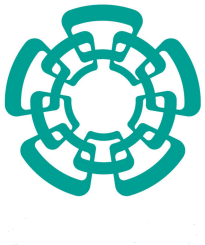


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Tamaulipas

Modelo de acoplamiento transversal para estructuras de procesamiento

Tesis que presenta:

Juan Armando Barrón Lugo

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. Septiembre, 2020

This research was partially funded by project number - from “Fondo Mixto Conacyt-Gobierno del Estado de Tamaulipas”

La tesis presentada por Juan Armando Barrón Lugo fue aprobada por:

Dr. Mario Garza Fabre

Dr. Iván López Arevalo

Dr. José Luis González Compeán, Director

Cd. Victoria, Tamaulipas, México., 8 de Septiembre de 2020

A mi familia, mis amigos, mi pareja y Dios.

Agradecimientos

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Modelo de acoplamiento transversal para estructuras de procesamiento

por

Juan Armando Barrón Lugo
Unidad Tamaulipas
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2020
Dr. José Luis González Compeán, Director

Existe un amplio espectro de tareas para el procesamiento de datos. Estas tareas pueden combinarse con otras y crear estructuras de procesamiento, como lo pueden ser los flujos de trabajo (workflows) o las tuberías de procesamiento (pipelines). La estructuras de procesamiento se han convertido en una piedra angular para procesar grandes volúmenes de datos en estudios relacionados con el análisis del clima, cambios en el medio ambiente, productos de observación de la tierra, etc.

Actualmente, existe la necesidad de generar estructuras que no solo permitan a las organizaciones colaborar con otras organizaciones, sino también reutilizar la información procesada. En otras palabras, aprovechar datos previamente procesados, evitando la "re-ejecución' de una tareas, o bien, reutilizar estructuras de procesamiento ya implementadas en una infraestructura de TI. En el presente trabajo de tesis se propone un modelo de acoplamiento transversal para la gestión de múltiples estructuras de procesamiento existentes así como de las fuentes de datos utilizadas por las etapas de diferentes estructuras de procesamiento.

El modelo considera un esquema de manejo de meta-flujos de trabajo basado en grafos dirigidos de procesamiento transversal (DAG_{TP}), el cual es asistido por un conjunto de restricciones que permitirán modelar las relaciones transversales existentes entre las estructuras de procesamiento que conforman al meta-flujo. El modelo también considera estructuras llamadas servicios de procesamiento transversal (TPS) para gestionar los resultados obtenidos por la solución generada.

Abstract

Transversal Processing Model for Scientific Processing structures

by

Juan Armando Barrón Lugo

Cinvestav Tamaulipas

Research Center for Advanced Study from the National Polytechnic Institute, 2020

Dr. José Luis González Compeán, Advisor

There is a wide spectrum of tasks for data processing. These tasks can be combined with others and create processing structures, such as workflows or pipelines. These processing structures have become a cornerstone for processing large volumes of data in studies related to the analysis of climate, changes in the environment, products of earth observation, etc.

Currently, there is a need to generate structures that not only allow organizations to collaborate with other organizations, but also to reuse the processed information. In other words, take advantage of previously processed data, avoiding the 're-execution' of a task, or else, reuse processing structures already implemented in an IT infrastructure. In the present thesis work, a transverse coupling model is proposed for the management of multiple existing processing structures as well as the data sources used by the stages of different processing structures.

The model considers a meta-workflow management scheme based on directed graphs of transversal processing (DAG_{TP}), which is assisted by a set of restrictions that will allow modeling the transversal relations existing between the processing structures. The model also considers structures called transversal processing services (TPS) to manage the results obtained by the generated solution.

Nomenclatura

WF	Flujo de Trabajo
M-WF	Meta-flujo de trabajo
DAG	Grafo Acíclico Dirigido
TPS	Servicio de Procesamiento Transversal
EP	Estructura de Procesamiento

1

Introducción

En este capítulo se presentan la motivación y antecedentes que originaron esta tesis, además se plantea la problemática abordada, la pregunta de investigación que definió el rumbo del trabajo, así como los objetivos alcanzados.

1.1 Antecedentes

El procesamiento de grandes volúmenes de datos en estudios relacionados al análisis del clima [48], medio ambiente [30] [22] y productos de observación de la tierra [47] producen información crítica y útil [29] [64] para los procedimientos de toma de decisiones (por ejemplo, enfocados en prevenir desastres [43] [68] y analizar los patrones en los datos. [39] [65]).

En este contexto, existen las estructuras de procesamiento (**EP**). Una estructura de procesamiento consiste en un conjunto de tareas de procesamiento para la transformación de datos en información. Algunos ejemplos son:

- Flujos de trabajo (workflows): Conjunto de tareas, generalmente no lineales, que filtran o transforman datos, a menudo desencadenando eventos externos. La estructura de un flujo de trabajo puede ramificarse.

- Tubería de procesamiento (Pipeline): Serie de tareas, generalmente lineales, que filtran o transforman datos. Por lo general, los procesos se ejecutan de manera consecutiva. La estructura de una tubería de procesamiento no se ramifica ni se repite.

Una estructura de procesamiento está compuesta por cuatro componentes principales: *fuentes de datos*, *tareas* (módulos de procesamiento individuales o etapas), *interfaces de entrada/salida* y *almacenes de datos*.

En la práctica, los $tareas$ se conforman generalmente por aplicaciones, servicios o sistemas que funcionan como instancias de software. Estas tareas se encuentran interconectadas mediante interfaces de entrada y salida formando un grafo dirigido, cuyo inicio es la fuente de datos y el final suele ser un almacén de datos. En la Figura 1.1 se puede observar una tubería de procesamiento representada de forma gráfica, formada por tres procesos encadenados a través de cuatro fuentes de datos.

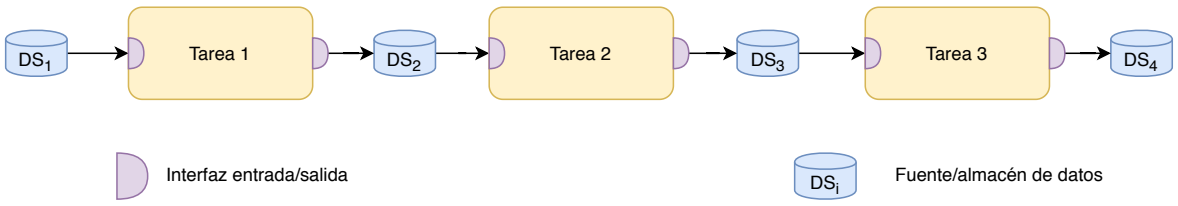


Figura 1.1: Ejemplo gráfico de una tubería de procesamiento (pipeline).

Una manera eficaz de modelar una estructura de procesamiento es a través de un Grafo Acíclico Dirigido (DAG, por sus siglas en inglés). Un DAG es un grafo dirigido el cual no contiene ningún ciclo, es decir, no existe ningún camino que empiece y llegue al mismo vértice (o nodo) [61]. A diferencia de otras estructuras de grafos existentes como lo puede ser un árbol, la característica principal de un DAG es que es unidireccional, es decir, las conexiones entre un vértice y otro van en una sola

dirección. Siguiendo este concepto, los vértices que conforman el DAG representan los procesos y fuentes de datos de la estructura de procesamiento, mientras que las aristas dirigidas representan la transición de los datos provenientes de un proceso y procedentes a otro, o en palabras más simples, las aristas representan las conexiones entre los componentes. Por tanto, una EP se puede representar como:

$$EP = \{TASK_1, TASK_2, TASK_3, ..., TASK_n\}$$

$$TASK_i = \{DS_1, PROC, DS_2, P\}$$

Donde EP consiste en un conjunto de tareas ($TASK_i$), las cuales están compuestas por una fuente de datos (DS_1), un proceso de transformación ($PROC$), un almacén de datos (DS_2) y un producto transformado (P) como resultado del proceso.

Asumamos una EP que incluye tres tareas: A = adquisición, B = pre-procesamiento y C = procesamiento. En este caso su notación se expresaría de la siguiente forma:

$$EP_1 = \{DS_0 \xrightarrow{P} A \xrightarrow{P_a} DS_1 \xrightarrow{P_a} B \xrightarrow{P_{ab}} DS_2 \xrightarrow{P_{ab}} C \xrightarrow{P_{abc}} DS_3\}$$

Donde el producto P es recuperado del almacén DS_0 por el proceso A, transformándose en el producto P_a para posteriormente ser depositado en el almacén DS_1, repitiéndose la misma secuencia de pasos para el resto de procesos.

En este contexto, se asume que las instancias de un grafo seguirán el modelo de procesamiento tradicional llamado ETC por Extracción-Transformación-Carga (ETL, por sus siglas en inglés, que representan un acrónimo para Extract, Transform and Load). Esto significa que cada una de las tareas $extrae$ datos de una fuente, $transforma$ los datos recibidos en una nueva versión de los valores de entrada, los cuales son entregados al almacén de datos o $cargados$ a otra tarea dependiendo del DAG que define a la EP.

1.2 Motivación

Las estructuras de procesamiento (*EP*) se han convertido en una piedra angular para crear servicios de procesamiento de grandes volúmenes de datos (big data). Algunos ejemplos de EP existentes son para el análisis de clima [29], estudios sobre contaminación [51], creación de productos de observación de la tierra [47] y manufactura y transporte de contenidos digitales [20].

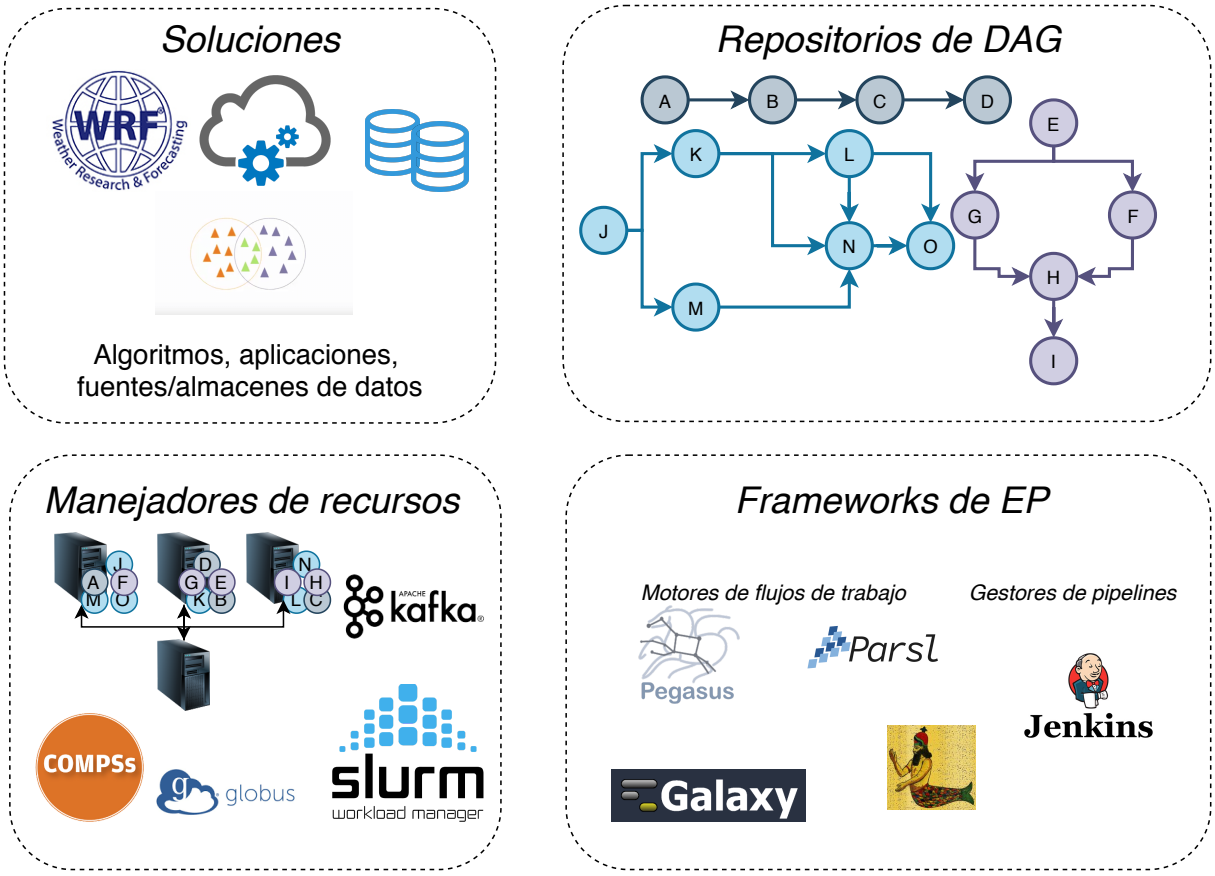


Figura 1.2: Ecosistema de soluciones.

Actualmente existe una gran variedad de algoritmos, servicios y aplicaciones para el procesamiento de datos. Estas aplicaciones pueden ser encadenadas entre si creando diversos EP, los cuales pueden ser almacenados, compartidos, y materializados mediante frameworks. Los frameworks tienen como objetivo coordinar la ejecución de cada una de las tareas, abstrayendo las EP como cajas negras. En

este sentido, las cajas negras toman como entrada un grafo de EP y entregan como resultado final los datos procesados al usuario. A su vez, los frameworks gestionan los recursos de cómputo disponibles utilizando gestores/manejadores de recursos. Los gestores de recursos permiten repartir las diferentes tareas de un EP en un conjunto de recursos físicos (por ejemplo un cluster de computadores, el cloud, computadora personal, etc), asignar recursos de cómputo (procesador, memoria y/o disco) o gestionar el envío de datos entre instancias de cómputo. A todos estos elementos que rodean la creación y materialización de un EP se le definió como *ecosistema de soluciones* (ver Figura 1.2).

En un ecosistema de soluciones se pueden realizar diversas operaciones:

- Diseñar soluciones bajo demanda.
- Reutilizar tareas o estructuras completas, aunque esto conlleve su instalación y configuración previa.
- Reutilizan los datos iniciales y los resultados finales.

No obstante, en este ecosistema de soluciones existen limitaciones en cuanto a operaciones se refiere. Por lo cual, actualmente no es posible:

- Compartir fuentes de datos, tareas o sumideros de datos entre múltiples EP.
- Acoplar nuevas EP (completas o parciales) con EP existentes.
- Crear nuevas soluciones basadas en múltiples EP existentes.
- Acceder a las versiones de los datos transformados por las etapas intermedias de una EP.
- Agregar o retirar nuevas EP a EP existentes que se encuentren en ejecución.

Estas limitaciones encontradas en los ecosistemas actuales representan la motivación del presente trabajo de tesis, el cual pretende abordar estas áreas de oportunidad mediante un modelo de acoplamiento de estructuras de procesamiento.

1.3 Planteamiento del problema

Las estructuras de software como las tuberías, los patrones, mallas de servicios y los flujos de trabajo se han convertido en la piedra angular para realizar procesamiento de datos. En el ámbito científico, se han desarrollado múltiples soluciones basadas en estas estructuras de procesamiento. Actualmente, una gran cantidad de estas soluciones ya están disponibles para que la comunidad científica procese y analice, por ejemplo, datos del clima [29], la contaminación [51], los datos de observación de la tierra [47], contenidos digitales, entre otros.

Actualmente existe la necesidad de estructuras que no solo permitan a las organizaciones colaborar con otras organizaciones, sino que también reutilicen tanto la información procesada como las EP ya implementadas en una infraestructura de TI.

En un escenario tradicional de bigdata, los datos son gestionados comúnmente a través de una serie de etapas (tareas), como la adquisición de datos de una fuente determinada; la preparación para ser utilizados en un formato dado; el preprocesamiento de datos para eliminar valores atípicos, reducir la dimensionalidad o enriquecerlos calculando los datos faltantes; el procesamiento para convertirlos en información (clasificación, agrupamiento o definición de reglas, por nombrar algunos); y la preparación de la información para ser consumida en un proceso de toma de decisiones. Este tipo de tareas tiende a repetirse en múltiples soluciones, por lo cual tiene sentido que los usuarios puedan reutilizar estructuras de procesamiento parciales o completas ya creadas, para construir soluciones integrales.

Sin embargo, las aplicaciones, servicios y sistemas utilizados en cada etapa pueden llegar a ser estructuras complejas. Reutilizarlos, acoplarlos a estructuras existentes, compartir datos producidos por ellos entre diferentes estructuras, cruzar información para procesar simultáneamente múltiples fuentes de datos, así como extraer, indexar, publicar y consumir información en cualquier etapa son tareas desafiantes.

Utilizar frameworks para crear estructuras de procesamiento como pipelines [27], flujos de trabajo

[6, 7, 36] o servicios [2, 12], no es una opción factible e inmediata para que las organizaciones consuman información procesada, así como para reutilizar EP que podrían implementarse en múltiples infraestructuras de TI y creadas por diferentes frameworks.

Además, en las soluciones tradicionales, las EP se crean como una caja negra donde una solución recupera datos de una fuente (por ejemplo, cualquier carpeta o ubicación en la nube), ejecuta un conjunto de tareas para convertir los datos entrantes en información útil que se depositan en un almacén de datos (por ejemplo, ubicación de almacenamiento en la nube). Esto significa que el usuario final solo tiene acceso a los resultados finales colocados en el almacén, pero no a los resultados obtenidos por etapas intermedias, que podrían ser útiles como datos de entrada para otras soluciones o para ser consumidos por usuarios finales o aplicaciones.

La creación de herramientas basadas en estas tareas representa un problema no resuelto o parcialmente resuelto. Resolver este problema en la práctica no es trivial ya que el lenguaje de programación utilizado por los frameworks que crean estructuras de procesamiento no es necesariamente el mismo, las soluciones ya disponibles podrían estar preparadas para ser ejecutadas en una plataforma dada, así como también se imponen requisitos de infraestructura y plataforma.

