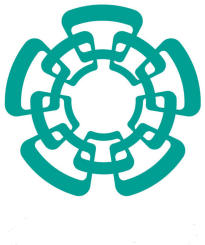


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Tamaulipas

Esquemas de almacenamiento de datos definidos por código

Tesis que presenta:

Diana Elizabeth Carrizales Espinoza

Para obtener el grado de:

Maestro en Ciencias en Ingeniería y Tecnologías Computacionales

Director de la Tesis:
Dr. José Luis González Compeán

Cd. Victoria, Tamaulipas, México Agosto, 2020

La tesis presentada por Diana Elizabeth Carrizales Espinoza fue aprobada por:

_____

Dr. Gregorio Toscano Pulido

Dr. Miguel Morales Sandoval

Dr. José Luis González Compeán, Director

Cd. Victoria, Tamaulipas, México, 06 de Agosto de 2020

Me gusta la expresión *"posibilidades perdidas"*.
Nacer significa estar obligado a elegir una época, un lugar y una vida.
Existir aquí, ahora, significa perder la posibilidad de ser otras innumerables personalidades potenciales.

-Hayao Miyazaki

Agradecimientos

- A mi madre y mi padre, que me han apoyado en cada una de mis decisiones a lo largo de este proceso, y me han brindado su comprensión y cariño incondicional.
- A mi director de tesis, el Dr. José Luis González Compeán por su dirección, experiencia, conocimiento, apoyo, tiempo, motivación y orientación durante la realización de este trabajo de investigación y esta etapa.
- A mi hermana y mi prima, Laura Carrizales y Perla Vázquez por su cariño, amor y apoyo incondicional.
- A mi amiga, Beatriz Pascual por su apoyo incondicional durante toda esta etapa.
- A Dante Sánchez, por su apoyo incondicional durante esta etapa, así como por todos lo momentos y aventuras compartidos.
- A mi hermano académico, Hugo Reyes por ser parte de este viaje compartiendo anécdotas, experiencias y conocimiento.
- Al Dr. Jesús Carretero Pérez por su apoyo, enseñanzas, conocimiento y dirección durante mi estancia en la Universidad Carlos III de Madrid y aún después de ello.
- A mis revisores, el Dr. Miguel Morales Sandoval y el Dr. Gregorio Toscano Pulido por sus valiosas enseñanzas y mejoras durante la revisión de este trabajo de investigación.
- A mis padres académicos, el M.C Santiago Gómez Carpizo y su esposa la Dra. Maria Esther Bautista Vargas por iniciarme en el camino de la investigación, y brindarme sus conocimientos y apoyo.
- A mis compañeros y amigos, Melissa Hinojosa y Edgard Díaz por todas las aventuras, momentos y experiencias vividas y compartidas durante esta etapa.
- A los chicos del grupo de investigación ARCOS de la Universidad Carlos III de Madrid por su apoyo durante mi estancia.
- 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

Índice de Algoritmos

Publicaciones

Carrizales, D., Sánchez-Gallegos, D. D., Reyes, H., Gonzalez-Compean, J. L., Morales-Sandoval, M., Carretero, J., & Galaviz-Mosqueda, A. (2019, Octubre). A Data Preparation Approach for Cloud Storage Based on Containerized Parallel Patterns. In International Conference on Internet and Distributed Computing Systems (pp. 478-490). Springer, Cham. [*Best paper Award*].

Sánchez-Gallegos, D. D., Carrizales-Espinoza, D., Reyes-Anastacio, H. G., Gonzalez-Compean, J. L., Carretero, J., Morales-Sandoval, M., & Galaviz-Mosqueda, A. (2020). From the edge to the cloud: A continuous delivery and preparation model for processing big IoT data. Simulation Modelling Practice and Theory, 102136.

Sánchez-Gallegos, D. D., Galaviz-Mosqueda, A., Gonzalez-Compean, J. L., Villarreal-Reyes, S.,Perez-Ramos, A.E., Carrizales, D., & Carretero, J. (2020, Abril). On the continuous processing of health data in edge-fog-cloud computing by using micro/nanoservice composition. In IEEE Access journal.

Resumen

Esquemas de almacenamiento de datos definidos por código

por

Diana Elizabeth Carrizales Espinoza
Unidad Tamaulipas
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2020
Dr. José Luis González Compeán, Director

Los sistemas de almacenamiento en la nube se están convirtiendo en una solución costo-beneficio *de facto* para las organizaciones que manejen grandes volúmenes de información. Sin embargo, estas soluciones no permiten a los usuarios finales cambiar, en forma dinámica, los parámetros que afectan a sus requerimientos no funcionales (seguridad, eficiencia, disponibilidad o confiabilidad por nombrar algunos). Esta limitación reduce la eficacia de soluciones de entrega continua tales como flujos de trabajo o flujos de procesamiento de datos que comúnmente son usadas por las organizaciones para compartir información con usuarios finales u otras organizaciones. Estas soluciones son populares porque se pueden crear y modificar en ambientes ubicuos bajo demanda e incluso se pueden destruir dependiendo de las necesidades del usuario final. En el presente documento de tesis, se describe el diseño, desarrollo e implementación de un modelo para la generación de esquemas de almacenamiento de datos definidos por código para soluciones de entrega continua de datos. Este modelo permite crear de forma flexible y dinámica, esquemas eficientes de almacenamiento considerando un conjunto reducido de requerimientos no funcionales definidos por organizaciones y/o usuarios finales. Los esquemas se presentan en forma de servicios que son automáticamente desplegables en la nube o en sistemas distribuidos de contenedores virtuales. Se implementó un prototipo basado en este modelo para crear esquemas de almacenamiento que fueron evaluados en tres estudios de caso: el primero basado en el manejo (que incluye la preparación de los datos, así como su almacenamiento) de datos organizacionales enviados a la nube, el segundo basado en el manejo de datos médicos en un sistema federado de distribución de contenidos y el tercero basado en escenarios dinámicos y heterogéneos

tales como borde, límite, nube (edge-fog-cloud) para el manejo de datos climatológicos provenientes del IoT. La evaluación experimental reveló la factibilidad y eficiencia de aplicar el modelo propuesto en la construcción de esquemas dinámicos de almacenamiento creados mediante secuencias de código.

Abstract

Data Storage Schemes Defined by Code

by

Diana Elizabeth Carrizales Espinoza
Unidad Tamaulipas
Center for Research and Advanced Studies of the National Polytechnic Institute, 2020
Dr. José Luis González Compeán, Advisor

Cloud storage systems are converting in de facto benefit-cost solutions to organizations that manage large volumes of data. Nevertheless, these solutions do not allow end-users to change, in a dynamic manner, the parameters that affect their non-functional requirements (security, efficiency, availability, or reliability to mention a few examples). This limitation reduces the efficacy of continuous delivery solutions, such as workflows or data processing flows that usually are used by organizations to share information with end-users or other organizations. These solutions are popular because they can be created and modified on-demand in ubiquitous environments, and even they could be destroyed based on end-user requirements. In this context, this thesis describes the design, development, and implementation of a model for the generation of code-defined data storage schemes for the continuous delivery of data solutions. This model enables the dynamically and flexibly creation of efficient storage schemes considering a reduced set of non-functional requirements defined by organizations and end-users. These schemes are presented in the form of automatically deployable services into the cloud or distributed system of virtual containers. It was implemented a prototype based on this model to create storage schemes, which were evaluated in three case studies: the first case study is based on the management (that includes the preparation and storage of data) of organizational data sent to the cloud, the second is based on the medical data management in a content distribution federated system, and the third is based on dynamical and heterogeneous scenarios, such as edge, fog, and the cloud for the management of climatologic data collected from IoT environments. The experimental evaluation revealed the feasibility and efficacy of applying the proposed model in the construction of dynamical storage schemes created by code sequences.

Nomenclatura

API	Interfaz de programación de aplicaciones.
BB	Bloque de construcción.
CD	Entrega continua.
CDF	Flujo continuo de datos.
CDN	Red de entrega de contenidos.
CI	Integración continua.
DAG	Grafo acíclico dirigido.
DBB	Bloque de construcción de datos.
ETL	Extracción, transformación y carga.
E/S	Entrada/Salida
FR	Requerimiento funcional.
IaC	Infraestructura definida por código.
IDA	Algoritmo de dispersión de información.
NFR	Requerimiento no funcional.
NIST	Instituto Nacional de Estándares y Tecnología.
PBB	Bloque de construcción de procesamiento.
SDS	Almacenamiento definido por software.
SO	Sistema operativo
VC	Contenedor virtual.
VM	Máquina virtual.

1

Introducción

En el presente capítulo se describen los antecedentes, motivación y planteamiento del problema abordado en esta tesis. Además, se presenta la hipótesis y objetivos planteados, así como la metodología de solución que se implementó para lograrlos.

1.1 Antecedentes y motivación

La producción de información ha crecido constantemente en años recientes creando un universo digital[1] que actualmente se encuentra en expansión debido a la incorporación de mercados emergentes en la producción de nueva información. Gantz y Reinsel [43, 44, 45, 105] estiman que el 75 % de la información dentro del universo digital es generada por individuos, y el 80 % de dicha información se encuentra bajo la responsabilidad de diversas organizaciones. Dicho crecimiento provocará una expansión exponencial en el tamaño del universo digital, por lo cual el IDC pronostica que de 33 zettabytes de datos existentes en 2018, el total de datos crecerá hasta los 175 zettabytes en 2025

[1] El universo digital es el conjunto de todos los datos digitales creados, replicados y consumidos.

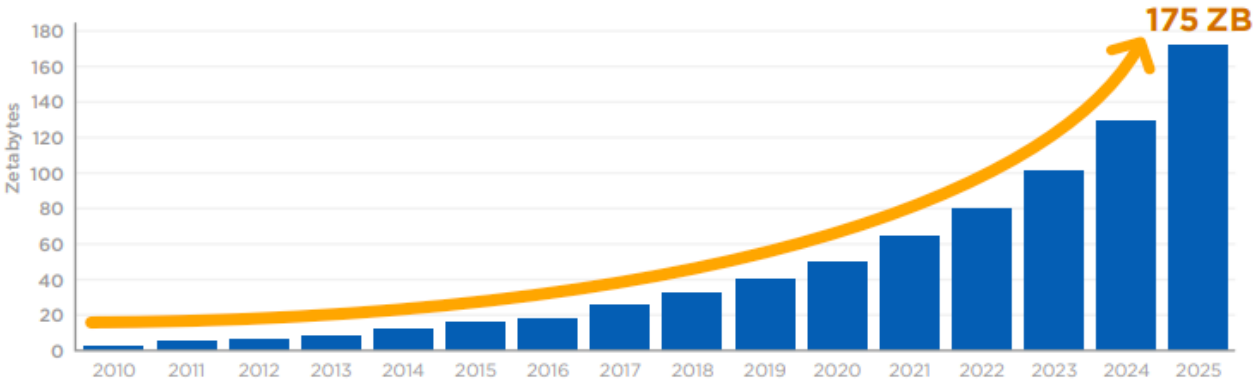


Figura 1.1: Crecimiento del universo digital. Fuente: *IDC's Digital Universe Study*, patrocinado por *Seagate*, Noviembre 2018 [105].

(ver Figura 1.1).

En este sentido, sus estudios concluyen que el universo digital duplica su tamaño cada dos años, por lo cual se prevé que la complejidad para administrar, almacenar, extraer valor y mantener seguros los datos también aumente. Por ejemplo, la Tabla 1.1 muestra la cantidad de datos que manejaban, hasta hace algunos años, diversas organizaciones, los cuales se encuentran en el orden de los terabytes hasta los zettabytes.

Tabla 1.1: Ejemplos de datos almacenados por organizaciones. Fuente: Stephen Kaisler et al. [60]. IEEE, 2013.

Conjunto de Datos/Dominio	Descripción
Gran colisionador de hadrones	13-15 petabytes en 2010
Comunicaciones en internet (CISCO)	667 exabytes en 2013
Redes sociales	Más de 12 Terabytes de tweets por día y creciendo. La media de retuits es de 144 por tweet.
Universo Digital	1.7 Zettabytes (2011) ->7.9 Zettabytes en 2015 (Gantz y Reinsel 2011)
Biblioteca Británica digital del Reino Unido	∼110 Terabytes por dominio para ser almacenados

Dado lo anterior, la mayoría de las organizaciones estiman que sus datos crecerán a lo largo de su ciclo de vida, a medida en que incrementan sus servicios, socios, clientes, negocios, instalaciones

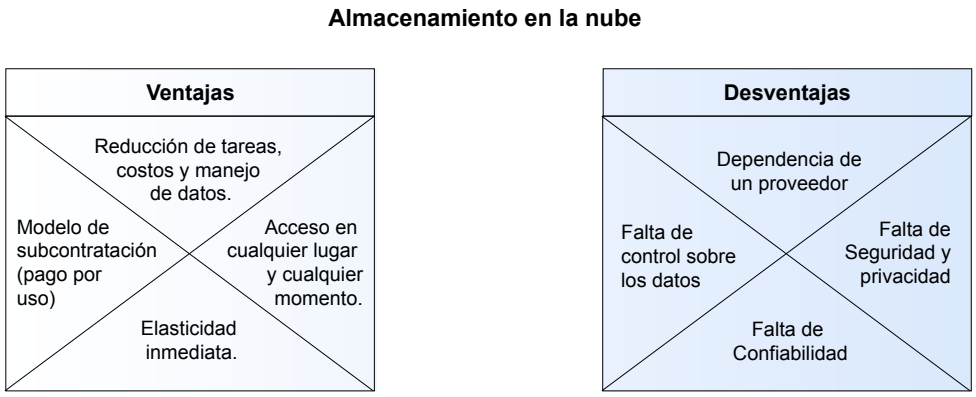


Figura 1.2: Ventajas y desventajas del almacenamiento en la nube.

y/o empleados. No obstante, muy pocas empresas consideran el impacto que tendrá dicha expansión sobre sus procesos de negocio o de entrega continua de resultados y/o productos a los usuarios finales, así como sobre sus sistemas de almacenamiento [60].

Los sistemas de almacenamiento tradicionales han sido construidos generalmente para ser utilizados por largos períodos de tiempo. Como resultado, dichos sistemas eran subutilizados en el inicio de su ciclo de vida y sobrecargados al final del mismo. Por lo tanto, estos sistemas solían producir incrementos en la probabilidad de fallas o disminución en la disponibilidad de dichos sistemas cuando aumentaba su utilización, así como su ubicación en la línea de tiempo en su ciclo de vida.

Esta tendencia ha orillado a las organizaciones a utilizar medios alternativos, tales como los servicios de almacenamiento en la nube, para preservar su información [4, 14, 46]. El almacenamiento en la nube se ha convertido en una serie de productos en línea populares entre las organizaciones y usuarios finales para archivar o respaldar información [14]. La Figura 1.2 resume las ventajas y desventajas del uso de la nube.

El almacenamiento en la nube se basa en un modelo de subcontratación de recursos (*outsourcing*) [13]. Este modelo les permite a las organizaciones delegar, al proveedor de almacenamiento en la nube, tareas tales como la entrega de contenidos a organizaciones, socios, clientes e incluso usuarios finales, así como la preservación de sus contenidos por largos períodos de tiempo [46].

Los proveedores del almacenamiento en la nube ofrecen a sus usuarios interfaces simples (por ejemplo, formularios y ventanas de monitoreo) para manejar y administrar sus datos siguiendo un modelo de pago por uso (*pay-as-you-go* en acepción anglosajona). Este modelo permite a los usuarios pagar únicamente por los recursos de almacenamiento que utiliza [122]. El pago de este servicio se define mediante el costo de almacenamiento, los intervalos de cobro, adquisición del mismo, y la previsibilidad de la demanda de almacenamiento [63].

Por ejemplo, si una organización tomara la decisión de almacenar sus datos de forma interna (utilizando sus propios recursos de hardware y software), tendría que realizar un estudio que estime su futura demanda de almacenamiento y por consiguiente adquirirla y administrarla de manera proactiva. Por otro lado, si la organización decide adquirir un servicio de almacenamiento basado en la nube, este le brindará la flexibilidad necesaria para aumentar su espacio de almacenamiento cuando sea requerido. Así mismo, esto brinda a la organización la opción de pagar solo por el volumen de almacenamiento que está utilizando realmente durante cada periodo [57].

Sin embargo, cuando una organización delega el manejo sobre su información a un tercero (proveedor de servicio), se delega también el control sobre dicha información [24]. Es por esto que este tipo de solución aún se sigue asociando a riesgos tales como inaccesibilidad [20], violaciones de seguridad [116] o dependencia con el proveedor, así como cuestiones relacionadas con la confiabilidad y seguridad de los datos [137], por nombrar algunos.

Por ejemplo, cuando una organización envía todos los datos a la nube, esta corre el riesgo de que sus datos sean analizados para diferentes fines, como es la monetización por parte de aplicaciones publicitarias, las cuales tienden a analizar y recopilar los datos de los consumidores para conocer el mercado al que se están dirigiendo y extender su red de publicidad (ejemplos de estas prácticas realizadas por compañías como Google, Amazon o Facebook han sido reportadas en la literatura) [24, 136].

Otro riesgo al que se enfrentan las organizaciones es el del costo que representan las interrupciones del suministro eléctrico en los centros de datos. Las corporaciones suelen sufrir los efectos secundarios

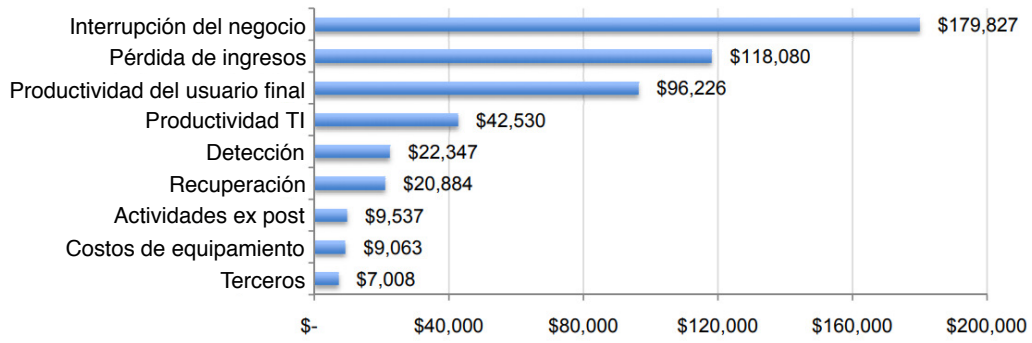


Figura 1.3: Costo promedio de interrupciones no planificadas en un centro de datos para nueve categorías. Fuente: Instituto Ponemon© [106], 2011.

de dichas interrupciones. Por ejemplo, Gartner [98] estima que estas interrupciones conforman un total de 87 horas por año aproximadamente, mientras que Emerson [99] reportó que el costo promedio de dichas interrupciones es de $5,600$ dólares por minuto. El Instituto de Procesos de TI [59], opina que el tiempo promedio de recuperación para una interrupción es de 200 minutos y el costo promedio por las interrupciones, en una estimación realizada por el Instituto de Ponemon entre 41 centros de datos, reporta que el costo asumido por los corporativos es de $\$20,735,602$ de dólares al año.

La Figura 1.3 muestra la variación que existe entre nueve tipos diferentes de costos. En ella se puede observar que el menor costo lo compone la categoría de participación de terceros, como lo son los consultores, los cuales ayudan en la resolución de dichos incidentes. Por otro lado, la categoría más costosa es aquella asociada con la interrupción del negocio, como lo son los daños a la reputación y la rotación de clientes [106].

Lo anterior provoca que los clientes que deleguen el manejo de sus datos a proveedores de almacenamiento en la nube no tendrán garantías (a menos que paguen por ellas) de que sus datos serán procesados para proveer los requerimientos no funcionales necesarios en escenarios no previstos por los contratos signados con dicho proveedor [136].

Las organizaciones, por tanto, deben prever que este tipo de riesgos se pueden eventualmente presentar y deberían prepararse para evitar perder control sobre los mismos. Lo anterior se vuelve crítico en ambientes dinámicos donde las organizaciones intercambian datos con socios u otras

organizaciones y entregan productos digitales a usuarios finales. De la misma forma, esto último también aplica a gobiernos, los cuales deben gestionar información de ciudadanos y/o contribuyentes.

Al respecto, se ha observado que diversas leyes de manejo de información y creación de repositorios digitales han sido promulgadas en diversos países para proteger, salvaguardar y asegurar la integridad de la información manejada o preservada por las organizaciones e instancias de gobierno [103].

Las soluciones de almacenamiento actuales en la nube no permiten a los usuarios finales cambiar los parámetros que afectan a sus requerimientos no funcionales [111] (seguridad, eficiencia, disponibilidad o confiabilidad por nombrar algunos) en forma dinámica y sin generar un nuevo contrato de servicio. Esta limitación reduce la eficacia de soluciones de entrega continua de datos que comúnmente son usadas por las organizaciones para gestionar datos. Estas soluciones son populares porque se pueden crear, modificar e incluso destruir bajo demanda dependiendo de las necesidades del usuario final, entre las que se encuentran los requerimientos no funcionales.

En el estado del arte se encuentran soluciones disponibles para garantizar la disponibilidad de los datos enviados a la nube mediante el establecimiento de distintas políticas de acceso y preservación de los datos [49, 120], así como diferentes niveles de seguridad para los mismos [108]. Sin embargo, cada una de estas soluciones procesa los datos enviados a la nube para cumplir con requerimientos funcionales específicos, mientras que los requerimientos no funcionales son establecidos por defecto o por contrato.

En este contexto, en el presente trabajo de tesis se propone el diseño, desarrollo e implementación de un modelo para la generación de esquemas de almacenamiento definidos por código para soluciones de entrega continua de datos. Se proyecta que dicho modelo permita crear de forma flexible y dinámica, esquemas eficientes de almacenamiento considerando un conjunto reducido de requerimientos no funcionales definidos por organizaciones y/o usuarios finales. Además, se pretende que los esquemas creados por este modelo se implementen en la forma de servicios, los cuales sean automáticamente desplegables en la nube o en sistemas distribuidos de contenedores virtuales.

1.2 Planteamiento del problema

En los últimos años han surgido los términos de "almacenamiento definido por software" (SDS, del inglés *Software Defined Storage* [18]) e "infraestructura definida por código" (IaC, del inglés *Infraestruture as Code* [89]). Las herramientas de SDS permiten a los usuarios generar políticas personalizadas para el manejo de recursos de almacenamiento virtualizados mediante software. La declaración de políticas permite al sistema de almacenamiento agrupar, manipular y administrar recursos mediante una solución unificada. Estas soluciones se crean de acuerdo con las necesidades declaradas por los usuarios finales. En cambio, las herramientas de IaC permiten a las organizaciones manejar recursos de hardware mediante el uso de un software especializado basado en esquemas declarativos en la forma de pseudocódigo, código, notaciones o *scripts*.

Sin embargo, tanto las herramientas actuales de SDS como IaC solo permiten a los usuarios finales construir soluciones basadas en requerimientos funcionales, los cuales incluyen, en el mejor de los casos, requerimientos no-funcionales que no son posibles cambiar (soluciones ad-hoc). Lo anterior, reduce la eficacia de las soluciones de entrega continua comúnmente utilizadas por las organizaciones para gestionar sus datos.

Algunos de los principales requerimientos no funcionales comúnmente usados en el manejo y almacenamiento de datos son, pero no limitados a, seguridad, confiabilidad, y eficiencia. Por ejemplo, los servicios de seguridad son requeridos cuando los datos e información son compartidos con múltiples usuarios a través de entornos no controlados y poco confiables, como es el caso de las nubes públicas (e.g., Amazon EC2, Google Cloud, o Digital Ocean), donde los usuarios pierden control sobre sus datos [12, 41, 42, 91, 102, 114, 119]. Mientras que los servicios de confiabilidad, son requeridos para mitigar problemas relacionados con fallas en la infraestructura donde los datos son almacenados, produciendo que estos no puedan ser accedidos [9, 29, 52, 59, 99, 106]. Finalmente, los servicios de eficiencia son clave para resolver problemas relacionados con los costos de almacenamiento y transporte de datos a través de diferentes entornos (e.g., desde la computadora

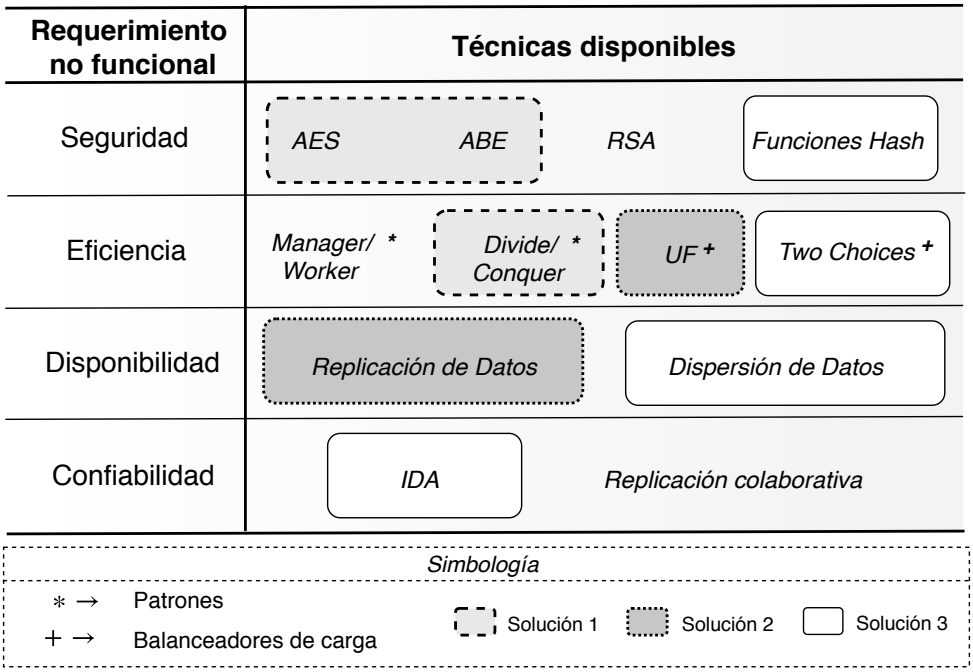


Figura 1.4: Ejemplo de tres soluciones de almacenamiento con diferentes requerimientos no funcionales.

de un usuario a la nube), así como para reducir la latencia producida al entregar los contenidos almacenados [3, 31, 65, 104].

La Figura 1.4 muestra tres soluciones de almacenamiento que contemplan distintos requerimientos no funcionales, así como las técnicas para cubrir dichos requerimientos. Por ejemplo, un conjunto de usuarios u organizaciones podría requerir asegurar sus datos mediante técnicas criptográficas de cifrado y descifrado de datos (seguridad), y mejorar el procesamiento mediante el uso de patrones de procesamiento de datos en paralelo (eficiencia). Otro conjunto de usuarios podría estar interesado en mejorar el balanceo de carga entre sus equipos de almacenamiento (eficiencia) y crear réplicas de su información (disponibilidad). Finalmente, un tercer conjunto de usuarios podría estar interesado en verificar la integridad de su información mediante funciones hash (seguridad), distribuir sus datos de forma equitativa entre los recursos de almacenamiento disponibles mediante técnicas de balanceo de carga (eficiencia), segmentar sus datos y distribuirlos en diferentes nodos de almacenamiento (confiabilidad) y asegurar que sus datos sean confiables.

Figura 1.5: Desafíos presentados al desarrollar soluciones que cumplimenten con requerimientos no funcionales.

Cuando se desarrollan soluciones para cumplir con los requerimientos no funcionales de los usuarios en forma dinámica, se presentan problemas tales como flexibilidad, eficiencia y accesibilidad a los datos (ver Figura 1.5).

El problema de flexibilidad se presenta porque las soluciones son típicamente diseñadas para usar un conjunto de requerimientos no funcionales o un conjunto de parámetros de dichos requerimientos, es decir soluciones ad-hoc que no pueden ser reutilizadas por organizaciones con diferentes requerimientos. Para poder llevar a cabo las modificaciones de dichos parámetros se suele modificar el código fuente de las aplicaciones o algoritmos que proveen dichos requerimientos no funcionales. Estos sistemas son esencialmente estáticos, porque no poseen la habilidad para realizar este tipo de procedimiento en tiempo de ejecución, lo cual evita que las soluciones se adapten al cambio de parámetros en forma dinámica. En este sentido, otro factor a considerar es que no siempre se cuenta con el código fuente de dichos algoritmos o aplicaciones, por lo cual realizar algún cambio en ellos resulta imposible.

Mientras, que el problema de eficiencia se observa cuando existen componentes en el sistema o servicios que generan cuellos de botella, los cuales incrementan el tiempo de respuesta de la

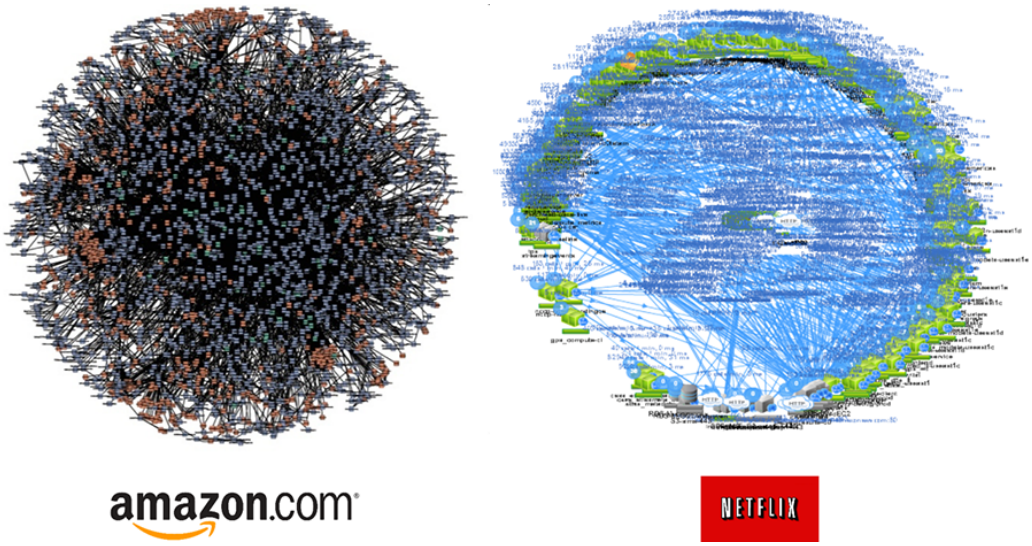


Figura 1.6: Redes de servicios (Amazon y Netflix). Fuente: OEX Divante [62].

solución. En este sentido, identificar los componentes del código que puedan ser desplegados de forma paralela resulta ser la solución más obvia. Sin embargo, la paralelización de las soluciones requiere que los desarrolladores exploren o conozcan a fondo el código de las mismas, invirtiendo tiempo en paralelizar y gestionar los recursos necesarios para que dichas soluciones mejoren el rendimiento de las aplicaciones.

El problema de accesibilidad a los datos se observa cuando las soluciones deben intercambiar datos. En tal escenario los componentes de dichas aplicaciones deben conocer las rutas de acceso a los mismos. Para intercambiar datos, es necesario que los bloques de construcción de una solución interactúen entre ellos. Esta interacción entre bloques de construcción y datos subyacentes se ve afectada por el modelo y topología de despliegue de los bloques, así como por los tiempos de servicio de la infraestructura. Por ejemplo, en la Figura 1.6 se observan dos modelos de despliegue de bloques de construcción utilizados por dos organizaciones proveedores se servicios por Internet: Amazon y Netflix.

Como se puede observar en la Figura 1.6, en Amazon todos los servicios se encuentran centralizados. Lo anterior se debe a que Amazon tiene control sobre su propia infraestructura

desplegada en diferentes centros de datos, por lo cual conoce el tiempo de servicio de cada una de sus computadoras, permitiendo el despliegue de sus servicios siguiendo un modelo jerárquico conocido como arquitectura de N-capas [30]. Esto permite establecer un control sobre la infraestructura, así como un control sobre el lugar en el que los bloques de construcción/servicios serán desplegados. Sin embargo, en la práctica las organizaciones desconocen en donde son desplegados sus servicios, y mucho menos el tiempo de servicio de las computadoras donde son ejecutados. Esto se debe a que por lo general se suele utilizar el modelo de subcontratación de infraestructura, los cuales ofrecen diferentes paradigmas como la nube. Este es el caso de Netflix, que cuenta con una infraestructura distribuida en la cual sus datos se encuentran almacenados en múltiples centros de datos distribuidos subcontratados. En dichos centros de datos se coloca el contenido altamente demandado por una región, evitando que los usuarios tengan que pasar por múltiples capas para poder acceder al contenido que deseen. Aunque lo anterior reduce el tiempo de servicio, al contar con una infraestructura cambiante, es necesario mantener un control sobre la interacción de cada uno de sus centros de almacenamiento y el intercambio de datos que existe entre ellos.

Actualmente, las redes de distribución de contenido (CDN, del inglés *Content Delivery Network* [26, 46]) hacen uso de los SDS para generar esquemas de almacenamiento basados en los conceptos de entrega continua (CD, del inglés *Continuos Delivery* [23, 28, 40, 112]), y flujo de datos continuo (CDF, del inglés *Continuos Data Flow* [129]), los cuales aseguren que sus componentes se encuentren funcionando constantemente, y que se garantice la ingesta de datos respectivamente. Sin embargo, los sistemas CDN, los sistemas de CD e integración continua (CI, del inglés *Continuos Integration* [40, 112]) y los sistemas de almacenamiento se encuentran desacoplados y en cada caso las organizaciones deben definir sus requerimientos no funcionales en tiempo de configuración, lo cual implica que las definiciones de restricciones diseñadas para una CDN no apliquen automáticamente a los sistemas de almacenamiento de CD y CI. Lo anterior podría resultar en riesgos al momento en el que los datos sean enviados a la nube.

Tomando en cuenta la problemática antes mencionada, en el presente tema de tesis, se propuso

el desarrollo de un modelo para la generación de esquemas de almacenamiento definidos por código para CD. En este contexto, el almacenamiento definido por código permite gestionar una serie de requerimientos no funcionales para almacenar datos en espacios de almacenamiento distribuido mediante la generación de secuencias de código. A través de las secuencias de código es posible realizar el acoplamiento de diferentes piezas de software (por ejemplo, aplicaciones y algoritmos) para crear distintos esquemas de almacenamiento de datos que cumplimenten con los requerimientos no funcionales de los usuarios, y que a su vez sean desplegables en distintos ambientes (por ejemplo, la nube). Además, mediante las secuencias de código es posible actualizar o retroceder las piezas de software a distintas versiones del mismo cuando sea requerido. Dicho modelo fue diseñado para permitir la creación de esquemas eficientes de almacenamiento para soluciones de CD. Para solventar los problemas de eficiencia de los esquemas de almacenamiento, se contempla la utilización de patrones de paralelismo basados en contenedores virtuales para la reducción del impacto de los cuellos de botella que aparezcan en el intercambio de datos, lo cual se espera mejoren la accesibilidad de los datos para las soluciones de CD.

1.3 Hipótesis

A partir de las problemáticas identificadas en la Sección 1.2, se establecen las siguientes premisas:

Es posible crear, mediante secuencias de código[2] (escritas por el usuario e interpretadas por un sistema), servicios de almacenamiento que se adecuen a los requerimientos no funcionales (esencialmente eficiencia, seguridad y confiabilidad), en el caso en el que se cumplan las siguientes premisas:

1. Se define un conjunto de sentencias que represente componentes de un sistema de

[2]Las secuencias de código permiten a las organizaciones crear y manejar los esquemas de almacenamiento de datos. Una secuencia de código es un conjunto de instrucciones escritas utilizando un lenguaje programación o de desarrollo de software. En este sentido, un esquema se codifica utilizando distintas abstracciones, las cuales se encuentran descritas en la Sección 4.5.

almacenamiento en la forma de código;

2. En la secuencia de código se asigna una notación que represente requerimientos no-funcionales a una codificación dada;

3. Un gestor convierte dicha representación en un esquema de almacenamiento de datos que pueda materializarse en la forma de un servicio.

En función de estas premisas se formulan las siguientes preguntas de investigación:

1. ¿Cómo se pueden crear dinámicamente soluciones de almacenamiento para múltiples usuarios con diferentes requerimientos no funcionales?

2. ¿Cómo un sistema de almacenamiento heterogéneo[3] como el antes mencionado puede manejar la carga de trabajo en forma balanceada?

A partir de estas preguntas se establece la siguiente hipótesis:

Los esquemas de almacenamiento definidos por código y notación de requerimientos no-funcionales materializados en la forma de esquemas de distribución, permitirán la creación de sistemas de almacenamiento dinámicos bajo demanda.

1.4 Objetivos generales y particulares

General

Obtener un modelo para la construcción de esquemas dinámicos de almacenamiento de datos definidos por código.

Particulares

[3]Un sistema de almacenamiento heterogéneo, es un sistema con múltiples componentes de almacenamiento que comunican y coordinan acciones para aparecer como un sistema coherente y único para el usuario final. Los dispositivos de almacenamiento tienden a tener diferentes capacidades. Además, el ancho de banda disponible de cada componente puede ser diferente dependiendo del tráfico de usuarios actual en cada uno de ellos [61].

1. Definir una notación que permita la construcción de sistemas de almacenamiento definidos por código.

2. Diseñar una arquitectura de construcción, despliegue y operación implícita de esquemas dinámicos de almacenamiento de datos definidos por código.

1.5 Metodología

En esta Sección se describe la metodología a seguir para alcanzar los objetivos planteados en la Sección 1.4. Para ello, se presenta un acercamiento conceptual a la solución de la problemática planteada en la Sección 1.2. Dicha problemática será utilizada como guía para desarrollar los productos finales de la tesis. Posteriormente, se presenta la metodología de solución en donde se describen las etapas en las que se dividirán las actividades para llevar a cabo la tesis.

1.5.1 Solución conceptual

Con la finalidad de modelar la solución propuesta en el presente trabajo de tesis, se han definido las siguientes etapas de construcción del modelo para la generación de esquemas de almacenamiento definidos por código:

1. Definición y formalización de un modelo de construcción de bloques modulares de almacenamiento:

   En esta etapa se hará la definición y formalización del modelo de construcción de bloques modulares de almacenamiento. Para ello, se realizará una taxonomía en la que se identificarán todos los aspectos funcionales y no funcionales que estarán disponibles en el sistema, así como la definición para la invocación de cada uno de ellos. Además, se creará un repositorio con dichos requerimientos (funcionales y no funcionales), los cuales tendrán que estar encapsulados en contenedores virtuales para su uso.

