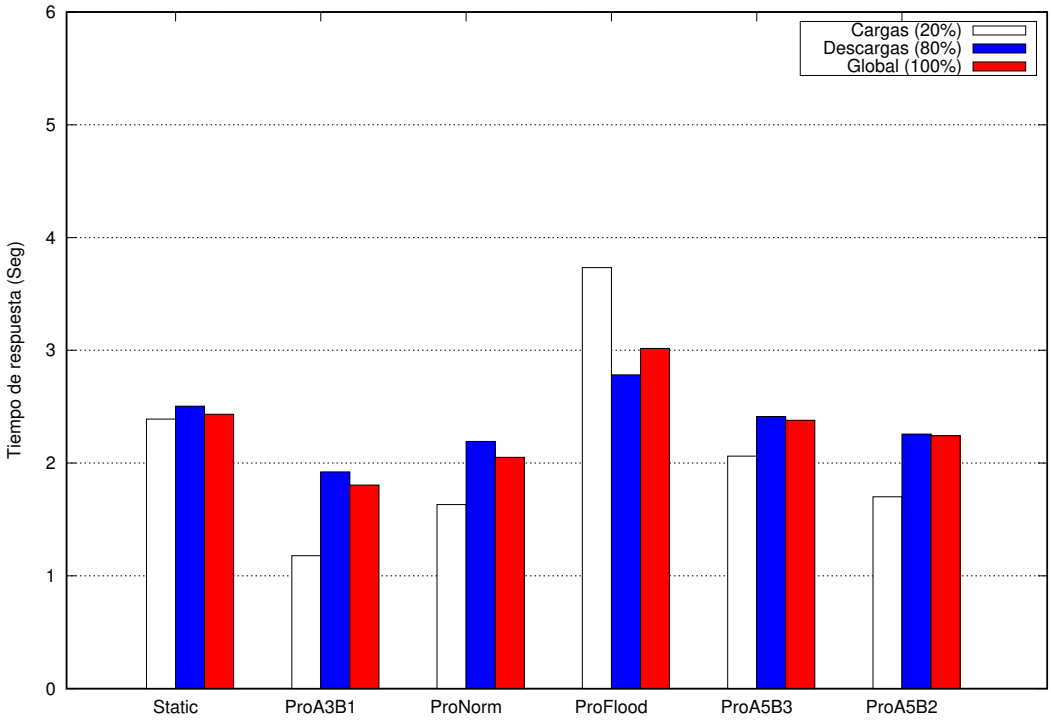


Figura 5.7: Tiempo de respuesta del SCA considerando la replicación basada en la popularidad de los temas de contenido.

de $RF = 3$ puede mejorar una aplicación genérica de las buenas prácticas, especialmente para los temas más populares (Temas de Contenido Alfa, ProA3B1). La figura 5.8 muestra el impacto en el rendimiento (en términos porcentuales) de las configuraciones de ProActive con respecto a la línea base, es decir, la configuración estática. Podemos ver cómo casi todas las configuraciones ProActive mejoraron el rendimiento del SCA, excepto la configuración de ProFlood, que tiene un impacto negativo en el rendimiento tanto en las operaciones de carga como de descarga, con un efecto importante en las operaciones de carga.

La experiencia del usuario también puede medirse por el rendimiento obtenido del SCA. En este sentido, La Figura 5.9 refuerza los resultados mostrados en las Figuras 5.7 y 5.8, en los que casi todas las configuraciones de ProActive mejoraron la experiencia de servicio de los usuarios con respecto a buenas prácticas (Estático), excepto el que usa inundación (ProFlood).