Esto representa un obstáculo para que usuarios construyan soluciones y procesos críticos de toma de decisiones de manera flexible y dinámica. En este contexto, los usuarios finales terminan creando una nueva solución en lugar de aprovechar las soluciones existentes.

Actualmente existen aproximaciones de solución que parcialmente enfrentan esta problemática: La primera se enfoca en la construcción de soluciones basadas en niveles de jerarquía y la segunda se enfoca en la composición de nuevas soluciones reutilizando fragmentos de flujos de trabajo.

Las soluciones basadas en el enfoque de nivel jerárquico [66] consiste en la composición de diferentes EP como una única solución basada en dos niveles de jerarquía. Es decir, las EP son utilizadas como tareas de otra EP. No obstante, se encuentran áreas de oportunidad, puesto que las EP son utilizadas en su totalidad (no parcialmente) y a su vez limita la posibilidad de reutilizar datos producidos por instancias previamente ejecutadas y etapas intermedias.

Por otro lado se encuentran las soluciones que reutilizan fragmentos de flujo de trabajo para la composición de nuevas soluciones. Estas se centran principalmente en algoritmos que buscan fragmentos de tareas dentro de repositorios de flujos de trabajo aptos para ser reutilizables y construir nuevas soluciones [70] [18]. Sin embargo, a pesar de que algunos de estos trabajos brindan el diseño automático de soluciones mediante el uso de fragmentos, aún existen limitaciones y áreas de oportunidad de este tipo de enfoques. No obstante, esto es comprensible dado el enfoque de generar algoritmos es localizar fragmentos, no en todo lo que implica su aplicación.

En base a estas observaciones se exploró el concepto de transversalidad. Transversal es definido por la Real Academia Española como "aquello que se halla o se extiende atravesado de un lado a otro", o bien "aquello que se cruza en dirección perpendicular con aquello de que se trata". En la Figura 1.3 se puede observar una linea transversal cruzando las lineas l_1 y l_2. Abstrayendo estas lineas como si fuesen EP, surge la idea de generar soluciones basadas en EP existentes mediante el acoplamiento de estas a través de una conexión virtual definida en cualquier punto de una EP (cualquier etapa). En base al lo anterior descrito, surge el concepto de procesamiento transversal, el cual fue definido como "proveer la propiedad de transversalidad a las EP al permitir su integración con múltiples tareas y/o EP existentes mediante puntos transverales".

1.4 Hipótesis

Dada la problemática descrita previamente, se plantearon las siguientes preguntas de investigación:

- ¿Cómo se pueden crear soluciones que incluyan múltiples EP (parciales o completas)?.
- ¿Cómo se pueden gestionar las intersecciones de soluciones que incluyen múltiples EP sin producir ciclos, mientras se mantienen los prerrequisitos de los DAGs de cada EP que hace parte de una solución?.

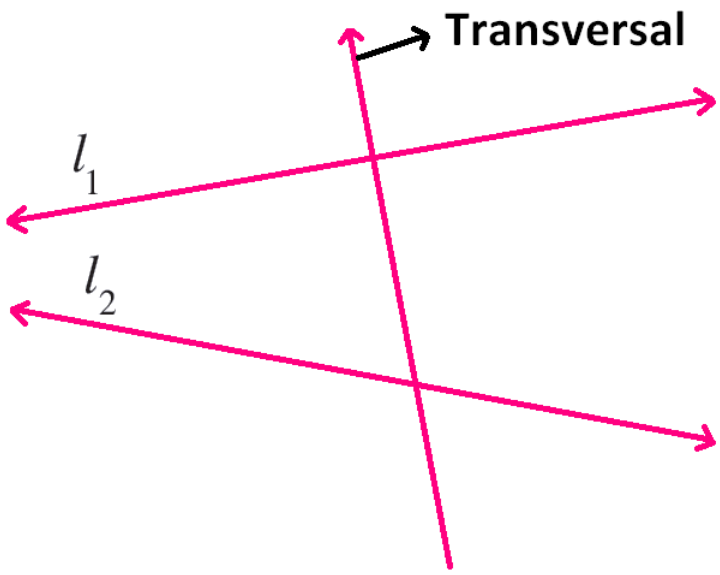


Figura 1.3: Representación conceptual de la propiedad de transversalidad.

- ¿Cómo se pueden materializar soluciones que incluyen múltiples EP si estos ya se encuentran en ejecución?.

Con el fin de dar respuesta a las preguntas antes formuladas se estableció la siguiente hipótesis de investigación:

Se puede construir una solución compuesta por múltiples estructuras de procesamiento mediante la creación de un flujo transversal, creado a partir del acoplamiento secuencial de estructuras aglutinantes que generen mapas de las intersecciones de los flujos que la conforman, y además verifiquen que las topologías de los DAGs sean preservadas, lo cual resultará en un nuevo DAG que evitará la creación de ciclos y que un motor de creación de flujos podrá materializar como una nueva solución.

1.5 Objetivos

En esta sección se describe el objetivo general y los objetivos particulares para hacer frente al problema planteado anteriormente.

Objetivo general

Definir un modelo de acoplamiento transversal para la cohesión de múltiples estructuras de procesamiento.

Objetivos particulares

- Definir un esquema de manejo de meta-flujos de trabajo basado en grafos dirigidos de procesamiento transversal (DAG_{TP}).
- Definir reglas y restricciones que permitan modelar relaciones transversales entre múltiples flujos de trabajo.
- Definir estructuras de servicios de procesamiento transversal (TPS) para consolidar resultados producidos por meta-flujos.

1.6 Metodología de investigación

En esta sección se describe la propuesta general para bordar el problema planteado. Asimismo, se describe un método a seguir que consta de cuatro etapas.

1.6.1 Modelo conceptual de la solución propuesta

Con base en los desafíos que se presentan y la problemática descrita anteriormente, se propone un modelo de acoplamiento transversal para estructuras de procesamiento, es decir, un modelo para la construcción de meta-flujos. En este sentido, un meta-flujo tomaría como insumo EPs existentes, y permitirá tomar ventaja de la información procesada de cada tarea que lo conforma través de servicios

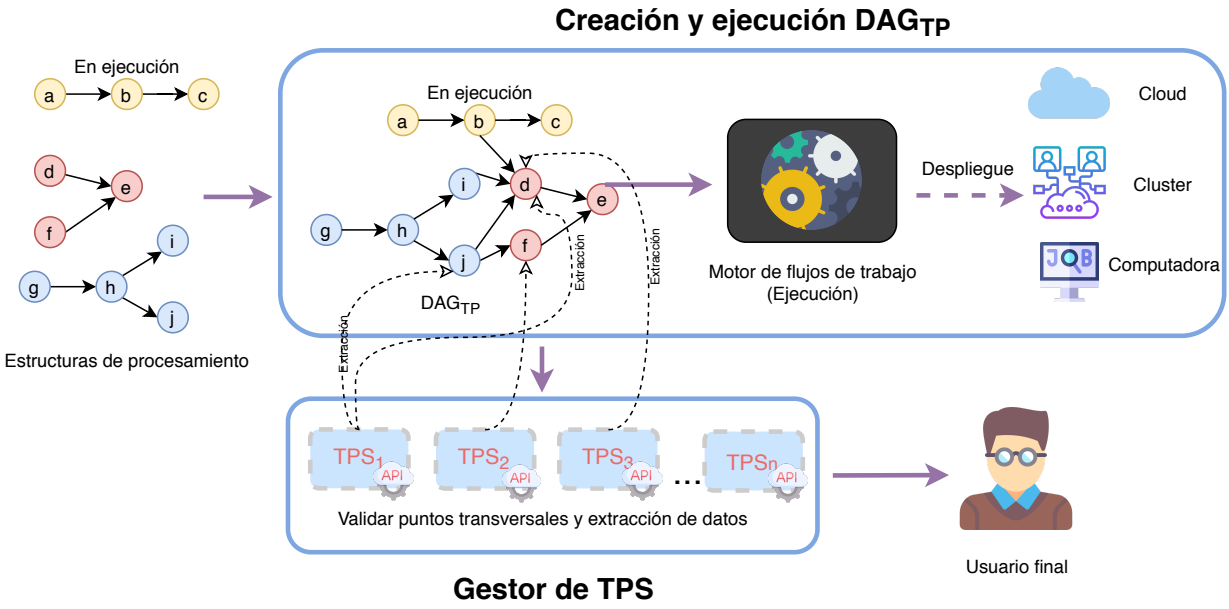


Figura 1.4: Diagrama conceptual de la solución.

de análisis que permitan generar resultados o "productos" para diversos fines. A estos servicios de análisis los nombramos *Servicios de Procesamiento Transversal* (Transversal Processing Services, por sus siglas en inglés, TPS) (ver en Figura 1.4).

El modelo considera un esquema de manejo de meta-flujos de trabajo basado en grafos acíclicos dirigidos de procesamiento transversal (ver DAG_{TP} en Figura 1.4), el cual es asistido por un conjunto de restricciones que permitirán modelar las relaciones transversales existentes entre las EP que conforman al meta-flujo.

La solución propuesta consta de tres fases y dos componentes:

1. Declaración de EP: En esta fase, el usuario interviene en la declaración de las EPs mediante el uso de DAGs y utilizando una notación, especificando dependencias, entradas y salidas.

2. Construcción de DAG_{TP} : Se construye un solo DAG de manera automática a partir de los DAGs declarados por el usuario en la etapa anterior, siguiendo así mismo una configuración dada por el usuario para la transversalidad entre las EP.

3. Ejecución del DAG_{TP} y despliegue de servicios transversales: El DAG_{TP} es ejecutado en un

motor de flujos de trabajo y se levantan los servicios de procesamiento transversal (TPS), permitiendo generar productos transversales en cualquiera de los componentes del meta-flujo de trabajo.

Los componentes de la solución propuesta se describen a continuación:

- **Generación de meta-flujo transversal (DAG_{TP})**: Dada la existencia de un conjunto de DAGs representando a diferentes flujos de trabajo, este componente realizará el establecimiento de relaciones basándose en diversas reglas y restricciones, generando un DAG de procesamiento transversal o DAG_{TP} representando un meta-flujo de trabajo. Adicionalmente, este componente será el encargado de gestionar la correcta ejecución del meta-flujo.

- **Servicios de procesamiento transversal (TPS)**: Asumiendo la existencia de un meta-flujo transversal, se generan los servicios de procesamiento transversal. Estos servicios son los encargados de consolidar los resultados producidos por una solución DAG_{TP}.

1.6.2 Metodología de investigación

En esta sección se especifican cuatro etapas y sus respectivas actividades para la realización del proyecto de investigación. En la Figura 1.5 se muestra la metodología descrita a manera de diagrama.

A continuación se describen cada una de estas etapas:

E1.- Estado del arte y marco teórico

El objetivo de esta etapa es realizar una revisión profunda sobre propuestas similares y definir los objetivos y problemas específicos que se abordarán. Una vez realizada la revisión de la literatura, se comenzó con la redacción del protocolo de tesis.

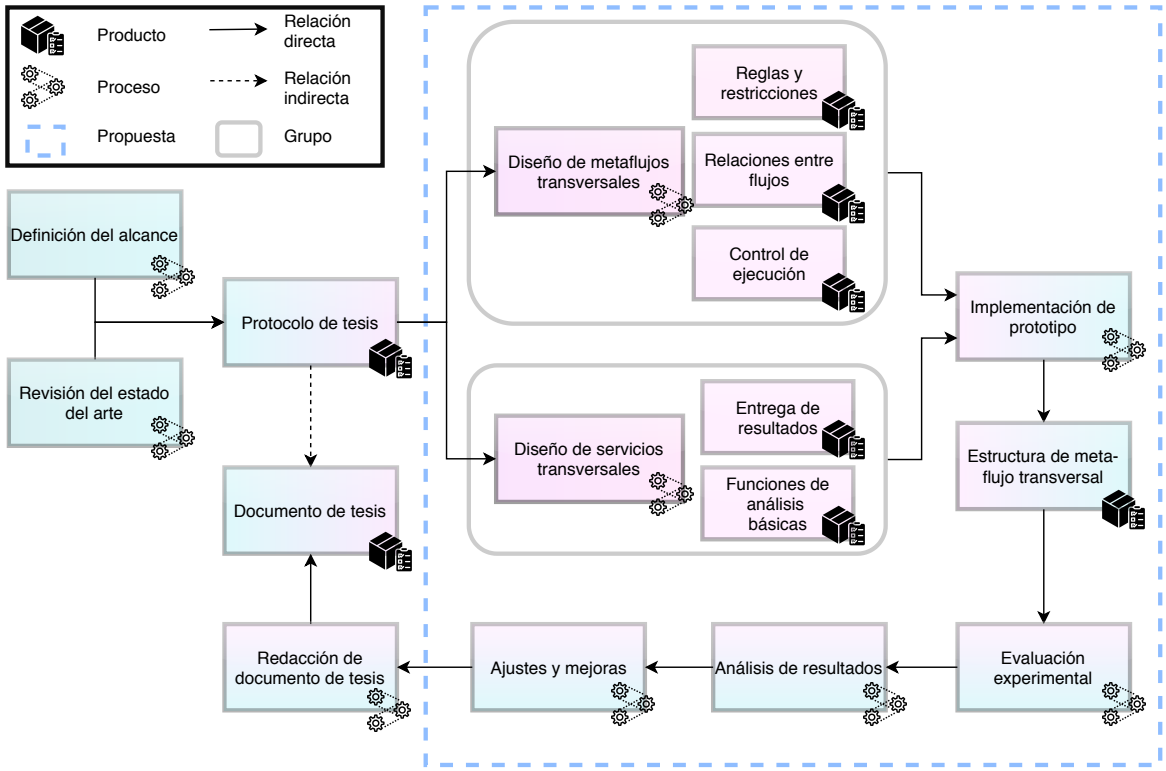


Figura 1.5: Diagrama de metodología de investigación.

E2.- Obtención de un modelo de acoplamiento transversal

Esta etapa tiene como objetivo obtener el modelo para la generación de meta-flujos transversales, para ello se plantearon las siguientes actividades:

- Diseñar meta-flujos transversales (DAG_{TP}): En esta etapa se diseñará el modelo para la generación de un meta-flujo o DAG_{TP}. Esto conlleva:
  - Reglas y restricciones. En este contexto, se entiende por regla como un precepto establecido para ejecutar y construir el DAG_{TP}.
  - Establecer relaciones entre flujos de trabajo (DAGs).
  - Control de ejecución del meta-flujo basándose en una configuración.
- Diseñar servicios transversales: Servicios encargados de realizar las operaciones de

transversalidad dentro de las etapas del meta-flujo de trabajo, esto incluye:

- Definir relación entre servicios transversales y meta-flujo de trabajo.
- Entrega de resultados.
- Funciones básicas para análisis de datos (operaciones transversales tales como ANOVA, clustering, diferencial, etc).

- Implementación del prototipo.
  - Implementar manejador de flujos de trabajo.
    1. Implementar reglas para la declaración de flujos de trabajo (DAGs).
    2. Generación de meta-flujo o DAG_{TP}.
    3. Gestión de ejecución de meta-flujos transversales.
  - Implementar servicios transversales (TPS) para análisis y cruza de información.
    1. Declaración de variables transversales.- Mecanismos para definir variables de interés por parte del usuario. La declaración de estas variables permitirá definir los datos a ser operados por procesos transversales, esto incluye implementación de restricciones, reglas y relaciones entre variables de una misma fuente y de una fuente de datos a otra.
    2. Implementar operaciones básicas para el procesamiento transversal.

E3.- Evaluación experimental

Las actividades a realizar durante esta etapa serán las siguientes:

- Diseño de baterías de experimentos.
- Evaluación de baterías de experimentos.
- Análisis preliminares y ajustes del modelo.

E4.- Análisis de resultados y Conclusiones

El objetivo de esta etapa es realizar un análisis completo de los resultados obtenidos en la etapa de evaluación experimental, con el fin de realizar una discusión y obtener conclusiones acerca de la veracidad de la hipótesis. Esta etapa está conformada por las siguientes actividades:

- Análisis crítico de resultados.
- Discusión y conclusiones finales.
- Redacción del documento de tesis.

1.7 Organización de la tesis

La organización de esta tesis esta compuesta de la siguiente forma: el capítulo 2 presenta los conceptos básicos para la facilitación de la comprensión del tema de tesis; en el capítulo 3 se describen aquellos trabajos relacionados al tema presentado; en el capítulo 4 se describe el diseño de la solución que se propone, así como la descripción e implementación de cada uno de los componentes que la conforman; en el capítulo 5 se describe el proceso de experimentación, resultados obtenidos y la descripción de los casos de uso utilizados; finalmente, en el capítulo 6 se señalan las conclusiones obtenidas.

2

Marco teórico

En este capítulo se presentan conceptos fundamentales asociados a este trabajo de tesis. Se hace especialmente énfasis en los flujos de trabajo, meta-flujos de trabajo y la característica de transversalidad.

2.1 Flujos de trabajo

Un flujo de trabajo, como es definido en [24], es la automatización de un proceso, en su conjunto o parte, durante el cual los datos, información o tareas se pasan de un participante (un recurso) a otro tras realizar una tarea dada de acuerdo con un conjunto de reglas de procesamiento. En otras palabras, se puede entender que los flujos de trabajo científicos computarizados son estructuras lógicas que permiten modelar una secuencia ordenada de acciones de procesamiento. La principal característica de un flujo de trabajo es la ramificación de tareas. En este sentido, un flujo de trabajo puede no tener definida una sola tarea inicial o final.