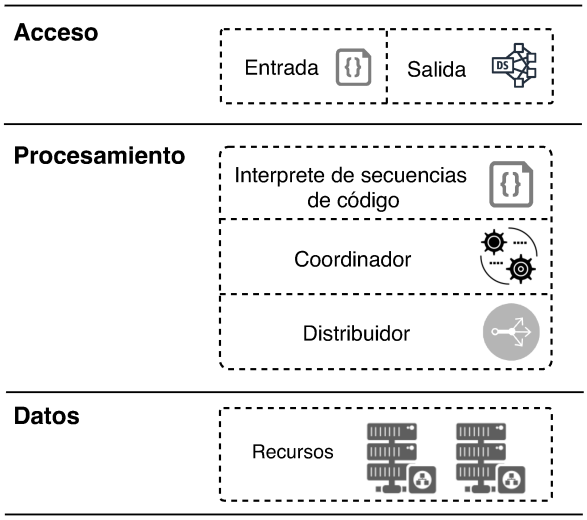


Figura 1.7: Aproximación conceptual de la arquitectura de construcción de esquemas de almacenamiento definidos por código.

2. Diseño de estructuras de codificación, control y sincronización para bloques modulares de almacenamiento:

   En esta etapa se realizará la definición y diseño de las estructuras de codificación, control y sincronización para bloques modulares de almacenamiento. Para ello, es necesario hacer la codificación para la gestión de dichos bloques, así como la codificación de los requerimientos funcionales y no funcionales, es decir la generación de secuencias de código que ayuden a cumplimentar con dichos requerimientos. Además, se diseñarán los coreógrafos y orquestadores que permitan mantener el control de los bloques modulares de almacenamiento.

3. Despliegue:

   En esta etapa se realizará el despliegue de la solución en la infraestructura de software que se indique en la fase de diseño de estructuras de codificación, control y sincronización para bloques modulares de almacenamiento.

En la Figura 1.7 se muestra el diseño conceptual del modelo propuesto, el cual se encuentra compuesto de tres capas: *i)* acceso, *ii)* procesamiento y *iii)* datos.

- Capa de acceso:

  En la capa de acceso se encuentran las interfaces de entrada y salida. Recibe como entrada una secuencia de código, y se entrega como salida un esquema de almacenamiento.

- Capa de procesamiento:

  En la capa de procesamiento se encuentran tres componentes:

  - Interprete de la secuencia de código:

    El intérprete de secuencias de código recibe la secuencia de código desde la capa de acceso para su posterior interpretación y generación del esquema definido. La Figura 1.8 muestra la taxonomía para la construcción de secuencias de código. Como se puede observar, los esquemas generados pueden ser de dos tipos: esquemas de preparación/entrega y esquemas de recuperación. Para mejorar la eficiencia de dichos esquemas, se hace uso de patrones los cuales se construyen utilizando bloques de construcción de procesamiento que tienen interfaces de entrada y salida para su encadenamiento (memoria, sistema de archivos o red), y que toman datos desde bloques de construcción de datos.

  - Coordinador:

    La Figura 1.9 muestra la taxonomía del coordinador, el cual se encarga de la creación, preparación, configuración, despliegue y ejecución de los esquemas generados, así como de la asignación y acoplamiento de las entradas y salidas, y el manejo de mensajes.

  - Distribuidor:

    El distribuidor se encarga de la distribución de datos, tareas y peticiones en los esquemas de almacenamiento generados.

- Capa de datos:

  En esta capa, se encuentra definida toda la infraestructura de los recursos de almacenamiento donde se desplegarán los esquemas generados.

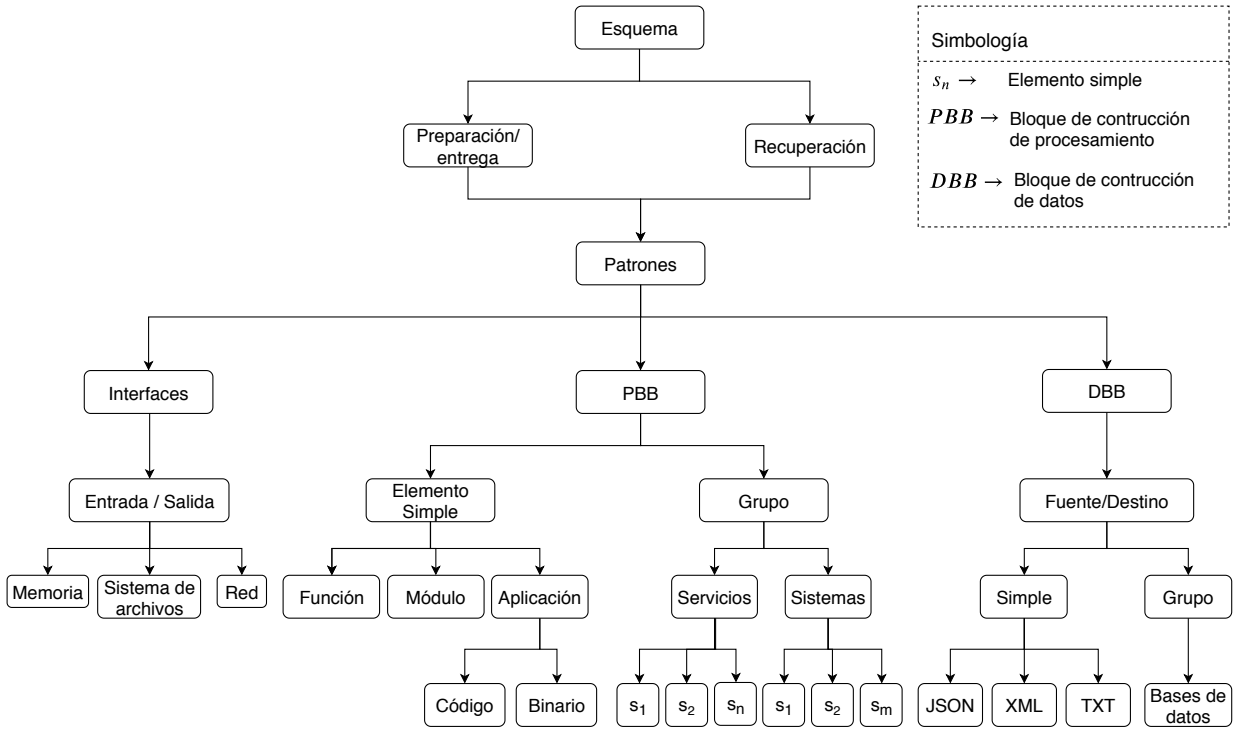


Figura 1.8: Taxonomía para la construcción de secuencias de código.

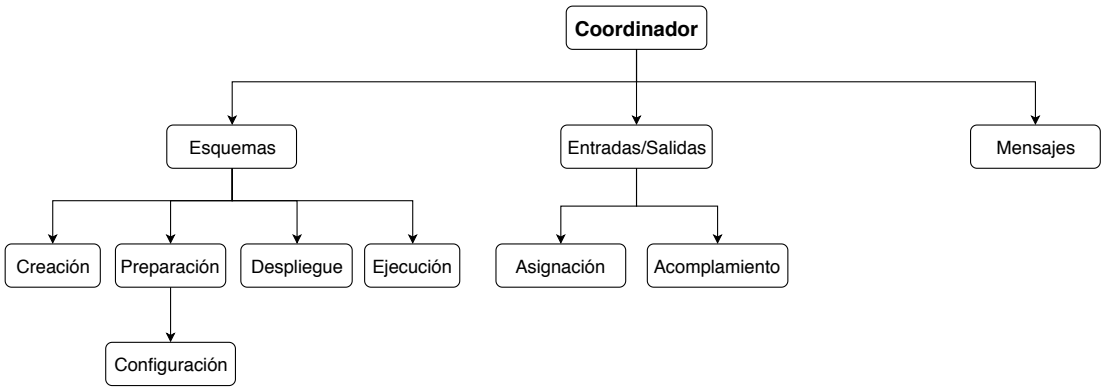


Figura 1.9: Taxonomía del coordinador.

1.5.2 Metodología de la solución

Para llevar a cabo la solución propuesta, se definieron las siguientes etapas:

1. Estado del arte/conceptualización.

   En esta etapa se realizó la recopilación de los trabajos relacionados con la problemática

descrita y la solución propuesta. Esto se hará mediante la recolección y análisis de distintas fuentes bibliográficas. Además, se definirán los conceptos de mayor relevancia para ayudar a la comprensión del trabajo de tesis y la solución propuesta en ella.

2. Modelado y diseño.

   En esta etapa se realizó la definición y formalización del modelo de construcción de bloques modulares de almacenamiento. Para ello se definirá una taxonomía que incluya los requerimientos funcionales y no funcionales con los que contará el modelo. Así mismo, se especificarán las entradas y salidas para cada uno de ellos.

   Además, se realizó el encapsulamiento en contenedores virtuales de las aplicaciones que ayuden a cumplir con los requerimientos funcionales y no funcionales identificados en la taxonomía. Los contenedores virtuales serán almacenados en un repositorio para su posterior uso al momento de generar los esquemas de almacenamiento.

3. Diseño de estructuras de codificación, control y sincronización para bloques modulares de almacenamiento.

   Durante esta etapa, se diseñó la estructura de codificación para la invocación de bloques modulares de almacenamiento. En ella, se realizó la codificación para la gestión del repositorio de bloques modulares de almacenamiento mediante la codificación de los requerimientos funcionales y no funcionales. Dentro de los requerimientos funcionales se contempla el manejo de metadatos de almacenamiento, redes de distribución de contenido, nodos de almacenamiento y distribución de carga. Por otro lado, dentro de los requerimientos no funcionales se contempla la seguridad, confiabilidad y eficiencia.

   Además, durante esta etapa se realizó la definición de estructuras de control y sincronización para los bloques modulares de almacenamiento. Para ello, se diseñaron coreógrafos y orquestadores que permitan la sincronización y control de dichos bloques.

   De la misma forma, en esta etapa se definió la notación para la codificación de secuencias de

código, a partir de las cuales se generarán los esquemas de almacenamiento.

4. Implementación y prototipado.

   Esta etapa se divide en dos partes:

   - Desarrollo del prototipo basado en el método propuesto.

     En esta etapa se desarrollará un prototipo utilizando el modelo de construcción de bloques modulares de almacenamiento, el cual contemplará las características funcionales y no funcionales.

   - Despliegue del prototipo.

     En esta etapa el prototipo diseñado será desplegado para validar su funcionamiento, y así encontrar posibles errores de programación o diseño para su inmediata corrección.

5. Evaluación experimental.

   En esta etapa se evaluará el prototipo desarrollado mediante la variación de sus parámetros de entrada.

   Se implementó un prototipo basado en este modelo para crear esquemas de almacenamiento que fueron evaluados en tres estudios de caso: el primero basado en el manejo (que incluye la preparación de los datos, así como su almacenamiento) de datos organizacionales enviados a la nube. Dado que en este estudio de caso se manejan datos organizacionales y personales, en los cuales se pueden encontrar copias de un mismo archivo, para este estudio de caso se construyó un sistema de almacenamiento que incluye un sistema de deduplicación (eficiencia), confiabilidad (IDA) y seguridad. Lo anterior con el objetivo de evaluar el impacto del sistema de deduplicación en el resto del esquema, y así en el mejor de los casos reducir el volumen de datos a almacenar. El segundo caso se basa en el manejo de datos médicos en un sistema federado de distribución de contenidos. Dado que en este estudio de caso se manejan datos médicos los cuales son importantes (para el personal asociado a áreas de salud y pacientes) y ayudan a la

toma de decisiones, es necesario mantenerlos seguros (por ejemplo, cifrándolos), y al ser datos de gran tamaño (superior a 1 GB) es necesario aplicar métodos que permitan eficientizar el transporte de estos (por ejemplo, aplicando técnicas de compresión para reducir su tamaño). Finalmente, el tercer caso se basa en escenarios dinámicos y heterogéneos tales como borde, límite, nube (edge-fog-cloud) para el manejo de datos climatológicos provenientes del IoT. El objetivo de este estudio de caso fue evaluar el costo de los componentes del esquema de preparación al procesar distintos volúmenes de datos, así como evaluar si el rendimiento de las soluciones de preparación/recuperación de datos desarrollada en este estudio es competitivo contra las soluciones encontradas en el estado del arte.

Dentro de las métricas consideradas en esta evaluación se encuentra el rendimiento de las soluciones generadas, así como su tiempo de servicio y respuesta. El tiempo de respuesta se evaluará como la sumatoria del tiempo de inicialización, tiempo de despliegue y tiempo de operación.

6. Conclusiones.

   En esta etapa se presentarán las conclusiones obtenidas a partir de los resultados recabados durante la experimentación que se efectuará para evaluar la solución propuesta para el presente tema de tesis.

1.6 Organización de la tesis

La presente tesis está conformada por seis capítulos, los cuales se encuentran organizados de la siguiente manera. En el Capítulo 2 se realiza una descripción del fundamento teórico requerido para el desarrollo de esta tesis. El Capítulo 3 presenta una revisión del estado del arte respecto a los sistemas de almacenamiento basados en la nube, así como de soluciones punto-apunto y flujos de trabajo. El Capítulo 4 describe el modelo propuesto, así como sus componentes. En el Capítulo 5 se describe la metodología a seguir para evaluar el prototipo construido. Además, se describen los estudios de caso

conducidos para evaluar el modelo propuesto, y se discuten los resultados obtenidos. Finalmente, en el Capítulo 6 se muestran las conclusiones obtenidas durante el desarrollo del presente trabajo de tesis.

2

Marco teórico

En la presente sección se describen los conceptos clave involucrados en el desarrollo del presente trabajo de tesis.

2.1 Requerimientos de software

Según el Glosario estándar de terminología de ingeniería de software, publicado en 1990 por la IEEE, un requerimiento es una condición o capacidad necesitada por un usuario para resolver un problema o alcanzar un objetivo. Una condición o capacidad es aquella que debe ser reunida o poseída por un sistema o un componente del sistema para satisfacer un contrato, estándar, especificación, o alguna otra formalidad impuesta en el documento/contrato [58]. Existen dos tipos de requerimientos de software: funcional y no-funcional (ver Figura 2.1).

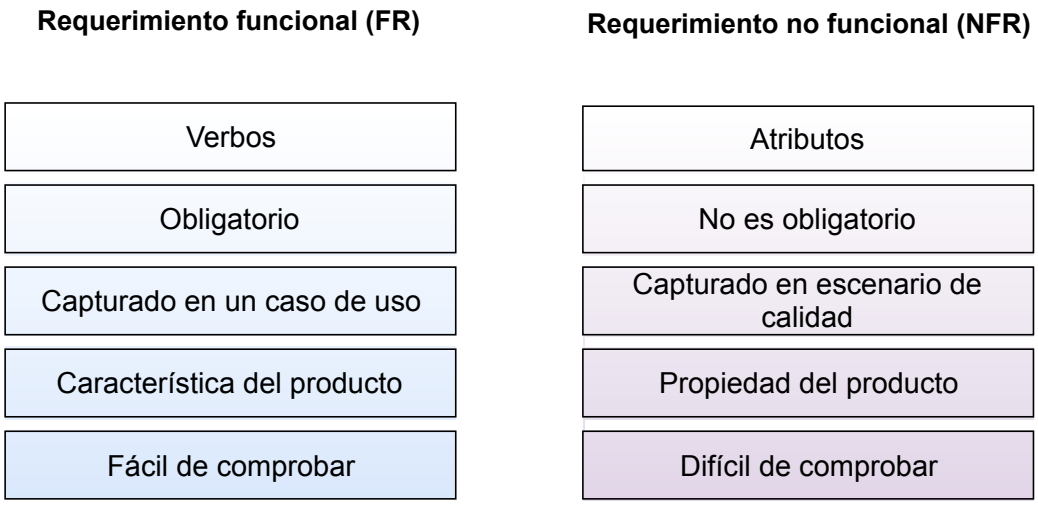


Figura 2.1: Diferencia entre requerimientos funcionales y requerimientos no funcionales.

2.1.1 Requerimientos funcionales

Un requerimiento funcional (ver Figura 2.1, columna izquierda) es aquel que especifica una función que el sistema o componente del sistema debe ser capaz de realizar. Estos, definen el comportamiento del sistema, es decir, el proceso fundamental de transformación que los componentes de software y hardware del sistema realizan en las entradas para producir las salidas [25].

De esta forma, un requerimiento funcional se define como una tarea, acción o actividad necesaria que debe ser realizada por el sistema para funcionar de forma correcta [67].

2.1.2 Requerimientos no funcionales

Un requerimiento no funcional (ver Figura 2.1, columna derecha), no es aquel que describe lo que hará el software , sino que describe el cómo lo hará, por ejemplo, los requisitos de rendimiento del software, los requisitos de la interfaz externa del software, las restricciones de diseño del software y los atributos de calidad del software. Los requisitos no funcionales son difíciles de probar, por lo tanto, generalmente se evalúan de forma subjetiva [25].

Figura 2.2: Representación conceptual de la metáfora ball-into-bins.

2.2 Sistemas de almacenamiento de tradicionales

Cualquier dispositivo en el que la información pueda ser mantenida o preservada, es considerado un sistema de almacenamiento tradicional. Las computadoras actuales poseen dos tipos principales de almacenamiento: la memoria RAM (por sus siglas en inglés, *random access memory*), y los discos duros u otros dispositivos de almacenamiento externo. Algunos otros tipos de almacenamiento incluyen la memoria ROM (por sus siglas en inglés, *read-only memory*) y los búferes [1].

Un dispositivo de almacenamiento es un aparato para almacenar datos de manera permanente o semipermanente en una computadora. La memoria RAM es considerada la memoria principal de cualquier computadora, mientras que el resto de los dispositivos de almacenamiento son considerados como dispositivos de almacenamiento auxiliares [1].

2.2.1 Metáfora balls-into-bins

Para poder comprender qué es un sistema de almacenamiento, es posible utilizar la metáfora balls-into-bins (ver Figura 2.2). Dicha metáfora, consta de dos componentes principales: las *balls*

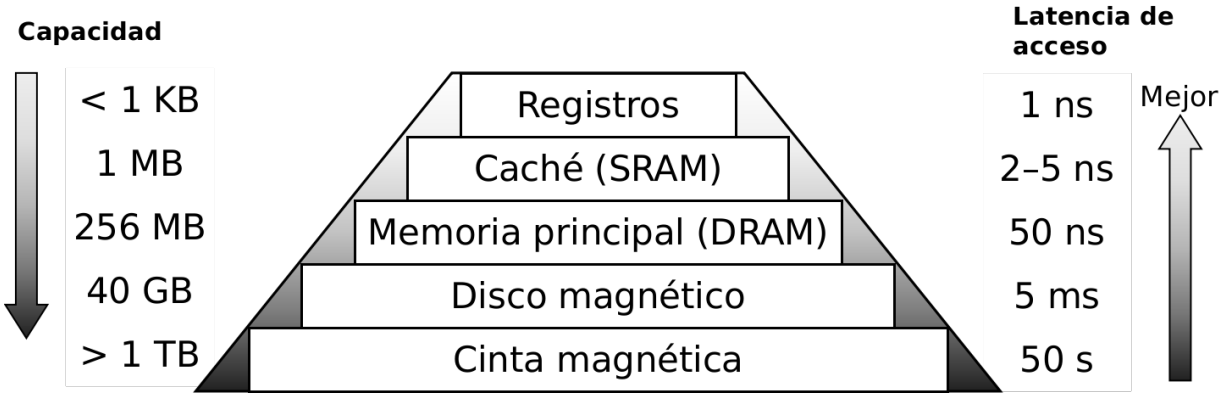


Figura 2.3: Jerarquía de memoria.

(esferas o elementos), que pueden ser archivos, bloques, objetos, usuarios, por mencionar algunos; y los *bins* (las cajas), que son aquellas entidades de almacenamiento a cargo de recibir *balls* (las particiones, carpetas compartidas y la nube son algunos ejemplos) [88].

En estos esquemas de almacenamiento se realizan operaciones funcionales basadas en mapas *ball-bin*. En donde se realizan dos operaciones principales: la colocación, en la cual se busca un bin para un *ball*; y la localización, en la cual se busca un *Ball* en un conjunto de *bins* [88].

2.2.2 Jerarquía de memoria

Dado que la memoria principal tiene un tamaño reducido, la memoria en una computadora se organiza de forma jerárquica (ver Figura 2.3). En el tope de dicha jerarquía, se encuentran las memorias de alta velocidad y poca capacidad (por ejemplo, la memoria caché y la memoria RAM), mientras que, en los niveles más bajos, se utilizan memorias permanentes de alta capacidad y baja velocidad (por ejemplo, discos duros y sistemas de almacenamiento externos). La latencia es baja en los niveles superiores de la jerarquía, y aumenta en los niveles inferiores; mientras que la capacidad tiene el efecto inverso, es decir es grande en los niveles inferiores de la jerarquía y pequeña en los niveles superiores [19].

2.3 Cómputo en la nube

El Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés *National Institute of Standards and Technology*) define el cómputo en la nube como "un modelo que permite el acceso ubicuo, conveniente y bajo demanda a una piscina de recursos computacionales configurables que pueden ser rápidamente provistos y lanzados con el mínimo esfuerzo de manejo o interacción con el proveedor de servicio" [80]. Dicho modelo cuenta con cinco características esenciales: autoservicio bajo demanda, amplio acceso a la red, una piscina de recursos heterogéneos, rápida elasticidad y un servicio medido. Los modelos de servicio a los que se tiene acceso a través del cómputo en la nube son Software como Servicio (SaaS, del inglés *Software as a Service*) e Infraestructura como Servicio (IaaS, del inglés *Infraestructure as a Service*), los cuales se despliegan en cuatro modelos de nube diferentes: pública, comunitaria, privada e híbrida [54].

Sistemas de almacenamiento en la nube. Un sistema de almacenamiento en la nube es considerado como un sistema de almacenamiento distribuido a gran escala, el cual consta de muchos servidores de almacenamiento independientes. La solidez de los datos es un requisito importante para los sistemas de almacenamiento, y en este sentido el almacenamiento de datos en un sistema de nube de un tercero suele causar una gran preocupación sobre la confidencialidad de los datos [68].

2.4 Virtualización

La virtualización tiene un papel importante en el cómputo en la nube ya que permite brindar almacenamiento virtual a los clientes de la nube, así como de algunos otros recursos computacionales [131]. Virtualizar puede definirse como simular hardware y/o software sobre un equipo físico, creando imágenes que puedan ser utilizadas en diferentes equipos físicos. De igual manera, permite la escalabilidad de los recursos de manera virtual, dado que posibilita la partición

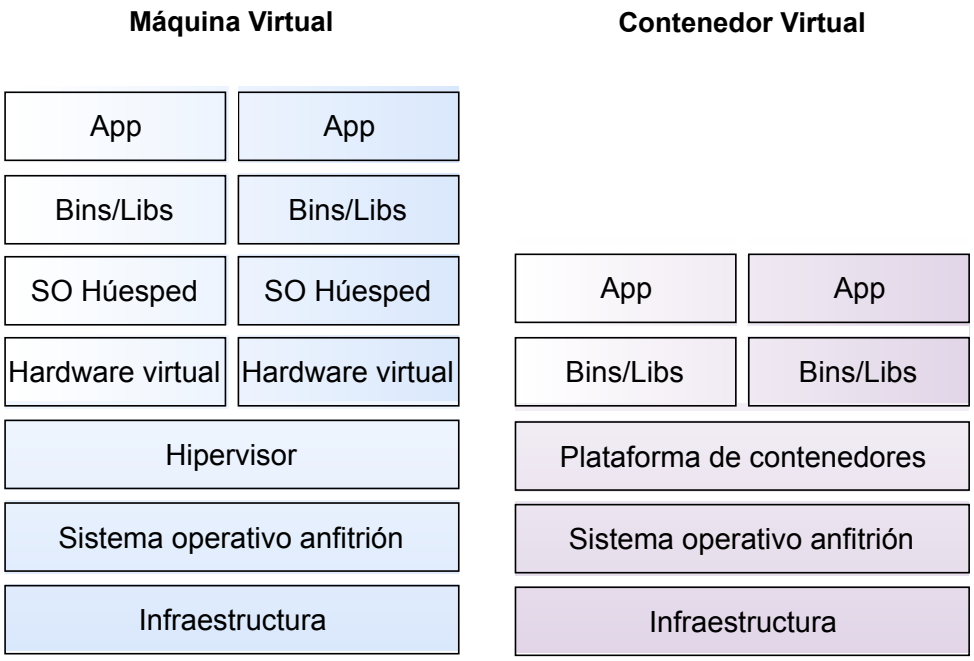


Figura 2.4: Comparación entre una máquina virtual y un contenedor virtual [134].

(de forma lógica) de un equipo físico [71]. Existen dos tipos de virtualización: virtualización a nivel de hardware y virtualización a nivel de sistema operativo.

La Figura 2.4 muestra los componentes de una máquina virtual (VM, por sus siglas en inglés *Virtual Machine*) (virtualización a nivel de hardware) y un contenedor virtual (VC, del inglés *Virtual Container*) (virtualización a nivel de sistema operativo). Como se observa en la Figura 2.4, el número de componentes que se encapsulan en un VC es menor en comparación con una VM, lo cual hace que estos sean de menor tamaño, y permite que estas puedan ser transportadas y desplegadas en diferentes infraestructuras.

2.4.1 Virtualización a nivel de hardware

Implica virtualizar el hardware en un equipo físico y crear máquinas virtuales (VM, del inglés *Virtual Machine*) que proporcionan la abstracción de una máquina física. Para llevar a cabo la virtualización, es necesario ejecutar un hipervisor (monitor de máquina virtual), el cual permite la

emulación de hardware virtual como la CPU, memoria, periféricos de E/S y dispositivos de red para cada VM. En cada VM se ejecuta un sistema operativo independiente, en donde se ejecutarán las aplicaciones [113] (ver Figura 2.4, columna izquierda).

2.4.2 Virtualización a nivel de sistema operativo

También conocida como virtualización basada en contenedores, implica la virtualización solo del núcleo (*kernel*) del sistema operativo, en lugar del hardware. En este tipo de virtualización las VMs, se denominan contenedores virtuales (VC, del inglés *Virtual Container*), y cada VC encapsula un grupo de procesos aislados de otros VCs y procesos del sistema huésped. El núcleo del sistema operativo es el responsable de organizar la compartición de los recursos de hardware como lo son la memoria, la CPU y la red por mencionar algunos [82, 113] (ver Figura 2.4, columna derecha).

2.4.3 Plataformas de gestión de Contenedores virtuales

En esta sección se describen diferentes plataformas que permiten la gestión de VCs, como lo son: *Linux VServer*, *OpenVZ*, *Linux Containers* (LXC) y *Docker*.

2.4.3.1. Linux-VServer

Linux-VServer fue la primera implementación basada en VCs para Linux. Los VCs de Linux-VServer realizan tareas tales como: el aislamiento de procesos, el aislamiento de redes, y el aislamiento de la CPU. Cada VC implementa sus propias capacidades en el núcleo de Linux [32].

El aislamiento de los procesos se lleva a cabo a través de un identificador de procesos, que oculta dentro del VC todos los procesos externos del sistema operativo, y prohíbe la comunicación entre diferentes procesos de diferentes VCs.

2.4.3.2. OpenVZ

OpenVZ ofrece algunas funcionalidades similares a LinuxVServer. La principal diferencia con los VCs de Linux-VServer, es que estos tienen su propio conjunto de recursos aislados, identificando cada contenedor virtual mediante un identificador de proceso [121]. En OpenVZ, cada contenedor tiene sus propios segmentos de memoria compartida, semáforos y mensajes, debido a la capacidad del espacio de nombres del núcleo IPC.

2.4.3.3. LXC

A diferencia de Linux-VServer, en LXC la gestión de recursos sólo está permitida a través de *cgroups*[1]. Desde el punto de vista de la red, *Cgroups* define la configuración de espacios de nombres de red. El sistema usa múltiples controladores sobre el programador estándar de la CPU de Linux [70].

2.4.3.4. Docker

Docker[2] es un proyecto de código abierto que provee un sistema para el despliegue automático de aplicaciones dentro contenedores virtuales. Los contenedores Docker, son una extensión de los contenedores LXC en la cual se agrega un API a nivel de núcleo y aplicación, que en conjunción ejecutan procesos de manera aislada en la CPU, memoria, E/S, entre otros [11].

2.5 Manejo de contenedores virtuales con Docker

En Docker, los VCs se crean utilizando imágenes de contenedor (ver Figura 2.5) como base. Una imagen de contenedor es inmutable e incluye todas las dependencias (como sistema operativo, librerías y *frameworks*), así como las configuraciones de despliegue y ejecución utilizadas por el contenedor en tiempo de ejecución. Usualmente, dichas imágenes se crean a partir de múltiples imágenes base,

[1] Cgroups es una herramienta de Linux para controlar la asignación de recursos a los procesos.
[2] www.docker.com

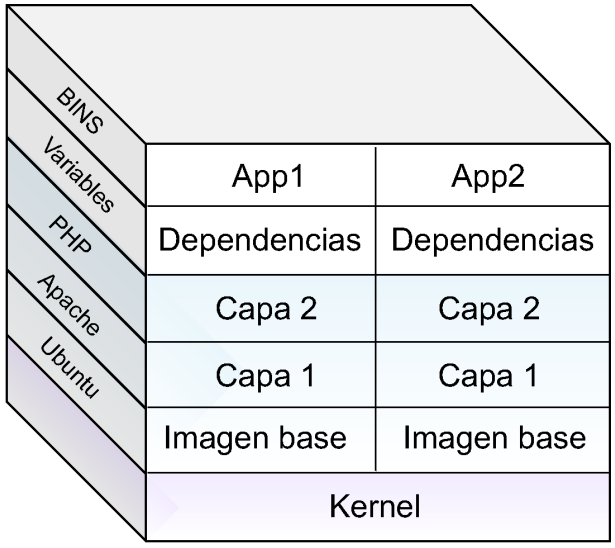


Figura 2.5: Ejemplo de una imagen de contenedor.

que son capas apiladas una sobre otra, para formar el sistema de ficheros del contenedor [33]. Cada comando ejecutado dentro del contenedor crea una nueva capa en la imagen. Los comandos en la imagen pueden ser ejecutados manualmente desde la línea de comandos o automáticamente, utilizando un archivo de configuración **Dockerfile** [11].

Los archivos *Dockerfile* contienen las instrucciones para construir una imagen de contenedor. En la primera línea del archivo, se establece la imagen base que será utilizada para generar la nueva imagen de contenedor. Posteriormente se colocan las instrucciones requeridas para instalar programas, copiar archivos, definir variables de entorno, volúmenes, entro otros requerimientos para configurar el entorno de trabajo de la imagen [35].

Un *contenedor* es una instancia de una imagen de contenedor, en el cual se ejecuta una aplicación, proceso o servicio, y se encuentra conformado por el contenido de una imagen de contenedor, un entorno de ejecución y un conjunto estándar de instrucciones [33].

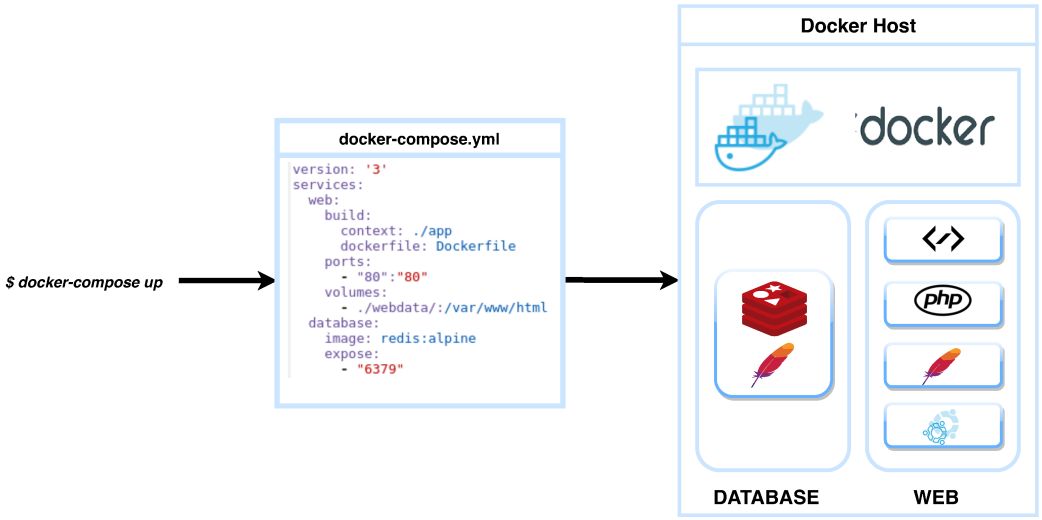


Figura 2.6: Despliegue de servicios con Docker-Compose [34].

2.5.1 Docker Compose

Docker Compose[3] es una herramienta que ayuda a definir y ejecutar aplicaciones multi-contenedor utilizando archivos de configuración YAML[4] (por su acrónimo recursivo en inglés *YAML Ain't Markup Language*, o en castellano "YAML no es un lenguaje de marcado").

Docker-Compose permite definir una sola aplicación utilizando múltiples imágenes de contenedor (lo cual puede realizarse con uno o más archivos YAML). Una vez que se han definido los servicios que conforman la aplicación, está puede ser desplegada con un solo comando (`docker-compose up`), el cual crea un contenedor para cada servicio definido en el archivo YAML [33, 35]. La Figura 2.6 muestra una representación conceptual del flujo de trabajo que se sigue al desplegar servicios con Docker-Compose a partir de un archivo YAML.

2.5.2 Docker Swarm

Docker Swarm es una herramienta que permite desplegar contenedores Docker en un clúster de computadores, mediante la consola interactiva de Docker. Docker Swarm posee las siguientes

[3]https://docs.docker.com/compose/
[4]Los archivos YAML, son un formato de serialización de datos legible por humanos.

características: *i)* administración y orquestación de contenedores utilizando Docker; *ii)* diseño descentralizado; *iii)* swarm permite escalar las aplicaciones, declarando el número de tareas que se desean ejecutar a partir de un contenedor; *iv)* el manejador de Swarm constantemente monitorea el estado del clúster para garantizar que el estado de los contenedores lanzados sea el mismo que el deseado durante su lanzamiento; *v)* descubrimiento de servicios; y *vi)* balanceo de carga.

2.6 Orquestación

La orquestación hace referencia a procesos ejecutables que pueden interactuar con otros servicios, tanto internos como externos. Dichas interacciones suelen ocurrir a nivel de mensaje, por lo cual, entre las tareas de un orquestador se incluyen la lógica y el orden de ejecución de las tareas. Esto puede abarcar desde aplicaciones hasta organizaciones, lo cual permite definir un modelo de proceso de larga duración, transaccional y de varios pasos. La orquestación siempre representa el control desde la perspectiva de una de las partes. Lo anterior difiere de los coordinadores, que son más colaborativos y permiten a cada parte involucrada describir su parte en la interacción [96].

En este sentido, la orquestación puede considerarse como la descripción de las interacciones y el flujo de mensajes entre servicios, en el contexto de un proceso [75]. En el pasado era conocida como flujo de trabajo.

2.7 Coordinador

Un coordinador se encarga de rastrear las secuencias de los mensajes transmitido a través de distintas etapas. Generalmente, se refiere al rastreo del intercambio de mensajes públicos que ocurre en los servicios web, en lugar de un proceso comercial específico que es ejecutado en una sola parte [96].

2.8 Flujo de trabajo

La formalización de un proceso científico es conocida como flujo de trabajo. Permite el despliegue automático de tareas en una infraestructura de hardware. Por lo general, los usuarios de flujos de trabajo deben seleccionar los datos y realizar algún proceso sobre estos para obtener los datos finales, y de ser necesario visualizarlos. Debido a ello, los sistemas de flujos de trabajo deben proveer distintas herramientas para el manejo de datos, creación y despliegue de tareas, manejo de relaciones entre datos y tareas, así como el monitoreo y recuperación de las fallas [123, 125, 126]. A través de un motor de flujos de trabajo, los usuarios deberían ser capaces de seleccionar tareas y relaciones que conformaran el flujo de trabajo.

En un sistema de flujos de trabajo, se puede encontrar la participación de diversas tecnologías heterogéneas, como lo es el lenguaje o herramienta para formalizar el flujo de trabajo, el motor que se encargara de orquestar la ejecución del flujo de trabajo, el sistema donde dicho flujo será ejecutado, y la infraestructura sobre la cual se ejecutan las tareas y se almacenan los datos [123].

Los flujos de trabajo pueden ser clasificados en dos tipos: *i)* orientados a flujos de datos, y *ii)* orientados a flujos de tareas. El primer caso se refiere a que los flujos de trabajo son definidos con base en las dependencias entre tareas. Es decir, el usuario tiene que definir explícitamente las dependencias entre tareas. En el segundo caso, los flujos son definidos mediante las dependencias entre los datos necesarios por cada tarea, y el motor del flujo de trabajo se encarga de resolver las dependencias entre tareas [86].