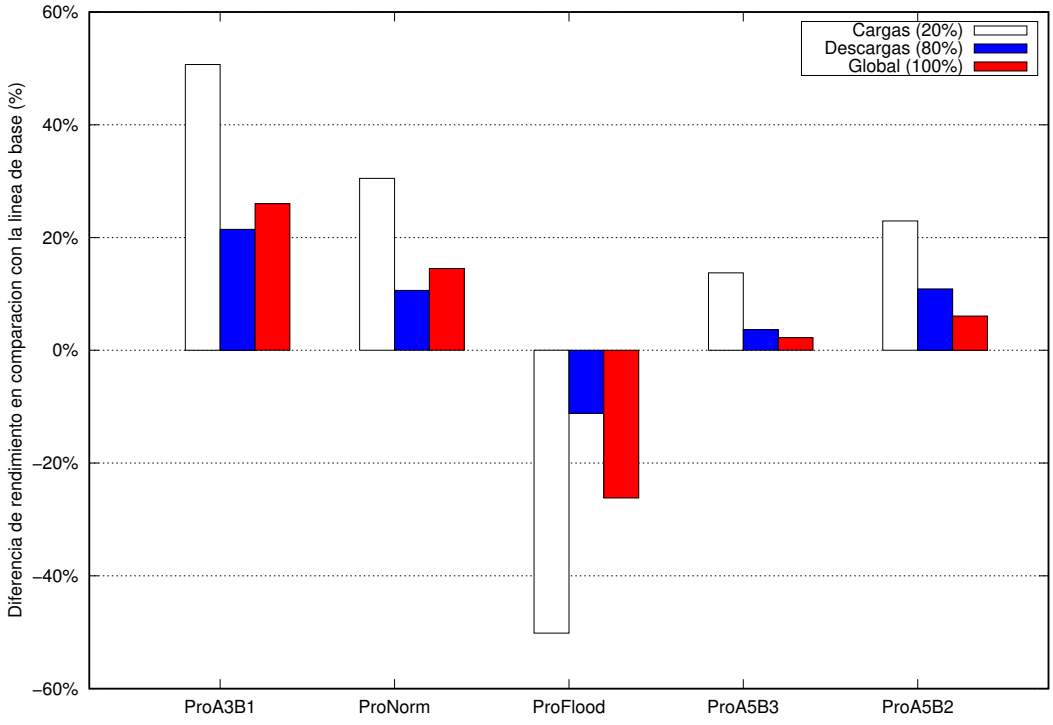
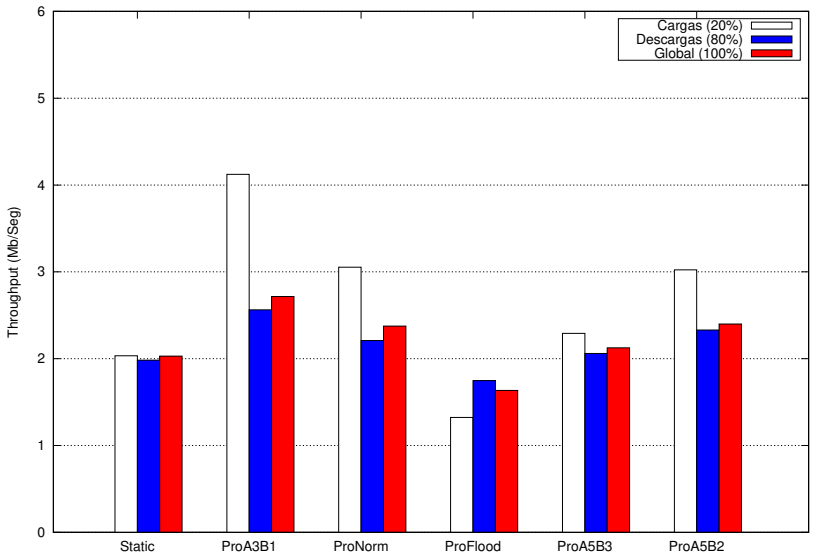
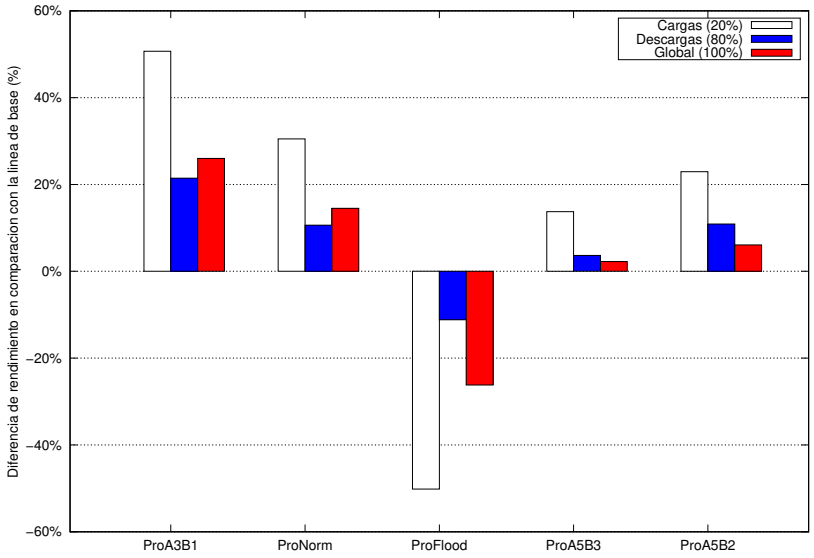