Los primeros flujos de trabajo (tanto en ciencias como en los negocios) eran utilizados para el

movimiento de entradas y salidas de una estación de trabajo a otra (tarea de procesamiento). El uso de scripts implicaban un procesamiento previo para transformar los datos obtenidos en información útil y comprensible para interpretación humana, siendo estas tareas más sofisticadas y exigentes con el paso del tiempo [55].

Con lo anterior, se puede entender que la función principal de estas estructuras es transformar datos en **información** útil para un usuario final, realizando por ejemplo las siguientes acciones:

- Acciones de procesamiento en datos: Adquisición, preparación (limpieza, enriquecimiento, etc), transformación (conversiones, codificación, reducción, clasificación, etc).
- Acciones de gestión en información: Indexación (mapeo, preservación), visualización y entrega.

2.1.1 Flujos de trabajo científico

Cuando un flujo de trabajo se aplica a la automatización de procesos de ámbito científico a gran escala (o e-Science), es acuñando el término flujo de trabajo científico [9]. Los flujos de trabajo empresarial, en general, son ejecutados a un nivel de abstracción diferente al que los científicos necesitan para ser aprovechados. En cambio, los flujos de trabajo científicos requieren herramientas de mayor nivel que permitan conectar componentes de resolución de problemas para la validación de hipótesis. Un flujo de trabajo científico intenta capturar una serie de pasos analíticos que describan el proceso de diseño de experimentos, proporcionando un entorno para ayudar al proceso de descubrimiento científico a través de la combinación de gestión, análisis, simulación y visualización de datos. El trabajo científico se centra en la realización de experimentos, por lo tanto, un flujo de trabajo de este ámbito debe permitir aplicar la metodología del científico sobre los recursos de cómputo con el fin de obtener resultados de apoyo a su investigación. Adicionalmente, el sistema de flujo de trabajo debe permitir que se muestre la misma información en varios niveles de abstracción, dependiendo de quién esté usando el sistema [5, 13].

Los flujos de trabajo han surgido como una forma de formalizar y estructurar el análisis de datos,

de tal forma que sea sencillo ejecutar cálculos necesarios dentro de diversos recursos de cómputo, recopilar información obtenida por dichos cálculos y compartir los resultados [55].

Algunas características de las estructuras de flujos de trabajo científicos computarizados son:

- Generalización: Permite gestionar múltiples tipos de datos a procesar y múltiples acciones de procesamiento.
- Modularización: La modularización de acciones de procesamiento, en la práctica, permite identificar problemas de rendimiento.
- Colaboración: Cada entidad participante en un flujo es responsable de una(s) etapa(s) del flujo, lo cual permite la colaboración que resulta clave para gestionar problemas de gran escala.

Un punto importante dentro de los flujos de trabajo en la comunidad científica implica el transporte y el análisis de grandes cantidades de datos. El financiamiento y la destinación y uso de recursos caros no pueden permitirse sobre flujos de trabajo que fallen a la mitad de una ejecución. Por lo tanto, es deseable que los sistemas que los soportan sean sólidos y confiables [13].

Algunos de los campos dentro de la ciencia donde es útil el uso de flujos de trabajo van desde la astronomía, ciencia de ondas gravitacionales, ecología, meteorología, terremotos, neurociencia [55], el manejo y análisis de datos medioambientales [30], estudios de largas secuencias de datos (e.g. ADN) [32, 38], química computacional [23], bioinformática [59] y estudios y análisis de datos históricos referentes a fenómenos naturales [22].

2.1.2 Modelo de procesamiento ETL

ETL (Extract-Transform-Load) [8] es un enfoque de procesamiento compuesto por 3 etapas:

- *Extracción (Extract)*: Consiste en la extracción datos desde una aplicación de origen o una fuente de datos.

- *Transformación (Transform)*: Los datos extraídos pasan a esta etapa y son transformados aplicando una serie de funciones o tareas de procesamiento.

- *Carga (Load)*: Los datos transformados son finalmente cargados a una fuente destino.

2.1.3 Componentes de flujos de trabajo

Un flujo de trabajo está compuesto en esencia por cuatro componentes principales [7, 36, 42]:

- **Tareas**: Una tarea consiste en una aplicación, servicio o sistema que funciona como instancia de software para realizar procesamiento sobre los datos. Cada tarea dentro de un flujo de trabajo realiza el procesamiento sobre los datos siguiendo el enfoque ETL.

- **Fuente de datos (data source)**: Es el repositorio donde se encuentran los datos de entrada que una tarea toma para realizar su procesamiento.

- **Almacenes de datos (data sink)**: Repositorio donde los datos son depositados.

- **Interfaces de entrada/salida**: Son los canales por los cuales se envían y reciben los datos, desde la fuente de datos hacia la tarea y desde la tarea hasta el almacén de datos. Las principales interfaces de entrada y salida son:

    - *Red*: Es aquella utilizada para la transición de datos a través de conexiones de red, ya sea cableadas o inalámbricas. Este tipo de interfaz es especialmente útil cuando se requiere transmitir datos hacia otra infraestructura física (e.g. un servidor). Por ejemplo, una tarea de procesamiento puede recibir los datos de entrada desde un servidor de internet a través de un socket.

    - *Memoria*: Se utiliza para establecer la comunicación y realizar transición de datos usando la memoria primaria de un computador (e.g. memoria ram). Por ejemplo, un proceso del sistema operativo accede a ciertos datos ubicados en un segmento de memoria, los procesa

y posteriormente ubica sus resultados en el mismo segmento para que otro proceso pueda tomarlos.

- *Sistema archivos*: Se utiliza para el intercambio de datos a través de la memoria secundaria, el disco. Por ejemplo, un aplicativo para el cifrado de archivos es ejecutado por un usuario, este proceso toma como datos de entrada el archivo a cifrar desde una carpeta ubicada en el sistema de archivos, posteriormente se aplica el algoritmo de cifrado y al final el nuevo archivo se deposita en una carpeta.

2.1.4 Representaciones

Existen múltiples maneras de representar a un flujo de trabajo, algunas de las principales formas son los Grafos Acíclicos Dirigidos (DAG) y Grafos Cíclicos Dirigidos (DCG), siendo la única diferencia el permitir ciclos dentro de un flujo de trabajo [55]. Por otro lado, existen diversas formas de realizar la formalización y descripción del proceso de ejecución de un flujo de trabajo:

- **Grafos**: Consiste en una estructura compuesta de vértices y aristas.
- **Redes de petri**: Una Red de Petri es una representación matemática o gráfica de un sistema. Fue definida en la década de los años 1960 por Carl Adam Petri. Es una generalización de la teoría de autómatas que permite expresar un sistema en una serie de eventos concurrentes [69].
- **SSDL**: El lenguaje de descripción de servicio SOAP (SSDL) es un lenguaje de descripción centrado en SOAP para servicios web. SSDL proporciona los conceptos básicos sobre los cuales se construyen los marcos para describir protocolos tanto para patrones simples de intercambio de mensajes como para flujos de trabajo. [63].
- **Calculo-pi**: El cálculo-pi es un álgebra de procesos para la descripción formal de diversos eventos. El cálculo-pi permite la comunicación entre diversos elementos cuya configuración de

comunicación puede cambiar [44, 63].

2.1.5 Grafo acíclico dirigido (DAG)

Un DAG (del inglés Directed Acyclic Graph), consiste en una estructura de grafo el cual no contiene ningún ciclo, es decir, no existe ningún camino dentro del grafo tal que partiendo de un vértice V se llegue al mismo vértice. Los DAG son utilizados para modelar problemas donde no tiene sentido que existan caminos que generen ciclos, fluyendo solamente en una única dirección. Un DAG es definido como una tupla $G = (V, E)$, donde V representa los vértices del grafo, mientras que el conjunto E es un conjunto de pares ordenados (x,y), es decir, $E = \{(x, y) \in V \times V | x \neq y\}$.

2.1.6 Orquestación y motores de flujos de trabajo

La orquestación de los flujos de trabajo se refiere a la actividad de definir la secuencia de tareas necesarias para completar un objetivo. Un flujo de trabajo es una plantilla para dicha orquestación [55]. Cuando un flujo de trabajo está diseñado para resolver un problema particular, se dice que es una instancia de un flujo de trabajo. En este contexto se encuentran los motores de flujos de trabajo, los cuales son herramientas para la orquestación de las tareas de una instancia de un flujo de trabajo. El motor del flujo de trabajo es aquel que proporciona la automatización de cada una de las etapas de un flujo de trabajo mediante la gestión de la secuencia de actividades. El motor realiza la invocación de cada etapa sobre un recurso tecnológico, gestionando la correcta ejecución y las entradas y salidas de cada una de estas.

2.2 Meta-flujos de trabajo

Un meta-flujo de trabajo consiste en un flujo de trabajo compuesto por dos o más flujos de trabajo. Un meta-flujo de trabajo es definido en la literatura [4] [1] [57] como un flujo de trabajo expresado a múltiples niveles, es decir, los flujos de trabajo pasan a ser tareas dentro de un flujo

de trabajo superior, formando estructuras más complejas que al final son tratadas como un flujo de trabajo. Para fines de este trabajo de tesis, se define a un meta-flujo de trabajo como una estructura de flujo de trabajo compuesta en parte o en su totalidad por diversos flujos de trabajo completos o parciales, dicho de otro modo, los flujos de trabajo que componen al meta-flujo no necesariamente se encuentran completamente autocontenidos en tareas, sino que el meta-flujo es conformado por las conexiones entre las tareas de los diferentes flujos de trabajo.

2.2.1 Flujos de trabajo colaborativos

El proceso de diseño de un flujo de trabajo científico puede involucrar a diferentes usuarios con diferentes roles. Así mismo, múltiples flujos de trabajo pueden realizar tareas muy específicas y diversas las cuales pueden ser difíciles de comprender para un solo científico, sin embargo, estos flujos pueden colaborar y trabajar en conjunto mediante la interconexión entre sus entradas y salidas. En este contexto surgen los flujos de trabajo colaborativos, los cuales son aquellos que, a través del intercambio de información, realizan sus tareas de procesamiento [21, 41, 67].

2.2.2 Flujos de trabajo multinivel

Un flujo de trabajo multinivel es aquel en el cual al menos una de las tareas de procesamiento es en esencia un flujo de trabajo encapsulado. Se dice que son multinivel debido a los diversos niveles de jerarquía que se manejan [66], es decir, la secuencia de ejecución de cada uno de los flujos de trabajo. Un flujo de trabajo multinivel puede ser considerado como un metaflujo de trabajo, debido a que este está compuesto por múltiples flujos de trabajo. La diferencia de un flujo de trabajo multinivel y un flujo de trabajo colaborativo radica en que mientras que el primero trata a los flujos de trabajo que lo componen como tareas, por lo cual, estos deben ser ejecutados en su totalidad y al momento de iniciar la tarea; en el caso del segundo, los flujos de trabajo funcionan en conjunto y por su cuenta, realizando intercambios de información generada por sus tareas de procesamiento.

2.3 Microservicios

Los microservicios son componentes autónomos que aíslan las capacidades específicas de una aplicación. Un microservicio generalmente se ejecuta en su propio proceso y se comunica mediante interfaces estandarizadas [17], es decir, son independientes unos de otros. Algunas de las características de los microservicios son:

- Autocontenido: Se refiere a la capacidad de ser desplegado y modificado sin afectar a otros elementos de software. Esto permite el ahorro de tiempo al realizar tareas de mantenimiento, así como el no ser necesario tener un amplio conocimiento sobre el funcionamiento de toda la aplicación, sino solamente del microservicio específico.

- Independiente: Cada microservicio es ejecutado con sus propios recursos, por lo cual se evita la sobrecarga y la caída total de la aplicación.

- Alta cohesión: Dado a los protocolos de comunicación estandarizados que se utilizan para la comunicación entre microservicios, es posible interconectar múltiples microservicios sin modificar el código fuente de cada uno de estos. Así mismo, permite el remplazo de piezas de software dentro de una aplicación de manera más sencilla.

Los microservicios tienen un estilo monolítico, capaces de ser implementados de manera independiente, por lo cual, puede entenderse a los microservicios como piezas de software con una alta cohesión, bajo acoplamiento, autocontenidas, monolíticas y autónomas que ofrecen a disposición un servicio.

2.3.1 Arquitecturas de microservicios

Una arquitectura de microservicios se refiere a un estilo como enfoque para el desarrollo de una aplicación como un conjunto de pequeños servicios, cada uno de los cuales se ejecuta de

manera independiente a los otros y se comunican con mecanismos ligeros, a menudo una API [17]. Mientras que en una arquitectura monolítica una aplicación es una única unidad altamente acoplada, una arquitectura de microservicios está desarrollada por pequeñas unidades de procesamiento encapsulando procesos específicos.

Algunas ventajas y desventajas [14, 60] que presentan los microservicios son:

- Ventajas:
  - Escalabilidad en la aplicación.
  - Despliegue independiente de cada microservicio.
  - Agilidad para el diseño, desarrollo y pruebas de cada microservicio.
  - Tolerancia a fallos, dada la separación de la aplicación en módulos.
- Desventajas:
  - Alto consumo de memoria.
  - Complejidad en la gestión de los múltiples microservicios.
  - Latencia en la comunicación.

2.3.2 Contenedores virtuales

Un contenedor virtual es una pieza de software estandarizada y autónoma la cual puede encapsular una aplicación junto con todas sus dependencias para su rápida ejecución de un entorno informático a otro. En otras palabras, un contenedor virtual consta de una aplicación empaquetada junto a todas sus dependencias la cual es fácil de implementar y actualizar. Al igual que las máquinas virtuales, los contenedores eliminan la dependencia del hardware, dado que múltiples contenedores virtuales pueden ser ejecutados en una misma infraestructura de hardware (e.g. un servidor) . Algunas de las características [10, 16] con las que cuentan los contenedores virtuales son:

- Flexible: Incluso las aplicaciones más complejas se pueden encapsular en contenedores.

- Estándar: Dada a la capacidad de los contenedores virtuales de abstraer el entorno en el que se ejecutan, estos son portables y pueden ejecutarse en diferentes infraestructuras y sistemas operativos.

- Ligero: los contenedores comparten el kernel del sistema operativo anfitrión, por tanto, no se requiere un sistema operativo por contenedor virtual, reduciendo el tamaño considerablemente en referencia a otros tipos de virtualización, como lo son las máquinas virtuales.

- Aislado: Dentro de un contenedor virtual, tanto la CPU, la memoria, el almacenamiento y los recursos de red son virtualizados al nivel del sistema operativo, lo que aísla a las aplicaciones contenidas en un contenedor virtual del resto en un entorno de pruebas, evitando afectar a las aplicaciones exteriores.

2.3.3 Orquestadores de contenedores virtuales

Para realizar el despliegue y control de contenedores virtuales dentro de una infraestructura de hardware es necesaria una plataforma de contenedores virtuales y de orquestadores. Las plataformas de contenedores virtuales permiten abstraer el manejo, creación y ejecución de contenedores, permitiendo gestionar y lanzar contenedores de manera rápida y sencilla. Por otro lado, los orquestadores abstraen las plataformas de contenedores, agregando funcionalidades nuevas para la gestión y control de los contenedores virtuales, por ejemplo, en ambientes distribuidos. Existen múltiples orquestadores de contenedores virtuales, algunos de ellos son Docker-compose, Docker-swarm y Kubernetes [10, 16].

Docker

Docker[1] [33] es una plataforma de código libre para desarrolladores y administradores de sistemas para construir, compartir y ejecutar contenedores. Docker encapsula las aplicaciones dentro de contenedores virtuales para posteriormente ser desplegadas en cualquier sistema operativo que tenga instalada la plataforma de Docker. Docker distingue tres componentes principales para la virtualización de aplicaciones:

- Contenedores: Un contenedor es un proceso en ejecución con características de encapsulación aplicadas para mantenerlo aislado del host y de otros contenedores.
- Imágenes: Consiste en una plantilla estática desde la cual se crean instancias de contenedores virtuales, es decir, una imagen de contenedor incluye todo lo necesario para la ejecución de una aplicación, desde el código o binario, dependencias, configuraciones, etc.
- Registro (docker-hub): Es un sistema de almacenamiento para imágenes Docker.

En cuanto a la orquestación de múltiples contenedores virtuales a la vez, Docker cuenta con las herramientas Docker-compose y Docker-swarm. Estas herramientas permiten el lanzamiento y gestión de múltiples contenedores virtuales de manera simultanea y distribuida, gestionando desde la comunicación, control de ejecución, administración de recursos, entre otros [40] .

[1] https://www.docker.com

3

Trabajos relacionados

En esta sección se presentan los trabajos relacionados revisados en la literatura con respecto al objeto de estudio. Para la búsqueda de trabajos relacionados se siguió una taxonomía considerando las palabras clave de este trabajo, siendo las principales la gestión sobre flujos de trabajo, reutilización de información, creación de meta-flujos y microservicios.

3.1 Gestores de flujos de trabajo

A lo largo de los años se han desarrollado tecnologías para atacar algunas de las problemáticas descritas, como el manejo y ejecución de múltiples flujos de trabajo, generar patrones de paralelismo para mejorar la eficiencia, facilitar la creación y ejecución de flujos de trabajo científicos para usuarios poco experimentados en el área, etc.

3.1.1 Swift

Swift presentado por Wilde *et al.* [62] es un lenguaje para la composición de aplicaciones secuenciales en aplicaciones en paralelo que puedan ser ejecutadas en supercomputadoras con múltiples procesadores, clusters, grids o clouds. Swift provee un lenguaje de tipo script basado en la sintaxis de C para la declaración de flujos de trabajo, permitiendo describir cómo cada aplicación se conecta con el resto, para posteriormente ejecutarse de manera paralela. Estas características permiten regularizar los procesos independientes junto con sus entradas y salidas para el despliegue de estos de manera distribuida. A pesar de lo anterior, no permite la integración de servicios o software externo, es decir, se centra solamente en la paralelización de aplicativos dejando de lado a aplicaciones encapsuladas como lo son aquellas ofrecidas como un servicio (as-a-Service).