2.9 Bloques de construcción

Un bloque de construcción (BB) es una unidad reutilizable, autocontenida y modular, basada en los principios de abstracción, encapsulamiento y transparente. Tanto el código, como la representación de los datos implementados en un bloque de construcción, se encuentran encapsulados de tal manera

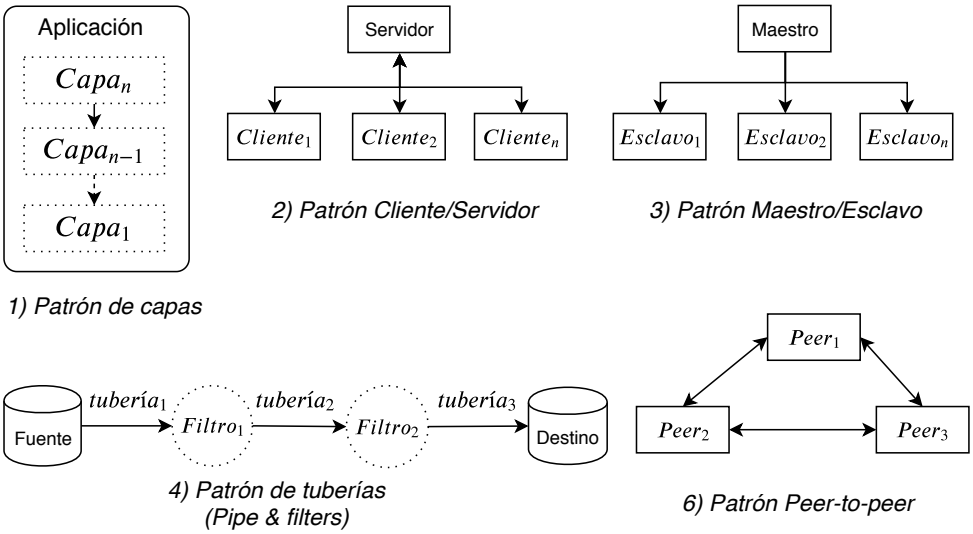


Figura 2.7: Ejemplo de patrones de arquitectura de software [72].

que todos sus detalles de implementación se encuentran ocultos. El usuario no puede ver, acceder o modificar el código o la representación de los datos. Los bloques de construcción de software tienen características similares a los utilizados en los chips electrónicos, dado que tienen interfaces de entrada y salida bien definidas, y solamente es necesario conocer la tarea que realizan y especificaciones para utilizarlos [64].

2.10 Patrones arquitectónicos de software

En arquitectura de software, los patrones son construidos para mejorar la eficiencia de los sistemas mediante el encadenamiento de aplicaciones o tareas. Dichos patrones tienen que ser generales y reusables para resolver problemas comunes en arquitectura de software [77]. La Figura 2.7 muestra algunos patrones presentes en la literatura los cuales son: patrón de capas, patrón cliente/servidor, patrón maestro/esclavo, patrón de tuberías (*pipe & filters*), y patrón peer-to-peer [72]. Construir dichos patrones no resulta una tarea fácil debido a que se deben desarrollar estructuras que controlen la comunicación y manejo de cada componente en el patrón.

A continuación, se describen los patrones mostrados en la Figura 2.7 [72]:

1. **Patrón de capas**. En este patrón los programas son estructurados en capas, donde capada capa provee diferentes servicios a los niveles más altos. Comúnmente se encuentran tres capas: capa de acceso, capa de procesamiento y capa de datos.

2. **Patrón Cliente-Servidor**. Está conformado por dos elementos: un servidor y múltiples clientes. En donde el servidor se encarga de proveer diferentes servicios a los clientes.

3. **Patrón Maestro/Esclavo**. Consta de dos entidades: el maestro y los esclavos. El maestro se encarga de repartir el trabajo entre los esclavos que son idénticos. Los esclavos procesan el trabajo y regresan un resultado al maestro, el cual calcula el resultado final.

4. **Patrón de tuberías**. En este patrón, cada proceso es considerado como un filtro (un componente de procesamiento). Los datos son transportados de un filtro a otro a través de tuberías. Además, es posible usar dicho patrón para estructurar sistemas que producen y procesan una secuencia de datos.

5. **Patrón *Peer-to-peer***. En este patrón, los componentes son llamados *peer*, los cuales cumplen dos funciones: cliente y servidor. En este patrón, la información no se encuentra concentrada en un solo punto, si no que está distribuida. Debido a ello, si falla un nodo es posible seguir utilizando el patrón de forma normal. Además, debido a que la información no se solicita directamente a un mismo punto, sino que se encuentra distribuida, por lo que los servidores no se saturarán.

2.11 Patrones de paralelismo

Los patrones de paralelismo permiten mejorar, en la mayoría de los casos, la eficiencia de las soluciones mediante la creación y acoplamiento de réplica de tareas, y entrega datos entre los BB's, servicios, módulos, funciones o aplicaciones que conformen un patrón [94].

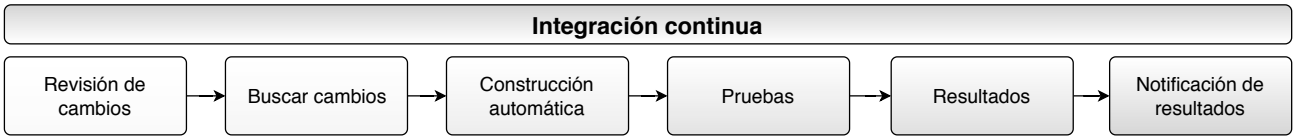


Figura 2.8: Esquema de integración continua (CI). Fuente: DevopsTI. Imagen disponible en la web [39].

2.11.1 Manejador/Trabajador

En el patrón *Manejador/Trabajador* [94], el *Manejador* procesa los datos en fases, tales como clonación, distribución de tareas, y supervisión de la ejecución de tareas. En la fase de clonación, el *Manejador* crea instancias de VC de una etapa incluida en los esquemas de preparación y recuperación. En la fase de distribución de tareas, el *Manejador* lee de manera recursiva el contenido almacenado en un directorio o en una fuente de datos DSr. Además, en esta fase el *Manejador* crea una lista de rutas $\mathbf{P} = \{Path_1, Path_2, \cdots, Path_n\}$ y distribuye cada ruta a cada trabajador de forma balanceada utilizando el algoritmo *Two Choices* [84]. En la fase de supervisión, el *Manejador* verifica que todos los trabajadores envíen los resultados de las tareas previamente asignadas por él.

2.11.2 Divide y vencerás

En el patrón *Divide y vencerás* ($D\&C$, del inglés *Divide&Conquer*) [94], una instancia de software llamada *Divide* segmenta los datos de entrada en s segmentos, los cuales son enviados a s número de trabajadores previamente clonados y lanzados en tiempo de configuración. Una instancia de software llamada *Consolidador* recibe los resultados producidos por los trabajadores y consolida los resultados en uno solo, enviando este nuevo resultado a la siguiente etapa en el esquema, o a un destino de datos.

2.12 Integración continua

La integración continua (CI, del inglés *Continuos Integration* [40, 112]), es un enfoque en donde se requiere la integración del código en un repositorio compartido durante varias veces al día, por parte de los desarrolladores. En ella, cada registro suele ser verificado mediante una compilación automatizada, lo cual permite a los equipos detectar problemas en una etapa temprana (ver Figura 2.8). Lo anterior, permite detectar errores de forma rápida y sencilla. Debido a que la integración se realiza con frecuencia, el seguimiento que se le debe dar para descubrir donde surgió un problema es más corto, lo que permite dedicar más tiempo al desarrollo de funciones [112]. La CI es barata, y por el contrario no integrarse continuamente suele ser caro. No obstante, sino se sigue un enfoque continuo, se tendrán periodos más largos entre cada integración lo cual hace que la dificultad para encontrar y solucionar problemas crezca de forma exponencial. Los problemas que suelen generarse debido a ello pueden provocar que un proyecto no funcione o falle por completo. Los beneficios que brinda la CI se listan a continuación [40]:

1. Reduce los tiempos de integración.
2. Proporciona una mayor comunicación aumentando la visibilidad.
3. Permite encontrar los problemas de forma temprana y solucionarlos desde la raíz.
4. Disminuye el tiempo de depuración e incrementa el tiempo para el desarrollo de funciones.
5. Proporciona una base sólida.
6. Acorta los tiempos para poder ver el código y funcionamiento.
7. Atenúa los problemas de integración, lo cual permite entregar el software de forma rápida.

2.13 Entrega continua

La entrega continua (CD, del inglés *Continuos Delivery* [23, 28, 40, 112]), es un enfoque de ingeniería de software en el que los equipos de desarrollo producen de forma continua software

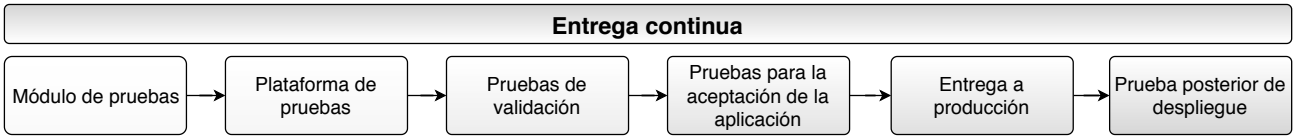


Figura 2.9: Esquema de entrega continua (CD). Fuente: DevopsTI. Imagen disponible en la web [39].

valioso en ciclos cortos (ver Figura 2.9). Dicho enfoque debe asegurar la confiabilidad del software al momento de ser lanzado. Aunque algunos desarrolladores aseguran que los CD permiten a las organizaciones llevar al mercado mejoras del servicio de forma rápida, eficiente y confiable, su implementación suele significar un gran reto, especialmente para las grandes empresas.

La utilización de la CD pretende optimizar la administración de la infraestructura, así como el equilibrio del tiempo y los recursos. La validación del software, implica múltiples entradas de variables como lo son, diferentes versiones de software, diferentes sistemas operativos, entre otros [127].

2.14 Flujo de datos continuo

Los flujos de datos continuos (CDF, del inglés *Continuos Data Flow* [129]), son aquellos que complementan los flujos de trabajo científico [5]. Permiten la composición de la ingesta de datos en tiempo real y de los procesos analíticos con el fin de procesar flujos de datos de sensores generalizados e instrumentos científicos. Dichos flujos de datos no pueden encontrarse en inactividad, son considerados de misión crítica, además de que deben funcionar de manera consistente y suelen ser de larga ejecución. Es probable que al paso del tiempo deban realizarse actualizaciones para corregir errores o agregar nuevas funciones.

[5]La formalización de un proceso científico es definida como flujo de trabajo. En él se permite el despliegue automático de tareas en una infraestructura de hardware. Los usuarios de los flujos de trabajo deben seleccionar los datos, y posteriormente realizar algún proceso sobre los mismos con el fin de obtener los datos finales y visualizarlos, de ser necesario [123, 125, 126].

2.15 Redes de distribución de contenidos

Las redes de distribución de contenido (CDN, del inglés *Content Delivery Network* [26, 46]), almacenan pequeños fragmentos de información y los distribuyen a lugares cercanos que los solicitan. Gracias a ello, los usuarios pueden observar en el proceso de entrega de contenido una reducción en la latencia y sobrecarga. Mediante el uso de este tipo de servicios, los proveedores en la nube aprovechan el almacenamiento en caché de contenido. Los servicios de almacenamiento basados en la nube permiten a las organizaciones la creación de catálogos de contenidos basados en URLs. Por otro lado, los editores y usuarios finales podrán acceder a ellos sin restricciones de descargas simultáneas mediante el uso de cualquiera de los navegadores web, sincronizadores basados en flujos HTTP, o aplicaciones FTP. No obstante, en este tipo de soluciones las organizaciones no suelen tener ningún tipo de control sobre la gestión de su contenido [24], esto se debe al hecho de que el control lo tiene el proveedor de la nube [109], el cual a su vez relega dicho control a un proveedor de CDN. Por lo tanto, este tipo de soluciones representa un riesgo para las organizaciones, ya que al no tener acceso a sus contenidos durante las interrupciones del servicio provoca un fuerte impacto económico en la misma [116].

2.16 Infraestructura definida por código

Las infraestructuras definidas por código (IaC, del inglés *Infraestruture as Code* [89]) son un enfoque basado en prácticas de desarrollo de software. Los cambios en la infraestructura son hechos por definiciones desplegadas en los sistemas a través de procesos no supervisados que son incluidos a través de la red. En este enfoque la infraestructura es tratada como software y datos, esto permite a las organizaciones aplicar o utilizar herramientas de desarrollo de software, tales como los controladores de versiones, librerías para pruebas automáticas y herramientas de orquestación [89].

2.17 Almacenamiento definido por software

El almacenamiento definido por software (SDS, del inglés *Software Defined Storage* [18]) emergió como una propuesta para una nueva categoría de productos de software de almacenamiento. Pueden ser una tecnología independiente, o bien un elemento dentro de un centro de datos definido por software.

El término SDS se deriva del término redes definidas por software[6], la cual fue utilizada para describir un enfoque de tecnología de red que abstrae varios elementos de redes, y además crea una abstracción en el software. El enfoque SDS abstrae y simplifica la gestión de redes en servicios virtuales. A continuación, se muestra una lista de los atributos de los SDS que normalmente se encuentran en el mercado [18]:

- Permite que los clientes diseñen y construyan una solución de acuerdo con sus necesidades.
- Puede funcionar con cualquier hardware o con hardware desarrollado específicamente para este tipo de soluciones.
- La mayoría de estas soluciones permite la ampliación del espacio de almacenamiento.
- Permite la construcción incremental de servicios de almacenamiento.
- Proporciona una gestión automatizada.
- Incluyen una interfaz para que los usuarios puedan generar su solución.
- Permite el etiquetado de los metadatos para controlar el tipo de almacenamiento y servicios de datos aplicados, esto mediante una gestión a nivel de servicio. Además, el nivel de granularidad puede ser modificado con el tiempo.

[6]Es un paradigma de red en donde la inteligencia de la red está lógicamente centralizada en los controladores basados en software, y los dispositivos de red se convierten en simples dispositivos de reenvío de paquetes que se pueden programar a través de una interfaz abierta [92].

- Permite el establecimiento de políticas para administrar los servicios de almacenamiento y datos.

La diferencia de este paradigma con el paradigma de infraestructura definida por código es que la *Iac* permite programar estructuras de hardware como un código ejecutable que puede adaptarse, duplicarse, eliminarse y versionarse fácilmente en cualquier momento. Mientras que las *SDS* permiten extraer los elementos que controlan las solicitudes de almacenamiento, y no lo que realmente se está almacenado. Es un nivel de software entre el almacenamiento físico y la solicitud de datos que permite manipular la manera y el lugar donde se almacenaran dichos datos.

2.18 Resumen

En este capítulo se dieron a conocer conceptos tales como el almacenamiento en la nube e infraestructuras de almacenamiento definidas por código, los cuales se encuentran relacionados con el presente trabajo de tesis. Así mismo, se describen otros conceptos, tales como el de los bloques de construcción, los cuales son utilizados para construir las soluciones de almacenamiento de acuerdo con los requerimientos no funcionales que el usuario final especifique.

3

Estado del arte

En esta sección se da a conocer una recopilación de los trabajos relacionados con la presente propuesta de tesis. En esta se contemplan trabajos referentes a herramientas de almacenamiento de datos, así como diferentes esquemas de desarrollo de software tales como CD/CI y los bloques de construcción. El principal propósito del repaso del estado del arte es dar a conocer la forma en que se han solucionado diversos problemas para los diferentes tipos de sistemas de almacenamiento de datos y tecnologías relacionadas a los mismos.

3.1 Almacenamiento en la nube

Los ambientes de nube pública y privada se han vuelto soluciones populares entre las organizaciones y los usuarios finales para mantener y preservar sus datos. Una de las soluciones de nube pública más populares en el mercado es la plataforma *Dropbox*, la cual es un sistema de almacenamiento basado en la nube que permite reducir la cantidad de datos intercambiados mediante el uso de la codificación de datos. *Dropbox*, mantiene una base de datos de información de metadatos

en cada dispositivo de forma local y comprime los fragmentos antes de enviarlos. Además, posee dos componentes principales en su arquitectura: i) el control, el cual se encuentra bajo el cargo directo de *Dropbox* Inc, y ii) los servidores de almacenamiento de datos, los cuales se encuentran bajo el cargo de Amazon Elastic Compute Cloud (EC2) y Simple Storage Service (S3).

En 2012, Drago et al. [36] presentaron un análisis de la plataforma *Dropbox*, en el cual se evaluó el desempeño de la plataforma mediante un estudio realizado en cuatro puntos estratégicos de Europa, durante un periodo de 42 días. Dicho estudio revelo que el rendimiento del servicio de la plataforma se encuentra estrechamente relacionado con la distancia que existe entre los clientes y el centro de almacenamiento de datos en el que se estén preservando los datos. Así mismo, se pudo observar que el uso creciente de la plataforma de *Dropbox*, por parte de los usuarios finales, deriva en el tráfico de grandes volúmenes de datos, lo cual equivale a una tercera parte del tráfico de la plataforma *YouTube*. Es importante considerar que al contar con una arquitectura de nube pública, en esta plataforma no existe garantía de que los datos no sean accedido por terceros. Además, el almacenamiento gratuito es de 2GB, y si se requiere más espacio hay que pagar por un plan de renta mensual.

En este contexto de nube pública, Calder et al. [17] presentan *Windows Azure*, un servicio de almacenamiento en la nube de alta disponibilidad. *Windows Azure* permite a los clientes almacenar grandes cantidades de datos. En él, los datos son almacenados de forma "duradera" utilizando técnicas de replicación local y geográfica, lo cual facilita la recuperación de los mismos en caso de algún desastre (por ejemplo, falla en los nodos de almacenamiento, caída del sistema, fallas eléctricas, por mencionar algunos). *Windows Azure* es escalable, cuenta con consistencia de datos entre los usuarios finales, y permite la disponibilidad de los mismos, así como tolerancia a fallos. *Windows Azure* utiliza técnicas para el balanceo de carga de forma automática, lo cual permite dividir y fusionar los rangos de partición de acuerdo con las demandas de tráfico de las aplicaciones.

No obstante, para poder utilizarlo en toda su capacidad es necesario contar con experiencia en la plataforma, para así poder garantizar que todas las piezas móviles funcionen de manera eficiente. Así

mismo, es necesario considerar que a medida en que más y más empresas continúan moviendo sus datos a la nube, rastrear un proveedor que ofrezca las mejores características para la organización que lo contrate, se vuelve una tarea complicada.

3.2 Bloques de construcción (BB's)

En la literatura se han propuesto los BB's como unidades de software reutilizables, que permiten la construcción de sistemas más complejos mediante la interconexión de los mismo, en forma de patrones de software. En 2018, González-Compeán et al. propusieron Sacbe [47], un enfoque de negocios para la construcción eficiente y flexible para soluciones de almacenamiento punto a punto, la cual agrega la propiedad de confiabilidad a los datos. En Sacbe se propone el uso de una arquitectura de software modular, la cual permite encapsular módulos de software en sus contenedores. Dichos contenedores son llamados bloques de construcción (BBs), y se encuentran interconectados a través de las interfaces de comunicación I/O, así como una red para el almacenamiento, compartición y seguridad de datos en la nube. A diferencia de las tuberías de procesamiento de Unix, esta propuesta fue realizada para aplicaciones punto a punto de almacenamiento en la nube, en lugar de herramientas del sistema operativo para sistemas de aplicaciones de almacenamiento como la propuesta realizada por Grawinkel et al. [50].

En Sacbe los módulos de cobertura y gestión son encapsulados por micro-aplicaciones independientes en BBs. Estos BBs pueden realizar diversas tareas, las cuales son: adquisición, entrega y procesamiento. En la etapa de procesamiento, los datos son procesados por instancias de software llamadas unidades de procesamiento (PUs, por sus siglas en inglés *Processing Units*). Los contenedores que encapsulan PUs son llamados *data coverage building blocks* (DC-BB), mientras que los contenedores que encapsulan *PUs deploying management modules* son llamados *management building blocks* (M-BB).

Sacbe provee continuidad en la entrega de datos, confiabilidad basada en la replicación de BBs

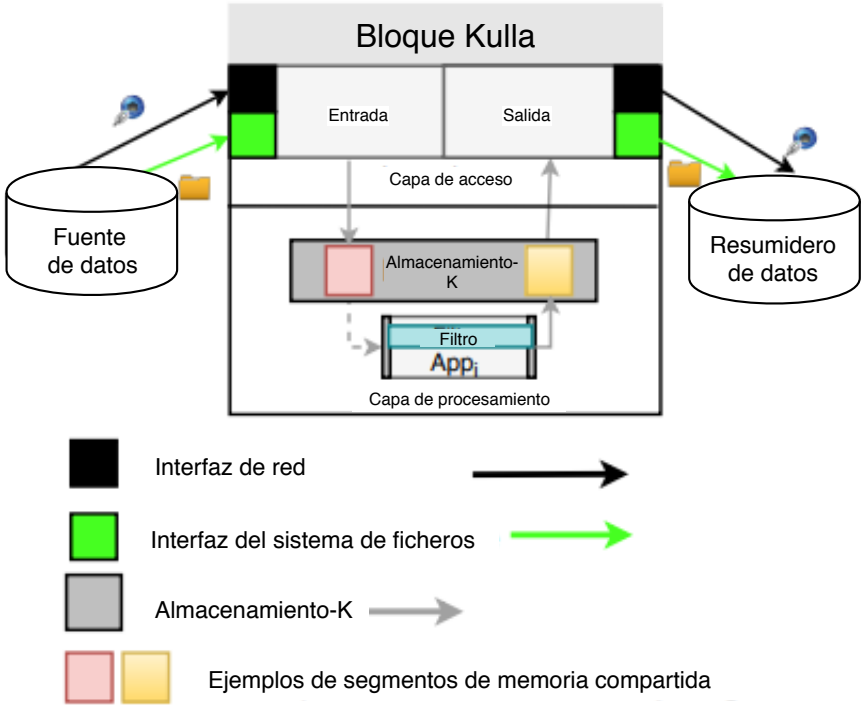


Figura 3.1: Representación conceptual de un bloque de construcción en Kulla. Fuente: *Reyes et al.* [107].

(tolerancia a fallos), adaptabilidad en los BBs en tiempos de configuración, y un modelo basado en almacenamiento para mejorar la eficiencia del intercambio de datos entre BBs.

De igual manera, en Reyes et al. [107] propusieron Kulla, un modelo de construcción basado en entrega continua y aplicaciones agnósticas. En este modelo, las aplicaciones se encuentran encapsuladas en contenedores virtuales, en los cuales se instalan las dependencias de la aplicación, así como sus configuraciones y variables de entorno para la generación de BB's. El modelo de construcción está basado en grafos acíclicos dirigidos, en donde los bloques de construcción son unidos mediante la entrada y salida de datos, para formar patrones de procesamiento de datos. La Figura 3.1 muestra la representación conceptual de un bloque de construcción en Kulla, en donde la aplicación es encapsulada en un contenedor, así como con otras estructuras de control para el manejo de datos, y la interoperabilidad con otros bloques.

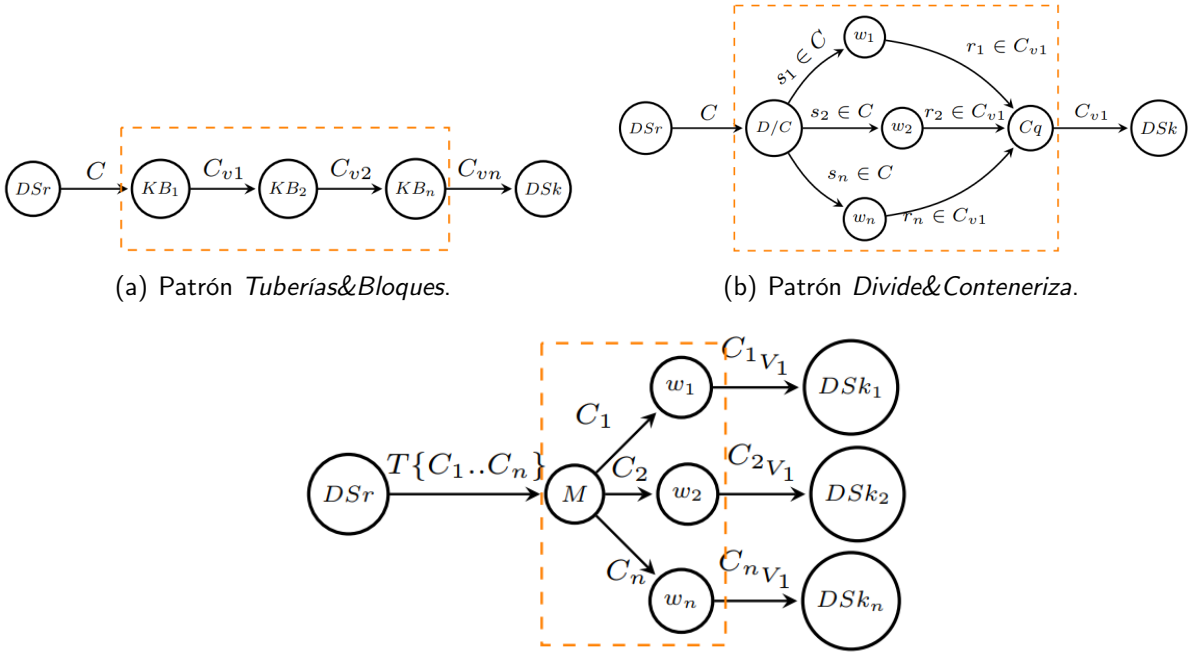
Al igual que Sacbe, Kulla implementa interfaces de E/S basadas en almacenamiento en memoria.

Sin embargo, para sacar mayor provecho de este modelo de intercambio de datos Kulla permite la construcción de patrones de paralelismo basados en datos y tareas.

3.3 Patrones de paralelismo basados en contenedores virtuales

En la literatura, los patrones de paralelismo han sido utilizados para mejorar la eficiencia de tareas y datos. En kulla [107], son descritos diferentes patrones de paralelismo tales como *Divide y Conteneriza* (paralelismo de datos), *Tuberías&Bloques* (streaming), y *Manejador/Bloques* (paralelismo de tareas) los cuales se encuentran encapsulados en contenedores virtuales. El patrón *Tuberías&Bloques* (ver Figura 3.2(a)) es utilizado para crear el patrón tradicional *Pipe&Filter*, mediante el encadenamiento de aplicaciones, a través de sus interfaces de entrada y salida. En tiempo de ejecución, este patrón crea un flujo de datos, desde una fuente de datos hasta un sumidero de datos. Por otro lado, el patrón *Divide&Conteneriza* (ver Figura 3.2(b)) es un método que implementa el algoritmo tradicional de divide y vencerás. La meta de este patrón es clonar los bloques de construcción y ejecutar todas estas replicas en paralelo. En él, una entidad llamada *Divide* segmenta un contenido y distribuye cada uno de los segmentos a n trabajadores que aplican una tarea de procesamiento a los segmentos. Y finalmente, una entidad llamada *Conquer* se encarga de consolidar los resultados en uno solo. Por último, el patrón *Manejador/Bloques* (ver Figura 3.2(c)) está basado en el patrón *Manejador/Trabajador* en el cual una entidad llamada manejador se encarga de leer una fuente de datos o contenido, y distribuir de manera balanceada los contenidos a n trabajadores para procesarlos en paralelo.

No obstante, Kulla no cuenta con componentes que aseguren la integridad de los datos. Es decir, no se garantiza que dentro de los BBs los datos no sean modificados más allá de la tarea para la que fue diseñada, o que no se pierdan.



(a) Patrón *Tuberías&Bloques*.

(b) Patrón *Divide&Conteneriza*.

(c) Patrón *Manejador/Bloques*.

Figura 3.2: Patrones de paralelismo manejados en Kulla [107].

3.4 Integración continua

En ingeniería de software la CI permite el despliegue y acoplamiento automático de nuevas aplicaciones y componentes en una solución ya existente. Es comúnmente utilizado en ambientes organizacionales en donde los requerimientos varían a través del tiempo.

En 2017, Ioannis et al. [90] presentaron *Jenkins-CI*, un sistema de CI de código abierto, como plataforma de procesamiento de imágenes y datos científicos. Dicha plataforma proporciona diferentes complementos para realizar tareas de cálculo estándar, además su diseño permite la rápida integración de diversas aplicaciones científicas externas. La plataforma de *Jenkins-CI* es accesible mediante la web, proporciona facilidad de acceso y de la compartición de recursos informáticos de alto rendimiento. También, permite la automatización de tareas de procesamiento de imágenes y datos. Sin embargo, dicha plataforma no permite el intercambio de datos mediante memoria, lo cual podría

generar cuellos de botella entre las diferentes etapas de procesamiento. Además, si se compara con los sistemas desarrollados específicamente para los flujos de trabajo computacional, *Jenkins-CI* se queda corto en la disponibilidad de módulos prefabricados para estadísticas, minería de datos y visualización.

3.5 Entrega continua

En ingeniería de software la CD de datos permite a los desarrolladores integrar de manera continua las piezas de código en un entorno de desarrollo o producción en las organizaciones, lo cual permite la automatización de las pruebas, y la verificación de las soluciones antes de ser entregadas a los clientes.

En 2015, Armenise [6] presentó *Jenkins* con un enfoque en CD, lo cual permitió la automatización de la compilación, del lanzamiento y del proceso de entrega del producto. *Jenkins* originalmente era una plataforma integración continua, y de código abierto. Su objetivo inicial fue la automatización de la construcción y prueba de procesos, sin embargo, la necesidad de una mejora continua en el ciclo de vida en todas las fases permitió abordar la introducción del enfoque de CD. Gracias a la aplicación de dicho enfoque, es posible controlar la ejecución de la tubería, proporcionando una vista agregada de los diferentes pasos, junto con los resultados de su ejecución. Sin embargo, esto aun presenta retos como lo es realizar el control de versiones de los artefactos que deben ser enviados continuamente, o rastrear el entorno en el que el artefacto fue creado.

El principal inconveniente de Jenkins en CD, es la necesidad de tener que instalar múltiples plugins que permiten la generación de tuberías de CD. Lo anterior debido a que el enfoque original de esta herramienta es CI, por lo cual se tuvo que adaptar mediante el uso de plugins. Además, aún es necesario mejorar las herramientas para el control de versiones que se encuentran disponibles actualmente.

3.6 Flujo de datos continuo

Los flujos de datos continuo permiten la ingesta continua de datos en una tubería de procesamiento o en un flujo de trabajo. La entrega de los datos debe de ser ininterrumpida y durar por largos periodos de tiempo.

En 2003, Chandrasekaran et al. [21] desarrollaron *TelegraphCQ*, un sistema de procesamiento de flujo de datos continuo. Este sistema se enfoca en enfrentar los desafíos que surgen en el manejo de grandes flujos de consultas continuas, realizadas sobre flujos de datos de gran volumen y altamente variable. Esta herramienta permite realizar consultas continuas sobre una combinación de tablas y flujos de datos. Para tratar dichos flujos de datos cuya longitud no tiene límites, ciertas operaciones como las uniones, solo pueden ser ejecutadas en ventanas finitas en estos flujos.

Sin embargo, en este trabajo solo se presenta un enfoque que permite el flujo de datos continuo para consultas de gran escala sobre un sistema. Es decir, permite a un sistema atender solicitudes *cuasi* ininterrumpidamente. En este trabajo no se contemplan sistemas complejos como lo pueden ser las tuberías de procesamiento o los flujos de trabajo, en los que cada una de sus componentes recibe solicitudes de manera constante, además de que es requerido entregar continuamente datos de una etapa a otra, las cuales pueden estar distribuidas a pequeña escala (en diferentes segmentos de memoria) o a gran escala (en diferentes computadores o incluso en diferentes lugares geográficos como centros de datos).

3.7 Redes de distribución de contenidos

Las CDN son plataformas de servidores distribuidos que permiten la entrega de contenido de forma optimizada. Se encuentran distribuidas a través de servidores físicos como virtuales, lo cual permite que los usuarios finales puedan descargar y compartir contenidos (vídeos, imágenes, PDFs, por mencionar algunos) de forma rápida y segura.

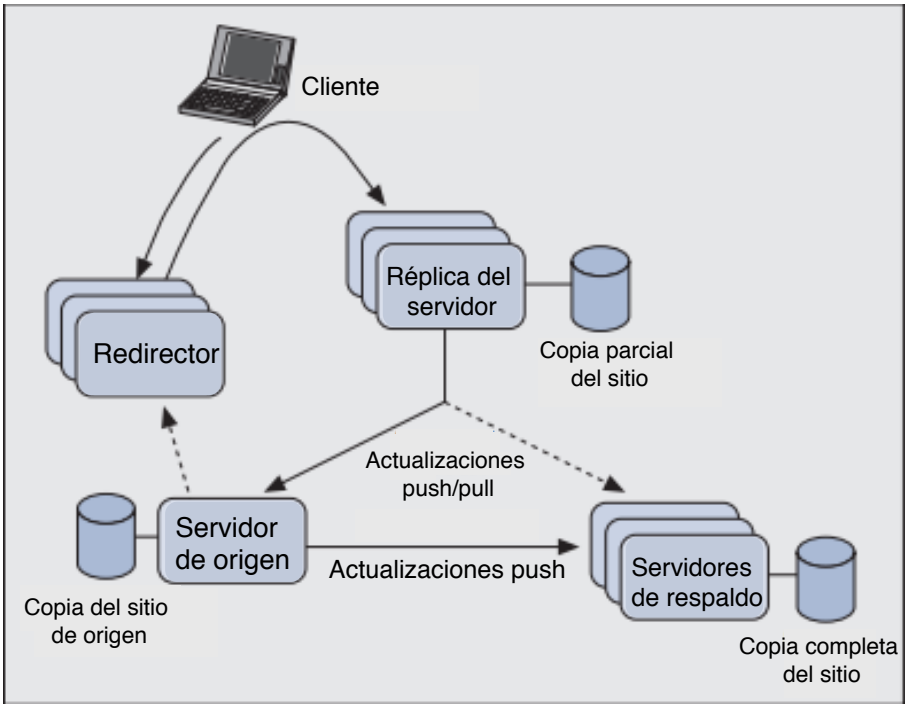


Figura 3.3: Modelo de Globule. Fuente: Pierre y Maarten [97]. IEEECommunications Magazine, 2006.

Pierre y van Steen [97] propusieron *Globule*, una red de entrega de contenido colaborativa. *Globule* fue desarrollado para sistemas UNIX y adaptado a sistemas Windows. En este CDN se manejan dos abstracciones conocidas como $sitios$ y $servidores$, donde un $sitio$ es una colección de documentos que pertenecen a un usuario, y un $servidor$ es un proceso ejecutándose en un equipo conectado a la red. Cuando el servidor origen no se encuentra disponible, los servidores de respaldo se encargan de responderlas solicitudes de los clientes. Estos servidores mantienen una copia del sitio, garantizando la accesibilidad al mismo. Además, para mejorar el desempeño se tienen servidores replicas, los cuales se mantienen distribuidos a través de la red y mantienen una copia del sitio (ver Figura 3.3).

En Globule se mantienen replicas completas de los contenidos para aumentar su disponibilidad, sin embargo, en ambientes organizacionales los recursos de almacenamiento son limitados, por lo que mantener copias de archivos de gran volumen es inviable. Técnicas de compresión de datos mezclado con las técnicas de disponibilidad de Globule pueden, en el mejor de los casos, disminuir el volumen de datos a almacenar. Otra alternativa, son las técnicas de confiabilidad basadas en redundancia

(ej. IDA) las cuales permiten generar segmentos de un archivo y recuperarlo a partir de un número menor de segmentos. Por otro lado, en Globule, los recursos se encuentran centralizados, por lo cual al recibir una gran cantidad de peticiones se pueden generar cuellos de botella, disminuyendo la eficiencia en la entrega/recuperación de datos.

De igual manera, *SkyCDS* es una CDN presentada por González et al. [46] basada en el modelo de publicación/subscripción. En ella se utilizan abstracciones llamadas catálogos, las cuales son un componente básico de metadatos que permite a las organizaciones mantener un control sobre los nuevos contenidos, el consumo del contenido y el manejo de este. Además, posee una capa en una plataforma de almacenamiento en múltiples nubes basada en la dispersión de información. Esta capa permite mantener un uso eficiente del espacio de almacenamiento y una alta confiabilidad. *SkyCDS* cuenta con un flujo de trabajo que se encarga de la entrega y adquisición de los datos. Dicho flujo se encuentra diseñado para trabajar con los diferentes núcleos. Un inconveniente identificado en *SkyCDS* es que el flujo de trabajo implementado en el cliente no considera esquemas de CD por lo que las peticiones suelen encolarse, lo que aumenta el tiempo de espera.

Otra de las soluciones CDN más utilizadas que se encuentran en el mercado la tiene Amazon [74], el cual dispone de una solución de CDN basada en servicios web. Dicha solución se encuentra integrada con otros servicios de AWS (por ejemplo, Amazon S3, Amazon EC2, elastic load balancing y amazon route 53), que permite distribuir contenido a los usuarios finales de las aplicaciones desplegadas con AWS. Esta solución dispone de la entrega de sitios web, así como datos, vídeos, aplicaciones y APIs. Su desventaja es que al ser un servicio con infraestructura de nube pública los datos no se encuentran totalmente protegidos, y se pueden perder fácilmente. Así mismo, no hay una garantía de que los datos sean accedidos por terceros, además de que hay un desconocimiento de la ubicación geográfica de los datos, lo cual es una desventaja frente a cualquiera de los fallos que pueden ocurrir en un sistema, y la recuperación de estos en distintos puntos, por distintos usuarios.

```
resource "aws_instance" "example" {
  ami           = "ami-40d28157"
  instance_type = "t2.micro"
}

resource "dnsimple_record" "example" {
  domain = "example.com"
  name   = "test"
  value  = "${aws_instance.example.public_ip}"
  type   = "A"
}
```

Figura 3.4: Ejemplo del contenido de un archivo de configuración de *Terraform*. Fuente: Brikman [15]. O'Reilly Media, Inc., 2017.

3.8 Infraestructura definida por código (IaC)

En ambientes organizacionales donde la infraestructura de almacenamiento es dinámica y cambiante, se requiere mantener un control y gestión sobre los recursos de almacenamiento disponibles. Para ello es necesario utilizar diferentes técnicas como lo son las IaC, la cuales permiten a las organizaciones tratar su infraestructura como software mediante la generación de secuencias de código. Actualmente existen en el mercado algunas soluciones de IaC como lo es Terraform, OpenStack Swift, y AWS Cloud Formation.

Terraform [15] es una herramienta de código abierto desarrollada en el lenguaje de programación GO por HashiCorp. Permite construir y desplegar una infraestructura desde una computadora, y no es necesaria ninguna otra infraestructura adicional para ejecutarlo. *Terraform* cuenta con un archivo binario que permite "traducir" el contenido del archivo de configuración, en llamadas a los distintos proveedores de nube (por ejemplo, Google Cloud, Amazon Web Service, Open Stack, por mencionar algunos). La Figura 3.4, presenta un ejemplo de un archivo de configuración de *Terraform*.