Figura 5.8: Diferencia de rendimiento teniendo en cuenta la popularidad del tema de contenido.

La Figura 5.10 muestra la mejora del rendimiento con respecto al consumo de almacenamiento considerando los enfoques ProActive (valores medianos globales). Podemos ver que casi todas las configuraciones ProActive mejoran el rendimiento, excepto la estrategia ProFlood, que produce el peor de los dos mundos, afecta negativamente el rendimiento y requiere un alto consumo de almacenamiento. El mejor equilibrio entre el consumo de almacenamiento y el rendimiento lo proporciona ProA3B1, que solo genera réplicas de contenido de los temas de contenido más populares (temas Alfa).



(a) Rendimiento del SCA.



(b) Comparación de rendimiento.

Figura 5.9: (a) Rendimiento del SCA utilizando diferentes políticas de replicación teniendo en cuenta la popularidad del tema de contenido, (b) Rendimiento en comparación con la línea de base.

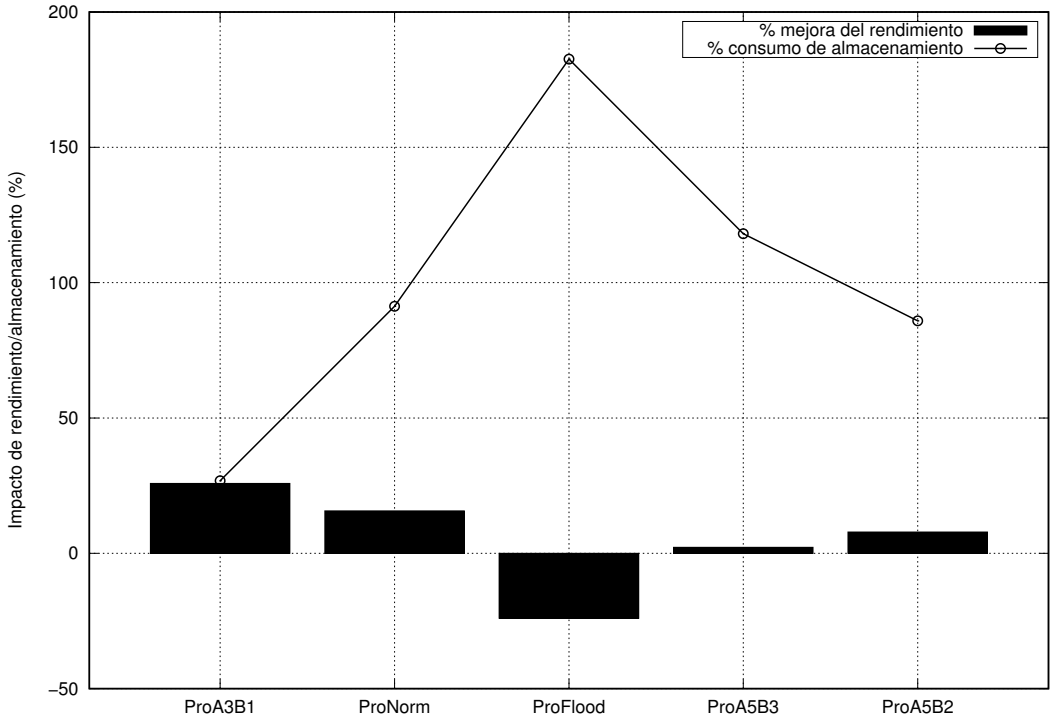


Figura 5.10: Impacto en el rendimiento y el almacenamiento con respecto a la línea de base (popularidad del tema de contenido).

5.5 Discusión

La redundancia y la replicación de contenido son los enfoques más aceptados para proporcionar disponibilidad y tolerancia a fallas; sin embargo, estos enfoques también se convierten en un desafío no trivial para el administrador del SCA debido al aumento del consumo de almacenamiento y el tiempo de servicio que causan. Este capítulo presenta dos nuevas estrategias para replicar contenido digital considerando la popularidad de los productores/proveedores de contenido y la popularidad del tema del contenido. En este enfoque, la popularidad de los productores y el tema de los contenidos se clasificaron en tres clases, Alfa (alto), Beta (medio) y Gamma (bajo). Cada clase determinará el factor de replicación (RF) utilizado para un contenido en particular. Se implementaron diferentes políticas de replicación en un entorno de almacenamiento en nube privado, y los resultados mostraron que la