3.1.2 PyComps

Otro proyecto existente para el despliegue en paralelo de flujos de trabajo es *PyComps*, presentado por Tejedor *et al.* [56]. PyComps es una distribución de Comps [7], un framework[1] para el desarrollo de computación paralela. PyComps permite la paralelización y ejecución de flujos de trabajo en múltiples procesadores, además de tener la capacidad de ser desplegado en diferentes arquitecturas de red como los grid, cloud o clusters. La declaración de los procesos del flujo de trabajo dentro de una aplicación de python se hace a través de decoradores (e.g. $@app$), permitiendo definir funciones que requieren de un alto poder de procesamiento como "tareas" independientes. El orquestador de pycomps se encarga de realizar el proceso de paralelización de manera automática, detectando las dependencias de cada tarea, volviendo su uso más fácil a usuarios que no pertenecen al área. Desafortunadamente, a pesar de que Comps tiene distribuciones para C/C++, java y python, se limita solamente a funciones escritas en esos lenguajes (existe una distribución para cada lenguaje), por consiguiente, no es posible incorporar servicios externos al flujo de trabajo. PyComps sirve como

[1] Conjunto de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

una capa adicional o framework para la paralelización de tareas hechas en lenguaje python, en otras palabras, la paralelización se realiza mediante el núcleo de Comps (hecho en C), por lo cual, es necesario que el usuario tenga todas estas dependencias de Comps instaladas para generar flujos de trabajo en python con PyComps.

3.1.3 Parsl

Parsl [6] (Paralel Scripting Library) es una biblioteca basada en Swift desarrollada en python para el desarrollo y ejecución de flujos de trabajo orientados a flujos de datos de manera paralela y distribuida. Parsl puede ser usado para el desarrollo de aplicaciones en python para desplegarse en paralelo, esto se logra mediante el uso de decoradores y funciones los cuales declaran el flujo de los datos y la comunicación entre las funciones mediante las entradas y salidas. Al mismo tiempo, aplicaciones externas pueden unirse a estos flujos de trabajo mediante la declaración de las entradas y salidas en un script de python. Las características más relevantes de Parsl son: la elasticidad, ya que permite el control de los recursos de cada aplicativo o proceso del flujo de trabajo; el soporte para la ejecución de instancias en múltiples recursos, como lo puede ser un cloud, un cluster de computadoras o el fog (ejemplo, computadora personal); y el acoplamiento de aplicaciones externas. Estas características permiten a Parsl realizar la ejecución de múltiples instancias de procesamiento o incluso, ejecución de múltiples flujos de trabajo como uno solo, sin embargo no es ese el objeto de estudio principal en Parsl, sino la paralelización y construcción de flujos de trabajo a partir de scripts de python o servicios externos.

3.1.4 DagOn*

En el año 2018 Raffaele Montella *et al.* [36] aportan *DagOn** (Direct acyclic graph On anything). DagOn* es una biblioteca de código abierto en python para la ejecución de flujos de trabajo sobre cualquier tipo de recurso de cómputo. DagOn* abstrae el concepto de flujo de trabajo y los representa

como DAG, en donde los nodos de este grafo son vistos como tareas, donde cada tarea consume datos de entrada y produce datos de salida (siguiendo un enfoque de procesamiento ETL). Algunas de las principales características de esta herramienta es el manejo de dependencias por parte de tareas ejecutadas en recursos externos y la posibilidad de ejecución de software externo. DagOn* tiene la capacidad de incorporar diversos recursos de software a un flujo de trabajo. Un flujo de trabajo se declara mediante uso de las herramientas que DagOn* provee mediante su biblioteca en python, siendo capaz de integrar tanto funciones nativas de python, tareas o servicios web (web task), recursos de software utilizando SSH[2], tareas embebidas dentro de clouds o máquinas virtuales, tareas dentro de contenedores virtuales[3], etc. Para el manejo de dependencias a recursos remotos, DagOn* utiliza un esquema denominado *workflow://*, el cual consiste básicamente en un sistema de archivos virtual (similar a ftp[4]). Por último, DagOn* cuenta adicionalmente con un servicio web el cual permite a los usuarios monitorear el estatus de los flujos de trabajo ejecutados, y además permite implementar interacciones entre distintos flujos de trabajo desplegados a través de web task. Gracias al manejo del denominado esquema *workflow://*, para DagOn* podría ser posible realizar una colaboración entre distintos flujos de trabajo, o bien, generar estructuras más complejas de tareas, conectadas a través de este esquema de compartición de datos, sin embargo, esta característica no se encuentra implementada a la fecha.

3.1.5 Decaf

Para 2017 Matthieu Dreher y Tom Peterka [15] presentaron *Decaf*, un sistema para la comunicación en paralelo de tareas sobre flujos de trabajo in situ[5]. Funciona como un middleware[6] enfocado en los datos, el cual permite la comunicación y el intercambio de información entre diferentes tareas dentro de un flujo de trabajo. Adicionalmente, Decaf puede controlar la ejecución de un flujo

[2]Secure Shell. Protocolo de administración remota.
[3]https://www.docker.com/resources/what-container.
[4]Protocolo de transferencia de archivos.
[5]Dentro de un mismo computador.
[6]Capa que provee de un servicio a una aplicación.

de trabajo a través de mensajes. Las tareas realizan un intercambio de mensajes para la coordinación de ejecución. Cuando una tarea recibe todos los mensajes declarados por sus tareas predecesoras, esta puede iniciar su proceso. De esta manera, se genera un control de ejecución tanto para tareas secuenciales como para tareas en las que se comparten fuentes de información. Como detalle adicional, Decaf da soporte a tareas cíclicas, es decir, tareas que regresan al mismo punto o a la misma tarea.

Decaf trabaja como una capa extra sobre flujos de trabajo que realizan su comunicación a través de memoria, es decir, aquellos que son ejecutados en supercomputadoras. Dado a esto, la paralelización y despliegue en múltiples recursos puede realizarse con el apoyo de otras herramientas como Parsl o PyComps. Decaf realiza el intercambio de información entre tareas (módulos del flujo de trabajo) a través de mensajes y recursos llamados *links*, los cuales permiten la transformación y redistribución de los datos entre tareas.

Dado que Decaf trabaja mediante conexiones usando la memoria de la máquina, no es posible generar flujos de trabajo y distribuir tareas a través de alguna infraestructura en red (e.g un cluster de computadoras, el cloud, etc). No obstante, una de las principales ventajas de Decaf es la velocidad de comunicación entre sus tareas. Decaf no es precisamente un motor de flujos de trabajo, pero dada su naturaleza de "cohesionador" de aplicativos, es capaz de realizar algunas tareas de un motor de flujos de trabajo.

3.2 Reutilización de flujos de trabajo y resultados

Los flujos de trabajo científicos, en contraste de los flujos de trabajo en el ámbito empresarial, tienden a manejar un enfoque más "abierto" a los datos manejados para experimentos, o bien, resultados obtenidos en la ejecución de un flujo de trabajo. Esto es debido a que la replicación de experimentos es un concepto importante a la hora de generar nueva información pues permite corroborar los resultados obtenidos por un investigador. A continuación se muestran diferentes herramientas y soluciones propuestas siguiendo el enfoque de "compartir" flujos de trabajo.

3.2.1 PyComps-Decaf

Orcun Yildiz *et al.* proponen una solución para la composición de múltiples flujos de trabajo basados en dos niveles de jerarquía:

- El primero se basa en Decaf [15] para la composición de flujos de trabajo *in situ*, flujos de trabajo con tareas que intercambian mensajes a través de la memoria, compuestos por varios campos de datos que pueden o no ser utilizados en su totalidad.

- El segundo es el desarrollo de la nueva solución (Meta-DAG) utilizando PyComps [56] [7]. PyComps permite reutilizar los DAG producidos por Decaf y ejecutar los flujos de trabajo *in situ* como tareas en un entorno distribuido. Este framework también se encarga de la gestión de dependencias y el transporte de datos entre tareas.

3.2.2 Fragflow

Daniel Garijo *et al.* presentan Fragflow [18], un algoritmo para la detección automática de fragmentos de tareas reutilizables en un conjunto de flujos de trabajo. Fragflow centra sus esfuerzos en solamente realizar la detección de fragmentos de tareas en un conjunto de flujos de trabajo, es decir, busca patrones en tareas que se repiten, extrayéndolas como un fragmento reutilizable. Sin embargo, existen limitaciones y áreas de oportunidad en este tipo de enfoque. No obstante, esto es comprensible dado el enfoque está en la detección, no en todo lo que implica su aplicación.

3.2.3 Taverna

Duncan Hull *et al.* [25] (2006) presentan Taverna, una herramienta para la construcción y ejecución de flujos de trabajo como servicios. Taverna fue creado para la ejecución de flujos de trabajo bioinformáticos, ofreciendo una variedad de procesos (como servicios) en un catálogo que puede ser utilizado para la construcción de diversos flujos de trabajo. Adicionalmente, Taverna permite

la invocación de servicios externos para incorporarlos a sus flujos de trabajo, así como una interfaz sencilla para la construcción y ejecución de estos procesos, permitiendo que usuarios sin mucha experiencia en áreas de computación puedan construir sus flujos de trabajo de procesamiento. Como característica adicional, la separación de los procesos como servicios permite despliegue de estos sobre otras máquinas, clusters, clouds, etc. En 2010 Paolo Missier et al. [34] presentaron una actualización de la herramienta llamada T2 (Taverna 2). En ella proponían una nueva arquitectura para mejorar la escalabilidad de ejecución de flujos de trabajo y facilitar el diseño de flujos de trabajo para diversos escenarios. Actualmente Taverna es un proyecto en incubación de apache[7] y se encuentra en la versión 2.5.0. Taverna permite la publicación de flujos de trabajo (estructura y parámetros), datos de entrada y resultados, para el uso de distintos usuarios. Esto permite que los experimentos realizados puedan ser replicados y corroborados, además de ser usados como un nuevo componente para otros experimentos.

3.2.4 DfAnalyzer

En 2018 Vítor Silva *et al.* presentaron DfAnalyzer [53], una herramienta para monitorear y analizar flujos de datos. DfAnalyzer surge bajo el contexto de los flujos de trabajo científicos y simuladores, dada la recurrente generación de datos crudos (sin procesar) que se generan, existiendo la posibilidad de rastrear y realizar análisis sobre estos datos. DfAnalyzer permite integrar diferentes archivos de datos con variables similares de un mismo flujo de trabajo, posteriormente los datos son indexados en una base de datos estructural desde la cual es posible realizar consultas. Este proceso es realizado de manera asíncrona al flujo de trabajo, permitiendo realizar consultas sin necesidad de que el flujo de trabajo termine su procesamiento [52]. A pesar de no ser reportado en el artículo, los resultados obtenidos por un análisis de datos crudos podría ser de utilidad para diferentes usuarios, es decir, podría ser un área de oportunidad el publicar estos resultados para su uso.

[7] https://taverna.incubator.apache.org/

3.2.5 Galaxy

Galaxy [19] es un enfoque integral para apoyar la reproducible y transparente generación de experimentos en las ciencias enfocadas a la vida. Galaxy es una plataforma web que permite a los científicos diseñar flujos de trabajo usando una interfaz gráfica, reutilizar componentes para el análisis de datos (preestablecidos), reutilizar conjuntos de datos para experimentos, y posteriormente, publicar la estructura del flujo en conjunto de los datos de entrada y resultados, dando pie a la reproducción de sus experimentos por cualquier usuario. El enfoque principal de Galaxy es la reproducción de experimentos, esto difiere del enfoque que se plantea en este trabajo de tesis. Para Galaxy, si un usuario desea reutilizar los datos generados por una etapa intermedia en un flujo de trabajo publicado, es necesario descargar (o importar) dicho flujo, y rediseñarlo para ser ejecutado a su conveniencia.

3.3 Microservicios

James Lewis y Martin Fowler [17] definen a los microservicios como una aproximación al desarrollo de aplicaciones con una pequeña cantidad de servicios corriendo por su cuenta y utilizando métodos ligeros para el intercambio de información. Los microservicios tienen un estilo monolítico, capaces de ser implementados de manera independiente. Por otro lado, Johannes [58] define un microservicio como una aplicación que puede ser desplegada, escalada y probada independientemente. Con estas definiciones puede entenderse un microservicio como una instancia de software monolítica y autónoma que ofrece y pone a disposición un servicio. Una arquitectura conformada por microservicios tiene diversos beneficios, por ejemplo [45]:

- Dado que los servicios son pequeños, el mantenimiento es sencillo.
- Cada servicio puede ser desplegado (ejecutado) de manera independiente.
- Los servicios son escalables (es posible agregar más con cambios mínimos).

- Permite adoptar nuevas tecnologías (por ejemplo, diversos servicios en distintos lenguajes de programación).

3.3.1 MERRA analytic service

En 2014 John L. Schnase *et al.* [48] mencionan CAaaS *(Climate Analytics-as-a-Service)*, una forma de enfrentar el análisis de grandes volúmenes de datos climáticos. CAaaS se basa en el concepto de *process-as-a-service*, el cual consiste en procesamiento en forma de "outsourcing"[8] empleando cloud computing. Schnase et al. construyen CAaaS bajo el concepto de *generatividad*, el cual ellos definen como la capacidad de auto ensamble, en otras palabras, la capacidad de conectar aportes de diversas personas o grupos, que puedan o no estar trabajando en conjunto. Esta idea es introducida como una forma de afrontar los problemas de "big data" [3, 54]. CAaaS es construido bajo cinco conceptos los cuales, en palabras de los autores, contribuyen a la generatividad:

- **Map reduce**[9]**:** Distribuye el procesamiento de datos a través de grandes conjuntos de computadores. Consta de dos operaciones básicas: *Map*, toma las entradas y las reduce en problemas más pequeños; *Reduce*, recoge las soluciones de todos los subproblemas y las combina de alguna manera [3].
- **iRODS:** Middleware que facilita el acceso a los datos en un ambiente distribuido.
- **vCDS (Virtual Climate Data Server):** iRODS modificado para el manejo de datos climáticos.
- **Canonical Ops:** Operaciones comunes con las cuales se pueden crear otras más complejas (e.g. Media, varianza, etc.).
- **API**[10]**:** Expone al usuario las "Canonical Ops" y las funciones para el manejo de los datos.

[8] Delegación funciones de una empresa
[9] http://hadoop.apache.org/
[10] Interfaz de programación de aplicaciones

Las principales aportaciones de CAaaS son la adaptabilidad (debido a las operaciones comunes), generatividad, accesibilidad y transferibilidad, gracias al API y al vCDS.

3.3.2 Geoscience data analytics

En 2015 Zhenlong Li *et al.* [31] proponen un framework para flujos de trabajo científicos para la analítica de datos geocientíficos, esto con el objetivo de enfrentar tres retos que supone el pre-procesamiento y análisis en esta área: a) big data, refiriéndose al manejo y control de grandes conjuntos de datos en un orden de terabytes; b) computación intensiva, dado a la gran cantidad de recursos computacionales que son necesarios para procesar los datos; c) complejidad, debido a que la analítica de este tipo de datos requiere una serie de pasos complejos con secuencias específicas. Para atacar el problema de "big data" se propuso el uso de "Map reduce" a través de una infraestructura de nodos de Hadoop en el cloud, de esta forma se ataca el problema de computación intensiva, distribuyendo la carga de trabajo entre los diversos nodos del cloud. Por último, proponen el uso de flujos de trabajo compuestos por servicios para atacar el problema de complejidad, así mismo, proponen la instalación de dependencias de estos servicios en conjunto con apache Hadoop en "imágenes" para máquinas virtuales, de esta forma pueden ser desplegados los servicios con todas sus dependencias en diversos nodos de procesamiento, y adicionalmente, pudiendo ser controlados e invocados mediante Oozie[11], un planificador de flujos de trabajo manejado por Hadoop.

3.3.3 OceanTEA

Oceanographic Time Series Exploration (OceanTEA) [26] es un software desarrollado por Arne N. Johanson *et al.* presentado en el 2016. OceanTEA surge como una herramienta para el análisis de fuentes de datos con multidimencionalidad, siendo esto un problema debido al incremento constante en la cantidad de series de datos climáticos. OceanTEA está construido utilizando una arquitectura

[11] https://oozie.apache.org/

de microservicios, lo cual da como principales ventajas el despliegue en arquitecturas de cómputo como un cloud. En base a lo descrito en el artículo, el análisis de distintas fuentes de datos tiene una importancia significativa, además de ser un reto dado al creciente volumen de datos.

3.3.4 GeRDI

En 2018 Tavares *et al.* [11] presento *GeRDI* (Generic Research Data Infrastructure), una infraestructura genérica para datos de investigación. GeRDI surge como una solución a la variedad de cambios posibles que se pueden presentar en una infraestructura de datos científicos, dado que es común el aumento de requerimientos o funcionalidades necesarios en los flujos de trabajo científicos. Con esto en mente, los autores proponen una arquitectura de microservicios, la cual provee la modularidad y facilidad de cambios de las diferentes etapas. GeRDI cuenta con las siguientes etapas:

- **Archive:** Repositorio de datos (data source).
- **Harvest:** Recolecta los metadatos[12] de la fuente de datos.
- **Bookmark:** Tratamiento de metadatos importantes para facilitar búsquedas y procesamientos posteriores.
- **Store:** Almacenamiento local (sistema de archivos del computador del usuario) o remoto.
- **Preprocess:** Filtro de los datos: procesos de normalización, limpieza, etc.
- **Analyze:** Análisis de los datos para generar conocimiento científico.
- **Publish:** Almacén de datos.

La característica principal de esta infraestructura es un modelo genérico para el tratamiento de datos científicos de manera modular, permitiendo adaptarse a diversas situaciones y diferentes tipos de datos manejados (con respecto al contexto de los datos).