La versión gratuita de *Terraform* se encuentra limitada, por lo cual es necesario pagar una licencia

para adquirir características superiores de manejo. Además, con *Terraform* no se puede volver a un estado anterior del sistema, por lo cual, si se presentan errores es necesario identificarlos y corregirlos al momento, o en su defecto volver a desplegar toda la infraestructura desde cero, lo cual implica la perdida de las configuraciones y datos previamente cargados y almacenados.

Por otro lado, *OpenStack Switf* [7] es un sistema de almacenamiento de objetos (*Object Storage System*) altamente escalable, durable y multiusuario diseñado para almacenar grandes cantidades de datos no estructurados (por ejemplo, documentos, contenido web, respaldos, entre otros), a bajo costo. *OpenStack Switf* permite almacenar, recuperar y eliminar objetos en la nube y sus metadatos a través de una API HTTP. Los programadores pueden utilizar el API de *OpenStack Switf* o alguna librería existente en lenguajes como *Java*, *Python*, *Ruby* y *C#*. Por otro lado, en *OpenStack Switf* todo el tráfico va a través de servidores proxy, lo cual provoca que la velocidad de transferencia disminuya y aumente la latencia. Estos problemas de latencia pueden causar que las réplicas de los datos no sean actualizadas simultáneamente y, por lo tanto, algunos usuarios tendrán versiones desactualizadas de los datos, lo que puede derivar en un conflicto de versiones.

Amazon también cuenta con una solución de IaC llamada *AWS CloudFormation* [118], el cual es un servicio que permite modelar y desplegar recursos de *Amazon Web Services* mediante la creación de plantillas (archivos YAML o JSON), los cuales describen los recursos que se desean administrar, y la infraestructura que se desea desplegar. La construcción de la infraestructura y aplicaciones se realiza sin necesidad de procesos manuales, y permite regresar a un estado anterior de forma automática en caso de errores. La Figura 3.5, muestra las etapas de construcción de una solución con *AWS CloudFormation*.

En la primera etapa se crea el archivo de configuración, el cual pasara a ser revisado y compilado en la segunda etapa. Posteriormente, en la tercera etapa, se podrá utilizar *AWS CloudFormation* desde la terminal o desde un navegador para ejecutar el archivo de configuración previamente creado. Finalmente, en la cuarta etapa, *AWS CloudFormation* construirá y configurará los recursos especificados.

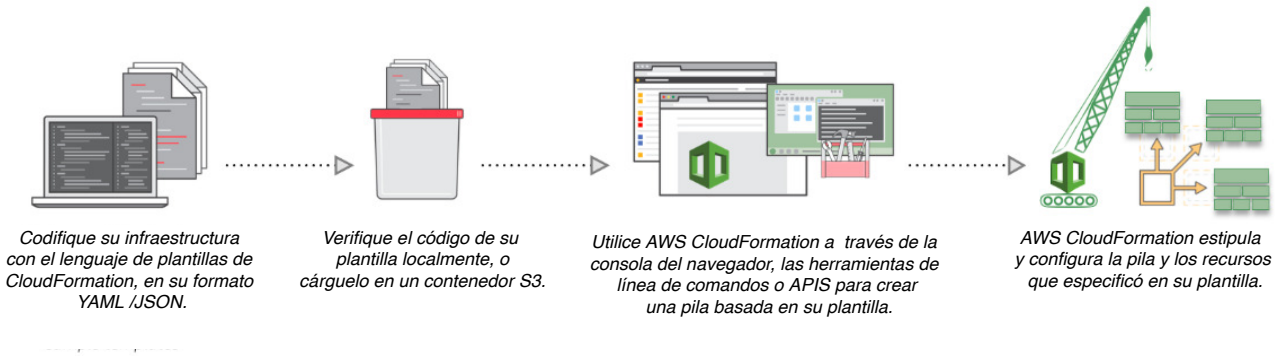


Figura 3.5: Etapas de construcción de una solución con *AWS CloudFormation*. Fuente: Amazon Web Services, Inc. o sus empresas afiliadas [5].

No obstante, el uso de archivos *JSON* representa una desventaja para este sistema, lo cual se debe a que, al no ser un lenguaje de codificación, los archivos *JSON* pueden volverse muy grandes (comparados con los archivos de configuración que se crean en otras IaC, como Terraform por ejemplo) y, por ende, difíciles de manejar por los usuarios. Por otro lado, cuando se realizan modificaciones en el entorno, *AWS CloudFormation* no especifica la parte en donde realizó dicha modificación, sino que solo muestra el objeto que fue modificado. Estas soluciones solo permiten el manejo de infraestructura contratada en AWS, no permitiendo el manejo de infraestructura privada (propia o de algún otro servicio en la nube).

3.9 Almacenamiento definido por software (SDS)

Parecidos a los IaC, los SDS son un enfoque que permite el manejo de la infraestructura física como abstracciones de software, lo cual proporciona soporte a las arquitecturas basadas en la nube (públicas, privada o híbrida), brindando la agilidad y escalabilidad que ellas requieren. Por ejemplo, aplicaciones de escritorio o servicios que se encargan del manejo de los recursos y planificación de peticiones de escritura y lectura. Así mismo, las SDS permiten aprovechar al máximo los recursos de almacenamiento disponibles en la infraestructura.

En este contexto, IBM [128] desarrolló *IBM System Storage DS8000* el cual fue diseñado como

un sistema de almacenamiento de alto rendimiento, capacidad y resistencia. Además, ofrece alta disponibilidad, brinda soporte a múltiples plataformas, y provee herramientas de administración simplificadas. El consolidar múltiples sistemas de almacenamiento en un solo equipo ayuda a simplificar el entorno de almacenamiento. No obstante, su interfaz es difícil de usar y no especifica de forma clara los requerimientos de uso, además de ser costosa, lo cual lo hace difícil de administrar y mantener. Además, *IBM System Storage DS8000* es un software propietario, el cual solo funciona con equipos IBM.

Por su lado, red Hat [115] presentó *Ceph*, un proyecto de código abierto que utiliza la tecnología SDS. *Ceph* proporciona soluciones unificadas definidas por software de almacenamiento, archivos y almacenamiento de objetos. Su idea se centra en proporcionar un sistema de almacenamiento distribuido escalable y de alto rendimiento. Además, provee características tales como confiabilidad, codificación de borrado, niveles de caché y conteo. Sin embargo, *Ceph* posee características de configuración muy limitadas, y no brinda un reporte detallado de los errores que suelen ocurrir en el sistema. Por otro lado, no existe un mecanismo que distribuya el archivo de configuración a todos los nodos existentes, por lo cual se vuelve una tarea muy tediosa entre más grande sea el sistema, lo cual lo vuelve ineficiente a la hora de querer expandirlo.

3.10 Discusión

En esta sección, se presenta una discusión acerca de los trabajos encontrados en el estado del arte relacionados al presente tema de tesis. Así como un análisis comparativo de dichos trabajos.

3.10.1 Manejo de requerimientos no funcionales en herramientas de almacenamiento de datos

Se generó una taxonomía en donde se muestran y clasifican los dos tipos de soluciones identificadas con base en los sistemas de almacenamiento de datos (ver Figura 3.6).

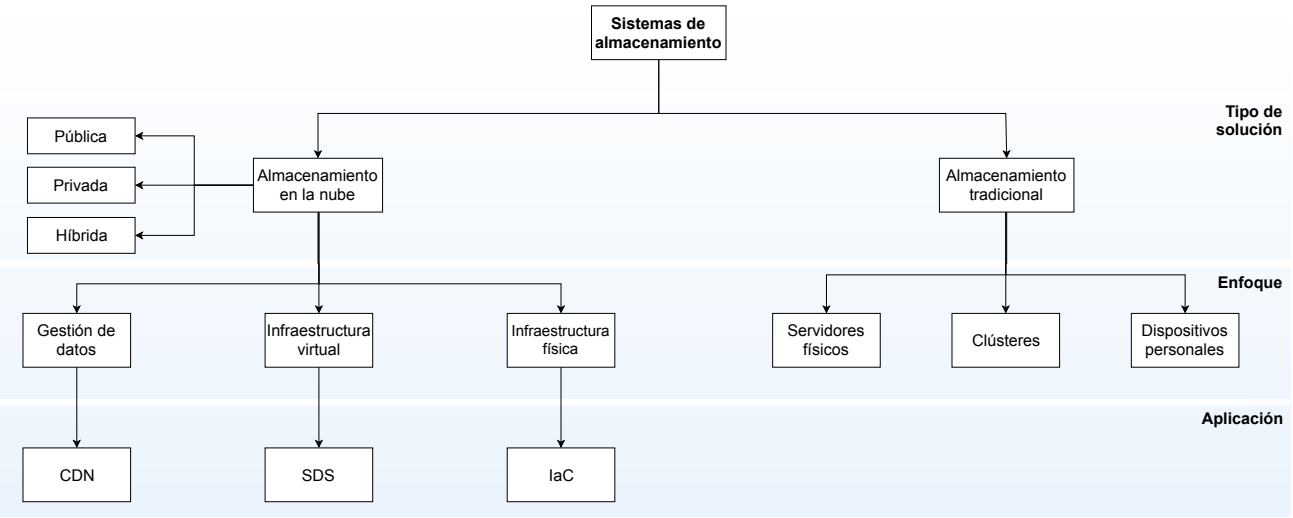


Figura 3.6: Taxonomía de la clasificación de los sistemas de almacenamiento.

La Figura 3.6 presenta dos tipos de soluciones de almacenamiento: las soluciones basadas en la nube y las soluciones tradicionales. Las primeras son todas aquellas que hacen uso de proveedores de nube pública, privada o híbrida para realizar el manejo y almacenamiento de los datos. Además, en este tipo de soluciones se cuenta con tres tipos de enfoque: gestión de datos, infraestructura física e infraestructura privada. En la gestión de datos se encuentran herramientas para el transporte de datos, entre infraestructuras o entre dos puntos utilizando como intermediario la nube. Por ejemplo, los sistemas CDN se encargan de acercar los datos a los usuarios para atenderlos de forma rápida, mediante URLs o clientes (por ejemplo, SkyCDS [46], Globule [97] y Amazon CDN [74]). En la infraestructura física existen mecanismos para el manejo, y la división de los recursos físicos existentes de manera lógica. Por ejemplo, las IaC permiten gestionar una infraestructura de acuerdo con las configuraciones creadas por los administradores (por ejemplo, Terraform [15], OpenStack Swift [7], AWS CloudFormation AWS CloudFormation [118]). La infraestructura virtual permite el manejo de los recursos de almacenamiento como máquinas virtuales en una capa subyacente a los recursos físicos de almacenamiento. En este sentido los SDS, proporcionan interfaces que permiten a los usuarios gestionar los recursos que poseen de manera virtual, creando divisiones lógicas de los mismos (por ejemplo, Ceph [115]).

Tabla 3.1: Comparación cualitativa de los requerimientos no funcionales de los trabajos de almacenamiento de datos.

		Requerimientos no funcionales											
		Disponibilidad		Confiabilidad		Consistencia de datos		Eficiencia		Escalabilidad		Flexibilidad	
Trabajo	**Enfoque**	*Manejo*	*Construcción*	*Manejo*	*Construcción*	*Manejo*	*Construcción*	*Manejo*	*Construcción*	*Manejo*	*Construcción*	*Manejo*	*Construcción*
Windows Azure [17]	CS	E	Conf	E	Conf	E	Conf	NA	NA	E	Conf	NA	NA
Dropbox [36]	CS	E	Conf	NA	NA	E	Conf	NA	NA	E	Conf	NA	NA
Globule [97]	CDN	E	Conf	NA	NA	NA	NA	E	Conf	E	Conf	NA	NA
SkyCDS [46]	CDN	E	Conf	E	Conf	E	Conf	E	Conf	E	Conf	NA	NA
Amazon CDN [74]	CDN	E	Conf	NA	NA	NA	NA	NA	NA	E	Conf	NA	NA
IBM sytem storage DS800 [128]	SDS	E	Conf	NA	NA	NA	NA	E	Conf	E	Conf	NA	NA
Ceph [115]	SDS	E	Conf	E	Conf	E	Conf	E	Conf	NA	NA	E	Conf
Terraform [15]	IaC	E	Cod	NA	NA	NA		E	Cod	E	Cod	E	Cod
OpenStack Swift [7]	IaC	E	Cod	NA	NA	NA	NA	NA	NA	E	Cod	E	Cod
AWS CloudFormation [118]	IaC	E	Cod	NA	NA	E	Cod	NA	NA	D	Cod	E	Cod
Solución propuesta ⋆	SDC	D	Cod	D	Cod	D	Cod	D	Cod	D	Cod	D	Cod

D = dinámico **E** = estático **Conf** = por configuración **Cod** = por código **NA** = no aplica

CS: Cloud Storage, CDN: Content Delivery Network, SDS: Software-Define Storage, IaC: Infraestructure as Code & SDC: Storage-Define by Code

Por otro lado, entre las soluciones tradicionales de almacenamiento se encuentran los servidores físicos, clúster y dispositivos personales. Por ejemplo, una solución integral con servidores físicos es la que ofrece IBM con IBM System Storage DS8000 [128], el cual brinda una solución de almacenamiento de alto rendimiento conformada por un conjunto de servidores físicos.

Se llevo a cabo una comparación cualitativa de los requerimientos no funcionales entre los distintos trabajos encontrados en el estado del arte, con relación al presente tema de tesis (ver Tabla 3.1).

En esta comparación se consideran diferentes requerimientos no funcionales (disponibilidad, confiabilidad, eficiencia, escalabilidad, flexibilidad y consistencia de los datos). En el análisis se compararon diferentes trabajos de almacenamiento de datos con cuatro enfoques diferentes (CS, CDN, IaC y SDS), para identificar las características con las que cumple cada uno de ellos.

Todos los trabajos encontrados cumplen con la característica de disponibilidad y escalabilidad, ya que brindan al usuario la capacidad de acceder a la información o recursos disponibles en cualquier ubicación, y en el formato correcto. Además, son capaces de funcionar aun cuando se agreguen o quiten recursos al sistema. Esto se debe a que estas soluciones suelen ser desplegadas en la nube.

Respecto a la confiabilidad, se encontró que solo Windows Azure [17], SkyCDS [46] y Ceph [115] cumplen dicha característica. Esto se debe a que ellos garantizan que la tarea que se está realizando

por el usuario, sea terminada de forma exitosa.

La característica de eficiencia fue cubierta solo por cuatro de los trabajos. Globule [97] e IBM System Storage DS8000 [128], los cuales reducen la cantidad de recursos a utilizar para producir mejores resultados. Por su parte, SkyCDS [46] y Ceph [115] aplican métodos de eficiencia en el procesamiento de datos a la hora de cargarlos a la nube, maximizando los recursos locales del cliente de carga y descarga.

Se encontró que los trabajos que cumplen con el requerimiento de flexibilidad son todos aquellos que tienen un enfoque en IaC, ya que estos tienen la capacidad de adaptarse a requisitos nuevos, diferentes o cambiantes en el sistema.

Referente al requerimiento de consistencia de datos, seis de los 10 trabajos, lograron cubrirlo: Windows Azure [17], Dropbox [36], SkyCDS [46], OpenStack Swift [7], AWS CloudFormation [118] y Ceph [115]. Estos sistemas permiten tener un control de las versiones de los datos, así como su correcta administración.

Como se puede observar en la Tabla 3.1, los sistemas de almacenamiento basados en los enfoques de *CS, CDN* y *SDS* son manejados de forma estática, y su construcción se realiza mediante una configuración predeterminada. Por otro lado, los sistemas de almacenamiento basados en el enfoque *IaC* son manejados de forma estática, no obstante, su construcción se realiza mediante el uso de codificación.

3.10.2 Manejo de requerimientos no funcionales en herramientas de procesamiento de datos

Actualmente, en la literatura se han propuesto diferentes tecnologías para el modelado de flujos de trabajo de procesamiento de datos. Estas tecnologías permiten abordar el problema de la representación de procedimientos científicos, empresariales y de ingeniería como tareas de procesamiento computacional y distribuido entre diferentes infraestructuras (por ejemplo,

computadoras personales, servidores, estaciones de trabajo).

Tradicionalmente, estas tecnologías y lenguajes modelan los flujos de trabajo como grafos acíclicos dirigidos (DAGs, por sus siglas en inglés *Directed Acyclic Graph*). En esta representación, los nodos representan procesos de negocios o tareas de procesamiento, y las aristas representan el intercambio de datos entre dos nodos. Un ejemplo de este tipo de lenguajes es el lenguaje de modelado de procesos de negocios (BPML, por sus siglas en inglés *Business Process Model Language*). BPML es un modelo conceptual estándar basado en lenguaje XML, el cual permite modelar y ejecutar procesos de negocios. Además, considera distintos componentes como actividades, procesos, contextos, propiedades y señales. Este modelo genera la ejecución coordinada de actividades en diferentes contextos, y el intercambio de información en otro [124].

En esta sección, se generó una taxonomía que muestra las características, interfaces de entrada y salida (E/S), y el manejo de soluciones punto-a-punto. Como se puede observar en la Figura 3.7, las interfaces de una solución punto-a-punto pueden ser de tres tipos: i) memoria, ii) red, y iii) sistema de archivos. Mientras que su manejo se realiza a través de bloques de construcción (BBs) (los cuales pueden ser máquinas virtuales (VMs, del inglés *Virtual Machine*) o contenedores virtuales (VCs, del inglés *Virtual Containers*)), de aplicaciones, plantillas o de funciones. Por otro lado, las características de este tipo de soluciones se dividen en dos: i) requerimiento no-funcionales, y ii) requerimientos funcionales.

Dentro de los requerimientos funcionales se encuentran las aplicaciones, así como el manejo de dichas aplicaciones, las tareas y procesos, así como los nodos de almacenamiento sobre los que se almacenaran los datos de las soluciones.

Mientras que dentro de los requerimientos no-funcionales se encuentran la seguridad mediante el uso de técnicas para la verificación de integridad y el control de acceso a los datos (por ejemplo, técnicas de hash o autenticación), la eficiencia mediante el uso de técnicas de compresión de datos o de paralelismo de tareas y procesos (por ejemplo, algoritmos de compresión sin perdidas como LZ4 o patrones de paralelismo el patrón *manejador/trabajador*), y la confiabilidad mediante técnicas de

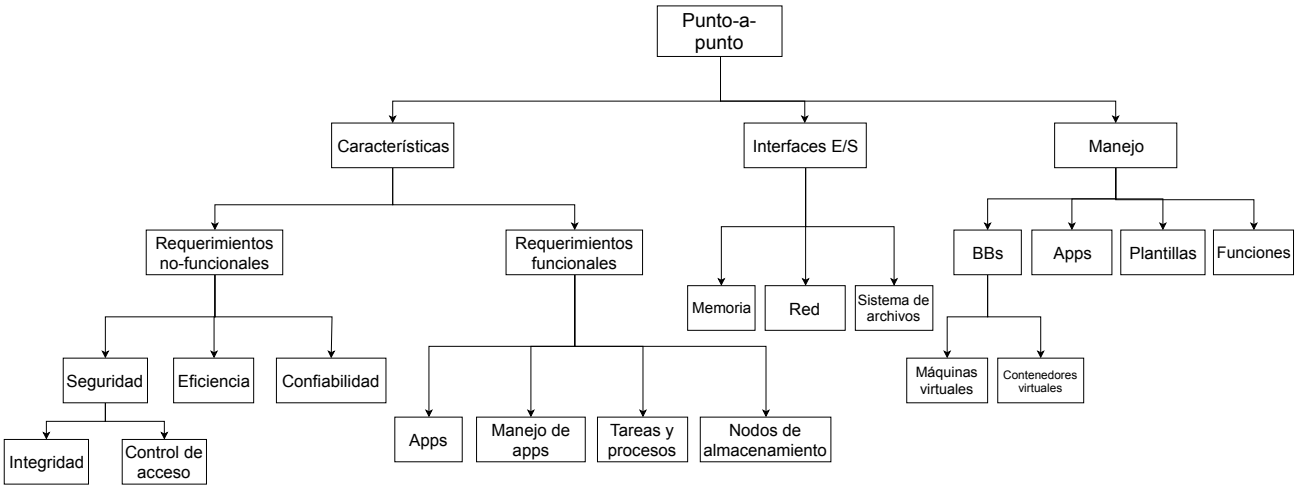


Figura 3.7: Taxonomía de la clasificación de soluciones punto-a-punto.

tolerancia a fallos (por ejemplo, algoritmos de dispersión de información).

Además, se analizaron diferentes soluciones punto-a-punto y de flujos de trabajo, las cuales fueron clasificadas de acuerdo a los requerimientos no funcionales que manejan. Para ello, se consideraron características tales como eficiencia de procesamiento y almacenamiento de datos, integridad de datos, confiabilidad basada en codificación de datos y confiabilidad de procesamiento (tolerancia a fallos), así como seguridad basada en control de acceso y confidencialidad.

Para proveer eficiencia en el procesamiento de datos, en la literatura se consideran técnicas de paralelismo basadas piscinas de hilos, patrones paralelos (por ejemplo, *manejador/trabajador* o *divide&conquer*), y esquemas en memoria compartida. Dichas técnicas, en el mejor de los casos, permiten mejorar el rendimiento en el procesamiento de datos reduciendo el impacto de los cuellos de botella en las tuberías de procesamiento, así como en los flujos de trabajo. En este contexto, soluciones como Makeflow [2], Jenkins [117], DagOn* [85] y Parsl [8] proporcionan paralelismo implícito basado en hilos para la ejecución de tareas en un flujo de trabajo/tubería. También, se han propuesto esquemas en memoria compartida para tubería y flujos de trabajo no distribuidos [47, 110]. Por otro lado, las técnicas para la eficiencia de almacenamiento de datos, también llamados esquemas de almacenamiento costo-eficiencia, basados en técnicas como la deduplicación y la compresión de

datos permiten reducir el volumen de datos a transportar por la red y a almacenar en la nube [130]. Las técnicas de deduplicación permiten encontrar datos replicados en un conjunto de datos mediante su huella digital (hash) [78]. Ejemplos de sistemas de deduplicación incluyen StorReduce [83] y Dedupv1 [79] .

Soluciones de integridad de datos se han propuesto para verificar la consistencia de los datos en conexiones punto-a-punto. Estas soluciones permiten identificar alteraciones ocurridas en los datos durante el traslado de los mismos (por ejemplo, cuando un archivo es enviado a través de la red puede ser accedido y modificado por terceros) [69]. Además, las soluciones de integridad de datos consideran aplicar técnicas de suma de verificación (por ejemplo, SHA3 o MD5) a los datos antes de la transmisión de estos para detectar alteraciones [22].

La confidencialidad es definida como la capacidad de una solución para preservar la privacidad de los datos entre las etapas de procesamiento y los nodos de un flujo de trabajo [100]; mientras que el control de acceso se define como la capacidad de la solución para dar acceso a los datos solo a aquellos usuarios autorizados [87]. En la literatura se han propuesto servicios como los presentados por Wiseman et al. [130], Zhang et al. [133] y Morales et al. [87].

Técnicas de tolerancia a fallas de datos utilizando técnicas de dispersión de información y codificación de datos son ejemplos de soluciones de confiabilidad [47, 73]. En este sentido, las tuberías de confiabilidad se han estudiado principalmente en dos direcciones: i) relanzar la tubería de procesamiento completa y comenzar desde el principio el procesamiento de datos, que es el caso de soluciones como Parsl [8] y DagOn* [85]; o ii) recuperar la etapa que presento la falla, y reiniciar el tubería desde el último estado seguro antes de que la falla ocurriera, como lo es el caso de Sacbe [47].

La Tabla 3.2 muestra una comparación de los trabajos descritos anteriormente. La tabla se divide en dos subcategorías: el alcance de la solución (de punto-a-punto, tuberías de CD/CI, motor de flujo de trabajo o sistema de deduplicación) y los requisitos no funcionales aplicados en la solución.

En la segunda subcategoría, se consideraron los requisitos no funcionales de eficiencia para el procesamiento de datos (subprocesos, patrones y en memoria) y el almacenamiento de datos

Tabla 3.2: Comparación cualitativa de soluciones punto-a-punto y motores de flujos de trabajo.

Trabajo	Enfoque	Eficiencia					Confiabilidad		Seguridad		
		Procesamiento			Almacenamiento						
		Hilos	Patrones	Memoria	Compresión	Deduplicación	Datos	Procesamiento	Control de acceso	Confidencialidad	Integridad
Wiseman et al. (2005) [130]	punto-a-punto	-	-	-	✓	-	-	-	-	-	-
CloudSeal (2011) [132]	punto-a-punto	-	-	-	-	-	-	-	✓	✓	✓
Jenkins (2011) [117]	Tuberías para CD/CI	✓	-	-	-	-	-	✓	✓	-	-
Makeflow (2012) [2]	Motor de FT	✓	-	-	-	-	-	✓	-	-	-
Zhang et al. (2015) [133]	punto-a-punto	-	-	-	-	-	-	-	✓	✓	-
Mao et al. (2015) [73]	punto-a-punto	-	-	-	-	-	✓	-	-	-	-
StorReduce (2018) [83]	Sistema de deduplicación	-	-	-	-	✓	-	-	-	-	✓
AES4SeC (2018) [87]	punto-a-punto	-	-	-	-	-	-	-	✓	✓	✓
Sacbe (2018) [47]	punto-a-punto	-	-	✓	-	-	✓	✓	-	-	-
Dedupv1 (2010) [79]	Sistema de deduplicación	-	-	-	-	✓	-	-	-	-	✓
Parsl (2018) [8]	Motor de FT	✓	✓	-	-	-	-	✓	-	-	-
DagOn* (2018) [85]	Motor de FT para IoT	✓	-	-	-	-	-	✓	✓	-	-

FT = flujo de trabajo

(compresión y deduplicación), confiabilidad para la tolerancia a fallas de las ubicaciones de almacenamiento de datos y aplicaciones de procesamiento, y seguridad aplicando técnicas de control de acceso de usuarios, confidencialidad de datos e integridad de datos.

Como es posible observar en la Tabla 3.2, la mayoría de las soluciones son de tipo punto-a-punto y están centradas (en su mayoría) en cubrir el requerimiento de seguridad (control de acceso, confidencialidad o integridad) que en el requerimiento de confiabilidad o eficiencia.

4

Diseño y desarrollo del modelo de programación de esquemas de almacenamiento de datos definidos por código

En este capítulo se describe el modelo de programación y despliegue de esquemas de almacenamiento de datos definidos por código.

4.1 Arquitectura de bloques de construcción (*PBB's* y *DBB's*)

La unidad básica de este modelo son los bloques de construcción ($BB's$), los cuales se dividen en dos estructuras constructivas modulares llamadas bloques de construcción de procesamiento (*PBB's*) y bloques de construcción de datos (*DBB's*) (ver Figura 4.1).

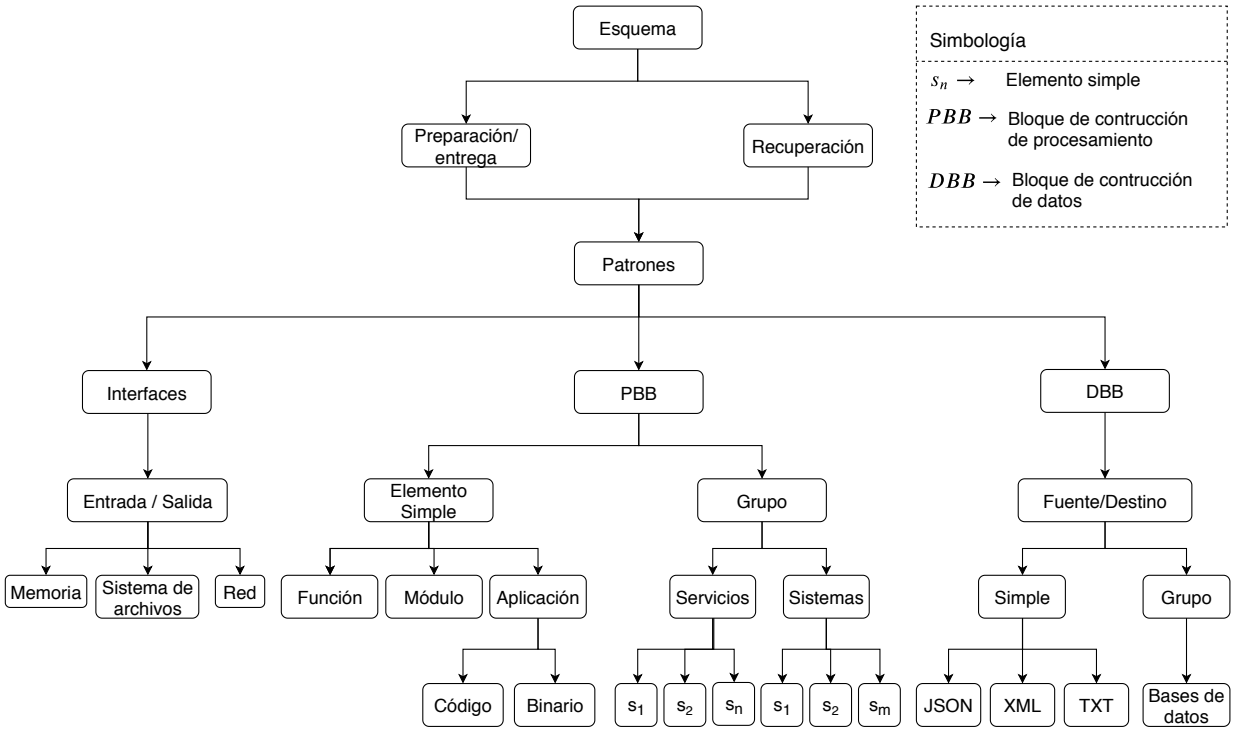


Figura 4.1: Taxonomía para la construcción de secuencias de código.

Los *PBBs* pueden ser de dos tipos: *i)* elementos simples y *ii)* grupos. Donde los elementos simples pueden ser funciones, módulos y aplicaciones (ya sean en código o binarios). Mientras que los grupos pueden ser servicios o sistemas, los cuales son un conjunto de n elementos simples. Por ejemplo, es posible tener un *PBB* que contenga un algoritmo de compresión de datos (como LZ4 [10] o ZIP [53]) para mejorar la eficiencia de almacenamiento, o un *PBB* que permita añadir un módulo de dispersión de información (como IDA [101]) para brindar tolerancia a fallos.

Los *DBB's* se refieren a todas aquellas fuentes o destinos de datos, los cuales pueden ser de dos tipos: *i)* simples (tales como archivos JSON, XML y TXT) o *ii)* grupos (tales como bases de datos). Por ejemplo, es posible tener una fuente de datos (*DBB*) que permita extraer datos médicos (por ejemplo, tomografías y mamografías), los cuales serán procesados en un *PBB*, para posteriormente ser almacenados en un destino de datos (otro *DBB*). Es importante destacar que los *DBB's* deben contar con un archivo de configuración, el cual permitirá especificar las credenciales de acceso que permitirán extraer o almacenar los datos en ellos. Algunos de los elementos que se especifican en

esto archivos de configuración, son los siguientes: contraseña, usuario y tipo, por mencionar algunos.

Los *PBB's* y *DBB's* son asociados a identificadores y su ciclo de vida (creación, modificación, acoplamiento, y eliminación) es gestionado mediante secuencias de código.

4.2 Esquemas de almacenamiento de datos

En este modelo, los esquemas de almacenamiento se crean mediante el acoplamiento de bloques, los cuales contienen los algoritmos que cumplimentan requerimientos no funcionales (NFR) tales como seguridad, confiabilidad, eficiencia e integridad.

Los esquemas se organizan en la forma de tuberías de procesamiento (para más detalles ver Sección 4.4.2) cuya función es unir las propiedades de cada bloque para agregarlas gradualmente a los datos, y de esta forma atender tantos requerimiento no-funcionales como bloques incluidos en un esquema existan. Dos tipos de esquemas se han definido en este modelo: los esquemas de preparación y los esquemas de recuperación de datos.

Los esquemas de preparación se crean acoplando bloques en una tubería que es utilizada antes de compartir, almacenar, transportar y/o descargar datos desde, o través de una ubicación remota. En forma proactiva, estos esquemas preparan los datos antes de almacenarlos en un sitio (e.g. nube). Estos esquemas se pueden entender como un proceso de codificación donde los bloques se ejecutan de acuerdo a una secuencia establecida en el código (por ejemplo, la tubería de procesamiento previamente descrita). Lo cual, como ya se ha establecido, aseguran el cumplimiento de los NFRs considerados por los bloques incluidos en un esquema.

Por ejemplo, en una organización se tienen diferentes clases de documentos clasificados de acuerdo con su nivel de importancia, los cuales requieren ser manejados utilizando diferentes requerimientos no funcionales. Para manejar dichos documentos, la organización desarrolló tres bloques de construcción

para generar diferentes esquemas de preparación de datos. Estos bloques son:

$$\begin{aligned} PBB_A =& DataCompressLZ4 \\ PBB_B =& IDA \\ PBB_C =& CPABE, \end{aligned} \tag{4.1}$$

donde el PBB_A contiene un algoritmo de compresión de datos, lo cual brinda eficiencia de almacenamiento, ya que reduce el volumen de datos a procesar y almacenar. El PBB_B que es un algoritmo de dispersión de información, el cual permite tolerar fallas brindando la característica de confiabilidad a los datos. Y el PBB_C, es un módulo de cifrado/descifrado de datos, el cual permite agregar la característica de seguridad a los datos. Estos bloques pueden ser combinados para generar diferentes esquemas de preparación de datos, por ejemplo una solución que incluya los tres bloques para documentos de prioridad mayor ($E_1 = PBB_A \rightarrow PBB_B \rightarrow PBB_C$), otro esquema para documentos de prioridad media ($E_2 = PBB_A \rightarrow PBB_B$), y esquemas para documentos de mínima prioridad ($E_3 = PBB_A$). Cada una de estas soluciones, tendrá una fuente y un destino de datos. En este ejemplo como fuente se tiene un clúster privado (DBB_1), y como destino un servidor privado en la nube (DBB_2).

La Figura 4.2 muestra la representación de las tres soluciones previamente creadas a partir de los bloques existentes como un DAG. En este contexto, se puede definir que es posible crear tantos esquemas como combinaciones de bloques existan.

Los esquemas de recuperación permiten administrar los datos que han sido preparados anteriormente y se crean bajo demanda ejecutando los bloques de un esquema de preparación en forma inversa. Estos esquemas se pueden entender como procesos de decodificación que permiten acceder a los datos que han sido previamente preparados para reunir ciertos requerimientos no funcionales (ver esquema previamente descrito, Figura 4.2).

En las siguientes secciones se describirán cinco componentes claves para la creación de esquemas

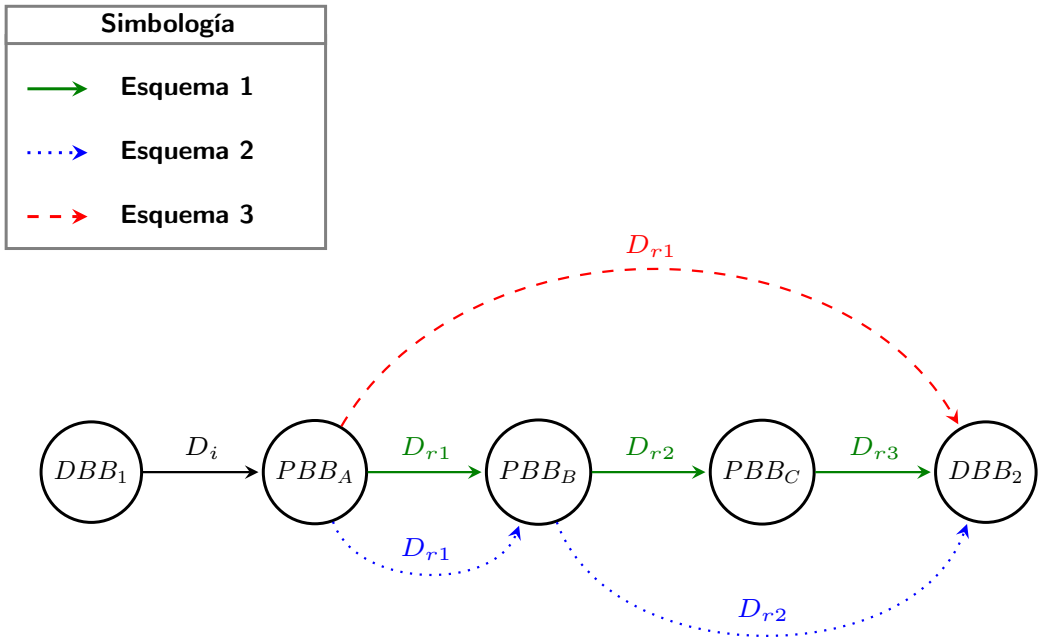


Figura 4.2: Representación de los esquemas creados a partir de los bloques existentes como un grafo acíclico dirigido (DAG).

de almacenamiento dinámicos. El primer componente es una arquitectura de gestión de esquemas. El segundo es un modelo de construcción de esquemas de preparación y recuperación desplegables en diferentes ambientes, el tercero es un modelo de programación de bloques para crear crear secuencias de código. El cuarto, son patrones recursivos de procesamiento. Y el último, un repositorio de esquemas de preparación y recuperación de datos.

4.3 Descripción general de la propuesta

En esta sección se describe una arquitectura para la creación de esquemas de almacenamiento mediante la interpretación de secuencias de código.

La Figura 4.3, muestra las tres etapas principales del método para la creación de esquemas de almacenamiento de datos definidos por código. Dichas etapas se describen a continuación:

- Desarrollo y diseño:

Figura 4.3: Etapas del método para creación de esquemas de almacenamiento.

En dicha etapa es necesario desarrollar código, el cual permita producir los PBB's y DBB's, para la construcción de los esquemas de almacenamiento.

- Configuración y despliegue:

  En esta etapa, es necesario desarrollar estructuras de manejo para producir un manejador de los BB's, así como los NFR-BBs adaptables, los archivos de configuración de los PBB's y DBB's, y sus respectivas imágenes de contenedor.

- Ejecución:

  En la etapa de ejecución es necesario desarrollar servicios de almacenamiento, los cuales permitan conseguir los servicios de PBB's y DBB's.