implementación del proveedor de contenido o la popularidad del tema de contenido permiten mejorar la experiencia del usuario (tiempo de respuesta) y ahorrar recursos de almacenamiento en comparación con las buenas prácticas. recomendado en literatura especializada, donde la mayoría del sistema aplica un factor de replicación igual a 3. Los experimentos también mostraron que cuando la prioridad para el administrador del SCA es el tiempo de respuesta, la mejor compensación entre el tiempo de respuesta y el consumo de almacenamiento los temas que pertenecen a la clase Alfa tienen un factor de replicación de 3 ($RF = 3$) y las clases Beta y Gamma tienen $RF = 1$ (configuración ProA3B1). Cuando el tiempo de respuesta es importante pero el consumo de almacenamiento representa un alto costo potencial, la mejor solución viene dada por la configuración ProA3B2, es decir, clase Alfa con $RF = 3$, clase Beta con $RF = 2$ y clase Gamma con $RF = 1$. En resumen, nuestras propuestas de replicación permiten al SCA obtener un mejor rendimiento con un bajo consumo de almacenamiento al compararlo con las buenas prácticas tradicionales (utilizando un $RF = 3$ para todos los contenidos) gracias a los siguientes aspectos: *a)* clasificación de proveedores de contenido y temas de contenido en diferentes niveles de popularidad; *b)* la estimación dinámica y la configuración del valor de RF en el contenido cargado según la clase de popularidad, y *c)* el mecanismo de replicación reactiva que devuelve el control a los clientes antes de que comience el proceso de replicación en los nodos de almacenamiento en la nube.

6

Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones a las cuales se llegaron y que están basadas en la experimentación. Asimismo se hace una revisión de resultados obtenidos y se plantea trabajo futuro de está investigación. Se describen las contribuciones que surgieron durante este trabajo de tesis.

6.1 Conclusiones

En esta tesis se abordaron los problemas a los que enfrenta el desarrollo de Sistemas de Compartición de Archivos (SCA) en la nube. En particular el uso de modelos en capas que han presentado limitaciones en cuanto a su poca flexibilidad para adaptarse dinámicamente a los cambios y a las nuevas necesidades que surgen en las organizaciones. Provocando desafíos de naturaleza técnica en el SCA que hacen que la base del código fuente sea considerablemente más compleja, difícil de mantener, y afectando la continuidad del negocio.