[12]Datos acerca de los datos

3.4 Discusión

En la Tabla 3.2 se muestra un resumen de los trabajos reportados en la literatura. La tabla muestra una comparativa entre tres características: función de la propuesta; orientación, refiriéndose hacia el tipo de escenarios donde se aplica; y por último, el tipo de producto generado por cada estudio.

En primera instancia se identificaron algunos de los principales motores de flujos de trabajo en la literatura. Entre los diferentes trabajos se encontraron sutiles diferencias, sin embargo, cada motor se enfoca principalmente en la resolución de un problema específico. Para el caso de Swift y Parsl, se centran en mejorar la ejecución y rendimiento en tiempo de ejecución mediante la paralelización de los flujos de trabajo, además de incorporar soporte para que diversos aplicativos y en diversos lenguajes puedan ser utilizados. DagOn* va más allá, tomando como base los esfuerzos realizados por Parsl y Swift, aplicando un enfoque de paralelización a las tareas de un flujo de trabajo, y adicionando mayor soporte a distintos tipos de tareas de procesamiento, siendo capaz de ejecutar aplicativos desarrollados en cualquier lenguaje de programación. Como un extra, DagOn* también se encarga del manejo de dependencias, es decir, los datos procesados entre una etapa y otra son detectados automáticamente y gestionados mediante el esquema denominado *workflow://*, el cual funciona como un sistema de archivos virtual. Revisando detenidamente este esquema, pudimos identificar como un área de oportunidad la adición de soporte para la colaboración entre distintos flujos de trabajo, puesto que, si el manejo de dependencias ya es realizado, se puede expandir para manejar estas dependencias entre distintos flujos de trabajo, pudiendo generar nuevas estructuras de flujos de trabajo o meta-flujos.

Decaf en conjunto con otras soluciones como PyComps, consiguen alcanzar un enfoque de meta-flujos. Mientras que Decaf se encarga de la composición de flujos de trabajo in situ (dentro de un mismo recurso de cómputo), PyComps se encarga de la composición de flujos de trabajo constituidos por tareas desplegadas en diversos recursos de cómputo, tomando como una etapa al flujo construido

por Decaf. Este enfoque de meta-flujo difiere en cierta medida al buscado por este tema de tesis, dado que la composición de los flujos de trabajo se realiza para generar a un flujo de trabajo superior, el cual, debe ser ejecutado en conjunto para la generación de los resultados buscados, mientras que lo que se busca en este trabajo es la reutilización de estos flujos, sin necesidad de tener que ser configurados nuevamente, o bien, ejecutados para generar nuevamente los resultados.

En cuanto a reutilización de resultados e instancias de procesamiento se refiere, se encontraron diversos trabajos. DfAnalyzer se aproxima a un análisis transversal, permitiendo acceder a los datos crudos de cualquier etapa del flujo de trabajo, sin embargo, este se enfoca más al análisis de los datos de manera comparativa y no como generación de productos útiles, es decir, se encarga solamente de la integración de múltiples archivos de información hacia una sola fuente (base de datos estructural), permitiendo realizar consultas de manera más sencilla. Adicionalmente, las consultas son únicamente realizadas sobre flujos de trabajo por separado, es decir, no existe algún tipo de cruce entre diversos flujos durante alguna etapa, no obstante, no se especifica que su uso sea exclusivamente para flujos por separado, o sea imposible combinar u operar múltiples fuentes de información provenientes de diversos flujos de trabajo (dado que solo es mencionado para casos de un flujo de trabajo en paralelo), dejando esto como un área de oportunidad.

Trabajos como Taverna y Galaxy se centran principalmente en un problema dentro de los flujos de trabajo científicos, la replicación de experimentos. Estas herramientas facilitan el trabajo a los científicos enfocados en el área de biología, para el diseño y ejecución de experimentos mediante el uso de aplicativos los cuales pueden ser encadenados para formar flujos de trabajo. Adicionalmente, permite al usuario publicar todos los elementos necesarios para que sus experimentos puedan ser reproducibles y corroborados por otros. En contraste al objetivo que se busca en este trabajo de tesis, si bien la publicación de flujos de trabajo y resultados es un concepto interesante y el cual retomamos, el enfoque que se busca es diferente, puesto que no se busca poner a disposición los elementos para que los procesos sean replicados (o no necesariamente), sino que se busca la reutilización de los datos y etapas de procesamiento por otras estructuras, que de forma independiente, colaboren entre si.

En otro orden de ideas, se revisaron aportaciones que, en su mayoría, realizan analítica de datos proponiendo arquitecturas de microservicios. Esta arquitectura trae muchas ventajas, ofreciendo una modularidad a aplicaciones, lo cual vuelve más sencilla la aplicación de cambios, así como la posibilidad de dispersar estas aplicaciones en infraestructuras como el cloud, mejorando el tiempo de procesamiento.

MERRA Analytic Service es un caso de analítica sobre microservicios, ofreciendo el concepto de CAaaS capaz de ser aplicado a grandes repositorios con grandes volúmenes de datos, como es el caso de MERRA. Geoscience Data Analytics y OceanTea son casos similares, ofreciendo diferentes soluciones basadas en microservicios para la realización de analítica sobre repositorios de datos. GeRDI por otro lado, ofrece un panorama más abstracto, proponiendo una arquitectura más general para la analítica de datos, que a simple vista y gracias a las características de los microservicios, puede ser un flujo de trabajo.

Trabajo	**Reutilización**					**Consolidación de resultados**	
	EP		*Datos*			*Indexación*	*Funciones como servicio*
	Completas	Parciales	Iniciales	Intermedios	Finales		
Taberna [25, 34]	C ✓		C ✓		C ✓		C ✓
DfAnalyzer [52]			C ✓	E ✓ C ✓	E ✓ C ✓	E ✓ C ✓	E ✓
Galaxy [19]	C ✓		C ✓		C ✓		E ✓ C ✓
Fragflow [18]		E ✓ C ✓					C ✓
Pycomps-Decaf [66]	C ✓	C ✓	C ✓		C ✓		
Modelo transversal	E ✓ C ✓	E ✓	C ✓	E ✓ C ✓	E ✓ C ✓	E ✓ C ✓	E ✓ C ✓

Tabla 3.1: Trabajos relacionados. E representa la acción en tiempo de ejecución; C representa la acción en tiempo de configuración.

Trabajos como el realizado por Orcun Yildiz *et al.*, Fragflow, DFAnalyzer, Taverna y Galaxy son aquellos que más se relacionan a este trabajo de tesis, puesto que atacan parcialmente el problema planteado. En la Tabla 3.1, se exponen las características y diferencias entre estos trabajos y el presente trabajo de tesis. Las características contemplan la reutilización de EP completos y parciales, la reutilización de datos (iniciales, intermedios y finales) y la consolidación de los resultados producidos por una solución. Estas características son marcadas y etiquetadas con la letra E si el

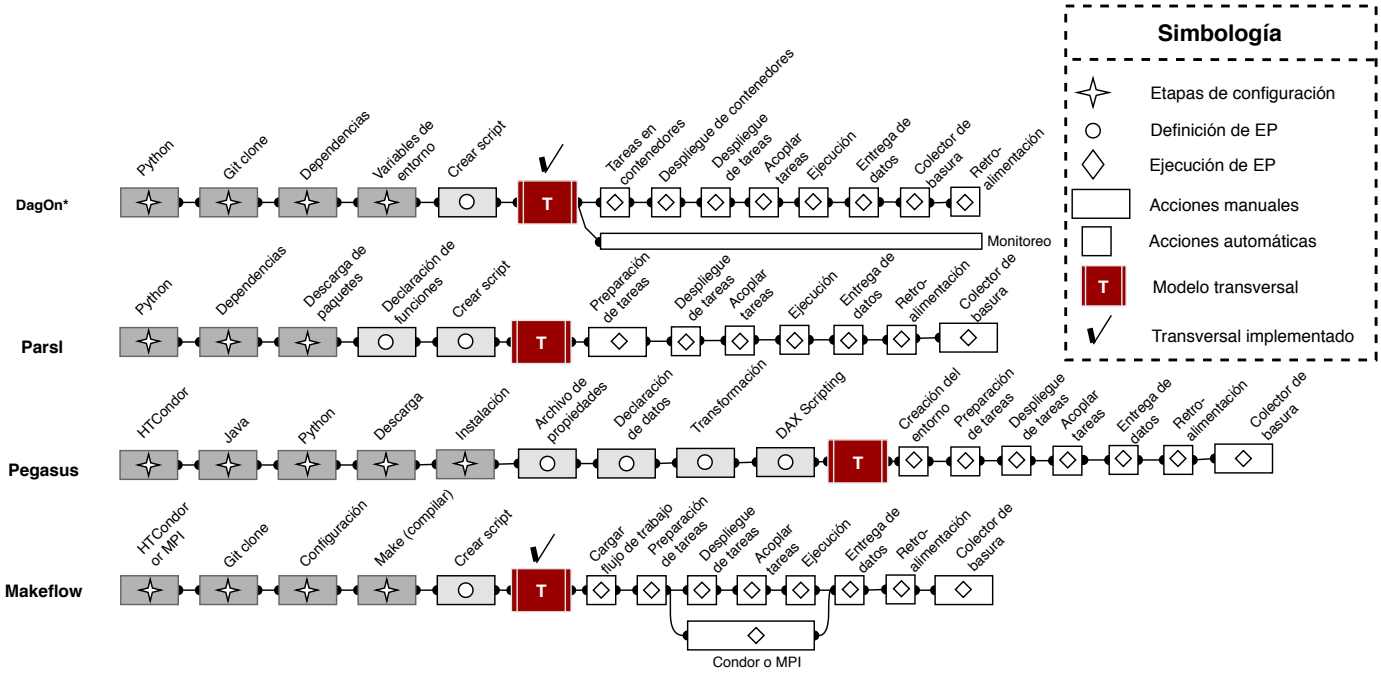


Figura 3.1: Figura extraída de [37]. Se agregó la propiedad de transversalidad.

proceso se realiza en tiempo de ejecución, y con una letra C si el proceso se realiza en tiempo de configuración.

Por otro lado, trabajos como DagOn*, Parls, Pegasus y Makeflow no atacan problema planteado para este problema de tesis, pero resultan de gran importancia para su resolución. En este contexto, estos trabajos pueden adoptar el modelo de acoplamiento transversal para generar soluciones construidas en base a múltiples EP. En la Figura 3.1 [37] se muestra una comparativa cualitativa de las etapas que sigue cada una de estas propuestas para realizar la configuración, definición, y ejecución de EP, y como una adición, se denota el punto donde el modelo transversal trabajaría en cada una de estas propuestas (siendo después de las etapas de definición y antes de las etapas de ejecución).

En resumen, los trabajo revisados de la literatura tienen diversos enfoques, cada uno ocupándose de atacar un problema diferente, pero a la vez, relacionado con lo que se busca en este tema de tesis. Si bien se comienzan a proponer conceptos tales como meta-flujos, compartición de información y

publicación de resultados, aún no consolidan esfuerzos para producir procesamiento transversal, así como propuestas que tomen ventaja sobre estos cruces transversales entre flujos de trabajo.

Estúdio	Objetivo	Objeto de estudio	Herramienta
Gestores de flujos de trabajo (motores)			
Swift [62]	Paralelizar, gestionar y desplegar flujos de trabajo	Aplicaciones	Framework
PyComps [7, 56]	Paralelizar, gestionar y desplegar flujos de trabajo	Aplicaciones y/o procesos	Framework
Parsl [6]	Paralelizar, gestionar y desplegar flujos de trabajo	Aplicaciones, procesos y/o servicios web	Biblioteca/ Framework
DagOn* [36]	Paralelizar, gestionar y desplegar flujos de trabajo	Aplicaciones, procesos, servicios web, aplicaciones externas, contenedores virtuales	Biblioteca/ Framework
Decaf [66]	Comunicación paralela entre procesos y tareas	Tareas o procesos	Middleware
Reutilización de flujos de trabajo y resultados			
Taverna [25, 34]	Ejecutar, construir y replicar flujos de trabajo	Servicios web	Framework
DfAnalyzer [52]	Análisis de datos "crudos" en flujos de datos	Fuentes y almacenes de datos	Framework
Galaxy [19]	Ejecutar, construir y replicar flujos de trabajo	Servicios	Framework
Fragflow [18]	Detecta fragmentos reutilizables de flujos de trabajo	Flujos de trabajo	Framework
Microservicios			
MERRA Analytic Service	Modelo para el análisis de datos climáticos	Microservicios	Principio Climate Analytics as a Service (CAaaS)
Geoscience Data Analytics	Analítica de datos geocientificos	Máquinas Virtuales/ Servicios.	Framework
OceanTEA	Analítica y visualización de datos climáticos	Microservicios	Framework
GeRDI	Arquitectura genérica para el análisis de datos científicos	Microservicios	Arquitectura

Tabla 3.2: Comparativa del estado del arte.

4

Diseño e implementación de un modelo de acoplamiento transversal para estructuras de procesamiento

En este capítulo se describe el proceso de diseño del modelo para la creación de soluciones de procesamiento transversales, y la implementación de este modelo en un prototipo funcional. La fase de diseño consta desde la formalización de los elementos clave, el diseño de una arquitectura para soluciones transversales y la generación de un diseño conceptual para una solución funcional. En el apartado de implementación se describe el proceso de codificación de un prototipo funcional basado en el modelo, la arquitectura y el diseño conceptual, así como una descripción de los módulos presentes dentro del prototipo.

4.1 Transversalidad

Según la Real Academia Española, el concepto de transversalidad se refiere a la cualidad de un objeto el cual se extiende atravesado de un lado a otro en dirección perpendicular, en referencia con aquello de lo que se trata. En palabras más simples, transversal se refiere a la noción de un cruce perpendicular a un objeto. Tomando este concepto, se define en este trabajo el *procesamiento transversal*, siendo este la integración de múltiples tareas y/o soluciones existentes/implementadas en nuevas soluciones de procesamiento que se exponen como un servicio (servicios de procesamiento transversal o TPS). Por tanto, un proceso de acoplamiento transversal es la unión virtual perpendicular entre un conjunto de estructuras de procesamiento (por ejemplo: flujos de trabajo o tuberías de procesamiento) a través de una entidad abstracta llamada *punto transversal*.

4.1.1 DAGs transversales

Se entiende por DAG transversal a una representación en forma de grafo acíclico dirigido donde existen un camino (conexiones) hacia otro DAG. En este sentido, un DAG transversal puede entenderse como la representación abstracta de un meta-flujo de trabajo, donde cada uno de los flujos de trabajo que lo componen está expresado como un DAG, mientras que las diversas conexiones entre estos representan la transversalidad entre los flujos de trabajo.

4.1.2 Cruza de información

Definimos el concepto de "cruza de información" como la reutilización de información por parte de un flujo de trabajo, la cual ha sido previamente procesada por alguna tarea perteneciente a otro flujo de trabajo. En este sentido, la cruza de información se refiere al cruce transversal entre flujos de datos, siendo los datos de salida de una tarea aprovechados por otra tarea en otro flujo de trabajo, sin afectar la ejecución del primero. La cruza de información se encuentra fuertemente arraigada al

concepto de reutilización, puesto que los datos se toman directamente del almacén de datos de la tarea correspondiente, evitando la re-ejecución de esta.

4.2 Diseño del modelo de acoplamiento transversal

Una EP es representada por un conjunto de etapas (V) y sus correspondientes conexiones (E) mediante la siguiente expresión:

$$EP = (V, E) \tag{4.1}$$

Donde:

- V: Representa un conjunto de etapas ($STAGE_i$).

$$V = \{STAGE_1, STAGE_2, \ldots, STAGE_v\} \tag{4.2}$$

- E: Consiste en un conjunto de pares ordenados compuestos por elementos de V, representando las interconexiones entre las etapas $E = \{(x, y) \in V \times V | x \neq y\}$.

Cada etapa ($STAGE_i$) realiza una tarea de transformación siguiendo el enfoque ETL, obteniendo datos de una fuente (ds_j) y depositando los datos transformados en un sumidero (ds_k) para su posterior uso, ya sea por el usuario final o por otra etapa. Tanto la fuente como el sumidero pertenecen al conjunto total de fuentes de datos del EP ($EPData$). En este sentido, podemos representar una etapa de la siguiente manera:

$$STAGE_i = ds_j \rightarrow Task \rightarrow ds_k | ds_j, ds_k \in EPData \wedge j \neq k \tag{4.3}$$

Donde:

- ds_j: Representa los datos de entrada los cuales sirven como insumo para la tarea de transformación.
- ds_k: Representa los datos de salida depositados en un almacén.
- $Task$: Representa una tarea, una actividad, o un proceso de transformación de datos.
- $EPData$: Es el conjunto de todas las fuentes de datos utilizadas por las etapas de una EP.

Para crear una representación de un DAG con las expresiones previas, se establece que los nodos representan a cada etapa ($STAGE_i \in V$) de una EP, mientras que las aristas representan el par ordenado correspondiente (E), el cual define la secuencia de ejecución de cada etapa. La Figura 4.1 muestra un ejemplo de DAG definido para una tubería de procesamiento EP, donde la ejecución ordenada de las tareas (A,B,C,D) de cada etapa produce diferentes versiones de datos (d_{V_x}) de una etapa a otra.

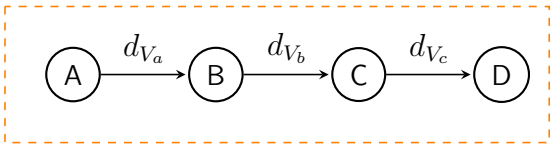


Figura 4.1: Un DAG describiendo una *tubería de procesamiento (EP)*.