En el presente trabajo de tesis los PBB's y DBB's se encontrarán encapsulados en VC's para dotar a las aplicaciones con las propiedades de portabilidad. Para ello, los VC's encapsulan el código fuente o binario de la aplicación, junto con estructuras de control que permiten el manejo y acoplamiento de los bloques de construcción.

Además, se desarrolló un intérprete de secuencias de código el cual se encarga de analizar las secuencias de código, en donde se especifiquen los requerimientos funcionales y no funcionales críticos

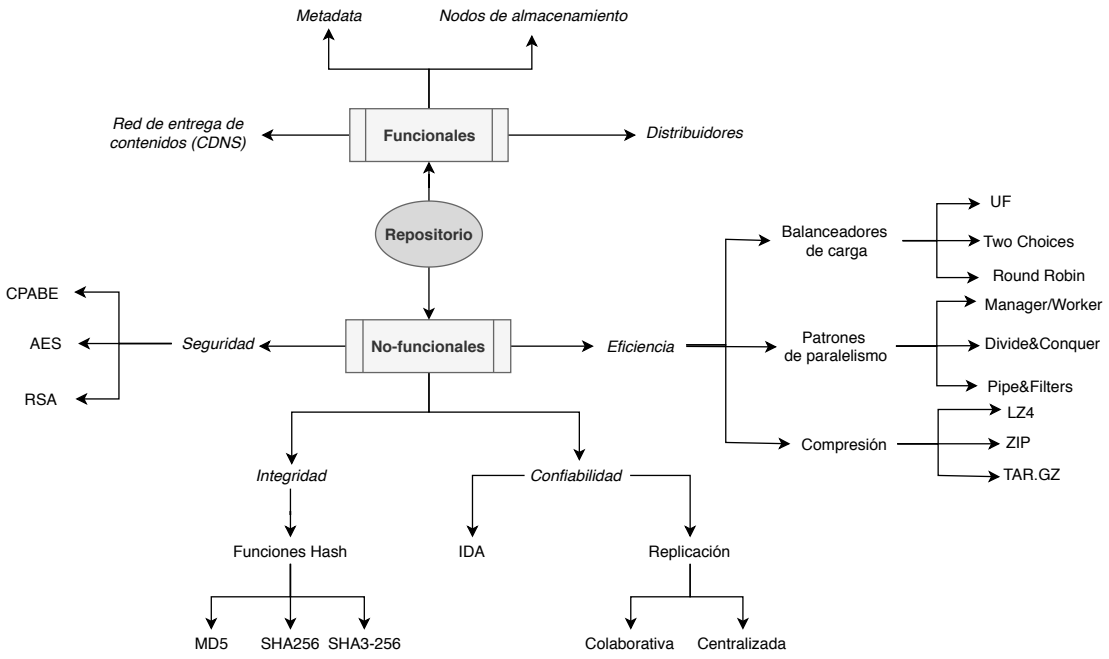


Figura 4.4: Taxonomía de requerimientos funcionales y no-funcionales para la construcción de esquemas de almacenamiento de datos.

para una organización. Posteriormente, un coordinador se encarga de manejar la comunicación entre cada BB y un distribuidor se encarga de repartir de manera continua tareas y/o datos entre ellos. Los esquemas de preparación/entrega y recuperación de datos, son generados mediante la concatenación de BB's, los cuales forman patrones para generar el esquema de almacenamiento, que posteriormente será desplegado en la infraestructura de almacenamiento especificada dicha secuencia.

Finalmente, con el objetivo de construir los esquemas propuestos en esta tesis, se generó un repositorio de requerimientos funcionales y no-funcionales. La Figura 4.4 muestra la taxonomía de los requerimientos contemplados en la presente tesis.

4.4 Modelo de construcción y programación de esquemas de almacenamiento programables

Esta sección presenta un modelo de procesamiento continuo de datos basado en abstracciones llamadas *cajas negras* ($BBox$), el cual permite a los desarrolladores construir estructuras de software para procesar datos siguiendo el principio de entrega continua utilizando bloques de construcción portables y agnósticos de la infraestructura, los cuales se encuentran encapsulados en contenedores virtuales llamados *building blocks* (BBs).

Para crear estas soluciones de preparación y recuperación de datos, son necesarios los siguientes pasos: i) encapsular un conjunto de aplicaciones con sus bibliotecas y dependencias, así como encapsular las estructuras en bloques de construcción (BBs); ii) asociar estructuras de BBs con cajas negras, las cuales incluyen estructuras de control; iii) aplicar el concepto de entrega continua, el cual se encuentra presente en la ingeniería de software, para acoplar cajas negras y crear soluciones de preparación y recuperación de datos [1]; y iv) realizar el procesamiento y entrega continua de datos a través de $BBoxes$ y BBs.

Para administrar, crear e implementar las soluciones de preparación y recuperación de datos, se propuso un modelo basado *grafos acíclicos dirigidos* (DAGs). Los nodos del DAG representan cajas negras, las cuales a su vez incluyen un DAG donde los nodos representan bloques de construcción. Los bordes representan interfaces de entrada/salida que se utilizan para realizar las interconexiones entre cajas negras (nivel superior), y las interconexiones entre los bloques de construcción (nivel inferior).

Estas interfaces de entrada/salida son administradas por estructuras de control integradas dentro de los $Bboxes$ y los BBs. En este modelo, las interfaces de entrada/salida se pueden configurar para

[1] La entrega continua es una técnica popular en la ingeniería de software que permite entregar actualizaciones de software de los desarrolladores a través de diferentes etapas de prueba a los usuarios finales de manera interrumpida [23, 56].

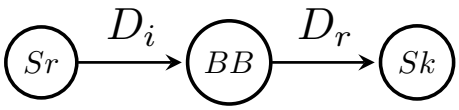


Figura 4.5: Representación del proceso de extracción, transformación y carga (ETL) de un bloque de construcción (BB) como un grafo acíclico dirigido (DAG).

usar cualquiera de las interfaces (memoria, red o sistema de archivos).

Para lograr que los nodos de estos DAGs se interconecten, las estructuras se definen mediante el uso de un modelo de procesamiento de transformación, extracción y carga (ETL, del inglés *Extraction, Transformation and Load*); como resultado, un $BBox$ se puede definir mediante la siguiente expresión:

$$BBox = (DSr, (BB_{i=1}^{n}, DAG), DSk), \tag{4.2}$$

donde DSr es una fuente de datos conectada a la entrada de $BBox$, el cual representa la etapa de extracción. $BB_{i=1}^{n}$ es un conjunto de bloques de construcción administrados por $BBox$ para realizar la etapa de transformación, DAG representa la topología de las conexiones BB y DSk es un destino de datos conectado a la salida de $BBox$ (el cual representa la etapa de carga). Por otro lado, un BB se modela utilizando el mismo modelo de procesamiento utilizado para los $BBoxes$ mediante la siguiente expresión:

$$BB = (in, task, out), \tag{4.3}$$

Como se puede ver, un BB incluye interfaces de entrada y salida, así como una aplicación para procesar datos, la cual puede ser cualquier programa, aplicación, binario o módulo. Donde in es la interfaz de entrada, $task$ es la unidad de procesamiento en el bloque de creación y out es la interfaz de salida. Cuando un $BBox$ y un BB son modelados por un ETL, se crea una tubería como se muestra en la Figura 4.5, donde se requiere $n \geq 1$ [16, 51]. La figura 4.5 muestra un ejemplo de BB definido por un DAG como modelo ETL, donde los datos D_i son extraídos de una fuente de datos (Sr) por una entidad transformadora (BB), que entrega los datos transformados en forma de resultados (D_r) a un sumidero de datos (Sk).

El objetivo del modelo propuesto es producir un flujo de datos a través de las aristas (interfaces de entrada/salida) para los nodos ($BBoxes$ y BBs), lo cual permite el procesamiento de los datos entrantes y producir resultados de manera ininterrumpida. Esto se logra mediante el uso del concepto de entrega continua para entregar datos de manera continua e ininterrumpida, y para establecer control sobre la ejecución de los nodos y aristas definidos en el árbol de DAG.

En este modelo, las abstracciones ($BBoxes$ y BBs) y las estructuras de preparación y recuperación de datos creadas por las combinaciones de $BBoxes$ y BBs, se administran como servicios, lo cual permite a los usuarios finales reutilizar los $BBoxes$ y BBs para construir múltiples soluciones, así como para extraer los datos de salida de cada caja negra conectando aplicaciones al servicio correspondiente. Las estructuras de control de los servicios se encuentran integradas dentro de $BBoxes$ y BBs, los cuales son descritos en la Sección 4.4.1.

4.4.1 Diseño de soluciones de preparación y recuperación de datos basadas en patrones

En esta sección se describen diferentes tipos de patrones para crear soluciones de preparación y recuperación de datos, como tuberías, patrones de paralelismo y flujos de trabajo que se pueden construir utilizando el modelo de procesamiento ETL descrito anteriormente.

4.4.2 Tuberías de procesamiento

En este modelo, una tubería de procesamiento es modela como un conjunto de $BBoxes$ y BBs (filtros) conectados de manera adyacente a través de rutas de entrada/salida (tuberías), denotadas por: $PBBoxes = (\{BBoxes_1, BBoxes_2, \ldots, BBoxes_n\})$ y $PBBs = (\{BB_1, BB_2, \ldots, BB_n\})$.

Los datos de entrada recibidos y procesados por los BBs / $BBoxes$ se denotan como $BB[in]$, mientras que los datos resultantes después del procesamiento de los BBs / $BBoxes$ se denotan como $BB[Out]$, y el procesamiento de tareas en los BBs se denotan como $BB[task]$. El cual es un

patrón de procesamiento ETL, denominado $Patt_{etl}$, y puede ser definido como:

$$Patt_{etl} = \{Sr_i \rightarrow BB_i[in], BB_i[task], BB_i[Out] \rightarrow Sk_i\}, \tag{4.4}$$

donde Sr representa una fuente de datos como entrada, y Sk representa un destino de datos para entregar los resultados producidos por el BB. La notación de un patrón ETL es generalizada como la definición de BBs encadenados. Para un determinado BB_j, en un patrón $Patt_{etl,1}$, su fuente Sr_j puede ser el destino Sk_i del BB_i. La salida del BB_j se puede asociar a la fuente del BB_k dado el $Patt_{etl,2}$. Lo anterior se denota como:

$$\{Patt_{etl,1}, BB_j\} = \{Sk_i, BB_j, Sk_j\}, \tag{4.5}$$

$$\{BB_j, Patt_{etl,2}\} = \{Sr_j, BB_j, Sr_k\}. \tag{4.6}$$

La Ecuación (4.5) representa el encadenamiento del BB_i con el BB_j y la Ecuación (4.6) representa el encadenamiento del BB_j con el BB_k. La notación anterior, es utilizada para la implementación, el acoplamiento y la ejecución de los BBs considerados en un patrón.

La Figura 4.6 muestra un grafo que representa un patrón creado mediante una tubería de procesamiento, el cual crea una secuencia de procesamiento de BBs (nodos) conectados de forma adyacente dentro de una caja negra, a través de las interfaces de entrada/salida (aristas). La notación de la caja negra para esta tubería se denota de la siguiente manera:

$$Pipeline = (DSr, (BB_{i=1}^{n}, Patt_{etl,x}), DSk), \tag{4.7}$$

donde $n \geq 1$, DSr está conectado al $BB_1[entrada]$, DSk está conectado al $BB_n[salida]$, n es el número de BBs en la tubería y $Patt_{etl,x}$ se define como:

$$Patt_{etl,x} = \{BB_i[Out] \rightarrow BB_{i+1}[In]\}_{i=1}^{n-1}. \tag{4.8}$$

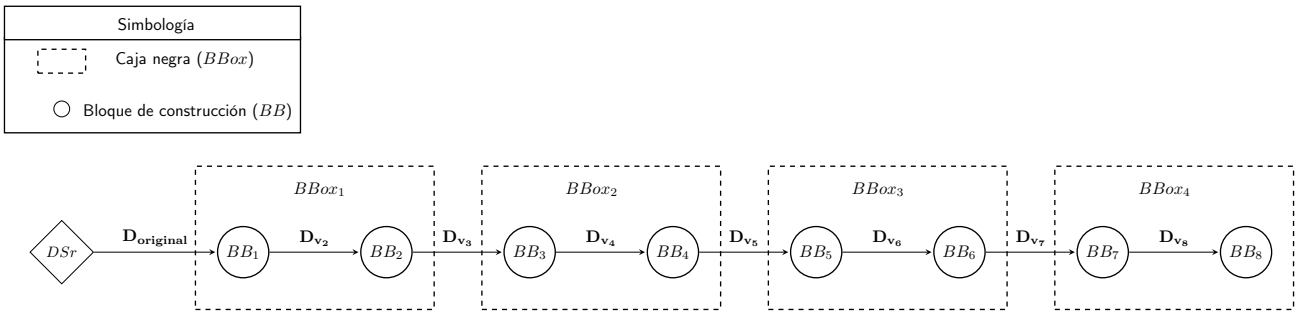


Figura 4.6: Representación de una estructura para procesar datos basados en $BBoxes$ y BBs en forma de grafo acíclico dirigido (DAG).

Dicho patrón crea un flujo de datos desde una fuente de datos (DSr) a un destino de datos (DSk). En dicho flujo de datos, cada BB_i transforma los datos recibidos de los datos de entrada, mientras que el BB_1 transforma este contenido en una nueva versión, y la reenvía a un destino de datos o a otro BB a través de una tubería.

4.4.3 Patrones de paralelismo implícitos a nivel de caja negra

Este modelo permite a los desarrolladores de soluciones de preparación/recuperación de datos crear patrones de paralelismo implícitos. Lo cual permite mejorar la eficiencia de las cajas negras de manera transparente al incluir diferentes estructuras de control como: $divisor$ (D) y $trabajador$ (w), y en algunos casos $consolidador$ (C) reservado para BBs.

La Figura 4.7 muestra un ejemplo de un patrón que incluye un divisor (D) que extrae datos de una fuente de datos (Dsr) y distribuye estos datos en forma de tareas ($P = \{c_1, c_2, \ldots, c_n\}$, donde n es el tamaño del conjunto de contenidos c_i) para los trabajadores generando w subconjuntos de contenidos ($p_w = \{c_1, c_2, \ldots, c_m\}$). Cada trabajador transforma estos subconjuntos en una nueva versión del contenido ($p'_w = \{c'_1, c'_2, \ldots, c'_m\}$), donde m es el tamaño del subconjunto distribuido a cada trabajador y $m \leq n$. En este ejemplo, los trabajadores procesan cada tarea recibida (c_i) para producir datos procesados (c'_i), que se almacenan en un destino de datos dado (DSk), el cual puede ser colocado en otra caja negra (por ejemplo, cualesquiera de los $BBox_2$, $BBox_3$ en una ubicación

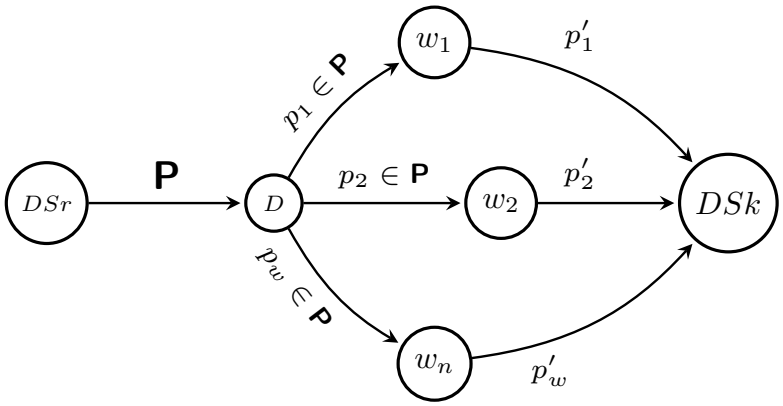


Figura 4.7: Representación del patrón Divide y Conteneriza como un grafo acíclico dirigido (DAG).

de almacenamiento previamente definida, o el $BBox_4$). Este método produce un patrón tradicional de $Manejador/Trabajador$ [94], donde los trabajadores pueden entregar los resultados a cualquier caja negra, destino de datos, otra división (de manera recursiva) o a una entidad $consolidador$.

Se requiere un BB $consolidador$ cuando los datos procesados no pueden ser utilizados por separado por un $BBox$. Esto sucede cuando la instancia de división divide los datos entrantes de una caja negra en segmentos, los cuales son enviados a los trabajadores para procesar los datos. Los trabajadores envían los datos procesados al consolidador, el cual organiza los resultados recibidos para devolver un resultado consolidado al control $BBox$ en una acción de reducción. Este método produce un patrón tradicional llamado *Divide y vencerás*, también llamado *división/unión* ($fork/join$ [94]), dependiendo de las estructuras de control utilizadas por los trabajadores y el consolidador. El mapeo de un patrón **Divide y vencerás** se denota de la siguiente manera:

$$BB_{Patt} = \{DSr, (D, (w_{i=1}^{n}, Patt), Cq), DSk\}, \tag{4.9}$$

donde $n > 1$, $Patt = \{D\&C \xrightarrow{s_i} w_i \xrightarrow{r_i} Cq\}_{i=1}^{n}$, y Cq es una entidad consolidadora. En el caso de un patrón *división/unión*, la notación es similar a la anterior, pero cambia Cq por $Join$, el cual también representa la entidad consolidadora. Como se puede ver, en este modelo se pueden construir diferentes tipos de combinaciones de patrones ($Patt$) para crear un BB, de la misma manera es

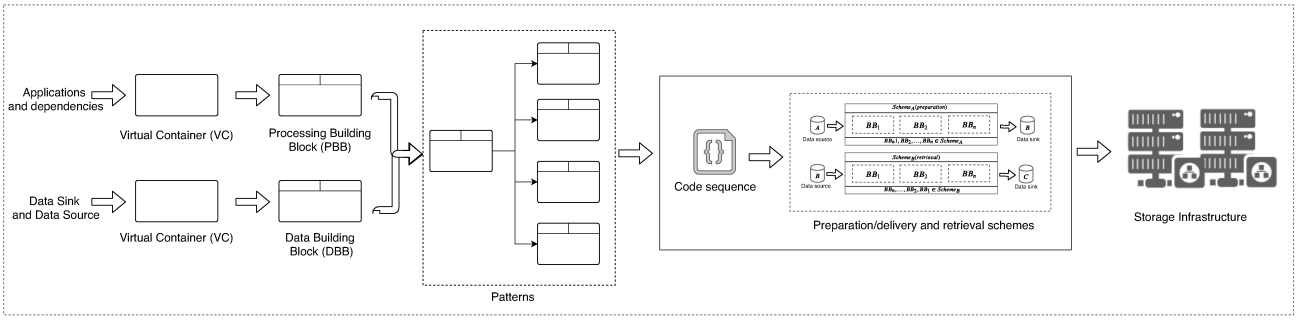


Figura 4.8: Representación conceptual del despliegue de esquemas de almacenamiento.

posible construir una tubería de software de BBs para crear una caja negra ($BBox$).

Hay que tener en cuenta que los patrones de paralelismo también son administrados como un servicio (un microservicio de manera formal), lo que significa que se agregan estructuras de control adicionales a los patrones para hacer cumplir la gestión implícita del procesamiento de datos. En este contexto, las estructuras de control consideradas para estas tareas son un equilibrio de carga de BB para la carga de trabajo de asignación, así como mensajes y administradores de entrada/salida de datos para aplicar el modelo ETL en los patrones, lo que incluye una biblioteca de recursos de datos en memoria para reducir operaciones enviadas a interfaces lentas (por ejemplo, sistema de archivos y red).

Los grafos acíclicos dirigidos utilizados en este modelo permiten la construcción de patrones de paralelismo para hacer frente a problemas de eficiencia que se presentan cuando se deben preparar o recuperar grandes volúmenes de datos. En esta tesis se contemplan patrones tales como *Manejador/Trabajador*, *Divide y vencerás*, y *División/Unión*.

4.4.4 Despliegue y acoplamiento de esquemas de preparación y recuperación de datos

La Figura 4.8 ilustra el proceso de despliegue de los esquemas de almacenamiento a partir de una secuencia de código. Las etapas de despliegue son las siguientes:

1. Las aplicaciones y sus dependencias son encapsuladas en VC's, lo que permitirá su portabilidad entre infraestructuras.

2. Estructuras de manejo son agregadas para generar bloques de construcción que serán encadenados para generar los esquemas.

3. Los bloques de construcción son encadenados de acuerdo cómo se indica en la secuencia de código para generar patrones de procesamiento de datos.

4. Los PBB's son acoplados de acuerdo con la secuencia de código la cual indica un flujo de datos desde un DBB fuente hasta un DBB de destino.

5. Los esquemas son desplegados en la infraestructura de almacenamiento indicada.

4.4.5 Configuración y despliegue de esquemas de almacenamiento

La Figura 4.9 muestra la arquitectura del servicio de configuración y despliegue de los esquemas de preparación y recuperación de datos. En la parte superior de la arquitectura se encuentra el intérprete de secuencias de código, el cual se encarga de validar que la estructura del código sea correcta. Posteriormente, se encuentra un orquestador que es el encargado de seguir la secuencia del código para identificar las etapas de procesamiento y sus interconexiones generando un DAG y un conjunto de mapas.

Además, el orquestador también se encarga de encapsular los componentes de los esquemas y su posterior despliegue en la infraestructura señalada. Finalmente, en la parte inferior de la arquitectura se encuentra el coreógrafo, que es el encargado de generar los BBs que componen el esquema y las soluciones y servicios de almacenamiento.

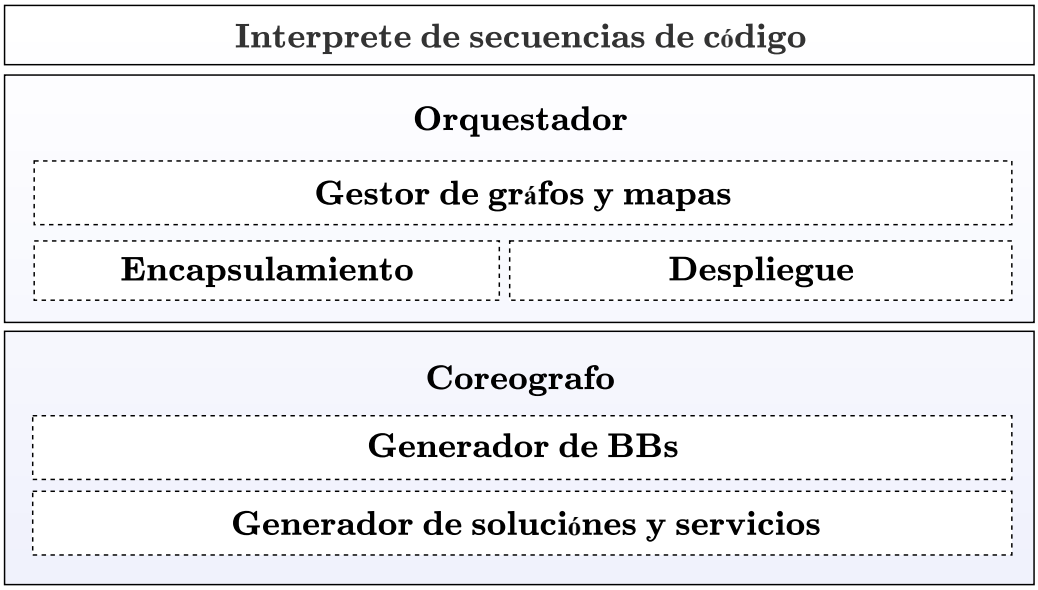


Figura 4.9: Arquitectura de pila del servicio para la configuración y despliegue de los esquemas de almacenamiento definidos por código.

4.5 Diseño de esquemas de almacenamiento mediante código

A partir del modelo presentado en la Sección 4.7 se definió una notación de secuencias de código, las cuales permiten a las organizaciones crear y manejar los esquemas de preparación/recuperación de datos.

En este sentido, un esquema se codifica utilizando las siguientes abstracciones, las cuales se encuentran basadas en el modelo ETL previamente descrito:

- $Extractor$: esta abstracción se encarga de realizar el proceso de extracción de datos para su posterior procesamiento a través de las diferentes interfaces de entrada (red, memoria o sistema de ficheros), o a los diferentes ambientes de trabajo (nube, niebla y borde).

- $Transformador$: esta abstracción se encarga de realizar el proceso de transformación de datos mediante el encapsulamiento de distintas aplicaciones, funciones y/o algoritmos para el

procesamiento de los datos. Dichas abstracciones permiten brindar a los datos las características de confiabilidad, disponibilidad, eficiencia y seguridad.

- $Cargador$: esta abstracción se encarga de realizar el proceso de carga de datos para su preservación o trasformación, mediante el uso de las diferentes interfaces de salida (memoria, red, sistema de archivos).

- $Pipeline$: esta abstracción se encarga de la generación de tuberías de procesamiento creadas a partir de la concatenación de las abstracciones anteriores ($Extractor \rightarrow Transformador \rightarrow Carga$).

- $Esquema$: esta abstracción se encarga de realizar la construcción de los esquemas de almacenamiento de datos a través de la generación de secuencias de código utilizando las abstracciones anteriores.

Cada una de estas abstracciones cuenta con una serie de funciones que permiten realizar la creación de los elementos necesarios para la generación de los esquemas de preparación/recuperación de datos. La Figura 4.10 muestra las funciones CRUD (Crear, leer, actualizar y eliminar, del inglés *Create, read, update and delete*) implementadas para el manejo de las abstracciones $Extractor$, $Transformador$ y $Cargador$, previamente descritas.

Adicionalmente se pueden crear patrones de control implícito, solo basta con indicar el bloque que se utilizará (de los existentes), el tipo de patrón (por ejemplo *Manejador/Trabajador*), su utilización y los parámetros del patrón.

4.5.1 Especificaciones de la sintaxis para la generación de secuencias de código

En esta sección se presentan las reglas léxicas y de sintaxis utilizadas para la generación de secuencias de código.

	Crear	Leer	Actualizar	Eliminar
Extractor	- CreateExtractors()	- ReadExtractors()	- UpdateExtractors()	- DeleteExtractors()
Transformador	- CreateBlocks() - CreatePipeline() - CreateTransform()	- ReadBlocks() - ReadPipeline() - ReadTransform()	- UpdateBlocks() - UpdatePipeline() - UpdateTransform()	- DeleteBlocks() - DeletePipeline() - DeleteTransform()
Cargador	- CreateLoader()	- ReadLoader()	- UpdateLoader()	- DeleteLoader()

Figura 4.10: Matriz de funciones para la creación de secuencias de código para la generación de esquemas de almacenamiento de datos.

4.5.1.1. Notación extendida de las secuencias de código

En la presente sección se presenta la notación BNF (notación de Backus-Naur) [2] extendida que se diseño para crear las secuencias de código propuestas.

En las reglas léxicas y de sintaxis que figuran a continuación, los caracteres de notación se encuentran en color azul.

Las alternativas se encuentran separadas por barras verticales: es decir, 'a | b ' significa "a o b".

El nombre de un elemento en la sintaxis se encuentra denotado por los símbolos de comparación: es decir, '<a>' representa un elemento dentro de la sintaxis.

Las llaves indican repetición: ' {a}' significa " a | aa | aaa | . . . ".

[2] BNF es un metalenguaje, es decir, un lenguaje que describe sintácticamente a otro lenguaje de programación. Al utilizar BNF es posible especificar qué secuencias de símbolos constituyen un programa sintácticamente válido en un idioma dado [76].

4.5.1.2. Reglas léxicas

A continuación se presentan las reglas léxicas y las palabras clave necesarias para la generación de secuencias de código.

letra ::= a | b | . . . | z | A | B | . . . | Z

dígito ::= 0 | 1 | . . . | 9

identificador ::= letra { letra | dígito | _ }

número ::= dígito { dígito }

cadena ::= "{ch}", donde ch es un carácter ASCII.

carácter ::= 'ch', donde ch es un carácter ASCII.

keywords ::= **createScheme** | **createTransoformers** | **createExtractors** | **createLoaders** | **createBlocks** | **createPipeline** | **TYPE** | **In** | **Out** | **processData** | **createPatt** | **processBB** | **dataBB** | **segmenter** | **twoChoices** | **roundRobin** | **random** | **UF** | **extractorsList** | **transformersList** | **tansformersListDetailed** | **transformerDetailed** | **extractorsListDetailed** | **extractorDetailed** | **loadersList** | **loadersListDetailed** | **loaderDetailed** | **blocksList** | **blocksListDetailed** | **blockDetailed** | **pipelinesList** | **pipelinesListDetailed** | **pipelineDetailed** | **updateTransformer** | **updateExtractorsMemory** | **updateExtractorsFS** | **updateLoaderMemory** | **updateLoaderCluster** | **updateLoaderFS** | **updateBlocks** | **updatePipeline** | **deleteTransformers** | **deleteExtractors** | **deleteBlocks** | **deletePipeline** | **createPatt** | **getSourceMemory** | **getSourceCluster** | **getSourceFS** | **putInMemory** | **putInCluster** | **putInFS** | **file_system** | **memory** | **network** | **managerWorker** | **divideConquer** | **forkJoin** | **conquer** | **join** | **updateExtractorsCluster** | **NULL**

4.5.1.3. Reglas de sintaxis

A continuación se presentan las reglas de sintaxis utilizadas para la generación de las secuencias de código. Los no terminales se muestran en *cursiva*; Los terminales se muestran en **negrita**.

Producciones gramaticales

<*importación de paquetes*>::= <*declaración de paquete*>; { <*declaración de paquete*>; }

<*declaración de paquete*>::= **cadena**

<*creación de un programa*>::= {<*creación de un esquema*>; | <*actualización de componentes*>;

| <*eliminación de componentes*>;}

<*creación de un esquema*>::= **createScheme (cadena) {** <*cuerpo_del_esquema*>**}**

<*cuerpo_del_esquema*>::=

<*transformadores*><*extractores*><*cargadores*><*bloques*><*tuberías*>

<*transformadores*>::= **createTransoformers () {** <*transformador*>; {<*transformador*>;} **}** |

<*funciones_lectura_transformador*>

<*transformador*>::= **identificador= processData(** <*imagen*>, <*comando*>, <*tipo_de_BB*>,

<*interfaz_de_entrada*>, <*interfaz_de_salida*>**);** | **identificador = createPatt (** <*fuente*>,

<*divisor*>, <*worker*>, <*Número trabajadores*>, <*tipo*>, <*consolidador*>, <*destino*>**);**

<*imagen*>= **cadena**

<*comando*>= **cadena**

<*tipo_de_BB*>::= **processBB** | **dataBB**

<*interfaz_de_entrada*>::= **file_system** | **memory** | **network**

<*interfaz_de_salida*>::= **file_system** | **memory** | **network**

<*fuente*>::= <*extractor_ID*>

<*divisor*>::= **segmenter** | <*balanceador_de_carga*>

<*balanceador_de_carga*>::= **twoChoices** | **roundRobin** | **random** | **UF**

<*worker*>::= <*tubería_ID*>| <*bloque_ID*>| <*patron_ID*>| <*transformador_ID*>

<*patron_ID*>::= **identificador**

<*Número trabajadores*>::= **número**

<*tipo*>::= **managerWorker** | **divideConquer** | **forkJoin**

<*consolidador*>::= **conquer** | **join** | **NULL**

<*destino*>::= <*cargador_ID*>

<*funciones_lectura_transformador*>::= **identificador = transformersList();** | **identificador = transformersListDetailed();** | **identificador = transformerDetailed (** <*transformador_ID*>**);**

<*extractores*>::= **createExtractors() {** <*extractor*>**;** {<*extractor*>**;**}**}**

<*extractor*>::= **identificador = getSourceMemory(** <*tipo_de_BB*>**,** <*dirección_de_memoria*>**);** | **identificador = getSourceCluster(** <*tipo_de_BB*>**,** <*ruta*>**,** <*usuario*>**,** <*token*>**,** <*contraseña*>**);** | **identificador = getSourceFS(** <*tipo_de_BB*>**,** <*ruta*>**);** | <*funciones_lectura_extractor*>

<*dirección_de_memoria*>::= **cadena**

<*ruta*>::= **cadena**

<*usuario*>::= **cadena**

<*token*>::= **cadena** | **número**

<*contraseña*>::= **cadena** | **número**

<*funciones_lectura_extractor*>::= **identificador = extractorsList();** | **identificador = extractorsListDetailed();** | **identificador = extractorDetailed (** <*extractor_ID*>**);**

<*cargadores*>::= **createLoaders() {** <*cargador*>**;** {<*cargador*>**;**} **}**

<*cargador*>::= **identificador = putInMemory(** <*tipo_de_BB*>**);** | **identificador = putInCluster(** <*tipo_de_BB*>**,** <*ruta*>**,** <*usuario*>**,** <*token*>**,** <*contraseña*>**);** | **identificador = putInFS(** <*tipo_de_BB*>**,** <*ruta*>**);** | <*funciones_lectura_cargador*>

<*funciones_lectura_cargador*>::=

identificador = loadersList(); | **identificador = loadersListDetailed();** | **identificador = loaderDetailed (** <*cargador_ID*>**) ;**

<*bloques*>::= **createBlocks() {** <*bloque*>**;** {<*bloque*>**;**} **}** | <*funciones_lectura_bloques*>

<*bloque*>::= **identificador =** <*extractor_ID*>$->$<*transformador_ID*>$->$<*cargador_ID*>**;**

<*funciones_lectura_bloque*>::= **identificador = blocksList();** | **identificador = blocksListDetailed();** | **identificador = blockDetailed**

(<*cargador_ID*>);

<*extractor_ID*>::= **identificador**

<*transformador_ID*>::= **identificador**

<*cargador_ID*>::= **identificador**

<*tuberías*>::= **createPipeline() { <*tubería*>; {<*tubería*>;} }** | <*funciones_lectura_tubería*>

<*tubería*>::= **identificador** = <*bloque_ID*>− > {<*bloque_ID*>};

<*bloque_ID*>::= **identificador**

<*funciones_lectura_tubería*>::=

identificador = pipelinesList(); | **identificador = pipelinesListDetailed();** | **identificador = pipelineDetailed** (<*tubería_ID*>);

<*tubería_ID*>::= **identificador**

<*actualización de componentes*>::= <*actualizar_transformador*>; | <*actualizar_extractor*>; | <*actualizar_cargador*>; | <*actualizar_bloque*>; | <*actualizar_tubería*>;

<*actualizar_transformador*>::= **updateTransformer** (<*transformador_ID*>, <*imagen*>, <*comando*>, <*tipo_de_BB*>, <*interfaz_de_entrada*>, <*interfaz_de_salida*>);

<*actualizar_extractor*>::= **updateExtractorsMemory** (<*extractor_ID*>, <*tipo_de_BB*>, <*dirección_de_memoria*>); | **updateExtractorsCluster** (<*extractor_ID*>, <*tipo_de_BB*>, <*ruta*>, <*usuario*>, <*token*>, <*contraseña*>); | **updateExtractorsFS** (<*extractor_ID*>, <*tipo_de_BB*>, <*ruta*>);

<*actualizar_cargador*>::= **updateLoaderMemory** (<*cargador_ID*>, <*tipo_de_BB*>); | **updateLoaderCluster** (<*cargador_ID*>, <*tipo_de_BB*>, <*ruta*>, <*usuario*>, <*token*>, <*contraseña*>); | **updateLoaderFS** (<*cargador_ID*>, <*tipo_de_BB*>, <*ruta*>);

<*actualizar_bloque*>::= **updateBlocks(** <*bloque_ID*>,<*extractor_ID*>, <*transformador_ID*>, <*cargador_ID*>);

<*actualizar_tubería*>::= **updatePipeline** (<*tubería_ID*>, <*bloque_ID*>, {<*bloque_ID*>});

<*eliminación de componentes*>::= **deleteTransformers** (<*transformador_ID*>); |

Tabla 4.1: Operadores asociativos y precedentes.

Operador	Asociatividad	**Descripción**
->	de izquierda a derecha	Concatenación
>	de izquierda a derecha	Push (No implementado)
<	de izquierda a derecha	Pull (No implementado)

deleteExtractors (<*extractor_ID*>); | **deleteLoader** (<*cargador_ID*>); | **deleteBlocks** (<*bloque_ID*>); | **deletePipeline** (<*tubería_ID*>);

Operadores Asociativos y Precedentes

En la Taba 4.1, se listan los operadores asociativos utilizados para la generación de secuencias de código y sus relaciones.

4.5.1.4. Reglas de escritura

A continuación se muestran las reglas de escritura para la generación de secuencias de código.

Declaraciones

1. Un identificador solo puede ser declarado una única vez en cualquier función, ya sea particular o no.

2. Las funciones de creación pueden ser declaradas máximo una vez, es decir, estas funciones pueden definirse como máximo una sola vez.

3. Si una función se encuentra declarada, entonces los parámetros en su definición deben coincidir (es decir, ser los mismos), en número y orden, a los parámetros del argumento en su definición.

4. La declaración de una función, si está presente, debe preceder a la definición de la función.

5. Un identificador puede aparecer como máximo una vez en la lista de parámetros formales en una declaración de función.

6. Los parámetros formales de una función tienen un alcance local para esa función.

Requisitos de consistencia de tipo

Todos los identificadores deben declararse antes de ser utilizados. Si se declara un identificador, cualquier uso de ese identificador dentro de esa función se refiere a la entidad con alcance global.

Las siguientes reglas guían la comprobación de la coherencia de tipos. La noción de que dos tipos son compatibles se define de la siguiente manera:

- Un extractor solo es compatible con la interfaz de entrada de un transformador;
- Un cargador solo es compatible con la interfaz de salida de un transformador;
- Un bloque puede ser concatenado con otros bloques para crear una tubería.

Expresiones

El tipo de una expresión d se define de la siguiente manera:

1. Si *d* es un identificador, entonces el tipo de *d* es el tipo de ese identificador.
2. Si *d* es una expresión del tipo extractor $->$ transformador $->$ cargador entonces *d* es un bloque.
3. Si *d* es una expresión del tipo bloque $->$ bloque entonces d es una tubería.