Como una solución a la problemática mencionada en este trabajo de tesis definimos una unidad mínima de abstracción, llamada Unidad de Construcción (UC). Que hizo posible la encapsulación de

micro-aplicaciones en contenedores virtuales facilitando su acoplamiento, a través de la integración de interfaces de comunicación, y la formación de servicios más especializados a los que llamamos Subservicios Virtuales. Asimismo, nuestra propuesta proporciona un esquema para definir estilos arquitectónicos de procesamiento, como maestro/esclavo, tuberías y flujos de trabajo que permitieron mejorar las prestaciones que ofrecen las micro-aplicaciones encapsuladas en UCs y que en su conjunto forman al SCA.

Como prueba de concepto de esta arquitectura se rediseñó un SCA llamado SkyCDS [23], el cual se logró transformar de un modelo basado puramente en capas a un modelo de micro-aplicaciones abstraídas en contenedores virtuales. Como parte de esta tesis se crea una nomenclatura de configuración y orquestación de Unidades de Construcción que permite la planificación, representación, despliegue, acoplamiento y operación de las UCs que serán parte del SCA que se desee crear.

Las pruebas de funcionamiento de las UCs y la nomenclatura de configuración mostradas en el Capítulo 4 muestran la factibilidad del uso del modelo basado en micro-aplicaciones encapsuladas en contenedores virtuales para el desarrollo de SCAs. Un caso de estudio, presentado en el Capítulo 5, muestra cómo un nuevo SCA, SkyCDS evolucionado, puede integrar, de manera modular, diferentes Unidades de Construcción que abstraen componentes de software en los que se implementan nuevos algoritmos que mejoran los aspectos no funcionales como balanceo de carga y disponibilidad del SCA.

Con este trabajo, se ha logrado contribuir al estado del arte con una nueva arquitectura de micro-aplicaciones modularizada y encapsulada en contenedores virtuales que permite la construcción de sistemas adaptables de compartición de archivos en la nube en forma flexible y eficiente. Se consiguió el objetivo de crear un esquema de nomenclatura que permite representar de manera abstracta a un SCA, contando con un mecanismo de acoplamiento que facilita la adición/remoción/integración de componentes no-funcionales. La facilidad de integrar nuevos componentes al SCA desarrollado con esta arquitectura nos permitió proponer y evaluar nuevos algoritmos (ver Capítulo 5), que logran mejorar las prestaciones de los componentes no-funcionales en términos de tiempos de

servicio/respuesta.

6.2 Contribuciones

Como producto del trabajo de tesis se obtuvo una arquitectura para construir sistemas de compartición de archivos basado en micro-aplicaciones, que permita la integración de aspectos no-funcionales y mejora la experiencia del servicio a los usuarios.

A partir de este trabajo de tesis se aporta lo siguiente:

- Una arquitectura de software modular para construir sistemas de compartición de archivos en la nube de forma dinámica, flexible y adaptable.
- Una nomenclatura que permite representar de manera abstracta los SCA.
- Un mecanismo que permita la agregación, actualización y remoción de aspectos no-funcionales a un SCA.
- Un método para construir, desplegar e implementar un SCA a partir del modelo basado en unidades de construcción.
- El diseño, implementación y evaluación de nuevos algoritmos para mejorar aspectos no-funcionales de un sistema de compartición de archivos.

6.3 Limitaciones

Este trabajo de tesis se realizó a partir de un plan de actividades que se ajustó a un periodo de un año, por lo que existen aspectos de esta tesis que quedaron limitados. Algunos de estos aspectos son los siguientes:

- La nomenclatura de configuración y orquestación que desarrollamos para crear los subservicios virtuales y construir arquitecturas de procesamiento como maestro/esclavo, tuberías, etc., es propietario, lo que hace que no sea compatible con otras herramientas de construcción de *workflows*.

- No se ha construido un flujo de trabajo utilizando nuestras Unidades de Construcción (UCs) a partir de herramientas existentes para construir flujos de trabajo, por lo que no tenemos garantías, de que, esto sería posible actualmente.