El modelo considera principalmente escenarios donde un conjunto de EP sean usadas para construir una solución transversal (DAG_{TP}). Un DAG_{TP} se crea utilizando un conjunto de puntos de procesamiento transversal (TPP). Se han definido dos tipos de TPP en este modelo:

- **Puntos de acoplamiento (TCP)** . Estos puntos crean una intersección abstracta entre tareas que pertenecen a diferentes EP.
- **Puntos de extracción de datos (TEP)**. Estos puntos crean interfaces de entradas / salidas para que las soluciones extraigan / entreguen datos a las soluciones existentes (ya sea en los tiempos de configuración o de ejecución).

Un DAG_{TP} es representado de la forma:

$$DAG_{TP} = (SolT, Links) \tag{4.4}$$

Donde $SolT$ es el conjunto de EP que compone una solución transversal, mientras que $Links$ es un conjunto de TPP los cuales representan interfaces virtuales en los EP. A diferencia de los elementos en E (donde todos sus elementos son aristas), cada TPP_i puede ser un punto de acoplamiento (TCP) o punto de extracción (TEP). Esto se representa como:

$$Links = \{TPP_1, TPP_2, \ldots, TPP_m\} | m \geq 1 \tag{4.5}$$

$$SolT = \{EP_1, EP_2, \ldots, EP_n\} | \tag{4.6}$$

$$(n = 1 \wedge TPP_i \text{ es un } TEP) \vee (n \geq 2 \wedge ((TPP_i \text{ es un } TCP) \vee (TPP_i \text{ es un } TEP)))$$

Los TCP representan una interconexión entre etapas mediante una fuente de datos perteneciente al conjunto $EPData_k$ de un EP_k ($ds_j \in EPData_k$) y una fuente de datos perteneciente al conjunto $EPData_l$ de un EP_l ($ds_h \in EPData_l$). Para definir un TCP es necesario que existan al menos dos EP ($n \geq 2$). Por otro lado, los TEP apuntan a una fuente de datos ds_j en una EP_k para su extracción, las cuales se encuentran en el conjunto total de fuentes de datos ($EPData_k$). Esto es representado como:

$$TCP = (x, y) | (x, y) \in EPData_k \times EPData_l | k \neq l \tag{4.7}$$

$$TEP = ds_j \in EPData_k \tag{4.8}$$

Con base en lo anterior, un TPP_i representa la linea transversal que une dos EP diferentes (Figura 4.2), o bien, la interfaz virtual de extracción de datos de una EP. La adición de nuevas conexiones

a los DAG existentes puede causar la generación involuntaria de ciclos, para lo cual la detección de cualquiera de estas rutas cíclicas dentro del DAG_{TP} es inherente. En este sentido, existe un ciclo en el grafo DAG_{TP} si existe un camino entre vértices que empiece y termine en el mismo vértice.

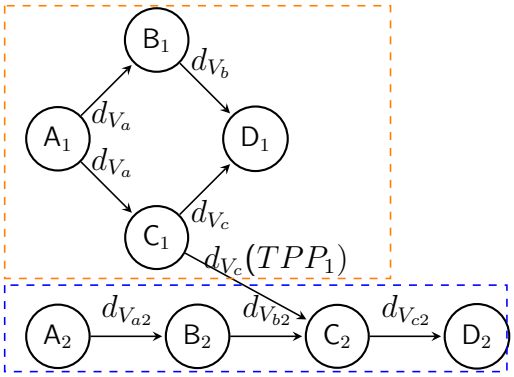


Figura 4.2: Representación de un DAG_{TP} mediante la unión de dos EP a través de TPP_1.

4.2.1 Servicios de procesamiento transversal (TPS)

El trabajo colaborativo entre diferentes EP no necesariamente implica la reutilización de etapas o información generada para ser aplicada en otra EP, sino que también puede ser utilizado por un proceso externo que provea de información útil al usuario final. Llamamos a este tipo de proceso como TPS o servicios de procesamiento transversal. Un TPS aprovecha la información generada por las EP, aplicando un proceso de transformación y generando información que puede ser útil para un usuario u otra etapa de procesamiento. Un TPS realiza la recopilación de fuentes de datos (ds_j) mediante puntos de extracción TEP. En este sentido un $TEP_i = ds_j \in EPData_k$. Un TPS puede usar hasta un total de dos TEP para realizar su procesamiento, realizando en este caso un proceso de *cruza de información* para unir las fuentes de datos extraídas. Para la unión de datos estructurados:

$$TEP_i = ds_j = (REG_j, ATT_j, KG_j) \tag{4.9}$$

Donde:

- REG_j es un conjunto de registros: $REG_j = \{reg_1, reg_2, \ldots, reg_n\}$.
- ATT_j es un conjunto de atributos: $ATT_j = \{a_1, a_2, \ldots, a_m\}$ $|a_k = (name_k, type_k, role_k)$.
    1. $name$. Nombre del atributo.
    2. $type$. Tipo de dato (int, double, char, string, etc).
    3. $role$: Rol del atributo, el cual puede asumir los valores de "value" o "keygroup".
- KG_j representa un conjunto de atributos a_k cuyo rol ($role$) asume el valor de "keygroup": $KG_j = \{a|a_k \in ATT_j, role_k = \text{“}keygroup\text{”}\}$.

Para hacer coincidir los datos de dos fuentes, es necesario crear una relación entre ellos. En el caso de los datos estructurados, se pueden generar "uniones" entre varios registros de una tabla a otra en función de las claves o los atributos de grupo (*keygroups*). Con base en lo anterior, dos puntos de extracción se consolidan en una sola fuente de datos uniendo los registros de ambas fuentes en base a los KG_j. Esto es expresado de la forma:

$$REG\tau = REG_j \cup REG_k | \forall a_i \in KG_j \exists a_h \in KG_k \wedge j \neq k \tag{4.10}$$

$$ATT\tau = ATT_j \cup ATT_k | j \neq k \tag{4.11}$$

$$KG\tau = \{a | a_i \in ATT\tau, role_i = \text{“}keygroup\text{”}\} \tag{4.12}$$

$$ds\tau = (REG\tau, ATT\tau, KG\tau) \tag{4.13}$$

Con lo anterior, podemos definir un TPS (ver ejemplo en la Figura 4.3) como una etapa de procesamiento que es externa a las EP.

Un TPS (TPS_i) sigue un proceso ETL con una entrada ($dsTPS_j$), una tarea de transformación ($Task$) y una salida ($dsTPS_k$). En este sentido, la entrada de un TPS puede ser tanto la unión de dos fuentes de datos ($ds\tau$), una fuente de datos obtenida por un punto de extracción (TEP) o la

salida de otro TPS ($dsTPS_l$). Esto es expresado como:

$$TPS_i = dsTPS_j \rightarrow Task \rightarrow dsTPS_k| \tag{4.14}$$

$$(dsTPS_j \text{ es un } ds\tau) \vee (dsTPS_j \text{ es un } TEP) \vee (dsTPS_j = dsTPS_l)$$

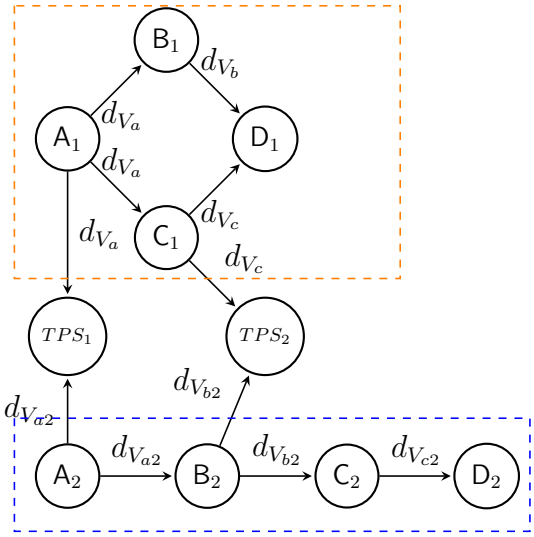


Figura 4.3: Representación de TPS en un grafo. TPS_1 y TPS_2 consolidan los resultados producidos por dos EP distintas.

Este modelo considera los siguientes componentes para crear y administrar los TPP de acoplamiento y extracción:

- **Acoplador**, crea puntos de procesamiento transversales (TPP) para acoplar tareas que pertenecen a diferentes soluciones o múltiples soluciones a través de entradas y salidas. Esta entidad crea nuevas soluciones como servicio.

- **Coordinador**, coordina la integración de los puntos de acoplamiento en nuevas soluciones (ya sea en los tiempos de configuración o de ejecución). Supervisa el surgimiento de ciclos en la solución resultante. Publica las soluciones resultantes como un servicio.

- **Extractores**, crean TPP para recuperar datos de cualquier intersección de tareas (etapas) de

una solución, posteriormente los datos son indexados y publicados como servicio.

4.2.2 Manejo de transversalidad en tiempo de ejecución

La secuencia de ejecución de las tareas puede variar según las diferentes situaciones que surjan, desde la forma de implementar la solución, las dependencias que tenga cada tarea y sus tiempos de respuesta. Para este trabajo, definimos una clasificación de tres escenarios para un TPP dentro de un DAG_{TP}:

- **Dependiente síncrono**: Cuando una tarea depende de otra que pertenece a otra solución dentro del mismo script de ejecución, por lo cual, son coordinadas como tareas en una única solución.

- **Dependiente asíncrono**: Cuando la dependencia de otra solución está en otro script de ejecución, es decir, la coordinación y ejecución de las soluciones en el DAG_{TP} son coordinadas por separado.

- **Independiente**: La ejecución de las tareas es independiente de TPP existentes. Esto se debe a que la transversalidad se da a través de un TPS, por lo cual, ninguna de las soluciones involucradas se ve afectada por la otra.

4.2.3 Soluciones como servicio

El **modelo de acoplamiento transversal** crea las siguientes soluciones de procesamiento transversal como un servicio:

- **Soluciones integradas como servicio (TPIS)**. Estos servicios representan la unión de tareas que pertenecen a diferentes EP o múltiples EP en una nueva solución DAG_{TP}.

- **Servicios de procesamiento transversal (TPS)**. Un servicio basado en ETL para consumir datos extraídos: **E**xtrae datos del punto transversal, **T**ransforma los datos en información útil y entrega/carga (**L**oad) en otro punto transversal.

4.3 Diseño del prototipo

El prototipo transversal es la implementación funcional del modelo, aplicando las características descritas para realizar procesamiento transversal sobre EP. Con base en el modelo y los trabajos encontrados en la literatura se asume que existe ya una variedad de herramientas que facilitan la ejecución y despliegue de flujos de trabajo (motores), por lo cual, el modelo propuesto busca agregar la funcionalidad de procesamiento transversal sin alterar por dentro la funcionalidad de los motores. La representación conceptual del prototipo puede apreciarse en la Figura 1.4.

El diseño del prototipo contempla a los motores de flujos de trabajo como un módulo independiente, siendo este llamado como un servicio externo, lo cual permitiría el trabajo en conjunto sin mantener una cohesión total entre ambas herramientas. Por otro lado se tiene el módulo de TPS, encargado de la gestión y control de los TPS a disposición del usuario. Basándonos en el modelo descrito anteriormente, se diseñó una arquitectura basada en pilas, en donde se describen las tres capas esenciales en un sistema de información (acceso, transformación y datos).

Como se observa en la Figura 4.4, la arquitectura se divide en 2 pilas, la arquitectura de configuración y la de ejecución. La arquitectura de configuración consiste en el proceso por el cual una solución basada en el modelo debe pasar desde el diseño de soluciones hasta la ejecución de estas en un motor de flujos procesados, mientras que la arquitectura de ejecución describe los pasos desde la ejecución hasta la obtención de resultados finales. La arquitectura de configuración describe el proceso de gestión de DAGs, el diseño realizado por parte del usuario entra desde la capa de acceso, pasando por procesos de detección de dependencias, publicación (en caso de ser una solución nueva) y/o subscripción (si se desea acceder a una solución existente), detección de posibles ciclos generados

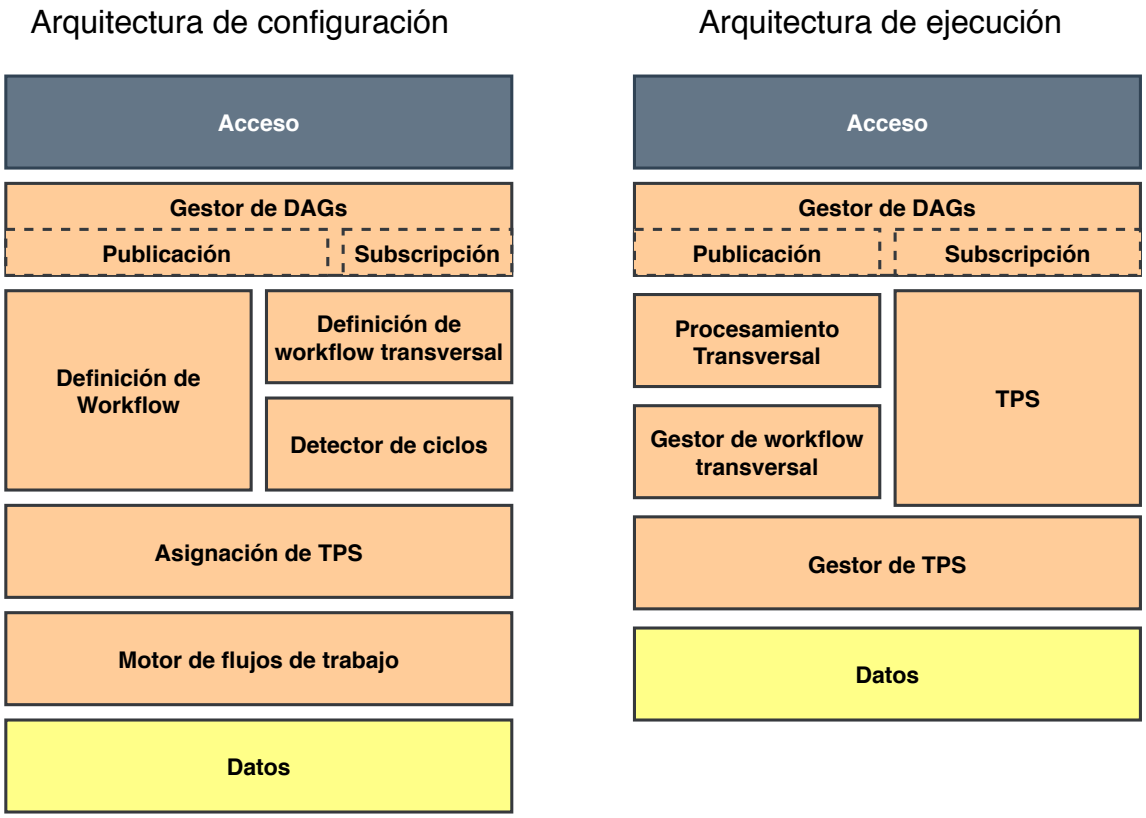


Figura 4.4: Arquitectura en forma de pila para la gestión de soluciones basadas en el modelo de acoplamiento transversal.

por las conexiones transversales y finalmente la entrega de tareas a un motor capaz de ejecutarlas. La arquitectura de ejecución describe el proceso de acceso a DAGs ejecutados, publicación de resultados o subscripción a resultados ya existentes, monitoreo de tareas ejecutadas por el motor, procesamiento de datos por parte de TPS, acoplamiento de TPS y entrega de resultados finales.

4.4 Implementación de un modelo transversal para estructuras de procesamiento

La implementación del prototipo (ver Figura 4.5) del modelo consistió en tres módulos diferentes: El módulo de almacenamiento de DAGs, encargado de proporcionar y actualizar el estado de ejecución de las EP: un módulo de transversalidad encargado de gestionar las conexiones transversales y la ejecución de tareas por parte del motor de flujos de trabajo, y un módulo gestor de TPS, encargado de extraer la información procesada por las EPs, almacenarla y aplicar los TPS requeridos por el usuario.

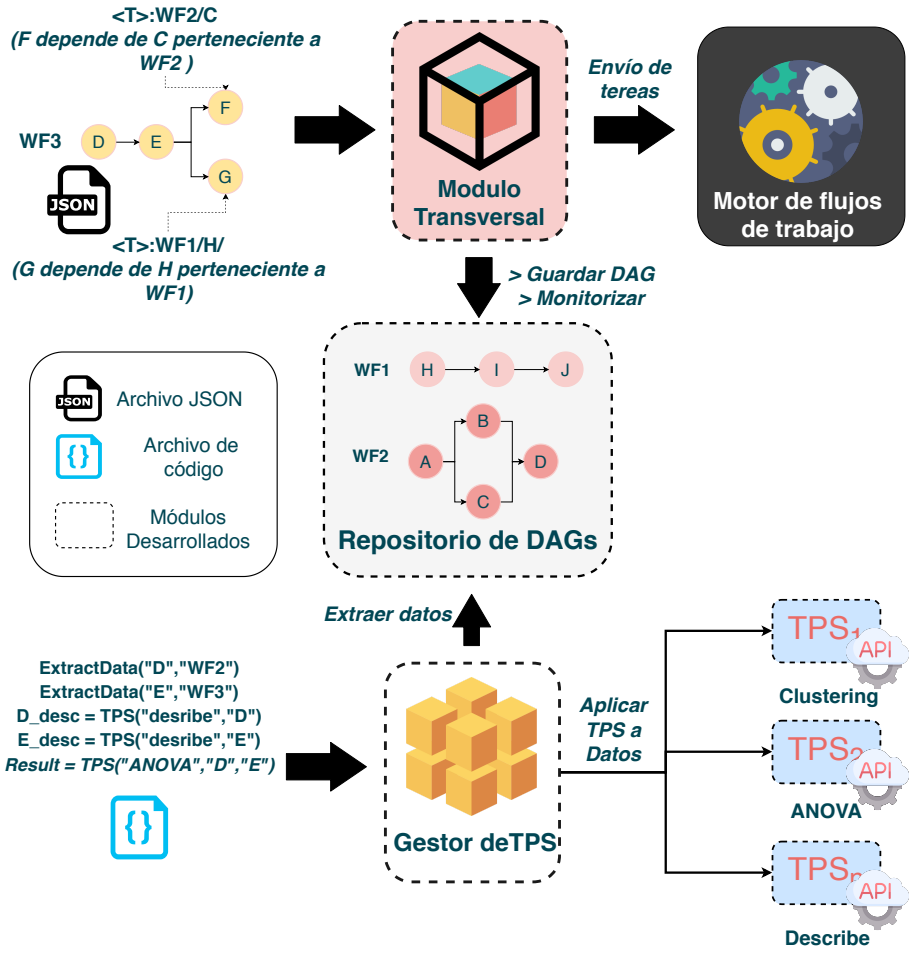


Figura 4.5: Prototipo del modelo.