Ejemplos de secuencias de código generadas a partir de las reglas definidas con anterioridad

La Figura 4.11 muestra un ejemplo de la secuencia de código propuesta, en la cual se definen los elementos que conformaran la $Pipeline$. Como se puede observar, es necesario declarar todos los elementos que conforman un BB, es decir, los $transformadores$, $extractores$, $cargadores$. Posteriormente, los BBs creados podrán ser concatenados para generar diferentes $Pipelines$.

```
2  IOLibrary.Kulla
3  IOLibrary.Geb
4  Transformers.Geb
5
6  CreateScheme (MyScheme) {
7      CreateTransformers() {
8          Lz4 = proccessData(lz4:image, "COMAND",  TYPE.processBB, In.Fs, Out.memory);
9          IDA = proccessData(IDA:image, "COMAND", TYPE.processBB,  In.memory, Out.Fs);
10     }
11
12     CreateExtractors() {
13         Init= getSourceCluster(TYPE.dataBB,"path","user", "token","password");
14         secondS = putInMemory(TYPE.dataBB, Through.getMemoryAddress());
15
16     }
17
18     CreateLoader() {
19         Through = putInMemory(TYPE.dataBB);
20         Ending = putInCluster(TYPE.dataBB, "path","user", "token","password");
21     }
22
23     CreateBlocks() {
24         Block1 = Init-> Lz4 -> Through;
25         Block2 = secondS -> IDA -> Ending;
26     }
27
28     CreatePipeline () {
29         Pipeline1 = Block1->Block2;
30     }
31 }
```

Figura 4.11: Ejemplo de una secuencia de código.

Después de declarar los componentes de la $Pipeline$, en necesario verificar la existencia de variables dependientes, y de los parámetros que componen a cada una de ellas.

Por otro lado, la Figura 4.12 muestra un ejemplo de una secuencia de código en donde se declara un patrón. En ella se declararon tres elementos de tipo *transformador*, en los cuales, uno de ellos es un patrón. El patrón se encuentra conformado por una fuente (*extractor*) y un destino (*cargador*) de datos, el número de trabajadores por los que estará conformado el patrón, el tipo de patrón el cual es un *Manejador/Trabajador* y por lo tanto un algoritmo de balanceo de carga (en este caso twochoices), y se ha indicado que la entidad consolidadora no será utilizada en este caso, por lo que se le ha dado un valor nulo (NULL).

```
2  IOLibrary.Kulla
3  IOLibrary.Geb
4  Transformers.Geb
5
6  CreateScheme (MyScheme2) {
7
8      CreateTransformers() {
9          AES = proccessData(AES:image, "COMAND", TYPE.processBB, In.Fs, Out.memory);
10         IDA = proccessData(IDA:image, "COMAND", TYPE.processBB, In.memory, Out.Fs);
11         Patt = createPatt ( source2, twoChoices, IDA , 3, managerWorker, NULL , sink2);
12     }
13
14     CreateExtractors() {
15         source1 = getSourceCluster(TYPE.dataBB,"path1","user1", "token","password_ejemplo");
16         source2 = putInMemory(TYPE.dataBB, sink.getMemoryAddress());
17     }
18
19     CreateLoader() {
20         sink1 = putInMemory(TYPE.dataBB);
21         sink2 = putInCluster(TYPE.dataBB, "path","user", "token","password");
22     }
23
24     CreateBlocks() {
25         securityBlock = source1 -> AES -> sink1;
26         reliabilityBlock = source2 -> Patt -> sink2;
27     }
28
29     CreatePipeline () {
30         myPipeline = securityBlock -> reliabilityBlock;
31     }
32 }
33
```

Figura 4.12: Ejemplo de una secuencia de código con patrones.

4.5.2 Interprete de secuencias de código

Se desarrolló un intérprete que permite leer la secuencia de código e identificar cada una de las abstracciones y funciones dentro de él, para la generación de los esquemas de almacenamiento de datos. Dicho interpreté funciona en tres etapas: *i)* verificación de la estructura; *ii)* resolución de la dependencia entre variables; y *iii)* construcción y despliegue de los BBs.

En la primera etapa, se realiza la verificación de las estructuras de código generadas por el usuario. Para ello, se comprueba que la estructura de cada una de las funciones se encuentre de forma ordenada, y que cumplan con todos los parámetros designados para ellas. De la misma forma, es necesario que cumplir con una secuencia de palabras reservadas en el orden correcto para su correcta compilación. En esta etapa es necesario cumplir con las siguientes reglas de codificación:

- Debe existir al menos un $Extractor$ de iniciación para la $Pipeline$.

- Debe existir al menos un $Cargador$ de finalización para la $Pipeline$.
- Las invocaciones de los $Extractores$, $Transformadores$ y $Cargadores$ son recursivas.
- Cada $Transformador$ debe ser elegible para poder ser puesto en un contenedor.
- Los $Transformadores$ pueden ser de dos tipos: Bloques únicos o Patrones.

La segunda etapa, permite la verificación de la dependencia de las variables utilizadas en cada una de las secuencias de código introducidas por el usuario. Esto permite determinar el orden en el que serán ejecutadas dichas secuencias, y al mismo tiempo, permite comprobar que todas las dependencias se encuentren correctamente referenciadas en la secuencia de código.

El Algoritmo 1 (ver Anexo A.1) muestra el proceso de identificación de los componentes necesarios para la construcción de un esquema de almacenamiento a partir de una secuencia de código. En este algoritmo, dichos componentes son identificados mediante un proceso iterativo, lo cual permite la creación de estos, así como su manejo. Durante este proceso, se verifican las dependencias de los componentes y el orden de la estructura de codificación que se ejecutara. Así mismo, se verifica que se cumpla con todos los elementos necesarios para construir un $Pipeline$.

Finalmente, en la tercera etapa se realiza la construcción y el despliegue de BBs, así como de los esquemas construidos a partir de la secuencia de código generada por el usuario.

4.5.3 Manejo de la ejecución de esquemas de almacenamiento

La Figura 4.13 muestra la pila arquitectural del servicio para el manejo de los esquemas de almacenamiento. En la parte superior, se encuentra una capa que permite mantener el control de acceso de los usuarios, posteriormente se tiene un manejador, el cual se encarga de la gestión de mapas y requerimientos no funcionales de los esquemas. En la tercera capa, se tiene un distribuidor que se encarga de la implementación de los balanceadores de carga. Dichos balanceadores de carga permiten la distribución cuasi uniforme de los datos entre cada etapa del esquema, las cuales se encuentran

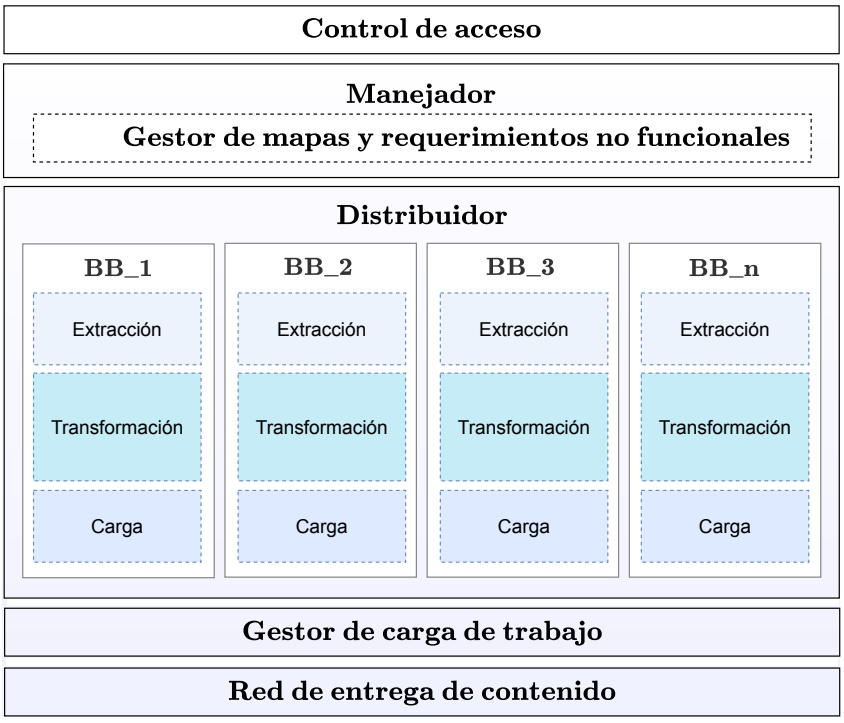


Figura 4.13: Arquitectura de pila del servicio para el manejo de los esquemas de almacenamiento definidos por código.

desplegadas siguiendo el modelo ETL. Finalmente, se encuentra el gestor de carga de trabajo que se encarga de administrar lógicamente los metadatos de los datos de entrada de cada bloque de construcción. Dicho gestor se encuentra comunicado con una red de distribución de contenido que se encarga de cargar y descargar datos de la nube.

El Algoritmo 2 (ver Anexo A.2) muestra el proceso de extracción, transformación y carga que los datos sufren al pasar por un BB. En él se realiza la extracción de los datos desde una fuente de datos, la cual puede ser de dos tipos: *i)* un almacén de datos (por ejemplo, Amazon, computadoras personales o clústeres), o *ii)* otro BB, es decir, otra etapa de la $Pipeline$.

Una vez que se han verificado las claves de acceso (que se obtienen mediante archivos de configuración) para acceder a la fuente de datos y que estos se han extraído, se realiza el proceso de transformación de los datos. Es decir, los datos son procesados para brindarles distintas propiedades (seguridad, confiabilidad, eficiencia, disponibilidad) dependiendo de la aplicación, algoritmo, o función

que se encuentre encapsulada en el BB.

Finalmente, los datos transformados son almacenados en un destino de datos, el cual puede ser de dos tipos: *i)* almacén de datos (por ejemplo, servidores o computadoras personales), o *ii)* otra etapa en la $Pipeline$ (otro BB).

4.6 Patrón recursivo de procesamiento

La Figura 4.14 muestra los patrones creados al usar los BBs solo para la confiabilidad $BBox$ (IDA). Este $BBox$ incluye una combinación de patrones. El primero considera un patrón simple de tubería y filtro construido por tres $BBoxes$ como el manejador, $ShFS$ (cualquiera de sistema de archivos, partición o ubicación en la nube) y entrega. Este patrón representa la adquisición de datos entrantes (O_1) y la entrega de los datos resultantes (O_2). El segundo es un $Manejador/Trabajador$ tradicional que incluye al manejador y un conjunto de trabajadores ($\{w_1, \ldots, w_n\}$), que reciben tareas del manejador para procesar los datos obtenidos de $ShFS$ invocando el tercer patrón: *Divide y vencerás*. Dicho patrón se clona tantas veces como los trabajadores creados por el Manejador. Estos trabajadores están a cargo de invocar un BB (Div), que divide el contenido enviado por los trabajadores en s segmentos ($\{s_1, \ldots, s_n\}$), que se entregan a los trabajadores de *Divide y vencerás* ($\{dw_1, \ldots, dw_n\}$). Los segmentos procesados por cada dw se envían a una etapa de conquista administrada por un patrón de recursos compartidos (ShM_1) que los esquemas crean en la memoria. El último es un patrón de recursos compartidos creado por $ShFS$, que consolida los resultados de cada patrón de **Divide y vencerás**. La $entrega$ puede recuperar los datos resultantes (O_2) de este patrón compartido y luego entregarlo al siguiente $BBox$.

En este contexto, los BBS recursivos permiten al desarrollador crear múltiples combinaciones de patrones dependiendo de sus necesidades (por ejemplo, el número de propiedades en la continuidad próxima, el rendimiento a entregar, el grado de sensibilidad, etc.). Esta característica de flexibilidad de los esquemas propuestos en este documento es bastante útil en escenarios reales.

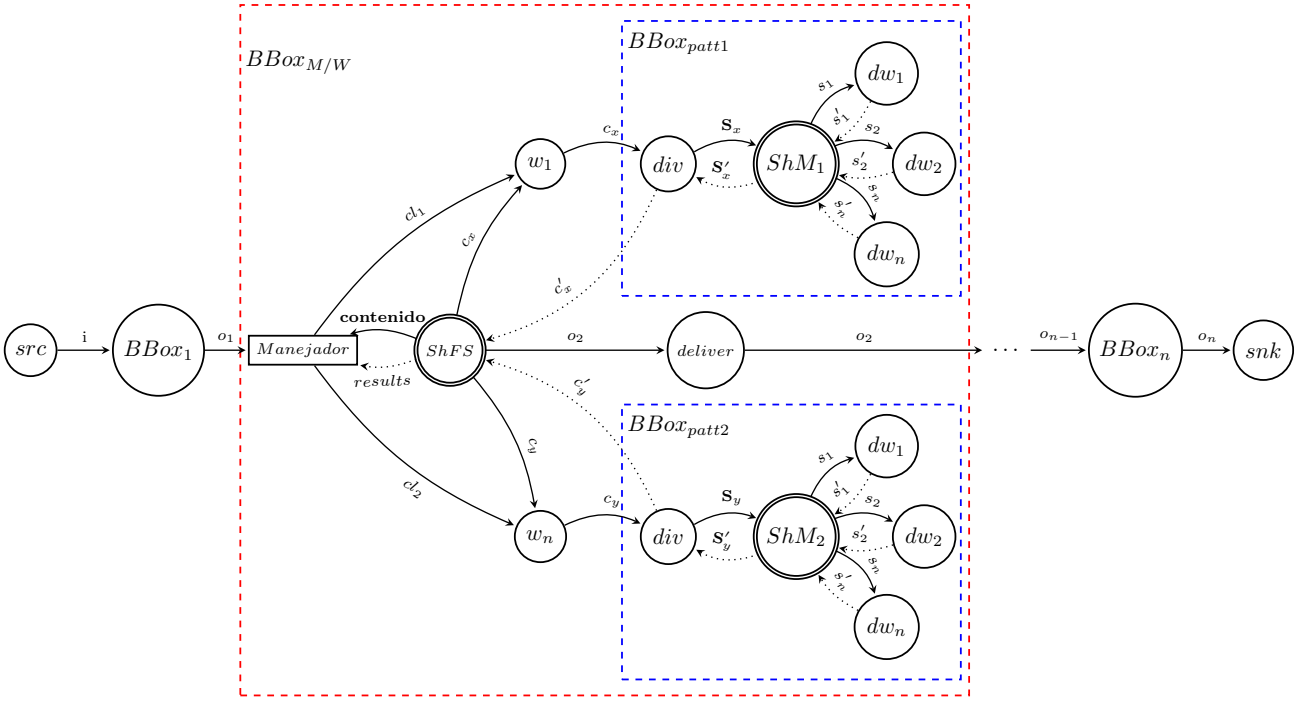


Figura 4.14: Representación de patrones recursivos para la confiabilidad como un grafo acíclico dirigido (DAG).

4.7 Repositorio de esquemas de preparación y recuperación de datos

Como se estableció en el estudio del trabajo relacionado, los requisitos no funcionales son cruciales para los escenarios de procesamiento de datos. La eficiencia, proporcionada en el modo de patrones de paralelismo, resulta crucial para los procedimientos críticos de toma de decisiones. En este sentido, los esquemas de preparación son implementados en el plano de preservación, para que los datos puedan ser preparados en un $BBox$ antes de enviarlos a los siguientes $BBoxes$ en una solución de preparación/recuperación de datos, o a los tomadores de decisiones/usuarios finales.

El objetivo es que las soluciones de preparación/recuperación de datos cumplan con los requisitos de seguridad y confiabilidad para administrar cualquier dato sensible, y al mismo tiempo resistir la falta de disponibilidad de datos, evitar violaciones de confidencialidad, detectar alteraciones en los

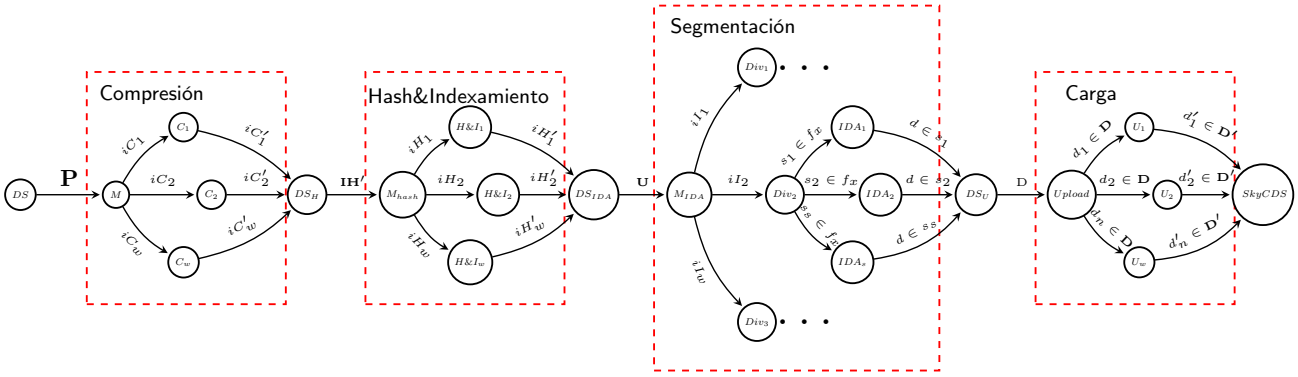
resultados y los datos durante su ciclo de vida, o presentar soluciones para no repudio.

La Figura 4.15 muestra el DAG de los esquemas de preparación/recuperación que incluyen cuatro BBs. El primer BB mejora la eficiencia en el transporte y almacenamiento de datos, reduciendo los costos de almacenamiento tanto en tamaño de datos como económicamente. Se encapsuló una aplicación de compresión en este bloque, que implementa un patrón *Manejador/Trabajador*, donde se recupera un conjunto de rutas P de la fuente de datos DS. P se divide en w sub-listas, que se distribuyen por el patrón, de forma balanceada mediante un algoritmo de balanceo de carga [88], a w número de BBs que ejecutan el algoritmo de compresión LZ4 (sin pérdidas de datos [10]), que se desarrolló en lenguaje C. Los w clones C_x de estos bloques de creación se crean para comprimir datos de manera concurrente para reducir la cantidad de datos que se enviarán a las siguientes etapas y se transportarán o compartirán.

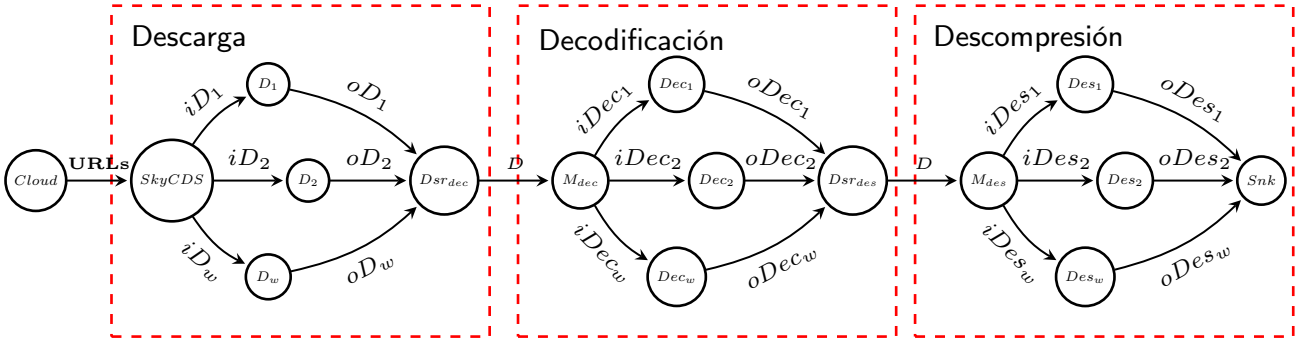
El segundo es la verificación de integridad y deduplicación (hash e indexación) para detectar alteraciones y datos replicados. El sistema de deduplicación para almacenamiento en la nube fue desarrollado con base en un algoritmo de hash h (SHA-3-256 [37]), el cual es utilizado para generar la huella criptográfica $h(C)$ que es utilizada para detectar archivos duplicados: sí $C_1 = C_2$ entonces $h(C_1) = h(C_2)$. Por lo tanto, h permite identificar contenidos replicados antes de enviarlos los esquemas de preparación. En este sistema, el hash es calculado y comparado con los hashes previamente descubiertos por el sistema (ver sistema de deduplicación en la Figura 5.2). El cálculo de hash de los archivos tiene dos objetivos: i) el primero es crear un índice de archivos, identificando lo previamente indexado y preparado, y ii) el segundo es verificar la integridad de los datos, identificando alteraciones durante el transporte de estos.

Los contenidos asociados a un hash previamente cargado en la nube son rechazados. Mientras, que los hashes no encontrados por el sistema de deduplicación (datos únicos) son indexados en una base de datos de metadatos (implementada en MongoDB) y estos archivos son enviados al sistema de preparación de datos (ver indexamiento en la Figura 5.2).

En el esquema de preparación de datos (ver Figura 5.2), el sistema de deduplicación es ejecutado



(a) Esquema de preparación.



(b) Esquema de recuperación.

Figura 4.15: Esquemas de preparación (a) y recuperación (b) basados en grafos acíclicos dirigidos (DAGs).

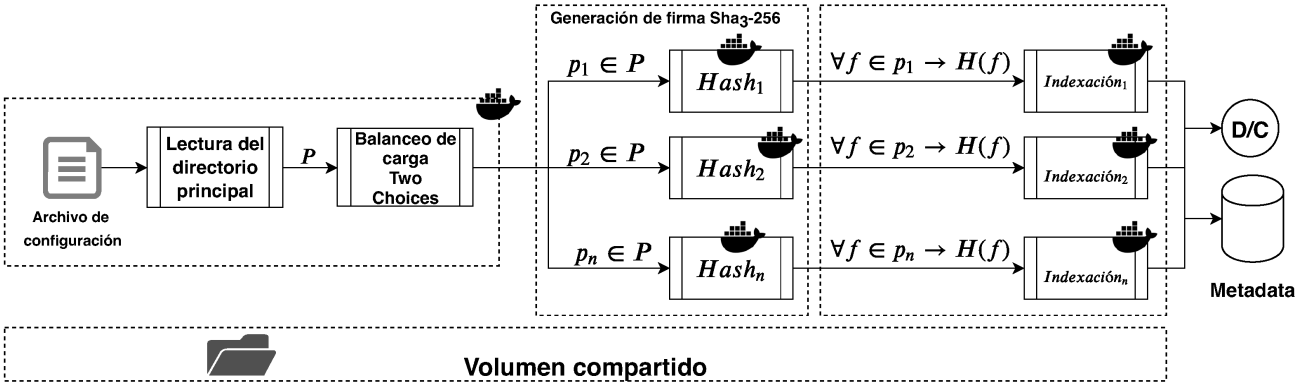


Figura 4.16: Patrón de deduplicación basado en contenedores virtuales (VCs).

cuando los contenidos son cargados a la nube, mientras el método de integridad es aplicado cuando los contenidos son descargados en operaciones de compartición de información, para que los usuarios finales puedan detectar y registrar alteraciones en los contenidos descargados.

La Figura 4.16 muestra la representación conceptual del servicio de deduplicación como un patrón de paralelismo desplegado sobre VC's. Cada trabajador incluye una tubería de procesamiento que realiza el cálculo de la firma hash y el indexamiento sobre un subconjunto de datos $p_i \in P$ de las rutas contenidas en cada subconjunto enviado al trabajador por el *Manejador*.

Los $Trabajadores$ realizan el cálculo de la firma hash $h(\cdot)$ del contenido en p_i ($\forall f \in p_i \rightarrow h(f)$) utilizando la función hash SHA3-256. La firma hash es calculada y enviada al contenedor de indexamiento, el cual es llamado $metadata$. El servicio de indexamiento obtiene el contenido con firmas de hash únicas, y los almacena en una tabla ($\mathbf{T}[f_u]$) y cada fila de dicha tabla tiene la forma $< h(f_u), [f_1, f_2, f_3, \ldots, f_n] >$, donde $H(f_u)$ representa la firma de hash de un archivo único, y $[f_1, f_2, f_3, \ldots, f_n]$ es una lista de rutas de los archivos con la misma firma hash ($h(f_1) == h(f_2) == h(f_3) == h(f_n)$). Para mantener la consistencia entre los archivos replicados a través del $cliente$, la tabla de firmas hash es enviada a la base de datos de $metadatos$, donde cada firma hash es registrada.

El tercer BB permite la codificación de los datos utilizando el algoritmo de dispersión de información (IDA) [101], que agrega redundancia a los datos. Este algoritmo segmenta los archivos en n segmentos llamados dispersos. En el esquema de preparación, la idea básica de usar IDA es agregar redundancia a cada disperso y colocarlos en n locaciones de almacenamiento diferentes. En el esquema de recuperación, la idea es recuperar el archivo original descargando m segmentos de los n creados en el esquema de preparación, en donde $m < n$. Como resultado, si alguno de los dispersos generados para un archivo no es enviado a la nube, es posible recuperarlo siempre y cuando se tengan disponibles $n - m$ dispersos del archivo.

En el esquema de preparación de datos, IDA fue implementado utilizando el patrón $D\&C$. La Figura 4.17 muestra un ejemplo de dicho patrón. Como es posible observar, la entidad $Divide$ segmenta cada dato (d_j) en c segmentos, los cuales son enviados a c $Trabajadores$. Cada $Trabajador$ ($W \in \{IDA_1, \cdots, IDA_n\} \in D\&C$) ejecuta el algoritmo IDA, procesando un segmento y produciendo 5 dispersos (en donde $n = 5$); como resultado, el patrón produce ($c * n$) porciones

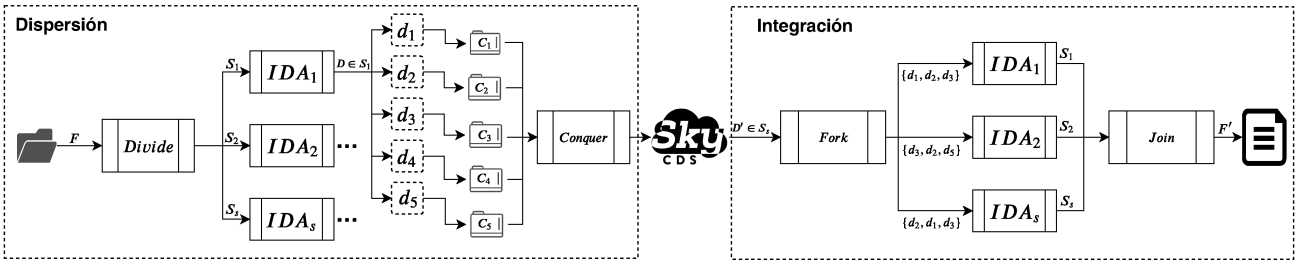


Figura 4.17: Procesos de confiabilidad y recuperación utilizando el patrón divide y vencerás ($D\&C$).

de redundancia, los cuales son colocados en memoria compartida. Dichas porciones son integradas por el $Consolidador$ desde la memoria compartida, el cual localiza los cinco dispersos en las cinco locaciones de almacenamiento. En esta configuración de IDA, el sistema puede soportar la falla en dos de las locaciones de almacenamiento ($n - m$). Esto se debe a que, en esta configuración, solo tres de los cinco dispersos son requeridos para recuperar el archivo original (d_j). La redundancia producida por este algoritmo es calculada como $R = (L/M) * (n - M)$.

Este patrón produce un paralelismo de datos, el cual es efectivo cuando se procesan grandes volúmenes de archivos (por ejemplo, imágenes satelitales y médicas, así como archivos multimedia y respaldos).

La etapa de integración del esquema de recuperación de datos fue implementada como un patrón *División/Unión*. En él un $Manejador$ recupera ($m = 3$) tres dispersos para cada uno de los archivos (d_j), y envía dichos archivos a los $Trabajadores$. Los $Trabajadores$ se encargan de reconstruir el archivo original (d_j). Posteriormente, el patrón envía d_j a la siguiente etapa en el esquema de recuperación (por ejemplo, la colocación) para que el usuario final pueda almacenarlos en un almacenamiento local o en algún otro sistema de almacenamiento disponible.

En este punto, se espera que los datos entregados al siguiente BB (cifrar y cargar) sean en el peor de los casos, 6 % más que el tamaño de los datos originales d y, en el mejor de los casos, incluso un 40 % menos que el tamaño original de d (dependiendo del ahorro logrado al comprimir el bloque de construcción). Además, se espera que se reduzca la cantidad de archivos a procesar codificando bloques de construcción dependiendo de la eficacia de la deduplicación. Los datos que llegan a la

cuarta etapa del esquema ($Cifrado\&Carga$), se envían a un demonio de cifrado y reenvío de la red de entrega de contenido llamada SkyCDS [46], la cual se encuentra implementada en Java. Este demonio ejecuta técnicas de cifrado basadas en AES [87] y CP-ABE [93] para cifrar los datos entrantes y crear un objeto cifrado que incluya el control de acceso. Esto se establece sobre los datos antes de enviarlos dependiendo de la dirección de la siguiente caja negra mediante una operación de carga.

Como es posible observar, cada bloque de construcción implementa patrones de paralelismo para reducir el tiempo de servicio de cada BB. El objetivo, por lo tanto, no es solo reducir los datos transportados a otra caja negra (a través de diferentes entornos) sino también reducir o incluso eliminar la sobrecarga al aplicar integridad, seguridad y confiabilidad a los datos en estos esquemas a través de diferentes entornos.

5

Evaluación experimental y resultados

En este capítulo se presenta el diseño experimental conducido para evaluar el modelo desarrollado, así como el diseño del prototipo, la infraestructura de experimentación, y repositorios de datos utilizados.

El objetivo de la experimentación fue evaluar la factibilidad del modelo para desarrollar diferentes esquemas de preparación y recuperación de datos que se adapten a los requerimientos de las organizaciones. Además, con la conducción de la presente experimentación se evaluó el rendimiento de los bloques de construcción utilizados para construir dichos esquemas.

Del mismo modo, el objeto de estudio es el modelo desarrollado y su capacidad de generar esquemas de preparación y recuperación de datos que se adapten a los requerimientos de las organizaciones que los utilicen.

Figura 5.1: Metodología de evaluación experimental.

5.1 Metodología de experimentación

En esta sección se describe la metodología con los pasos seguidos para conducir la experimentación descrita en el presente capítulo. En total, se definió una metodología de tres pasos, los cuales son descritos a continuación (ver Figura 5.1):

1. **Prototipo de experimentación:** Diseño de un prototipo basado en el modelo propuesto, el cual permite la construcción mediante código de esquemas de preparación y recuperación de datos que se adecúen a los requerimientos no funcionales de las organizaciones.

2. **Evaluación controlada**: En la segunda etapa, se realizó una evaluación controlada de los esquemas construidos con el prototipo, los cuales consideran las siguientes etapas de preparación: compresión de datos, indexamiento, codificación de datos para proveer confiabilidad, cifrado de archivos y carga de archivos a la nube. En este sentido, el esquema de recuperación realizó las tareas inversas, las cuales son: descarga de datos, decodificación de

archivos, verificación de integridad, y descompresión de datos.

Para evaluar la eficiencia de los esquemas construidos, se varió el número de trabajadores, así como el volumen de datos a procesar. Para ello, se generaron conjuntos de datos sintéticos de prueba variando tanto el volumen de los datos como la densidad de los mismos (número de archivos).

3. **Estudios de caso**: En esta etapa de la metodología, se condujeron tres estudios de caso que permitieron evaluar la factibilidad del modelo para desarrollar soluciones que se adapten a diferentes entornos organizacionales y los datos manejados en estos. Los estudios de caso son listados a continuación:

   a) **Estudio de caso para el manejo de datos organizacionales**.

   b) **Estudio de caso para el manejo de datos médicos en una federación de distribución de contenido**.

   c) **Estudio de caso para el manejo de datos meteorológicos obtenidos en ambientes de IoT**.

5.2 Perspectiva de la experimentación y cualidad evaluada

El experimento fue evaluado desde el punto de vista de investigadores, organizaciones y desarrolladores que requieren construir soluciones de preparación de datos antes de almacenarlos en la nube o en cualquier otra infraestructura de almacenamiento. Por ejemplo, dentro de las soluciones a evaluar se encuentran esquemas construidos para el manejo de requerimientos, tales como seguridad, integridad de datos, confiabilidad y eficiencia.

El principal efecto estudiado en este experimento fue el desempeño de los esquemas de preparación

Tabla 5.1: Infraestructura utilizada para la evaluación experimental.

Ubicación	Etiqueta	RAM (GB)	CPUs	Almacenamiento
us-east-1 (North Virginia)	vmec2	32	16	600 GB
Google Cloud	vmgc	16	8	800 GB
Cluster Cinvestav	Compute7	24	6	465.8GB, 931.5GB, 931.5GB
	Compute10	64	12	930.4 GB, 27.3 T
	Compute11	64	12	2.7 T, 2.7 T
	Compute12	64	12	2.7 T, 2.7 T
	Disys4	12	6	256 GB
Cluster UC3M	Compute 3-1	148	64	256 GB
	Compute 3-2	148	64	256 GB
	Compute 11-2	126	12	256 GB, 750 GB

y recuperación sobre diferentes conjuntos de datos. En este sentido, se evaluó el desempeño de los bloques de construcción utilizados para proveer diferentes requerimientos no funcionales variando el volumen de datos a procesar y la configuración (número de trabajadores) de los patrones de paralelismo desplegados en los bloques de construcción.

El experimento fue conducido utilizando diferentes infraestructuras de procesamiento y almacenamiento de datos. Para ello se desarrolló un prototipo de experimentación el cual permite el despliegue de esquemas de preparación y recuperación de datos, los cuales fueron desarrollados utilizando lenguaje C y la plataforma de contenedores virtuales de Docker para desplegar los bloques de construcción.

5.3 Materiales de experimentación

En la presente sección, se describen los conjuntos de datos e infraestructura utilizados para conducir la evaluación experimental.

Tabla 5.2: Conjuntos de datos utilizados en la evaluación experimental.

Etiqueta	Tipo de datos	Archivos		
		Total	Tamaño promedio	Tamaño total
CD1	Imágenes satelitales	219	277 MB	58.29 GB
CD2	Archivos personales	317465		302.16 GB
	Mamografías	102451	14.5 MB	11 GB
CD3	Resonancias magnéticas	2070	14.5 MB	30 GB
	Tomografías	1865	25.5 MB	47 GB
CD4	Registros meteorológicos de México	44	51.77 MB	2.2 GB
	Registros meteorológicos de España	4386	0.45 MB	2.1 GB

5.3.1 Infraestructura utilizada

En la Tabla 5.1 se muestra la infraestructura que se utilizó para evaluar la factibilidad del despliegue de los esquemas de procesamiento en diferentes organizaciones, así como la portabilidad de los bloques de construcción. Dicha infraestructura está compuesta por una máquina virtual en Amazon EC2, una máquina virtual en Google Cloud, un clúster de computadoras ubicado en el Cinvestav Tamaulipas y otro clúster ubicado en la Universidad Carlos III de Madrid.

5.3.2 Conjuntos de datos utilizados

Para conducir la evaluación experimental se utilizaron cuatro conjuntos de datos diferentes. En la Tabla 5.2 se muestran las características de los conjuntos de datos utilizados, mostrando su tamaño total y el tamaño promedio por archivo.

El primer conjunto de datos está compuesto por 219 archivos, de los cuales el $95.45\,\%$ son imágenes satelitales de la misión LandSat5 de 277 MB de tamaño promedio. Dichas imágenes fueron descargadas por la antena Eris de la Agencia Espacial Mexicana (AEM) [38], y se encuentra ubicada en el municipio de Chetumal, Quintana Roo. Las imágenes se encuentran bajo respaldo de almacenamiento en el Cinvestav Tamaulipas, y fueron utilizadas anteriormente para probar el funcionamiento de la herramienta FedIDS [48].

El segundo conjunto de datos corresponde a archivos obtenidos de un disco duro personal. Este

conjunto de datos está compuesto por $317,465$ archivos (302.16 GB), en donde $99.96\,\%$ de los archivos en este repositorio tiene un tamaño menor a los 250 MB.

El tercer conjunto de datos está compuesto por imágenes médicas que se encuentran públicas para su uso en el sitio *Cancer Imaging Archive*[1]. De este sitio se descargaron dos colecciones de imágenes, donde el primero corresponde a mamografías [66], y el segundo a resonancias magnéticas del páncreas [81]. La tercera colección de datos correspondiente a tomografías, se obtuvieron de *"The Digital database for ScreeningMammography"* [55].

El cuarto conjunto de datos corresponde a datos meteorológicos capturados por antenas desplegadas en los territorios de España y México. El conjunto de datos españoles contiene 44 archivos de registros correspondientes a datos capturados en el año 2011 con un tiempo de censado de 10 minutos entre ellos. Estos datos son de carácter privado, y fueron proporcionados con fines de investigación. El conjunto de datos mexicano corresponde a registros capturados por las estaciones meteorológicas automáticas (EMAS) de la CONAGUA [27], y contiene registros de datos diarios desde 1985 hasta el año 2018.

5.4 Prototipo de experimentación

Para evaluar los esquemas de preparación y recuperación se desarrolló un enfoque de preparación y recuperación de datos utilizando contenedores virtuales y patrones de paralelismo. Dicho enfoque se encuentra compuesto por dos componentes principales: un sistema de deduplicación de datos y un motor para la construcción de los esquemas de preparación y recuperación.

La Figura 5.2 muestra la representación conceptual de una aplicación *Cliente* que carga los contenidos en la nube utilizando el esquema de preparación (lado izquierdo de la figura). Además, en la Figura 5.2 se muestra como este cliente descarga los contenidos utilizando los esquemas de recuperación (lado derecho de la figura). El esquema de preparación incluye etapas como

[1]Disponible en: https://www.cancerimagingarchive.net/collections/

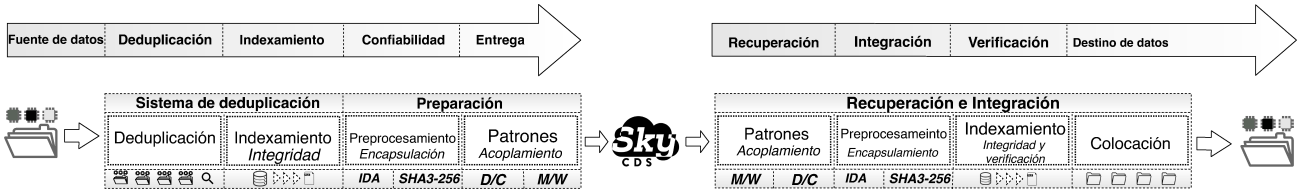


Figura 5.2: Representación conceptual de los esquemas de preparación y entrega de datos a la nube, así como de la recuperación y decodificación de datos desde de la nube.

deduplicación e indexamiento, así como de confiabilidad y entrega. La distribución de los archivos a la nube y a los usuarios finales es manejada mediante una red de distribución de contenidos llamada SkyCDS[46], la cual comprime y cifra los datos antes de ser enviados a ambientes multi-nube. Los esquemas de recuperación incluyen etapas tales como la recuperación/descarga de los datos utilizando SkyCDS, la decodificación de los datos codificados en el esquema de preparación, la verificación de la integridad, y la colocación de los datos en los sistemas de almacenamiento indicados por el usuario.