- No se cuenta con comunicación basada en memoria compartida para el caso en donde se utilizan dos contenedores virtuales en una misma computadora física. La interfaz de comunicación de memoria compartida funciona solo para micro-aplicaciones ejecutándose dentro de un mismo contenedor virtual.

6.4 Trabajo futuro

Como trabajo futuro se buscará resolver las limitaciones presentadas en la sección anterior, dando prioridad a las siguientes:

- Adaptar la interfaz de comunicación que haga posible que dos Unidades de Construcción encapsuladas en diferentes contenedores virtuales en una misma máquina física puedan intercambiar información utilizando memoria compartida.

- Adaptar la arquitectura de las Unidades de Construcción para que puedan ser utilizadas en la construcción de flujos de trabajo utilizando herramientas existentes para este fin.

- Crear una máquina generadora de subservicios virtuales que pueda recibir la configuración y la orquestación de Unidades de Construcción utilizando lenguajes formales de marcado existentes.

Bibliografía

[1] Abdelbaky, M., Diaz-Montes, J., Parashar, M., Unuvar, M., and Steinder, M. (2015). Docker containers across multiple clouds and data centers. In *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on*, pages 368–371.

[2] Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Microservices architecture enables devops: migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52.

[3] Bass, L., Weber, I., and Zhu, L. (2015). *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional.

[4] Bhudavaram, D. K., Naik, S., and Uthamanathan, S. (2016). File creation through virtual containers. US Patent 9,495,410.

[5] Boettiger, C. (2015). An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79.

[6] Brown, S. (2013). Software architecture for developers. *Coding the Architecture*.

[7] Cloud, S. (2017). Microservice architecture with spring boot, spring cloud and docker.

[8] Cosmina, I. (2017). Spring microservices with spring cloud. In *Pivotal Certified Professional Spring Developer Exam*. Springer.

[9] Coulouris, G. F., Dollimore, J., and Kindberg, T. (2005). *Distributed systems: concepts and design*. pearson education.

[10] Di Francesco, P., Malavolta, I., and Lago, P. (2017). Research on architecting microservices:

Trends, focus, and potential for industrial adoption. In *Software Architecture (ICSA), 2017 IEEE International Conference on*, pages 21–30.

[11] Docker (2017). Welcome to the docs. `https://docs.docker.com/`.

[12] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*, pages 195–216. Springer.

[13] Drive, G. (2017).

[14] Dropbox (2017).

[15] Dua, R., Raja, A. R., and Kakadia, D. (2014). Virtualization vs containerization to support paas. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 610–614.

[16] Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., and Newling, T. (2004). *Patterns: service-oriented architecture and web services*. IBM Corporation, International Technical Support Organization.

[17] Foster, I. (2005). Globus toolkit version 4: Software for service-oriented systems. In *IFIP international conference on network and parallel computing*, pages 2–13.

[18] Foster, I. (2011). Globus online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Computing*, 15(3):70–73.

[19] Foster, I. and Kesselman, C. (1999). The globus project: A status report. *Future Generation Computer Systems*, 15(5):607–621.

[20] Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S. (2002). Grid services for distributed system integration. *Computer*, 35(6):37–46.

[21] Gannon, D., Barga, R., and Sundaresan, N. (2017). Cloud-native applications. *IEEE Cloud Computing*, 4(5):16–21.

[22] Gonzalez, J. L. and Marcelin-Jimenez, R. (2011). Phoenix: A fault-tolerant distributed web storage based on urls. In *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, pages 282–287.

[23] Gonzalez, J. L., Perez, J. C., Sosa-Sosa, V. J., Sanchez, L. M., and Bergua, B. (2015). Skycds: A resilient content delivery service based on diversified cloud storage. *Simulation Modelling Practice and Theory*, 54:64–85.

[24] Gonzalez-Compean, J., Sosa-Sosa, V., Diaz-Perez, A., Carretero, J., and Yanez-Sierra, J. (2018). Sacbe: A building block approach for constructing efficient and flexible end-to-end cloud storage. *Journal of Systems and Software*, 135:143–156.