4.4.1 Módulo de almacenamiento: Repositorio de DAGs

Es el encargado de almacenar los grafos que describen a las EPs ejecutadas. Cuando una solución DAG_{TP} es ejecutada por el módulo de transversalidad, el grafo es almacenado en el repositorio, y el estado de ejecución es actualizado conforme las tareas terminen su procesamiento. El repositorio de DAGs está construido como un servicio web, por lo cual, es posible acceder a él mediante peticiones REST. De esta forma y mediante una API, el módulo de transversalidad accede a la información de las EP que han sido ejecutadas o se encuentran en ejecución.

Los atributos almacenados de cada DAG son:

- **Host**: Dirección del host donde la EP fue ejecutada (y donde se encuentran los resultados de la ejecución).
- **Name**: Nombre del la EP (identificador).
- **Created_on**: Fecha de creación.
- **Tasks**: Lista de tareas que componen la EP. Cada tarea está conformada por una serie de atributos:
  - *Name*. Nombre de la tarea.
  - *Command*. Comando de ejecución (procesamiento de la tarea).
  - *Status*. Estado actual en la que se encuentra la tarea.
  - *Input*. Lista de tareas de las cuales depende la tarea actual.
  - *Output*. Lista de tareas que dependen de la tarea actual.

  Los atributos created_on, status, input y output son generados por el módulo de transversalidad, mientras que el resto deben ser proporcionados en la definición de las EPs, o bien mediante archivos de configuración.

4.4.2 Módulo de transversalidad

El módulo de transversalidad es el encargado de coordinar la ejecución de cada una de las tareas de una o múltiples EP, respetando los tiempos de ejecución y transporte de datos, ya sea por dependencias propias de la misma EP, o bien, TPP provenientes de EP externas o previamente ejecutadas.

4.4.2.1. Biblioteca importable

El módulo de transversalidad fue desarrollado utilizando el lenguaje de programación python, permitiendo al usuario importarlo a sus proyectos a manera de biblioteca utilizando el comando *import*. De esta manera, permite a los diseñadores de EPs y flujos de trabajo generar scripts de ejecución más personalizables, utilizando todas las herramientas que el propio lenguaje posee. Si embargo, si bien es posible generar *scripts* mediante python, el módulo de transversalidad funciona a partir de estructuras JSON, respetando la estructura descrita en el apartado 4.4.1.

4.4.2.2. Detección de dependencias

Para convertir un conjunto de tareas en un grafo interconectado es necesario conocer las dependencias que una tarea tiene de otras. Para ello, la lista de tareas atraviesa por un proceso de detección de dependencias, ya sea de manera local o provenientes de otras EP. La manera en que las dependencias son definidas es a través de la etiqueta reservada `<T>:`. Esta etiqueta representa la ruta virtual en la cual se encuentran los resultados producidos por una tarea. De esta manera, un usuario puede establecer que los datos que una tarea necesita para su procesamiento están en `<T>:<workflow>/<task>/`, donde `<workflow>` es el identificador del flujo de trabajo del cual proviene la tarea `<task>`, la cual produce los datos (e.g. `<T>:workflow1/TaskA/`). El módulo de transversalidad detecta automáticamente estas etiquetas, generando así un grafo con base en las conexiones (TPP) que se encuentren. Estas etiquetas son posteriormente

remplazadas por la ruta real del sistema de archivos donde los datos producidos se encuentran, por lo cual, el usuario puede especificar las rutas que necesite utilizando esta etiqueta (e.g. `<T>:workflow1/TaskA/output\_folder/output.txt`).

4.4.2.3. Validación de ciclos

Después de realizar la detección de dependencias ya es posible comenzar el procesamiento y monitoreo de tareas con el fin de ejecutarlas, sin embargo, la definición de TPP dentro de las soluciones puede traer consigo la generación de ciclos, lo cual, puede ser no soportable por el motor que se quiera utilizar (Figura 4.6).

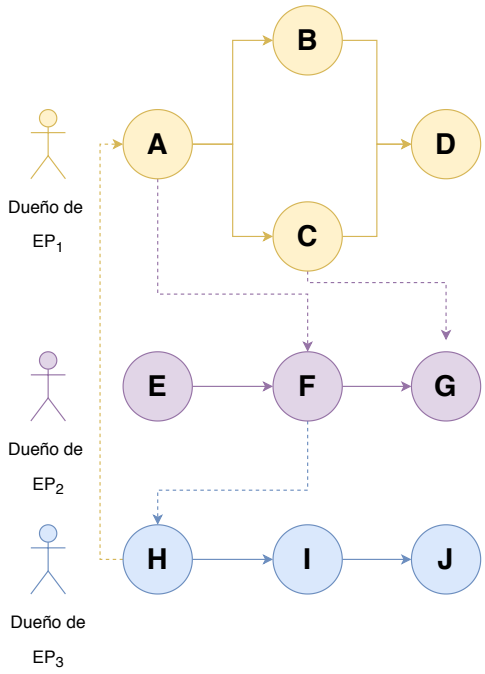


Figura 4.6: Ejemplo de ciclo causado por TPP.

Dado lo anterior, es necesaria la detección de caminos cíclicos dentro de la solución completa, con el fin de evitar ejecuciones infinitas o errores en la ejecución de tareas.

La detección de los ciclos se realiza mediante el análisis de los TPP existentes. Este proceso se realiza utilizando dos listas: publicadores y subscriptores. Si una tarea A depende de una tarea

B perteneciente a otra EP, B se considera como publicadora y A junto con todas las tareas que dependen de ella (dentro de la misma EP) se consideran subscriptoras. El algoritmo (ver Algoritmo 1) recorre únicamente aquellas tareas que presentan una dependencia hacia otra EP, es decir, que sean subscriptoras. Para cada dependencia se realizan dos validaciones, por ejemplo, para el caso de la validación de una tarea A que depende de B:

1. La tarea que A, ¿es un publicador? (es decir, que alguna otra tarea depende de ella).
2. La tarea publicadora (B), ¿esta subscrita de alguna otra tarea?.
3. ¿Existe algún indicio de ciclo?. En este caso, si otra tarea (diferente de A), detecta que B es publicadora, se considera un indicio.

En este sentido, si las sentencias 1 y 2 y 3 son verdaderas, significa que existe un ciclo en el grafo recorrido.

4.4.2.4. Validación de dependencias

Una vez que todas las dependencias de una tarea estén listas y se haya validado la existencia de ciclos, las tareas pasan a un proceso de validación de dependencias. La validación de dependencias se revisa el estado actual de cada tarea para posteriormente enviarlas a un proceso de ejecución. Las tareas pueden ser etiquetadas de cuatro maneras:

- PREPARADA. Esta etiqueta se le asigna a una tarea cuando está lista para enviarse al proceso de ejecución.
- EN-ESPERA. Esta etiqueta se le asigna a una tarea cuando aun no está lista para enviarse al proceso de ejecución.
- EN-EJECUCION. Esta etiqueta se le asigna a las tareas que se encuentran en ejecución.
- TERMINADA. Esta etiqueta se le asigna a las tareas cuando han terminado su ejecución.

Algorithm 1 Algoritmo de detección de ciclos

```
ListaEP = Lista de EP que conforman una solución
Subscriptores = Lista vacía para tareas que dependen de una tarea en otra EP.
Publicadores = Lista vacía que publican sus resultados a otras EP.
indicio = False
for all EP ∈ ListaEP do
  TareasEP = Lista de tareas de la EP
  for all tareaActual ∈ TareasEP do
    Entradas = lista de tareas de las cuales depende tareaActual
    Salidas = lista de tareas que dependen de tareaActual {(no incluye subscriptores, es decir,
    tareas en otras EP)}
    for all I ∈ Entradas do
      if I ∉ TareasEP then
        PublicadorEstaSubscrito = False
        SubscritoEsPublicador = False
        {I publica sus resultados y tareaActual esta subscrito}
        Publicadores.Agregar(I)
        if tareaActual ∈ Publicadores or salidas ∈ Publicadores then
          {TareaActual o sus salidas son también Publicadores}
          SubscritoEsPublicador = True
        end if
        if I ∈ Subscriptores then
          {si I depende de otra tarea transversal}
          PublicadorEstaSubscrito = True
        end if
        if PublicadorSubscrito and PublicadorEstaSubscrito and indicio then
          {se encontró un ciclo en la solución}
          return True
        else
          if I ∈ Subscriptores then
            {hay indicio de posible ciclo}
            indicio = True
          end if
          Subscriptores.Agregar(tareaActual)
          for all O ∈ Salidas do
            Subscriptores.Agregar(O)
          end for
        end if
      end if
    end for
  end for
end for
return False
```

Para determinar la etiqueta a asignar a cada una de las tareas, se realiza una validación basada en tres reglas:

1. Si la tarea no depende de ninguna otra, esta se marca con la etiqueta PREPARADA.
2. Si la tarea tiene una o más dependencias (depende de otra/s tareas):
   - Si todas las dependencias están marcadas con PREPARADA, la tarea se marca como PREPARADA.
   - Si al menos una de las dependencias esta marcada como EN-ESPERA, la tarea se marca como EN-ESPERA.
3. Si la tarea tiene una o más dependencias transversales (TPP):
   - Si al menos una de las dependencias transversales está marcada como PREPARADA o EN-EJECUCION, la tarea se marca como EN-ESPERA.
   - Si todas las dependencias transversales están marcadas como TERMINADA, la tarea se marca como PREPARADA.

Las reglas 2 y 3 son acumulativas, dado que una misma tarea puede tener múltiples dependencias o dependencias transversales, por lo cual la decisión de marcar a la tarea como PREPARADA depende del resultado de ambas reglas.

4.4.2.5. Segmentación del grafo y envío de tareas

Los TPP o dependencias transversales pueden causar que una solución DAG_{TP} no pueda ser enviada en su totalidad a un motor para su ejecución, puesto que en la práctica no hay manera de hacerle saber al motor que existe una tarea externa de la cual depende el flujo de trabajo, por lo cual, es indispensable la segmentación de la solución en *sub-flujos de trabajo* que puedan ser enviados para su ejecución. La segmentación se realiza con base en el estado de ejecución de cada tarea, separando

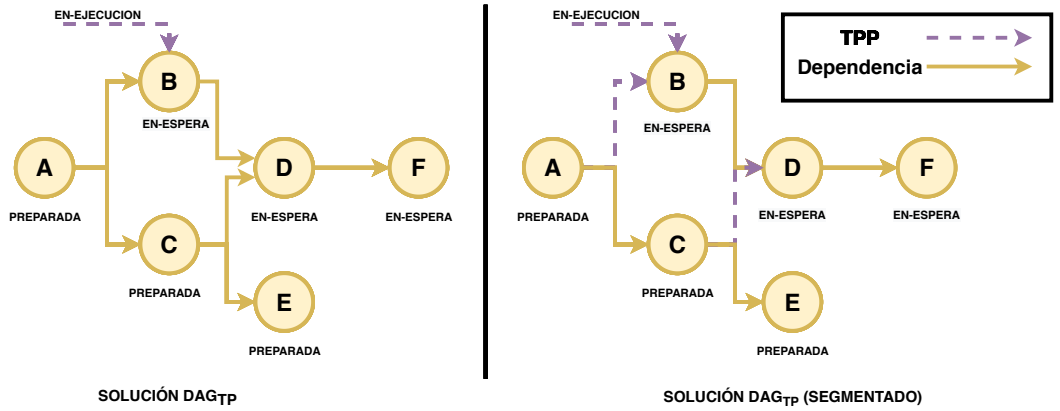


Figura 4.7: Segmentación de un DAG_{TP}.

aquellas tareas marcadas con PREPARADA. En este sentido, estas tareas marcadas se separan en un sub-grafo, donde las conexiones existentes entre las tareas PREPARADAS y EN-ESPERA dos pasan ahora a ser TPP, manejables por el módulo de transversalidad (Figura 4.7). El sub-flujo con las tareas preparadas pasan a un proceso intermediario de traducción (ver la Sección **??** para su despliegue, mientras que el sub-flujo con las tareas restantes (marcadas como EN-ESPERA) queda en un bucle de monitoreo, pasando múltiples veces por el proceso de validación de dependencias esperando a que más tareas estén listas.

4.4.2.6. Traducción y ejecución de tareas

Para que las tareas puedan ser enviadas al motor de flujos de trabajo, es necesario generar un script de ejecución que el motor pueda entender, de esta forma se traduce la notación del grafo a una que el motor pueda comprender. Esta traducción se realiza en relación con reglas definidas en un archivo de configuración llamado *Diccionario de motores* (ver apartado 4.4.4). El DAG traducido es enviado al motor de flujos de trabajo y ejecutado.

4.4.2.7. Actualización del estatus de las tareas

Mientras la ejecución de las tareas se lleva a cabo, el módulo de transversalidad es incapaz de conocer el estado actual en el que se encuentran las tareas en ejecución o han terminado su

procesamiento. En este caso, se realiza un monitoreo al motor de flujos de trabajo a través de un *escáner* y los resultados son publicados y actualizados en el repositorio de DAGs mediante una API. El escáner es un módulo desarrollado para la captura e interpretación de las líneas de salida de un motor de flujos de trabajo, es decir, captura el *output de consola* producido por el motor en tiempo real, interpretando las líneas en información que sean de utilidad para actualizar el estado actual de la solución (para más información ver la Sección 4.4.4). La información recopilada es indexada en el repositorio de DAGs mediante la API, con el fin de mantener informado a otras soluciones del estado actual de las tareas desplegadas.

4.4.3 Modulo gestor de TPS

Los TPS están diseñados para complementar las soluciones transversales, permitiendo conectar diferentes EP a través de estos microservicios, con el fin de contrastar resultados, enriquecer la información o bien, generar nuevos flujos. El gestor de TPS es el encargado de coordinar la extracción de información de las fuentes de datos, indexar la información extraída y gestionar la ejecución de los TPS, proveyendo la información requerida por el usuario.

4.4.3.1. TPS API

Mediante el uso de una API, el usuario final es capaz de controlar tanto la extracción de la información, como la ejecución de diversos TPS. La API desarrollada en python, permite al usuario realizar peticiones al módulo gestor de TPS, indicando las tareas y las EP de donde la información será extraída e indexada. Así mismo, el usuario puede definir un espacio de trabajo, con el fin de preservar la información extraída. Esto permite que múltiples usuarios puedan hacer uso de la misma información extraída, o bien, múltiples aplicaciones, permitiendo consumir los resultados generados por los TPS. Adicionalmente, el usuario puede definir los TPS que se aplicarán a los datos extraídos, ya sea de manera secuencial, simultánea o encadenada.

4.4.3.2. Extracción e indexado

Por parte del gestor de TPS, la extracción de los datos se realiza en forma paralela, mediante hilos de procesamiento. Al momento que el servicio recibe una solicitud de extracción, se despliegan N hilos de procesamiento los cuales acceden (con base a la información obtenida del repositorio de DAGs) a la fuente de los datos. La cantidad de hilos está dada por la cantidad de fuentes seleccionadas, por ejemplo, una solicitud de extracción de 10 fuentes de datos diferentes desencadena un total de 10 hilos trabajando en paralelo. Además de la extracción, cada hilo realiza un indexado de datos y metadatos (id de la fuente de datos, fecha, tamaño, etc.) en una base de datos no relacional. Este tipo de base de datos permite realizar búsquedas de manera más sencilla, además de permitir manejar los datos con una estructura clave-valor, ideal para el envío de datos a través de HTTP a los TPS. Por otro lado, en el caso de datos no estructurados (binarios), la base de datos no relacional permite que se almacenen a manera de archivo, manteniendo la misma secuencia de pasos.

4.4.3.3. Peticiones a TPS

Mediante la API de TPS un usuario es capaz de realizar peticiones a los distintos servicios disponibles. Los TPS están diseñados para trabajar en conjunto con los datos extraídos de las EPs, es decir, una vez que se realiza una petición de extracción (Sección 4.4.3.2), se puede realizar una petición a los TPS utilizando los datos extraídos como datos de entrada. Sin embargo, es posible adjuntar datos de entrada a una petición de un TPS, por lo cual, puede ser usado a pesar de no realizar extracción de datos de una EP, o bien, utilizar los resultados producidos por un TPS como datos de entrada hacia otro, realizando así un encadenamiento de servicios (Figura 4.8).