5.5 Evaluación controlada

La primera etapa de la evaluación experimental es una evaluación controlada de los esquemas de preparación y recuperación. En esta evaluación se varió el número de trabajadores en los patrones de paralelismo con el objetivo de observar el comportamiento de las soluciones al variar dicho valor.

5.5.1 Descripción

Se definió una evaluación experimental controlada utilizando cuatro conjuntos de datos sintéticos generados para mantener un control sobre el volumen y la densidad de los datos a procesar en los esquemas de preparación y recuperación de datos.

El objetivo de la evaluación controlada fue conocer el comportamiento de los esquemas en escenarios de alta densidad de datos y de gran volumen de datos.

5.5.2 Soluciones estudiadas

En este estudio de caso, el procesamiento de datos se implementó en las siguientes soluciones:

1. *Dagon**: es una plataforma que permite la creación de flujos de trabajo comúnmente utilizados para procesar datos climáticos y de internet de las cosas en la nube. Esta solución incluye un esquema de paralelismo para eficientizar el rendimiento de los flujos de trabajo.

2. *Solución propuesta*: Es la solución desarrollada a partir del modelo propuesto en la presente tesis.

5.5.3 Variación experimental

Para llevar a cabo la evaluación controlada, se generaron dos conjuntos de datos para evaluar el comportamiento de los esquemas al realizar el procesamiento de diferentes volúmenes y densidades de datos.

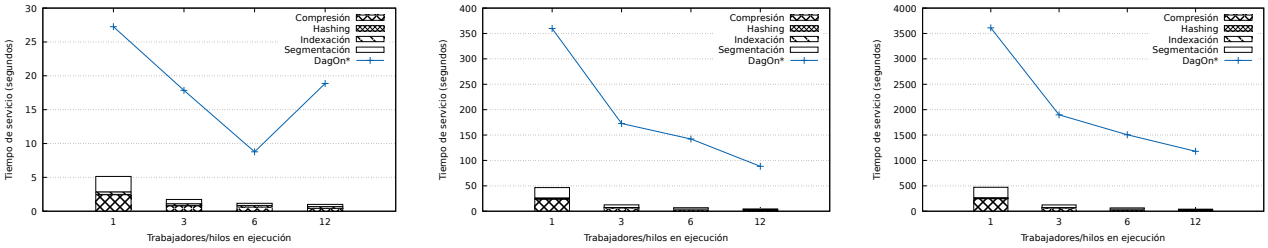
Para el escenario de densidad de datos, se generaron tres conjuntos de datos sintéticos homogéneos de 1 MB: *i)* conjunto de 10 archivos, *ii)* conjunto de 100 archivos, y *iii)* conjunto de 1000 archivos.

Mientras que, para el escenario de volumen de datos, se procesaron cuatro archivos únicos de forma individual. Los archivos tienen los siguientes tamaños: 1 MB, 10 MB, 100 MB y 1024 MB.

5.5.4 Resultados

A continuación, se describen los resultados obtenidos al realizar la evaluación experimental controlada.

La Figura 5.3(a) presenta los resultados obtenidos del procesamiento de 10 archivos de 1 MB. En el eje vertical se muestra el tiempo de respuesta observado utilizando 1,3,6, y 12 trabajadores en los patrones de paralelismo. Además, se puede observar en la línea azul el procesamiento de los mismos



(a) 10 archivos. (b) 100 archivos. (c) 1000 archivos.

Figura 5.3: Comparación del tiempo de servicio al procesar 10, 100, y 1000 archivos homogéneos en la solución propuesta y Dagon*.

archivos realizado por DagOn* (el cual es un motor de manejo de flujos de trabajo de procesamiento de datos), en el cual se consideraron las mismas etapas de procesamiento.

Como se puede observar en la Figura 5.3(a), a medida que se aumenta el número de trabajadores el tiempo de respuesta disminuye hasta llegar a 12 trabajadores que es el punto en donde aumenta el tiempo de respuesta, lo anterior debido a que el número de contenidos no es significante por lo que la sobrecarga generada por los patrones es mayor que el tiempo de procesamiento.

Además, se muestra la comparación del tiempo de respuesta de los esquemas propuestos utilizando 6 trabajadores con la versión de DagOn* de 6 hilos. En él se puede observar una ganancia en el tiempo de respuesta del 86.68 %.

Por otro lado, cuando se aumenta el número de archivos de 10 a 100, es posible observar que el tiempo de respuesta sigue disminuyendo aún con 12 trabajadores, observando una aceleración de 20x con una ganancia del 95.18 % (ver Figura 5.3(b)).

Finalmente, con 1000 archivos se observa un comportamiento similar, en donde la aceleración aumenta a 23x y la ganancia es de 95.66 % (ver Figura 5.3(c)).

Por otro lado, en el escenario de volumen, la Figura 5.4 presenta el tiempo de respuesta obtenido al procesar cuatro archivos de diferentes tamaños (1MB, 10MB, 100MB, 1024 MB). Como se puede observar el tiempo de respuesta obtenido con los esquemas es menor al tiempo obtenido con DagOn*. Lo anterior, representa una ganancia del 38 % para el procesamiento de un archivo de un gigabyte.

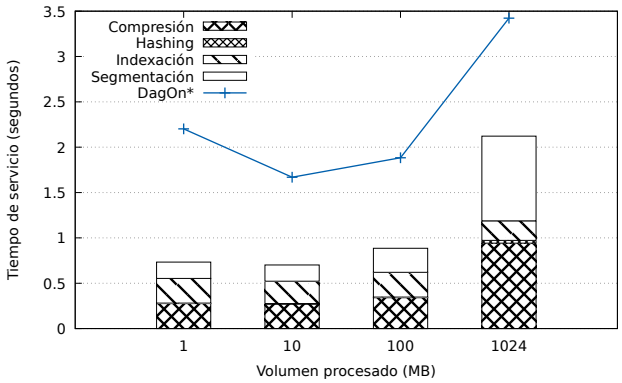


Figura 5.4: Tiempo de respuesta al procesar 4 archivos únicos de diferentes tamaños.

Mientras que, para un archivo de un mega, la ganancia es del 66

5.5.5 Conclusión

De la presente experimentación, se pudo observar el impacto que tienen dos variables al procesar datos: volumen y densidad. De esta forma es posible afirmar que, procesar un archivo único de un gigabyte producirá un menor tiempo de respuesta que procesar 1000 archivos de 1 MB.

5.6 Estudio de caso 1: manejo de datos personales y organizacionales

En esta sección se describe el primer estudio de caso, el cual se encuentra enfocado al manejo de datos organizacionales.

5.6.1 Descripción

Este estudio de caso está basado en repositorios reales de imágenes satelitales y archivos organizacionales, los cuales fueron preparados para ser migrados a la nube realizando procesos tales como compresión, cifrado, codificación para tolerancia a fallos y control de acceso.

Tabla 5.3: Resultados de la experimentación.

	Archivos			Tamaño (GB)			Porcentaje de duplicados (%)	
	Total	Únicos	Duplicados	Total	Únicos	Duplicados	Archivos	Capacidad
Compute7	219	130	89	58.29	34.18	24.10	40.63	41.35
Compute8	317465	148147	169318	302.16	146.20	155.96	53.33	51.61
Compute9	126115	43360	82755	300.15	82.54	217.61	65.61	72.49
Compute8	13537	6350	7187	11.63	9.32	2.31	53.09	11.90

La Tabla 5.3 muestra los resultados obtenidos de los cuatro servidores que almacenan el contenido de cuatro repositorios de datos diferentes. Los esquemas de preparación/recuperación fueron instalados en estos cuatro servidores para procesar los contenidos de un directorio dado. SkyCDS fue configurado para cifrar y comprimir los datos preprocesados y posteriormente cargarlos en la nube.

El primer repositorio de datos (almacenado en *Compute 7*) utilizado contenía 219 archivos (58.31 GBs). El $95.45\,\%$ de los archivos correspondía a imágenes satelitales de *LanSat5*, de 277 MB de tamaño medio (ver distribución de los datos en este repositorio en la Figura 5.5(a)). El segundo repositorio (almacenado en *Compute 8* incluía $317,465$ archivos personales (302.16 GB). En la Figura 5.6(a) muestra la distribución de los tamaños de los archivos en este repositorio. Como se puede observar, el $99.96\,\%$ de los archivos en este repositorio tiene un tamaño menor a los 250 MB. El tercer repositorio (almacenado en *Compute 9*) tenía $126,115$ archivos (300.15 GB). En este repositorio, el $99.85\,\%$ de los archivos corresponden a archivos de menos de 250 MB de tamaño. Este repositorio fue construido solo para propósitos de experimentación, recolectando y duplicando archivos de diferentes fuentes de datos.

5.6.2 Soluciones estudiadas

Los esquemas de preparación y recuperación fueron evaluados en dos etapas.

1. Fase 1: Análisis de los costos de deduplicación y verificación de la integridad de los datos.

2. Fase 2: Análisis del impacto de la deduplicación y la verificación de integridad en las etapas de

confiabilidad y recuperación de datos.

Los resultados de estos experimentos fueron comparados con un sistema incluyendo solo los esquemas de confiabilidad y recuperación utilizando los datos del repositorio en *Compute 7* (ver Tabla 5.1).

5.6.3 Variación experimental

1. En la primera fase, fueron evaluados los costos de deduplicación y verificación de integridad de datos. Los experimentos de esta fase fueron realizados variando el número de trabajadores desplegados en los patrones.

2. En la segunda fase, los esquemas de confiabilidad y recuperación fueron evaluados, incluyendo el sistema de deduplicación y verificación de integridad, así como con patrones de paralelismo. En esta fase fueron estudiadas las siguientes configuraciones:

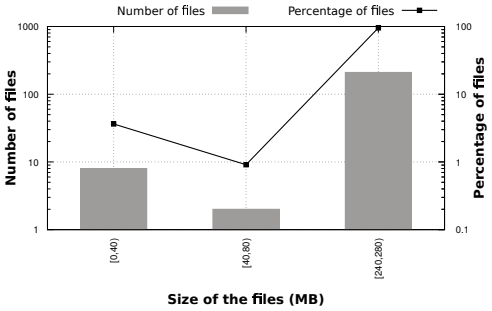
   1.- RegularIDA: esta configuración representa la implementación secuencial del algoritmo IDA.
   2.- Dedup: esta configuración representa la implementación de IDA incluyendo el sistema de deduplicación.
   3.- Dedup-IDA-Patterns: esta configuración representa un esquema de preparación de datos (incluyendo deduplicación y IDA utilizando el patrón D&C).

5.6.4 Resultados

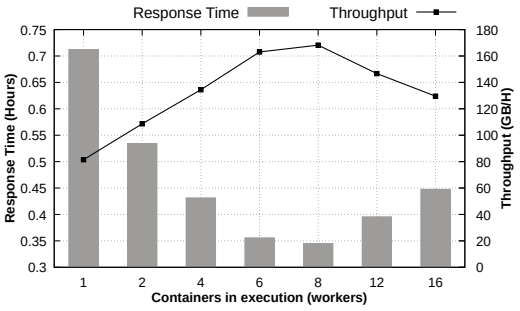
En la presenta sección, los resultados de los experimentos realizados en las dos fases de experimentación previamente descritas son presentados y analizados.

5.6.4.1. Fase 1

En esta sección, se muestran los resultados obtenidos en la primera fase de evaluación en donde se analizan los costos de deduplicación y verificación de la integridad de los datos. Las Figuras 5.5(b)



(a) Distribución del conjunto de datos en *Compute 7*.

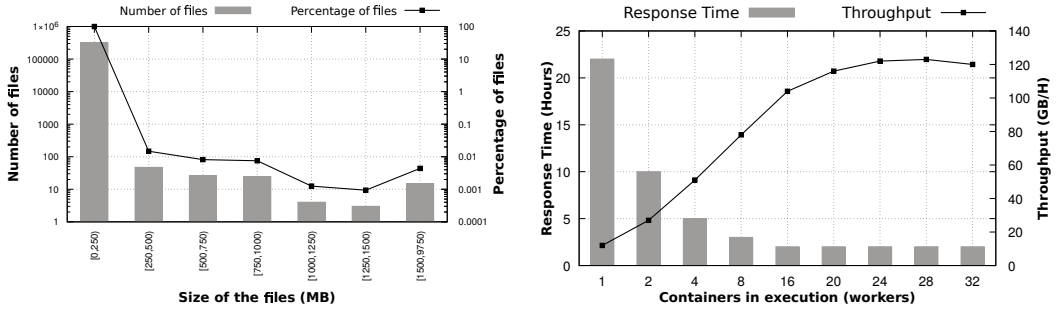


(b) Tiempo de respuesta en *Compute 7* utilizando diferente número de trabajadores.

Figura 5.5: Distribución del conjunto de datos en *Compute 7*, y el tiempo de respuesta para procesarlos.

y 5.6(b) muestran, en el eje vertical izquierdo, el tiempo de respuesta en horas para la etapa de deduplicación, incluyendo el análisis y generación del hash de los archivos en los conjuntos de datos *compute 8* y *compute 9* respectivamente. Los experimentos se ejecutaron utilizando patrones de paralelismo con diferentes números de trabajadores (eje horizontal). El eje vertical derecho, en ambas figuras, muestra el *throughput* medido en gigabytes por hora (GB/H).

Como se puede observar, los patrones de paralelismo de tareas basados en VCs, reducen significativamente el tiempo de respuesta de las etapas de deduplicación e indexamiento de datos, lo cual además reduce los tiempos de respuesta de las operaciones de carga y descarga realizadas en el cliente de almacenamiento en la nube. Esto aumenta el número de archivos procesados por hora. Por ejemplo, el esquema de preparación ejecutado en *Compute 7* proceso 58.31 GB en 0.71 horas (42.72 minutos) con solo un trabajador. Mientras que con seis trabajadores completó la misma tarea en 0.35 horas (21.34 minutos), lo cual significa que el patrón de paralelismo con seis trabajadores tiene una mejora del $50.05\,\%$ con una aceleración de $2x$ en comparación de la tarea ejecutada con un solo trabajador (ver Figura 5.5(b)). Además, se puede observar que la mejora en el *throughput* se detiene cuando el número de contenedores virtuales es mayor al número real de procesadores en una computadora. Por ejemplo, en la Figura 5.5(b), el tiempo de respuesta aumenta cuando el patrón lanzado con 12 trabajadores, mejorando el desempeño de la solución hasta un $44.46\,\%$ en



(a) Distribución del conjunto de datos en *Compute 8*.

(b) Tiempo de respuesta en *Compute 8* utilizando diferente número de trabajadores.

Figura 5.6: Distribución del conjunto de datos en *Compute 8*, y el tiempo de respuesta para procesarlos.

comparación de ejecutarlo con un solo trabajador.

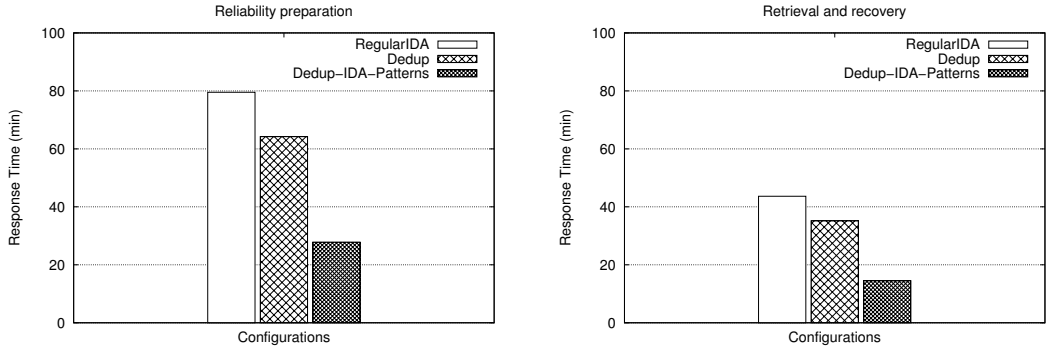
En *Compute 8*, los esquemas para el procesamiento de 285 GB tardaron 22.2 horas utilizando solamente un trabajador. Esto produce un *throughput* de 12.76 GB/H, mientras que el patrón *Manejador/Trabajador* con 28 trabajadores completan la misma tarea en 2.2 horas. Esto significa una mejora de $89.80\,\%$ con una aceleración de $9x$ utilizando 28 trabajadores en comparación con la versión de un solo trabajador (ver Figura 5.6(b)).

Como era de esperarse, el patrón de paralelismo de tareas basado en VCs reduce el tiempo de respuesta, incrementando el *throughput* de los procesos de preparación y recuperación. Se observo que esta mejora depende del número de núcleos físicos utilizados en el patrón.

5.6.4.2. Fase 2

En esta sección se muestran los resultados de la segunda etapa de evaluación en donde se analiza el impacto de la deduplicación y la verificación de integridad en las etapas de confiabilidad y recuperación de datos.

La Figura 5.7(a) muestra en el eje vertical, el tiempo de respuesta de la etapa de confiabilidad en la preparación de datos, y en el eje horizontal muestra las configuraciones evaluadas. El proceso de recuperación de datos es mostrado en la Figura 5.7(b).



(a) Dispersión. (b) Recuperación.

Figura 5.7: Tiempo de respuesta del esquema de confiabilidad para las etapas de dispersión (a) y recuperación (b) de datos.

Como se esperaba, la configuración *Dedup-IDA-Patterns* produce la mejor configuración en los esquemas de preparación y recuperación. El tiempo de respuesta producido por *RegularIDA* para la codificación de todos los archivos (incluyendo duplicados) fue de 79.51 minutos, el tiempo de respuesta producido por las configuraciones *Dedup* fue de 64.23 minutos, mientras que la configuración *Dedup-IDA-Patterns* (utilizando cuatro trabajadores) fue de 27.82 minutos. Esto significa una mejora en el tiempo de respuesta de $65\,\%$ para una aceleración de $2.8x$.

5.7 Estudio de caso 2: manejo de datos médicos en una federación de distribución de contenido

En esta sección se describe el segundo estudio de caso, el cual se enfocó en el manejo de datos médicos en una federación de distribución de contenido.

5.7.1 Descripción

En este estudio de caso se utilizó el conjunto de datos $CD2$ (ver Tabla 5.2). Mientras que el entorno experimental se construyó utilizando tres centros remotos conectados a través de Internet,

los cuales se representaron como tres máquinas diferentes (ver Tabla 5.1) y una red de entrega de contenido (SkyCDS) implementada en una nube, que se encargó de transferir los contenidos a través de las diferentes máquinas.

El experimento para validar la solución se llevó a cabo en tres fases:

- Fase1: los repositorios se procesaron utilizando los esquemas de preparación en donde se varió el número de trabajadores en los patrones de paralelismo.
- Fase 2: se evaluó la descarga de datos de la CDN y su recuperación utilizando el esquema de recuperación.
- Fase 3: el rendimiento de los esquemas se compara con soluciones del estado del arte: Rsync, una solución tradicional para enviar datos a la nube, e incluye la preparación de datos mediante compresión y cifrado de datos; y SCP, una solución tradicional que permite enviar datos a la nube de manera segura.

5.7.2 Soluciones estudiadas

El procesamiento de datos en este estudio de caso se implementó en las siguientes soluciones:

- *Rsync*: Esta solución incluye la preparación de datos al comprimir y cifrar datos durante el transporte de datos. No considera el preprocesamiento/procesamiento de datos, los controles de acceso basados en atributos de establecimiento, la verificación de integridad, ni el establecimiento de tolerancia a fallas.
- *SCP*: es una solución tradicional utilizada para enviar datos a la nube de manera segura ya que esta herramienta se basa en un modelo de clave pública / privada. Esta solución presenta la restricción de rsync en términos de requisitos no funcionales.
- *Solución propuesta*: esta solución representa la solución de preparación/recuperación de datos del modelo propuesto en este documento de tesis.

5.7.3 Variación experimental

La evaluación de cada fase se realizó de la siguiente manera:

- Fase1: En esta fase, se evaluó el rendimiento de los esquemas de preparación utilizando los conjuntos de datos asignados en diferentes computadoras y enviando los datos preparados a la CDN, haciendo que los datos estén disponibles para otros miembros de la federación.
- Fase 2: En esta fase, se evaluó el tiempo que le tomo al Hospital 1 (Compute 1) realizar la descarga del conjunto de datos DDSM, cargado por el Hospital 2 (Compute2).
- Fase 3: Se realizó la comparación de la solución propuesta con técnicas tradicionales de transferencia de datos encontradas en la literatura (SCP y rsync).

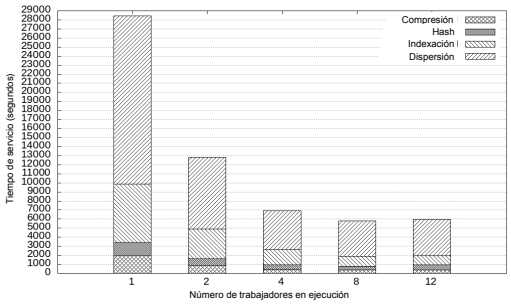
5.7.4 Resultados

En esta sección se dan a conocer los resultados obtenidos para las dos fases de experimentación realizadas en este estudio de caso.

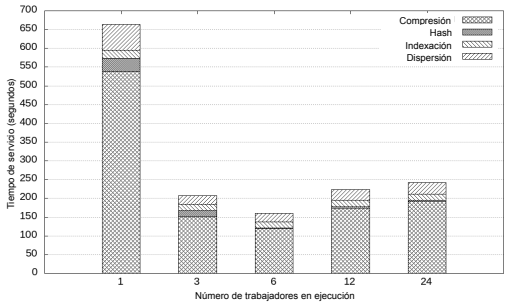
5.7.4.1. Fase 1

En esta fase, se procesaron los diferentes conjuntos de datos en diferentes centros para ver la influencia del conjunto de datos y las computadoras asociadas. La métrica principal medida fue el tiempo de servicio empleado para procesar el conjunto de datos variando el número de trabajadores en los patrones.

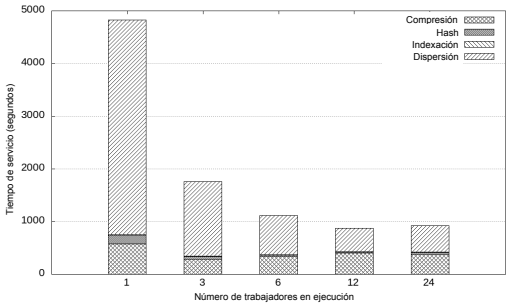
La Figura 5.8(a) muestra el tiempo de servicio al procesar el conjunto de datos QIN-Breast almacenado en Compute3. Como se muestra en la Figura 5, se logra una aceleración de 4.7x con 8 trabajadores para preprocesar los 11 GB del conjunto de datos. El tiempo de ejecución es mayor que para los otros conjuntos de datos, debido a la gran cantidad de archivos pequeños.



(a) Tiempo de servicio empleado por el esquema de preparación para procesar el conjunto de datos QIN-Breast.



(b) Tiempo de servicio empleado por el esquema de preparación para procesar el conjunto de datos CPTAC-PDA.



(c) Tiempo de servicio empleado por el esquema de preparación para procesar el conjunto de datos DDMS.

Figura 5.8: Tiempo de servicio empleado por el esquema de preparación para procesar los conjuntos de datos.

La Figura 5.8(b) corresponde al tiempo de servicio empleado por el esquema de datos de preparación para procesar el conjunto de datos CPTAC-PDA almacenado en Compute1. En Compute1, el esquema gastó 664.24 segundos para preprocesar 30 GB al lanzar solo un trabajador. Esto produjo un rendimiento de 45.16 MB/s, mientras que el M / W con 6 trabajadores completa la misma tarea en 165.88 segundos. Esto significa una mejora del 75 % con una aceleración de 4x.

De manera similar, la Figura 5.8(c) muestra el tiempo de servicio para procesar el conjunto de datos DDMS mediante los esquemas de preparación almacenados en Compute2. En este caso, el esquema preprocesó 47 GB con una mejora del 82 % con una aceleración de 5x con 12 trabajadores en comparación con 1 trabajador.

Como se esperaba, el paralelismo implementado en los esquemas redujo el tiempo de servicio de

cada tarea de preprocesamiento. Además, de los resultados se pudo observar que el tiempo de servicio comienza a aumentar cuando la cantidad de trabajadores es mayor que la cantidad de núcleos físicos.

El tiempo empleado para cargar los conjuntos de datos en la nube fue de 72 minutos para el conjunto de datos CPTAC-PDA en Compute1, y 160 minutos para los datos DDSM en Compute2. Como puede verse, el factor crítico para el costo de esas soluciones es, no solo el tamaño del conjunto de datos (volumen), sino principalmente el número de archivos del conjunto de datos (densidad).

5.7.4.2. Fase 2

En este estudio de caso, el proceso de recuperación de datos se activa automáticamente cuando se completa la carga de los datos, y la CDN se encarga de entregar los datos a las instituciones médicas suscritas para descargar los contenidos cargados. En este experimento se evaluó el tiempo que el Hospital 1 (Compute1) toma para descargar el conjunto de datos DDSM, cargado por el Hospital 2 usando Compute2.

El tiempo empleado por el cliente CDN para descargar los datos del almacenamiento en la nube fue de 64 minutos. Después de que este cliente descarga los datos, comienza el proceso de recuperación de los datos originales, esto es para decodificar los archivos de 4 de los 5 segmentos y descomprimir los archivos recuperados. La Figura 5.9 muestra los resultados obtenidos de este proceso de recuperación.

Como se puede observar en la Figura 5.9, el tiempo de servicio se reduce al aumentar el número de trabajadores en el patrón. Por ejemplo, el patrón con 12 trabajadores tardo 11.04 minutos para recuperar los datos, mientras que con solo 1 trabajador le tomo 26.25 minutos, lo que significa una mejora en el tiempo de servicio del 57.85 %.

5.7.4.3. Fase 3

En esta etapa se realizó una comparación directa con dos aplicaciones de transferencia de datos del estado de la técnica: SCP y rsync. Para este experimento, se utilizó el conjunto de datos DDSM,

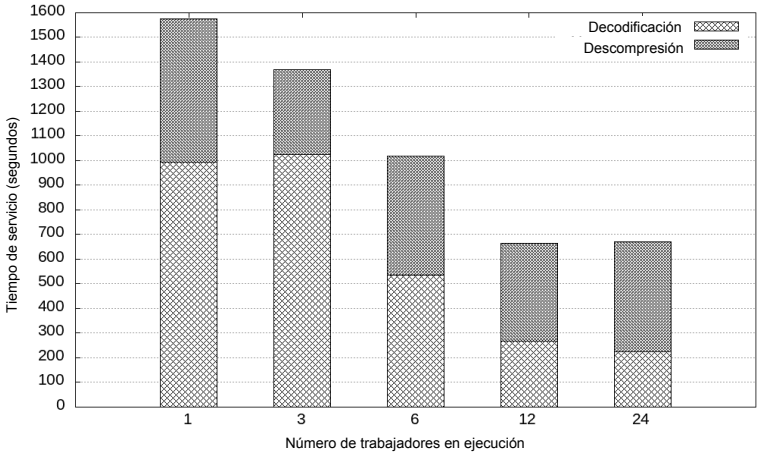


Figura 5.9: Tiempo de servicio empleado por el esquema de recuperación para procesar los datos descargados.

Tabla 5.4: Tiempo de servicio para la carga del conjunto de datos DDSM mediante el uso de diferentes soluciones disponibles en el estado del arte.

Solución	rsync	SCP	TAR+SCP	FCDS
Tiempo de servicio (minutos)	83.86	78.74	120.62	116.10
Rendimiento (MB/m)	568.40	605.36	395.18	684.28

transfiriendo los datos de Compute2 a Amazon EC-2 (ver Tabla 5.1).

La Tabla 5.4 muestra el tiempo de servicio observado para cada solución evaluada. En el caso de SCP, se evaluó la transferencia enviando todos los archivos sin comprimir el conjunto de datos y, en una segunda instancia, comprimiendo el conjunto de datos en un solo archivo.

Como se puede ver en la Tabla 5.4, la solución propuesta en la presente tesis produce el mayor rendimiento en comparación con las otras tres soluciones, a pesar de que esta solución produce el mayor tiempo de servicio. El tiempo de servicio de la solución propuesta es de 116.10 minutos, mientras que rsync tardó 83.86, SCP 78.74 y TAR + SCP 120.62 minutos. Lo anterior se debe a que la solución propuesta carga una mayor cantidad de datos en comparación con las otras soluciones debido a que agrega 1.7x de redundancia, lo que permite agregar a los datos la característica de confiabilidad y recuperación tolerante a fallos.

Los resultados del experimento realizado mostraron una buena escalabilidad y un buen rendimiento

en comparación con las soluciones tradicionales utilizadas. Los experimentos también muestran que la métrica crítica que aumenta el tiempo de distribución es el número de archivos en un conjunto de datos (densidad), más que el tamaño de los archivos en sí (volumen).

5.8 Estudio de caso 3: manejo de datos meteorológicos obtenidos en ambientes de IoT

En esta sección se describe el tercer estudio de caso, el cual se encuentra enfocado al manejo de datos meteorológicos obtenidos en ambientes de IoT.

Este estudio de caso está basado en datos capturados diariamente durante 33 años por sensores de estaciones terrestres ubicadas en los 32 estados de México, y registros capturados cada 10 minutos durante el año 2011 en 56 estaciones ubicadas en España.

El objetivo de realizar este estudio fue mostrar la viabilidad y flexibilidad del modelo propuesto en la presente tesis, así como realizar un estudio de comparación de rendimiento directo.

5.8.1 Descripción

Se definió un estudio de caso que consta de una metodología de dos fases para llevar a cabo una evaluación experimental basada en el procesamiento de datos climáticos producidos por sensores IoT de diferentes dimensiones.

Este estudio de caso se basó en la implementación de una solución de procesamiento de IoT para realizar un procedimiento analítico de clasificación de dos fuentes de datos ambientales producidos por sensores de estaciones terrestres.

Los conjuntos de datos contienen registros capturados por sensores de estaciones terrestres distribuidas a través del territorio de dos países: México y España (ver Tabla 5.2). El conjunto de datos mexicano incluye registros capturados por la red de estaciones meteorológicas mexicanas

(EMAS)[2], que cubre los 32 estados integrados en el territorio mexicano. Cada estación del sistema EMAS tiene métricas meteorológicas registradas, como temperaturas máximas y mínimas (medidas en grados Celsius), así como la precipitación (medida en milímetros), durante 33 años (de 1985 a 2018). El conjunto de datos en español incluye registros de 56 estaciones que cubren todo el territorio español. Cada estación terrestre tiene métricas meteorológicas registradas, como una marca de tiempo del registro, precipitaciones, velocidad del viento, temperatura y humedad relativa. Este conjunto de datos solo considera los registros del año 2011, que fueron capturados por sensores en un período constante de 10 minutos de manera 24/7.

5.8.2 Soluciones estudiadas

En esta sección se dan a conocer las soluciones estudiadas para las dos fases de experimentación realizadas en este estudio de caso.

- Fase 1: En la primera fase de este caso de estudio se realizaron los siguientes experimentos se realizaron con esta solución de preparación/recuperación de datos: se procesaron 10, 100 y 1000 trazas, cada traza contiene 1000 registros con un tamaño medio de paquetes de 20.9 KB, y se capturaron las métricas de cada experimento para calcular los efectos y los costos de los esquemas de preparación en el Rendimiento de la solución IoT. Estos experimentos se definieron para responder las siguientes preguntas: ¿Cuáles son los costos de cada BB del esquema de preparación al procesar distintos volúmenes de datos? ¿El rendimiento de las soluciones de preparación/recuperación de datos desarrollada en este estudio es competitivo contra las soluciones encontradas en el estado del arte?

- Fase 2: En la segunda fase del estudio de caso se llevó a cabo una comparación de la solución propuesta en la presente tesis con soluciones utilizadas tradicionalmente para el transporte de datos en la nube (SCP, y rsync), entrega continua de datos (Jenkins), y motores de flujo de

[2]https://smn.cna.gob.mx/es/observando-el-tiempo/estaciones-meteorologicas-automaticas-ema-s

trabajo (Makeflow y DagOn*). Los resultados de la comparación de rendimiento se analizaron para determinar la sobrecarga de los esquemas de preparación en soluciones de procesamiento de datos de IoT en comparación con soluciones que no proporcionan requisitos no funcionales en el transporte de datos (SCP, y rsync), así como la eficiencia de los patrones de paralelismo utilizados en el modelo en comparación con soluciones disponibles en el estado del arte (Jenkins, DagOn* y Makeflow) que proveen paralelismo de tareas.

5.8.3 Variación experimental

En esta sección se dan a conocer las variaciones experimentales utilizadas para las dos fases de experimentación realizadas en este estudio de caso.

En la primera fase del estudio, se evaluaron dos configuraciones:

- *Solución propuesta*: esta configuración representa la solución de preparación de datos definida en este estudio, implementada en el prototipo.
- *DagOn** [85]: esta configuración representa un motor de flujo de trabajo centrado en entornos IoT.

Para cada una de las configuraciones estudiadas se varió el número de trabajadores de procesamiento en paralelo. El objetivo de esta variación es conocer como escalan ambas soluciones al incrementar el número de trabajadores y el efecto en el tiempo de servicio de este cambio.

En la segunda fase el procesamiento de datos se implementó en las siguientes soluciones:

- *Rsync*: es una solución tradicional para enviar datos a la nube al sincronizar ubicaciones locales y en la nube. Esta solución también incluye la preparación de datos al comprimir y cifrar datos durante el transporte de datos. Esta solución no considera el preprocesamiento / procesamiento de datos, los controles de acceso basados en atributos de establecimiento, la verificación de integridad y el establecimiento de tolerancia a fallas.

- *SCP*: es una solución tradicional utilizada para enviar datos a la nube de manera segura ya que esta herramienta se basa en un modelo de clave pública / privada. Por lo tanto, incluye la preparación de datos no solo durante el transporte, sino también de extremo a extremo y produce un control de acceso reducido basado en un concepto clave primario. Esta solución presenta la restricción de rsync en términos de requisitos no funcionales.

- *DagOn** [85]: esta plataforma crea flujos de trabajo comúnmente utilizados para procesar datos climáticos e IoT en la nube e incluye un esquema de paralelismo para mejorar el rendimiento de los flujos de trabajo.

- *Jenkins* [6, 95]: es una plataforma que permite crear tuberías de software CI / CD que automatizan el proceso de construcción, prueba e implementación de soluciones de software. También incluye un esquema de paralelismo basado en hilos.

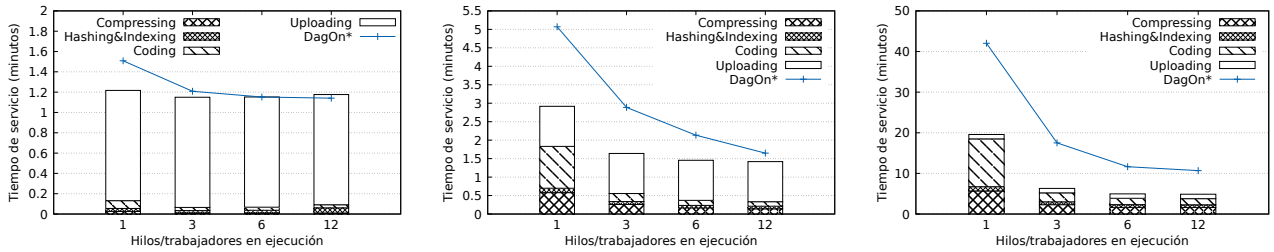
- *Makeflow* [2, 135]: es un motor de flujo de trabajo que permite la automatización en la ejecución de flujos de trabajo complejos en clústeres, nubes o entornos en contenedores, incluido también el esquema de paralelismo basado en subprocesos.

- *Solución propuesta*: esta solución representa la solución de preparación y recuperación de datos creada usando el modelo propuesto en este la presente tesis.

En esta fase, se analizó el rendimiento del flujo de trabajo de las cajas negras que implementa la solución propuesta y se compara con las soluciones encontradas en el estado del arte, implementadas para llevar a cabo este caso de estudio.

5.8.4 Resultados

En esta sección se dan a conocer los resultados obtenidos para las dos fases de experimentación realizadas en este estudio de caso.



(a) 10 trazas. (b) 100 trazas. (c) 1000 trazas.

Figura 5.10: Tiempo de servicio al procesar 10, 100, y 1000 trazas producidas por un simulador de sensores.

5.8.4.1. Fase 1

Los resultados de las configuraciones *solución propuesta* y *DagOn** se utilizarán para responder a las preguntas establecidas anteriormente.

La Figura 5.10 muestra los resultados obtenidos para la ejecución de la *solución propuesta* y *DagOn** al procesar los tres subconjuntos de 10 (Figura 5.10(a)), 100 (Figura 5.10(b)) y 1000 (Figura 5.10(c)) archivos producidos por un simulador de sensores. Como se puede ver, cuanta más carga de trabajo (trazas) y más trabajadores incluyeron los patrones, el tiempo de servicio disminuyó para las dos configuraciones estudiadas.

La respuesta a la primera pregunta establecida sobre los costos de los esquemas de preparación/recuperación (costo-eficiencia, integridad y tolerancia a fallas) es: cuanto más grande es la carga de trabajo, menos significativos son los costos del transporte entre los diferentes entornos (como la nube). En este contexto, cuando la cantidad de datos procesados es grande (por ejemplo, 1000 trazas de 1000 registros) los costos de procesamiento son más significativos que los costos de transporte, lo que hace que esta solución sea adecuada para grandes escenarios de datos de IoT.