[25] Google, I. y. L. (2017). Istio.

[26] Hsueh, H.-Y., Hua, J.-S., Huang, S.-M., and Will, H. J. (2009). Resource sharing behavior in a socialized peer-to-peer internet environment. In *Mobile Business, 2009. ICMB 2009. Eighth International Conference on*, pages 131–136. IEEE.

[27] Jedidiah Yanez-Sierra, Arturo Diaz-Perez, V. S.-S. and J.L.Gonzalez (2015). Towards secure and dependable cloud storage based on user-defined workflows. In *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, pages 405–410.

[28] Joseph, J. and Fellenstein, C. (2004). *Grid computing*. Prentice Hall Professional.

[29] Joy, A. M. (2015). Performance comparison between linux containers and virtual machines. In *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in*, pages 342–346.

[30] Kakadia, D. (2015). *Apache Mesos Essentials*. Packt Publishing Ltd.

[31] Kratzke, N. and Quint, P.-C. (2017). Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study. *Journal of Systems and Software*, 126:1–16.

[32] Krishnamoorthy, K. (2006). *Handbook of statistical distributions with applications*. Chapman and Hall/CRC.

[33] Lewis, J. and Fowler, M. (2014). Microservices: a definition of this new architectural term. *MartinFowler. com*, 25.

[34] Luczak, M. J., McDiarmid, C., et al. (2005). On the power of two choices: balls and bins in continuous time. *The Annals of Applied Probability*, 15(3):1733–1764.

[35] Marinescu, D. C. (2013). *Cloud computing: theory and practice*. Newnes.

[36] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

[37] Nadareishvili, I., Mitra, R., McLarty, M., and Amundsen, M. (2016). *Microservice Architecture: Aligning Principles, Practices, and Culture*. .º'Reilly Media, Inc.".

[38] Newman, S. (2015). *Building microservices: designing fine-grained systems*. .º'Reilly Media, Inc.".

[39] OneDrive (2017).

[40] Pablo Morales-Ferreira, Miguel Santiago-Duran, C. G.-D. J. G.-C. V. J. S.-S. and Lopez-Arevalo, I. (2018). A data distribution service for cloud and containerized storage based on information dispersal. In *Proc. On 12th IEEE Symposium on Service-Oriented System Engineering. IEEE Computer Society Conference Publishing Services*, pages 86–95.

[41] Pahl, C. (2015). Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31.

[42] Pérez, S., Facchini, H. A., Mercado, G., Taffernaberry, J. C., and Bisaro, L. (2012). Estudio sobre la distribución de tráfico autosimilar en redes wi-fi. In *XVIII Congreso Argentino de Ciencias de la Computación*.

[43] Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)*, 36(2):335–348.

[44] Reed, W. J. and Jorgensen, M. (2004). The double pareto-lognormal distribution—a new parametric model for size distributions. *Communications in Statistics-Theory and Methods*, 33(8):1733–1753.

[45] Richardson, C. (2018). *Microservice Patterns*. Manning Publications.

[46] Sayyadi, H. and Raschid, L. (2013). A graph analytical approach for topic detection. *ACM Transactions on Internet Technology (TOIT)*, 13(2):4.

[47] Sharma, P., Chaufournier, L., Shenoy, P., and Tay, Y. (2016). Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference*, page 1.

[48] Stellman, A. and Greene, J. (2005). *Applied software project management*. .ºReilly Media, Inc.".

[49] Taylor, I., Shields, M., Wang, I., and Harrison, A. (2007). The triana workflow environment: Architecture and applications. In *Workflows for e-Science*, pages 320–339. Springer.

[50] Terstyanszky, G., Kukla, T., Kiss, T., Kacsuk, P., Balaskó, Á., and Farkas, Z. (2014). Enabling scientific workflow sharing through coarse-grained interoperability. *Future Generation Computer Systems*, 37:46–59.

[51] U.S. National Library of Medicine, M. D. (2017).