Acerca de la segunda pregunta, sobre la comparación de rendimiento entre *DagOn** y la *solución propuesta*, la respuesta se puede obtener al observar la Figura 5.10(c), donde se puede observar que la *solución propuesta* procesó las 1000 trazas producidas por 1000 sensores simulados en 5.80 minutos utilizando 12 trabajadores en el patrón, mientras que *DagOn** usando 12 hilos procesó los mismos

datos en 10.68 minutos, lo cual significa que la *solución propuesta* tiene una mejora en el tiempo de servicio del $54.30\,\%$. Además, la *solución propuesta* permite agregar distintas propiedades a los datos. Esto significa que las organizaciones pueden soportar fallas en la ubicación del almacenamiento o la falta de disponibilidad de datos, así como también establecer la verificación de integridad y los controles de acceso no solo durante el transporte de datos, sino también al compartir datos en la nube y en la preservación en el almacenamiento en la nube, la cual no es ofrecido por *DagOn**. Además, se observó que cuanto mayor sea la gran carga de trabajo producida por los dispositivos IoT, mayor será la eficiencia de los patrones paralelos, lo que compensa el tiempo empleado por la *solución propuesta* para preparar los datos de IoT.

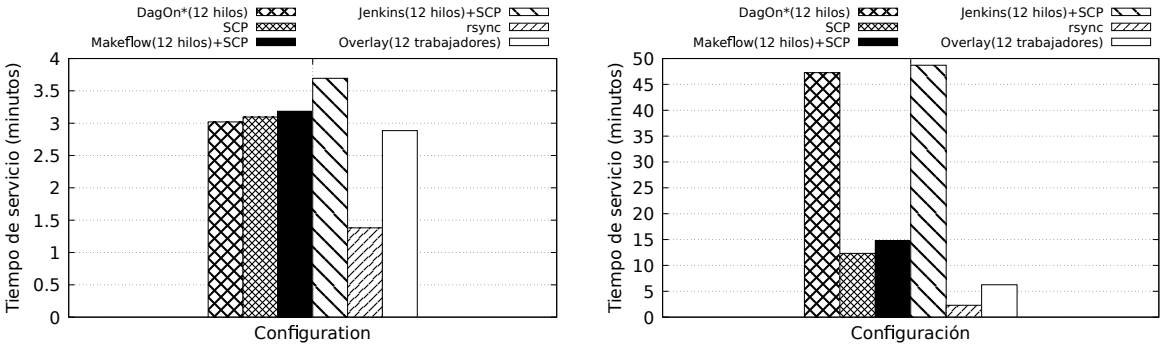
Hay que tener en cuenta que los patrones no solo procesan datos de forma concurrente, sino que también equilibran la carga por trabajador y reducen la cantidad de solicitudes de entrada/salida (sistemas de archivos y red) mediante el uso de esquemas en memoria, mientras que en el estado del arte solo produce una gestión implícita de hilos.

5.8.4.2. Fase 2

En esta fase, los experimentos se realizaron para responder a la pregunta sobre la eficiencia del modelo propuesto para crear un procesamiento continuo de forma segura y confiable en comparación con soluciones encontradas en el estado del arte. En este estudio se probaron dos tipos de soluciones: las primeras son herramientas tradicionales utilizadas para transportar datos a/en la nube, y las otras son herramientas disponibles para que los desarrolladores creen sus propias soluciones y flujos de trabajo de IoT basadas en un modelo de entrega continua de datos.

Además, se realizó una comparación directa de la configuración *solución propuesta* con $DagOn*$ [85], SCP, $rsync$, $Jenkins$ [6, 117] y $Makeflow$ [2]. En estos motores de procesamiento, la etapa de carga de datos se realizó utilizando SCP como se indica en la documentación de dichas soluciones.

Las Figuras 5.11(a) y 5.11(b) muestran, en el eje vertical, el tiempo de servicio empleado



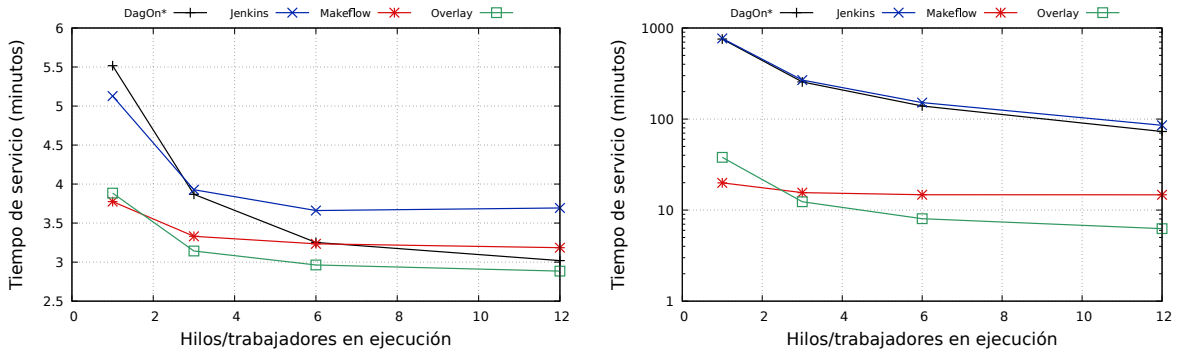
(a) Conjunto de datos españoles.

(b) Conjunto de datos mexicanos.

Figura 5.11: Comparación directa de la solución propuesta con soluciones encontradas en el estado del arte para procesar y transferir datos a la nube.

por las soluciones para procesar el conjunto de datos español y el conjunto de datos mexicano, respectivamente. La Figura 5.11 muestra que la configuración *solución propuesta* procesa y transfiere datos de IoT en menos tiempo que DagOn*, Jenkins, Makeflow e incluso SCP. Mientras que la Figura 5.11(a) muestra que el precesamiento de datos en la *solución propuesta* y la transferencia de datos se lleva a cabo en 2.88 minutos, mientras que DagOn*, Makeflow y Jenkins realizan la misma operación en 3.01 minutos, 3.18 minutos y 3.69 minutos respectivamente. Lo anterior significa un porcentaje de ganancia de $4.45\,\%$, $9.40\,\%$ y $21.89\,\%$ en comparación con DagOn* Makeflow y Jenkins respectivamente. Al comparar la *solución propuesta* con SCP, se puede observar una ganancia en el tiempo de servicio de $6.75\,\%$, dado que SCP transfiere los datos a la nube en 3.09 minutos. Rsync es la solución que brinda el mejor tiempo de servicio de las cuatro soluciones probadas, ya que solo carga los contenidos a la nube en 1.38 minutos. Sin embargo, esta solución solo considera el tiempo de entrega de los datos a la nube o usuarios finales, sin considerar el procesamiento de los datos.

La Figura 5.11(b) muestra el tiempo de servicio empleado por cada solución para procesar el conjunto de datos mexicano. La diferencia en el porcentaje de ganancia de rendimiento entre la *solución propuesta* y las otras soluciones que implementan la solución de IoT ($DagOn*$, $jenkins$ y $Makeflow$) aumenta en comparación con los resultados obtenidos al procesar el conjunto de datos en español. Este efecto es causado por el hecho de que el conjunto de datos mexicano incluye más



(a) Conjunto de datos españoles. (b) Conjunto de datos mexicanos.

Figura 5.12: Comparación directa con motores de flujos de trabajo mediante el uso de diferentes configuraciones de paralelismo en el número de tareas concurrentes.

registros que el español. La *solución propuesta* procesa este conjunto de datos en 6.25 minutos, DagOn * en 47.27 minutos, Makeflow en 14.71 minutos, Jenkins en 85.32 para un porcentaje de ganancia de rendimiento del $86.76\,\%$, $57.46\,\%$, $92.66\,\%$ respectivamente. Mientras que rsync y SCP solo cargan los datos procesados en 2.30 y 12.32 minutos respectivamente. Sin embargo, el tiempo producido por rsync solo representa el tiempo de transporte ya que esta solución solo envía datos a la nube mediante la sincronización de ubicaciones locales y en la nube. Estas soluciones solo se presentan para mostrar los gastos generales de las soluciones de procesamiento de IoT (sin transporte de datos).

Una vez mostrados los costos generales de las soluciones de IoT implementadas ($DagOn*$, $Jenkins$, $Makeflow$ y la *solución propuesta*), ahora se muestra una comparación directa de los mismos, como se puede observar en la Figura 5.12. La Figura 5.12 muestra, en el eje vertical, el tiempo de respuesta producido por las soluciones estudiadas al variar el número de trabajadores/hilos (eje horizontal) al procesar el conjunto de datos español (ver 5.12(a)) y el conjunto de datos mexicanos (ver Figura 5.12(b)). Como es posible observar, la configuración de la *solución propuesta* produjo un mejor tiempo de respuesta que Jenkins (12 hilos), Makeflow (12 hilos) y DagOn* (12 hilos) en un $4.31\,\%$, $21.95\,\%$, $9.43\,\%$ respectivamente al procesar el conjunto de datos en español. A su vez, las configuraciones de la *solución propuesta* produjeron un mejor tiempo de respuesta que Jenkins (12

hilos), Makeflow (12 hilos) y DagOn * (12 hilos) por un $92.67\,\%, 57.51\,\%$ y $91.43\,\%$ respectivamente al realizar el estudio de caso basado en el conjunto de datos mexicano.

Hay que tomar en cuenta que, en el modelo propuesto en la presente tesis, las operaciones de paralelismo se gestionan mediante una malla de servicio (malla de patrones) que incluye un algoritmo de balanceo de carga no determinista basado en la utilización de contenedores virtuales, así como una gestión de datos en memoria. A su vez, las soluciones tradicionales solo lanzan hilos por tarea en un contenedor virtual.

6

Conclusiones, limitaciones y trabajo futuro

En el presente capítulo se presentan las conclusiones del trabajo de tesis desarrollado, así como de la evaluación experimental. Además, se presentan las limitaciones de este trabajo, así como el trabajo futuro.

6.1 Conclusiones

En la presente tesis se presentó un modelo para la creación de esquemas de preparación/recuperación de datos definidos por código. Dicho modelo permite a las organizaciones construir soluciones que se adapten a sus requerimientos no funcionales (seguridad, confiabilidad, disponibilidad, eficiencia) para el procesamiento, transporte y compartición de datos entre las mismas y sus usuarios finales, así como para el almacenamiento de los datos.

Este modelo permite crear flujos de datos a través de flujos de trabajo de extremo a extremo, lo cual permite la ejecución ordenada de las etapas de un esquema de preparación de datos de forma implícita, transparente, y automática. Una etapa de compresión permite reducir el volumen de datos

enviados a la nube (o cualquier otro sistema de almacenamiento) para disminuir el costo de las tareas de administración y preservación de datos subcontratados, una etapa de cifrado mantiene la privacidad de los datos, una etapa de firma digital permite a los usuarios finales descubrir alteraciones de los archivos descargados, y una etapa de segmentación agrega redundancia a los datos , lo cual permite resistir la falta de disponibilidad de los mismos o ubicaciones de almacenamiento. Finalmente, una etapa de entrega y transporte de datos permite enviar los datos preparados a diferentes entornos.

Este modelo está basado en BBs, que incluyen conexiones de entrada/salida. Dichos BBs fueron creados mediante el uso de un modelo ETL, el cual ha demostrado ser flexible para la construcción de soluciones para el procesamiento de datos modelados como DAGs. Estas estructuras permiten a los desarrolladores encapsular distintos programas, aplicaciones o funciones en los BBs, que a su vez pueden ser acoplados a otros BBs utilizando el principio de entrega continua de datos a través de las interfaces de entrada/salida. Estas interfaces de entrada/salida de datos pueden ser configuradas para su uso, ya sea en memoria, por red o el sistema de archivos.

La evaluación se basó en tres estudios de caso que permitieron revelar la viabilidad de aplicar los esquemas de preparación/recuperación de datos a diferentes soluciones de procesamiento de datos mediante el uso de patrones de paralelismo definidos por código. A través de los estudios de caso se demostró que los esquemas de preparación/recuperación de datos son flexibles y eficientes, lo que permite establecer controles sobre la producción y el consumo de datos al agregar propiedades tales como seguridad y confiabilidad de forma rentable.

La evaluación experimental conducida en los tres casos de uso reveló la viabilidad de utilizar un enfoque de preparación/recuperación de datos para mitigar los riesgos que podrían surgir en la nube. La evaluación mostró que el uso de patrones de paralelismo y contenedores virtuales es beneficioso en todos los pasos de la preparación/recuperación de datos ya que cuanto mayor es la carga de trabajo, mayor será la eficiencia de los patrones de paralelismo utilizados, lo cual compensa el tiempo requerido para preparar/recuperar los datos.

6.2 Limitaciones

Las limitaciones del presente trabajo son las siguientes:

- En la versión actual de prototipo, se evaluó un esquema de preparación de datos que incluye el uso de LZ4 y una técnica de deduplicación para eficiencia, IDA para confiabilidad de datos, y SkyCDS para seguridad y carga de datos. En estos esquemas no se consideraron otras técnicas de confiabilidad (solo IDA) y disponibilidad de datos, lo cual es deseable para adaptarse a diferentes ambientes de almacenamiento, como lo son las multi-nubes.

- Actualmente, para construir los bloques recursivos de paralelismo se hace uso de una librería que implementa solamente los patrones *M/W* y *D&Q*. En este sentido, el usuario no puede construir sus propias secuencias de paralelismo utilizando el modelo de programación desarrollado, ya que éste solo puede llamar las funciones previamente implementadas.

6.3 Trabajo futuro

Como trabajo futuro se espera integrar las siguientes características al modelo desarrollado:

- Implementar mejoras funcionales de las operaciones de entrada y salida para extraer y cargar los datos en paralelo y así reducir la latencia de lectura de datos.

- Aplicar y evaluar las mejoras funcionales en ambientes de multi-nube (multicloud) y ambientes organizaciones.

A

Algoritmos para la interpretación y manejo de bloques de construcción.

A.1 Algoritmo de interpretación general de una secuencia de código.

Algorithm 1 Algoritmo de interpretación general de una secuencia de código.

Entrada: Secuencia de código (sc)
Salida: Esquema de almacenamiento ($pipe$)
{Se inicializan conjuntos vacíos para almacenar los elementos de los elementos que conformaran la pipeline.}
1: extractors = []
2: loaders = []
3: blocks = []
4: pipe = NULL
{Se realiza la validación de la secuencia de código.}
5: **si** validate(sc) **entonces**

6: sc_lines = split(sc) {Se hace una revisión linea por linea para identificar los elementos de un esquema en las secuencias de código.}
7: **para** $line$ in sc_lines **hacer**
8: **si** isExtractor($line$) **entonces**
9: $params$ = getParams($line$)
10: ext = createExtractor($params$)
11: extractors.push(ext)
12: **si no**, **si** isLoader($line$) **entonces**
13: $params$ = getParams($line$)
14: ld = createLoader($params$)
15: loaders.push(ld)
16: **si no**, **si** isBlock($line$) **entonces**
17: $params$ = getParams($line$)
18: bl = createBlock($params, extractors, loaders$)
19: loaders.push(bl)
20: **si no**, **si** isPipeline($line$) **entonces**
21: $pipe$ = createPipeline($blocks$)
22: **fin si**
23: **fin para**
{En caso de que no se encuentren errores, se genera la pipeline.}
24: **devolver** $pipe$
25: **si no**
26: **devolver** ERROR($Message$) {En caso de encontrar un error en la secuencia de código, se manda un mensaje.}
27: **fin si**

A.2 Algoritmo para el manejo de bloques de construcción en tiempo de ejecución

Algorithm 2 Algoritmo para el manejo de bloques de construcción en tiempo de ejecución

Entrada: Fuente de datos ($DSrc$), Credenciales de acceso ($tokens$), nombre del bloque ($Bname$), destino de datos ($DSnk$)

1: $verify =$ **falso**
2: $newData = []$
3: $data = []$
{Se verifica que se hayan introducido las credenciales de acceso a la fuente de datos.}
4: **si** $tokens! = NULL$ **entonces**
5: $verify =$ verify.accessTokens($DSrc$, $tokens$) {Se verifica que las credenciales sean validas.}
6: **si** $verify ==$**cierto entonces**
7: $data =$Extract.data($DSrc$, $tokens$) {Si son validas, se extraen los datos.} {Se verifica que existan datos a procesar.}
8: **si** $data! = NULL$ **entonces**
9: **para** d in $data$ **hacer**
10: $newData$.pull(transform.data(d)) {Si existen datos, se procesan.}
11: **fin para**
{Una vez procesados los datos, se cargan en el destino de datos especificado.}
12: loader.data($DSnk$,$newData$)
13: **devolver** $Message$("Task complete")
14: **si no**
15: **devolver** $Message$("Data list to process is empty") {En caso de no haber datos para procesar, se envía un mensaje al usuario.}
16: **fin si**
17: **si no**
18: **devolver** Message(.^ccess tokens are incorrects") {Si las credenciales de acceso no son correctas, se envía un mensaje al usuario.}
19: **fin si**
20: **si no**
21: **devolver** Message("Tokens is empty") {Si las credenciales de acceso no se ingresaron, se envía un mensaje al usuario.}
22: **fin si**

Bibliografía

[1] Aiken, P. (2002). *Microsoft computer dictionary*. Microsoft Press.

[2] Albrecht, M., Donnelly, P., Bui, P., and Thain, D. (2012). Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, pages 1–13.

[3] Almeida, F. and Calistru, C. (2013). The main challenges and issues of big data management. *International Journal of Research Studies in Computing*, 2(1):11–20.

[4] Amazon (2019a). Aws | almacenamiento de datos seguro en la nube (s3). https://aws.amazon.com/es/s3/. Accedido el 15 de julio, 2019.

[5] Amazon (2019b). Aws cloudformation – infraestructura como código y aprovisionamiento de recursos de aws. https://aws.amazon.com/es/cloudformation. Accedido el 15 de julio, 2019.

[6] Armenise, V. (2015). Continuous delivery with jenkins: Jenkins solutions to implement continuous delivery. In *Proceedings of the Third International Workshop on Release Engineering*, pages 24–27. IEEE Press.

[7] Arnold, J. (2014). *Openstack swift: Using, administering, and developing for swift object storage*. " O'Reilly Media, Inc.".

[8] Babuji, Y. N., Chard, K., Foster, I. T., Katz, D. S., Wilde, M., Woodard, A., and Wozniak, J. M. (2018). Parsl: Scalable parallel scripting in python. In *IWSG*, pages 1–6.

[9] Bala, A. and Chana, I. (2012). Fault tolerance-challenges, techniques and implementation in cloud computing. *International Journal of Computer Science Issues (IJCSI)*, 9(1):288.

[10] Bartík, M., Ubik, S., and Kubalik, P. (2015). Lz4 compression algorithm on fpga. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 179–182. IEEE.

[11] Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

[12] Bhushan, K. and Gupta, B. B. (2017). Security challenges in cloud computing: state-of-art. *International Journal of Big Data Intelligence*, 4(2):81–107.

[13] Böhm, M., Leimeister, S., Riedl, C., and Krcmar, H. (2011). Cloud computing–outsourcing 2.0 or a new business model for it provisioning? In *Application management*, pages 31–56. Springer.

[14] Bowers, K. D., Juels, A., and Oprea, A. (2009). Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198. ACM.

[15] Brikman, Y. (2017). *Terraform: Up and Running: Writing Infrastructure as Code*. " O'Reilly Media, Inc.".

[16] Buschmann, F., Henney, K., and Schmidt, D. C. (2007). *Pattern-oriented software architecture, on patterns and pattern languages*, volume 5. John wiley & sons.

[17] Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McKelvie, S., Xu, Y., Srivastav, S., Wu, J., Simitci, H., et al. (2011). Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157. ACM.

[18] Carlson, M., Yoder, A., Schoeb, L., Deel, D., Pratt, C., Lionetti, C., and Voigt, D. (2014). Software defined storage. *Storage Networking Industry Assoc. working draft*, pages 20–24.

[19] Carretero Pérez, J., De Miguel Anasagasti, P., García Carballeira, F., and Pérez Costoya, F. (2001). Sistemas operativos. una visión aplicada. *Mac Graw Hill*.

[20] Cérin, C., Coti, C., Delort, P., Diaz, F., Gagnaire, M., Gaumer, Q., Guillaume, N., Lous, J., Lubiarz, S., Raffaelli, J., et al. (2013). Downtime statistics of current cloud solutions. *International Working Group on Cloud Computing Resiliency, Tech. Rep*.

[21] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., et al. (2003). Telegraphcq: Continuous dataflow processing for an uncertain world. In *Cidr*, volume 2, page 4.

[22] Chen, H. C. and Lee, P. P. (2013). Enabling data integrity protection in regenerating-coding-based cloud storage: Theory and implementation. *IEEE transactions on parallel and distributed systems*, 25(2):407–416.

[23] Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2):50–54.

[24] Chow, Golle, Jakobsson, Shi, Staddon, Masuoka, and Molina (2009). Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85–90. ACM.

[25] Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. (2012). *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media.

[26] CloudFront, A. (2014). Amazon cloudfront. *URL: http://aws. amazon. com/cloudfront*. Accedido el 15 de julio, 2019.

[27] CONAGUA (2018). Estaciones meteorológicas automáticas (emas). [Online; accessed May 03, 2018].

[28] Cordeiro, L., Fischer, B., and Marques-Silva, J. (2010). Continuous verification of large embedded software using smt-based bounded model checking. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pages 160–169. IEEE.

[29] Dastjerdi, A. V. and Buyya, R. (2016). Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116.

[30] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220.

[31] Demchenko, Y., Grosso, P., De Laat, C., and Membrey, P. (2013). Addressing big data issues in scientific data infrastructure. In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pages 48–55. IEEE.

[32] des Ligneris, B. (2005). Virtualization of linux based computers: the linux-vserver project. In *19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, pages 340–346. IEEE.

[33] docker (2019). Docker terminology. https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/container-docker-introduction/docker-terminology.

[34] docker-compose (2019). Docker compose. http://neokobo.blogspot.com/2017/04/docker-compose.html.

[35] Docker Documentation (2019). Overview of docker compose. http://company.com/.

[36] Drago, I., Mellia, M., M Munafo, M., Sperotto, A., Sadre, R., and Pras, A. (2012). Inside dropbox: understanding personal cloud storage services. In *Proceedings of the 2012 Internet Measurement Conference*, pages 481–494. ACM.

[37] Dworkin, M. J. (2015). Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, NIST.

[38] ECOSUR (2020). Estación recepción administrada por ecosur (eris)). [Online; accessed Feb 10, 2020].

[39] Faura, O. (2014). Integración continua (CI), Entrega continua (CD) y Despliegue Continuo (CD). https://devopsti.wordpress.com/2014/09/26/integracion-continua-ci-entrega-continua-cd-y-despliegue-continuo-cd/. Accedido el 15 de julio, 2019.

[40] Fowler, M. and Foemmel, M. (2006). Continuous integration. *(Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 122:14.

[41] French-Baidoo, R., Asamoah, D., and Oppong, S. O. (2018). Achieving confidentiality in electronic health records using cloud systems. *International Journal of Computer Network and Information Security*, 10(1):18.

[42] Frustaci, M., Pace, P., Aloi, G., and Fortino, G. (2017). Evaluating critical security issues of the iot world: Present and future challenges. *IEEE Internet of things journal*, 5(4):2483–2495.

[43] Gantz, J. and Reinsel, D. (2011). Extracting value from chaos. *IDC iview*, 1142(2011):1–12.

[44] Gantz, J. and Reinsel, D. (2012a). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007(2012):1–16.

[45] Gantz, J. and Reinsel, D. (2012b). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007(2012):1–16.

[46] 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.

[47] Gonzalez, J. L., Sosa, V., Diaz, A., Carretero, J., and Yanez0, 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.

[48] Gonzalez-Compean, J., Sosa-Sosa, V. J., Diaz-Perez, A., Carretero, J., and Marcelin-Jimenez, R. (2018). Fedids: a federated cloud storage architecture and satellite image delivery service for building dependable geospatial platforms. *International Journal of Digital Earth*, 11(7):730–751.

[49] Gonzalez-Compean, J., Telles, O., Lopez-Arevalo, I., Morales-Sandoval, M., Sosa-Sosa, V. J., and Carretero, J. (2019). A policy-based containerized filter for secure information sharing in organizational environments. *Future Generation Computer Systems*, 95:430–444.

[50] Grawinkel, M., Mardaus, M., Süß, T., and Brinkmann, A. (2015a). Evaluation of a hash-compress-encrypt pipeline for storage system applications. In *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 355–356. IEEE.

[51] Grawinkel, M., Mardaus, M., Süß, T., and Brinkmann, A. (2015b). Evaluation of a hash-compress-encrypt pipeline for storage system applications. In *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*, pages 355–356. IEEE.

[52] Gunawi, H. S., Hao, M., Suminto, R. O., Laksono, A., Satria, A. D., Adityatama, J., and Eliazar, K. J. (2016). Why does the cloud stop computing?: Lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 1–16. ACM.

[53] Harnik, D., Kat, R., Sotnikov, D., Traeger, A., and Margalit, O. (2013). To zip or not to zip: Effective resource usage for real-time compression. In *Presented as part of the 11th {USENIX} Conference on File and Storage Technologies ({FAST} 13)*, pages 229–241.

[54] Hayes, B. (2008). Cloud computing. *Communications of the ACM*, 51(7):9–11.

[55] Heath, M., Bowyer, K., Kopans, D., Kegelmeyer, P., Moore, R., Chang, K., and Munishkumaran, S. (1998). Current status of the digital database for screening mammography. In *Digital mammography*, pages 457–460. Springer.

[56] Humble, J. and Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education.

[57] Ibrahim, S., He, B., and Jin, H. (2011). Towards pay-as-you-consume cloud computing. In *2011 IEEE International Conference on Services Computing*, pages 370–377.

[58] IEEE (1990). Ieee standard glossary of software engineering terminology. In *IEEE Standard Glossary of Software Engineering Terminology*, pages 1–84.

[59] Institute, T. P. (2019). The real cost of business interruption. http://calyxit.com/the-real-cost-of-business-interruption/. Accedido el 15 de julio, 2019.

[60] Kaisler, S., Armour, F., Espinosa, J. A., and Money, W. (2013). Big data: Issues and challenges moving forward. In *2013 46th Hawaii International Conference on System Sciences*, pages 995–1004. IEEE.

[61] Kari, C., Kim, Y.-A., and Russell, A. (2011). Data migration in heterogeneous storage systems. In *2011 31st International Conference on Distributed Computing Systems*, pages 143–150. IEEE.

[62] KWIECIEŃ, A. (2019). 10 companies that implemented the microservice architecture and paved the way for others.

[63] Laatikainen, G., Mazhelis, O., and Tyrväinen, P. (2014). Role of acquisition intervals in private and public cloud storage costs. *Decision Support Systems*, 57:320–330.

[64] Lenz, M., Schmid, H. A., and Wolf, P. F. (1987). Software reuse through building blocks. *IEEE Software*, 4(4):34–42.

[65] Li, M., Qin, C., and Lee, P. P. (2015). Cdstore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pages 111–124.

[66] Li, X. et al. (2016). Data from qin-breast [data set]. *The Cancer Imaging Archive*.

[67] Lightsey, B. (2001). Systems engineering fundamentals. Technical report, DEFENSE ACQUISITION UNIV FT BELVOIR VA.

[68] Lin, H.-Y. and Tzeng, W.-G. (2011). A secure erasure code-based cloud storage system with secure data forwarding. *IEEE transactions on parallel and distributed systems*, 23(6):995–1003.

[69] Liu, C., Yang, C., Zhang, X., and Chen, J. (2015). External integrity verification for outsourced big data in cloud and iot: A big picture. *Future generation computer systems*, 49:58–67.

[70] Ltd, C. (2017). Lxc. https://linuxcontainers.org.

[71] Malhotra, L., Agarwal, D., and Jaiswal, A. (2014). Virtualization in cloud computing. *J. Inform. Tech. Softw. Eng*, 4(2).

[72] Mallawaarachchi, V. (2018). 10 common software architectural patterns in a nutshell. https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013. Accedido el 13 de agosto, 2019.

[73] Mao, B., Wu, S., and Jiang, H. (2015). Improving storage availability in cloud-of-clouds with hybrid redundant data distribution. In *IPDPS 2015m*, pages 633–642. IEEE.

[74] Mathew, S. and Varia, J. (2014). Overview of amazon web services. *Amazon Whitepapers*.

[75] Mazzara, M. and Govoni, S. (2005). A case study of web services orchestration. In *International Conference on Coordination Languages and Models*, pages 1–16. Springer.

[76] McCracken, D. D. and Reilly, E. D. (2003). Backus-naur form (bnf). In *Encyclopedia of Computer Science*, pages 129–131.

[77] Medvidovic, N. and Taylor, R. N. (2010). Software architecture: foundations, theory, and practice. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pages 471–472. ACM.

[78] Meister, D. and Brinkmann, A. (2009). Multi-level comparison of data deduplication in a backup scenario. In *Proceedings of SYSTOR 2009.*, page 8. ACM.

[79] Meister, D. and Brinkmann, A. (2010). dedupv1: Improving deduplication throughput using solid state drives (ssd). In *MSST 2010*, pages 1–6. IEEE.

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

[81] Menegotto, A. B., Becker, C. D. L., and Cazella, S. C. (2019). Computer-aided hepatocarcinoma diagnosis using multimodal deep learning. In *International Symposium on Ambient Intelligence*, pages 3–10. Springer.

[82] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014.

[83] Miller, K. (2018). Cloud deduplication, on-demand: Storreduce, an apn technology partner: Amazon web services.

[84] Mitzenmacher, M. (2001). The power of two choices in randomized load balancing. *IEEE TPDS*, 12(10):1094–1104.

[85] Montella, R., Di Luccio, D., and Kosta, S. (2018a). Dagon*: Executing direct acyclic graphs as parallel jobs on anything. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pages 64–73. IEEE.

[86] Montella, R., Di Luccio, D., and Kosta, S. (2018b). Dagon*: Executing directed acyclic graphs as parallel jobs on anything. In *Supercomputing 2018*.

[87] Morales, M., Gonzalez, J. L., Diaz, A., and Sosa, V. J. (2018). A pairing-based cryptographic approach for data security in the cloud. *IJISP*, 17(4):441–461.

[88] Morales-Ferreira, P., Santiago-Duran, M., Gaytan-Diaz, C., Gonzalez-Compean, J., Sosa-Sosa, V. J., and Lopez-Arevalo, I. (2018). A data distribution service for cloud and containerized storage based on information dispersal. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 86–95. IEEE.

[89] Morris, K. (2016). *Infrastructure as code: managing servers in the cloud*. "O'Reilly Media, Inc.".

[90] Moutsatsos, I. K., Hossain, I., Agarinis, C., Harbinski, F., Abraham, Y., Dobler, L., Zhang, X., Wilson, C. J., Jenkins, J. L., Holway, N., et al. (2017). Jenkins-ci, an open-source continuous integration system, as a scientific data and image-processing platform. *SLAS DISCOVERY: Advancing Life Sciences R&D*, 22(3):238–249.

[91] Mukherjee, M., Matam, R., Shu, L., Maglaras, L., Ferrag, M. A., Choudhury, N., and Kumar, V. (2017). Security and privacy in fog computing: Challenges. *IEEE Access*, 5:19293–19304.

[92] Nunes, B. A. A., Mendonca, M., Nguyen, X.-N., Obraczka, K., and Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634.

[93] Odelu, V., Das, A. K., Rao, Y. S., Kumari, S., Khan, M. K., and Choo, K.-K. R. (2017). Pairing-based cp-abe with constant-size ciphertexts and secret keys for cloud environment. *Computer Standards & Interfaces*, 54:3–9.

[94] Ortega-Arjona, J. L. (2010). *Patterns for Parallel Software Design*. Wiley Publishing, 1st edition.

[95] Pathania, N. (2017). *Learning continuous integration with Jenkins: a beginner's guide to implementing continuous integration and continuous delivery using Jenkins 2*. Packt Publishing Ltd.

[96] Peltz, C. (2003). Web services orchestration and choreography. *Computer*, 36(10):46–52.

[97] Pierre, G. and Van Steen, M. (2006). Globule: a collaborative content delivery network. *IEEE Communications Magazine*, 44(8):127–133.

[98] Pisello, T. and Quirk, B. (2004). How to quantify downtime. *Network World*, 5.

[99] Power, E. N. (2011). Understanding the cost of data center downtime: an analysis of the financial impact on infrastructure vulnerability. *white paper*.

[100] Puttaswamy, K. P., Kruegel, C., and Zhao, B. Y. (2011). Silverline: toward data confidentiality in storage-intensive cloud applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 1–13.

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

[102] Rao, B. T. et al. (2016). A study on data storage security issues in cloud computing. *Procedia Computer Science*, 92:128–135.

[103] Regulation, P. (2016). Regulation (eu) 2016/679 of the european parliament and of the council. *REGULATION (EU)*, 679.

[104] Reichman, A. (2011). File storage costs less in the cloud than in-house. *Forrester Research, Cambridge, MA*.

[105] Reinsel, D., Gantz, J., and Rydning, J. (2018). The digitization of the world from edge to core. *IDC White Paper*.

[106] Report, P. I. R. (2011). Calculating the cost of data center outages. https://airandpowersolutions.com/wp-content/uploads/2015/09/Calculating-the-Cost-

of-Data-Center-Outages-Ponemon-Institute-White-Paper-R0211-SL-24659.pdf. Accedido el 19 de Septiembre, 2019.

[107] Reyes-Anastacio, H. G., Gonzalez-Compean, J., Sosa-Sosa, V. J., Carretero, J., and Garcia-Blas, J. (2020). Kulla, a container-centric construction model for building infrastructure-agnostic distributed and parallel applications. *Journal of Systems and Software*, 168:110665.

[108] Rimal, B. P., Choi, E., and Lumb, I. (2009). A taxonomy and survey of cloud computing systems. In *2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51. Ieee.

[109] Salveggio, E. (2004). Your (un) reasonable expectations for privacy. *Ubiquity*, 2004(April):1–1.

[110] Santiago-Duran, M., Gonzalez-Compean, J., Brinkmann, A., Reyes-Anastacio, H. G., Carretero, J., Montella, R., and Pulido, G. T. (2020). A gearbox model for processing large volumes of data by using pipeline systems encapsulated into virtual containers. *Future Generation Computer Systems*.

[111] Schroeder, B. and Gibson, G. A. (2007). Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? In *FAST*, volume 7, pages 1–16.

[112] Shahin, M., Babar, M. A., and Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

[113] 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. ACM.

[114] Singh, A. and Chatterjee, K. (2017). Cloud security issues and challenges: A survey. *Journal of Network and Computer Applications*, 79:88–115.

[115] Singh, K. (2016). *Ceph Cookbook*. Packt Publishing Ltd.

[116] Sloan, R. H. and Warner, R. (2013). *Unauthorized access: The crisis in online privacy and security*. CRC press.

[117] Smart, J. F. (2011). *Jenkins: The Definitive Guide: Continuous Integration for the Masses*. .º'Reilly Media, Inc."

[118] Stacks, B. and Instances, A. (2015). In this section. *AWS CloudFormation*, page 104.

[119] Stojmenovic, I. and Wen, S. (2014). The fog computing paradigm: Scenarios and security issues. In *2014 federated conference on computer science and information systems*, pages 1–8. IEEE.

[120] Subhash, C., Patel, S., Umrao, L., Ravi, S., and Singh, S. (2012). *Policy-based Access Control in Cloud Computing*. Phd thesis, Indian Institute of Technology.

[121] SWsoft, I. (2016). Openvz. https://openvz.org.

[122] Tang, S., Lee, B.-S., and He, B. (2014). Towards economic fairness for big data processing in pay-as-you-go cloud computing. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 638–643. IEEE.

[123] Terstyanszky, G., Kukla, T., Kiss, T., Kacsuk, P., Balasko, A., and Farkas, Z. (2014). Enabling scientific workflow sharing through coarse-grained interoperability. *Future Generation Computer Systems*, 37. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications.

[124] van der Aalst, W. M., Dumas, M., ter Hofstede, A. H., and Wohed, P. (2002). Pattern-based analysis of bpml (and wsci). Technical report, Citeseer.

[125] van Der Aalst, W. M., Ter Hofstede, A. H., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns. *Distributed and parallel databases*, 14(1):5–51.

[126] van der Aalst, W. M. P. (1998). The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8:21–66.

[127] Virmani, M. (2015). Understanding devops & bridging the gap from continuous integration to continuous delivery. In *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, pages 78–82. IEEE.

[128] Westphal, A., Dufrasne, B., Brandenburg, J., Jamsek, J., Jehnen, K., Joseph, S., Olivieri, M., Rendels, U., Rodriguez, M., et al. (2012). *IBM System Storage DS8000: Architecture and Implementation*. IBM Redbooks.

[129] Wickramaarachchi, C. and Simmhan, Y. (2013). Continuous dataflow update strategies for mission-critical applications. In *2013 IEEE 9th International Conference on e-Science*, pages 155–163. IEEE.

[130] Wiseman, Y., Schwan, K., and Widener, P. (2005). Efficient end to end data exchange using configurable compression. *ACM SIGOPS Operating Systems Review*, 39(3):4–23.

[131] Xing, Y. and Zhan, Y. (2012). Virtualization and cloud computing. In *Future Wireless Networks and Information Systems*, pages 305–312. Springer.

[132] Xiong, H., Zhang, X., Zhu, W., and Yao, D. (2011). Cloudseal: End-to-end content protection in cloud-based storage and delivery services. In *SecureComm*, pages 491–500. Springer.

[133] Zhang, J. and Zhang, Z. (2015). Secure and efficient data-sharing in clouds. *CCPE*, 27(8):2125–2143.

[134] Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L., and Zhou, W. (2018). A comparative study of containers and virtual machines in big data environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 178–185. IEEE.

[135] Zheng, C. and Thain, D. (2015). Integrating containers into workflows: A case study using makeflow, work queue, and docker. In *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing*, pages 31–38.

[136] Zhou, M., Zhang, R., Xie, W., Qian, W., and Zhou, A. (2010). Security and privacy in cloud computing: A survey. In *2010 Sixth International Conference on Semantics, Knowledge and Grids*, pages 105–112. IEEE.

[137] Zissis, D. and Lekkas, D. (2012). Addressing cloud computing security issues. *Future Generation computer systems*, 28(3):583–592.