4.4.4 Opciones para desarrolladores: Agregando soporte a motores

La capacidad de realizar procesamiento transversal a diversos motores de flujos de trabajo puede agregarse a través de la implementación del modelo propuesto, o bien, utilizando el módulo

```
from TPS.Builder import Builder

# DEFINIR ESPACIO DE TRABAJO
> metaworkflow=Builder("WF-TPS-EXAMPLE")

# PETICION DE EXTRACCION
> metaworkflow.TPS_data_extraction("DS_A",path="output_INT/",
    workflow=workflow_name)
> metaworkflow.TPS_data_extraction("DS_B",path="output_INT/",
    workflow=workflow_name_2)

# CREAR QUERY PARA PETICIONES TPS
> query_1 = metaworkflow.TPSapi.format_single_query("DS_A")
> query_2 = metaworkflow.TPSapi.format_single_query("DS_B")

# PETICIONES TPS
> RESULT_DSA = metaworkflow.TPSapi.TPS(query_1,"clustering",
    options={"k":"3","variables":"T2MMAX,T2MMIN",
    "group":"Station_code","algorithm":"kmeans"})
> RESULT_DSB = metaworkflow.TPSapi.TPS(query_2,"clustering",
    options={"k":"3","variables":"T2MMAX,T2MMIN",
    "group":"Station_code","algorithm":"herarhical"})

# ENCADENAR RESULTADOS
> CHAINED_RESULT = RESULT_DSA + RESULT_DSB
> result_var = metaworkflow.TPSapi.TPS("","ANOVA",
    options={"variables":"T2MMAX,T2MMIN"},
    workload=CHAINED_RESULT)
```

Figura 4.8: Ejemplo usando la API de TPS.

desarrollado como prototipo. Con respecto al prototipo, es necesario cumplir con una serie de requerimientos con el fin de adicionar el módulo de transversalidad correctamente a un nuevo motor.

4.4.4.1. Reglas de traducción

En primera instancia, el prototipo debe de ser capaz de traducir la sintaxis utilizada para el diseño de soluciones y trasladarla a una entendible por el motor. Para lograr esto, es necesario proporcionar a partir de un conjunto de reglas, una manera de convertir una estructura JSON con un formato en específico a otro tipo de estructura con formato similar. Las reglas aplicables a cada motor son las

siguientes:

- SCHEMA: Tomando a consideración la notación <T>: (mencionada en la Sección 4.4.2.2), los motores serían incapaces de interpretarla como un comando válido, por lo cual esta es remplazada por el valor definido en SCHEMA (si se deja en blanco, todas las etiquetas <T>: pasan a ser borradas).

- STRUCTURE: En este apartado es definida la estructura (a manera de JSON o cadena de texto) en la cual el motor interpreta un flujo de trabajo, y todas sus partes, separadas en las sub-secciones WORKFLOW y TASK, correspondientes a la estructura de la EP y de cada una de las tareas respectivamente.

- OPTIONS: Se definen opciones adicionales, desde la extensión del archivo producido, hasta la definición de sentencias para el remplazo de caracteres.

Cada motor definido cuenta con sus reglas propias, dependiendo de las necesidades de transformación del archivo producido, pudiendo ser tan complejas como simples. Adicionalmente, las reglas pueden hacer uso de sentencias reservadas, con el fin de complementar los archivos producidos con información propia de la solución. La siguiente lista de sentencias reservadas se utilizan para definir en el diccionario de motores donde se colocara la información de una EP:

- <w_name>: Nombre del la EP.

- <TASK>: Lista de tareas de la EP (según la estructura definida en STRUCTURE-TASK).

- <command>: Comando de ejecución de la tarea.

- <t_name>: Nombre de la tarea.

- <INPUTS>: Lista de nombres de tareas de las cuales depende la tarea actual.

- <OUTPUTS>: Lista de nombres de tareas las cuales dependen la tarea actual.

```
"ENGINES":{
    "DAGON":{
        "SCHEMA":"workflow://<workflow_dependency>/<task_dependency>/",
        "STRUCTURE":{
            "WORKFLOW":{
                "id": 0,
                "name": "<w_name>",
                "tasks": "<TASK>"
            },
            "TASK":{
                "<t_name>": {
                    "command": "<command>",
                    "name": "<t_name>",
                    "nexts": "<INPUTS>",
                    "prevs": "<OUTPUTS>",
                    "status": "READY",
                    "type": "batch",
                    "working_dir": null
                  }
                }
            },
        "OPTIONS":{
            "REPLACEMENT":[
            ],
            "EXTENTION_FILE":"json"
        }
```

Figura 4.9: Reglas usadas para DagOn*.

En la Figura 4.9 se puede observar un ejemplo con un conjunto de reglas para DagOn*, las cuales ayudan a la creación de una estructura JSON capaz de ser interpretada por el motor. En este sentido, en las reglas de DagOn* por ejemplo, el nombre de una EP (<w_name>) se colocara en el apartado name en la estructura de WORKFLOW. De esta manera, el módulo de transversalidad es capaz de transformar un JSON con la información y notaciones necesarias para la transversalidad (Figura 4.10) a un JSON procesable por DagOn* (Figura 4.11).

```
{
    "name": "workflow-A",
    "tasks": {
      "A": {
        "command": "sleep 20;mkdir output;hostname > output/f1.txt",
        "name": "A"
      },
      "B": {
        "command": "echo $RANDOM > f2.txt; cat <T>:/A/output/f1.txt >>
           f2.txt",
        "name": "B"
    ...
```

Figura 4.10: Definición simple de una EP.

4.4.4.2. Comando de ejecución

Es indispensable proporcionar la forma en que el motor de flujos de trabajo será ejecutado. Cuando el módulo genera un script interpretable por el motor que se desea ejecutar, el módulo debe conocer el comando de ejecución para realizar este proceso. Por defecto, el módulo de transversalidad busca un archivo de ejecución llamado ENGINE_launcher.sh (donde ENGINE es el nombre del motor), en el cual deben encontrarse los comandos necesarios para la ejecución del motor requerido. En la Figura 4.12 se observa un ejemplo con los comandos necesarios para la ejecución de DagOn*.

4.4.4.3. Escáner

Cuando las tareas son enviadas al motor y ejecutadas, el modulo de transversalidad debe tener la capacidad de interpretar los mensajes producidos, con el fin de gestionar el estatus en el cual se encuentran las tareas, así como otro tipo de información (e.g. donde se encuentran los datos producidos), y que esta pueda ser indexada en el repositorio de DAGs. Existen os maneras de realizar esta tarea:

- Mediante el uso de la API del repositorio de DAGs: En este caso, se debe adaptar la API directamente al motor de flujos de trabajo, con el fin de que deposite la información necesaria

```
{"id": 0,
"name": "workflow-A",
"tasks":
    {"A":
        {"command": "sleep 20;mkdir output;hostname > output/f1.txt",
        "name": "A",
        "nexts": [B,C],
        "prevs": null,
        "status": "READY",
        "type": "batch",
        "working_dir": null},
    "B": {"command": "echo $RANDOM > f2.txt; cat workflow:///A/output/
       f1.txt >>     f2.txt",
    "name": "B",
    ...
```

Figura 4.11: Transformación a sintaxis de DagOn*.

```
# $1 is the input file path
# $2 is the workflow name

cd $WITE_BASEPATH
export dagon_dir="Engines/dagonstar" # where is installed
export PYTHONPATH=$dagon_dir:$PYTHONPATH
. $dagon_dir/dagon-venv/bin/activate # virtual envirionemnt
python "Launchers/Dagon_WITE_Launcher.py" $1 #launching the engine
```

Figura 4.12: Comandos de ejecución para DagOn*. Archivo DAGON_Launcher.sh.

al repositorio.

- Mediante el uso de un escáner: Si el motor proporciona esta información como salida de la línea de comandos (la información que genera al ejecutarse), puede ser capturada por el módulo de transversalidad e interpretada por un escáner.

Los escáners deben ser desarrollados en lenguaje python y siguiendo un estilo de programación orientada a objetos, de esta manera, un objeto scanner es instanciado dependiendo del motor que se quiera utilizar. El objeto debe contar con al menos una función llamada *Scan_output()*: Scan_output() es la función principal, obtiene como entrada una cadena de texto correspondiente a un párrafo. El párrafo puede o no contener información de utilidad para ser indexada en el

repositorio de DAGs, desde un cambio de estatus (PREPARADA, EN-ESPERA, EN-EJECUCION, TERMINADA) o la ubicación de los datos producidos (working_dir). La función debe devolver una tupla con la información (TYPE,TASK,INFO), donde TYPE corresponde al tipo de información devuelta (status o working_dir), TASK es el nombre de la tarea a la que hace referencia e INFO es el texto capturado. Siguiendo este patrón de envío de información, el módulo de transversalidad e capaz de indexar la información en el repositorio de DAGs.

5

Metodología de evaluación experimental y resultados

En este capítulo se describen los materiales utilizados para el desarrollo de este proyecto de tesis, los métodos de evaluación experimental, así como los experimentos realizados y sus resultados obtenidos. En el apartado de materiales, se describen las aplicaciones y servicios utilizados para la construcción y evaluación del prototipo. En el apartado de métodos se detalla paso a paso el proceso de evaluación, y las métricas utilizadas. Por último, se describen los resultados obtenidos de cada experimento.

5.1 Materiales, recursos computacionales y estructuras de procesamiento

Para la evaluación del prototipo de generación de soluciones transversales, se utilizaron las siguientes EP:

- **MERRA-Crawler**: Pipeline construido para la adquisición, interpolación, e indexación de datos climatológicos producidos por el proyecto de la NASA, MERRA-2 [35] (ver Figura 5.1). El pipeline consta de las siguientes etapas de procesamiento:

  1. *Adquisición (M-Acq)*. Se adquieren los datos climatológicos de alguno de los productos que el proyecto MERRA-2 pone a disposición a través de diversas URL. La lista de URLs es obtenida y posteriormente recorrida para la descarga de datos en formato de archivo binario.
  2. *Interpolación (M-Int)*. En esta etapa se realiza la extracción e interpolación de datos contenidos en conjuntos de archivos de formato binario. Esta etapa recibe como entrada dos elementos, la ruta de la carpeta contenedora de archivos y la ruta de un archivo que contenga los puntos geográficos (latitud, longitud) a interpolar.
  3. *Formateo e indexado (M-F&I)*: En esta etapa se realiza un proceso de conversión de temperaturas al formato Celsius, y posteriormente se indexan los resultados en una base de datos.

- **Flujo de trabajo Landsat8**. Flujo de trabajo para la generación de productos derivados de imágenes de satélite (ver Figura 5.2) obtenidos por Landsat8[1]. Realiza la aplicación de filtros y correcciones a un conjunto de imágenes de satélite. El flujo de trabajo se compone principalmente de tres tareas:

[1] https://glovis.usgs.gov/app.

Figura 5.1: Estructura de MERRA-Crawler.

1. *Descompresión*: descomprime archivos "tar" que contienen imágenes y metadatos.
2. *Correcciones*: Obtiene un conjunto de correcciones (por ejemplo, radiométricas) e índices espectrales derivados de la reflectancia de la superficie, tales como: índice de vegetación de diferencia normalizada (NDVI), índice de vegetación mejorado (EVI), índice de vegetación ajustado al suelo (SAVI), índice de vegetación ajustado al suelo modificado (MSAVI), índice de humedad de diferencia normalizada (NDMI), relación de quemadura normalizada (NBR), y relación de quemadura normalizada dos (NBR2).
3. *Metadatos*: Obtiene información de los metadatos para conocer a la ubicación geográfica de la imagen.
4. *TIFF2JPG*: Realiza un cambio de formato de imagen de TIFF a JPG, reduciendo a su vez la resolución y el tamaño del archivo.

- **Tuberías de procesamiento RAMA & REDMET**: Tuberías para la adquisición de datos de las fuentes RAMA[2] [49] (Red Automática de Monitoreo Atmosférico) y REDMET[2] [50] (Red Meteorológica y Radiación Solar). RAMA contiene bases de datos anuales con información

[2]http://www.aire.cdmx.gob.mx/default.php?opc=%27aKBi%27.

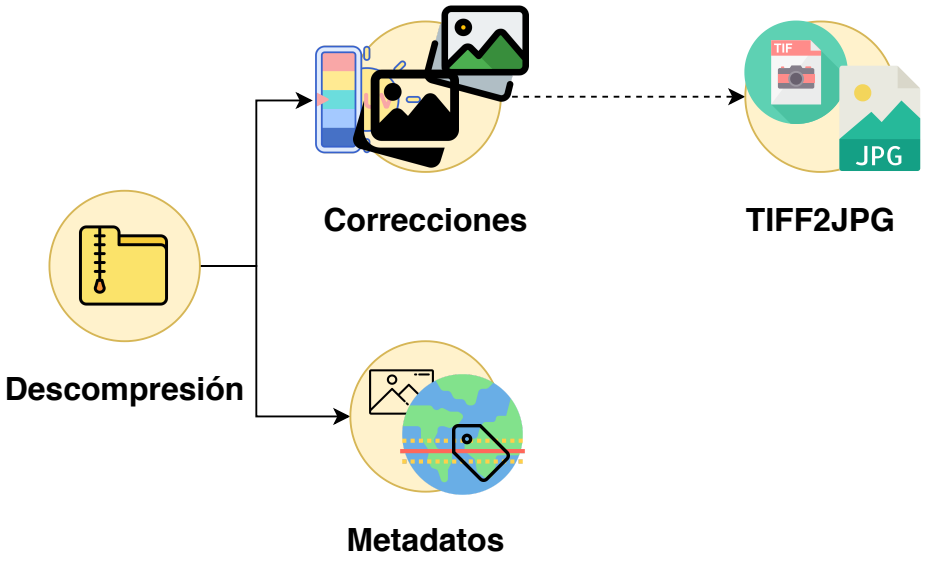


Figura 5.2: Estructura original del flujo de trabajo Landsat8.

sobre las concentraciones de contaminantes que se han registrado cada hora desde 1986. REDMET contiene información sobre parámetros meteorológicos que se han registrado cada hora desde 1986. La estructura de los tuberías se muestra en la Figura 5.3.

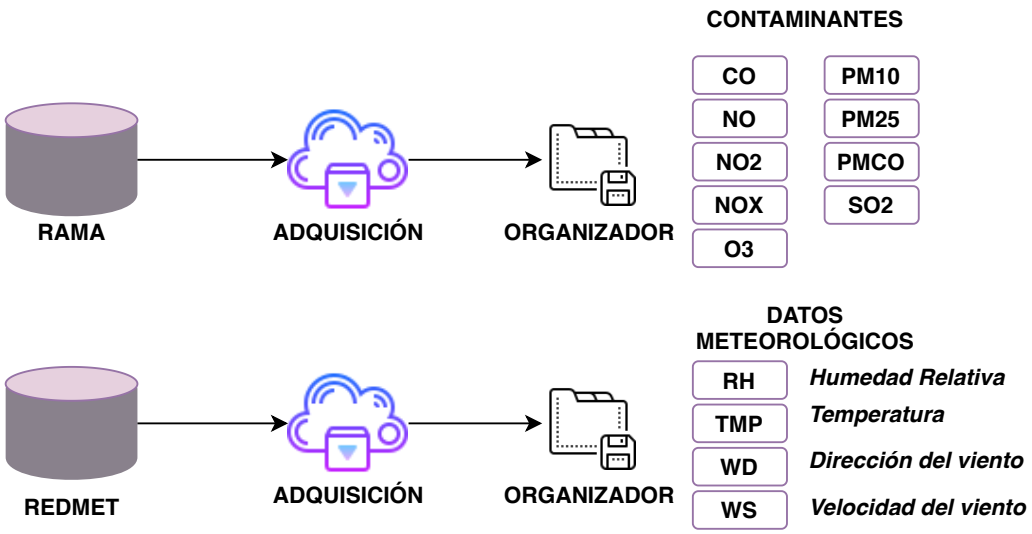


Figura 5.3: Tuberías de adquisición RAMA y REDMET.

5.1.1 TPS

Como parte de la implementación del prototipo, se realizó la implementación de un conjunto de servicios genéricos utilizando el enfoque de microservicios. A continuación se describe la lista de los TPS disponibles y utilizados para los diversos experimentos:

- **Gestor TPS:** Servicio en cargado de la gestión, indexación y ejecución del resto de servicios. Proporciona herramientas para el filtrado de datos, transformación y acceso.
- **Clustering:** Ofrece algoritmos de agrupamiento no supervisado.
- **Describe:** Ofrece servicios de analística básica para conjuntos de datos.
- **ANOVA:** Servicio de análisis de correlaciones, varianzas y covarianzas.
- **Cleaning Tools:** Servicio para la limpieza de datos, ofrece herramientas para la eliminación de datos atípicos, omisión, remplazo de datos (mediante interpolación, media, mediana o moda).
- **Cluster Validation:** Servicio para el análisis de clusters a partir de índices (dunn, silhouette, tau, gamma, Davies Bouldin, etc.).
- **Transform:** Ofrece herramientas para la reestructuración y agrupamiento de datos.
- **Graphics:** Servicio para la generación de gráficos estadísticos en dos y tres dimensiones (Linear, histograma, puntos, clusters, etc).

5.1.2 Infraestructura

Para la ejecución de los experimentos se utilizó una infraestructura compuesta por seis equipos de cómputo, los cuales cuentan con procesadores de 12 núcleos y 64 GB de RAM, así como la plataforma de contenedores Docker instalada para el despliegue de contenedores virtuales. Las características de cada uno de los recursos de cómputo se muestran en la Tabla 5.1.

Hostname	Socket	Núcleos /Socket	Hilos/ núcleo	Ram (GB)	Disco (GB)	Asignación
compute7	1	6	1	24	465, 931, 931	Docker CE
compute8	2	8	1	64	931, 931, 931, 931	Docker CE
compute9	1	12	1	64	930, 27	Asignados a la nube. Cuenta con Docker CE
compute10	1	12	1	64	2764	Asignados a la nube. Cuenta con Docker CE
compute11	1	12	1	64	2764 2764	Asignados a la nube. Cuenta con Docker CE
compute12	1	12	1	64	2764 2764	Asignados a la nube. Cuenta con Docker CE

Tabla 5.1: Recursos de cómputo disponibles.

5.2 Metodología de evaluación

La evaluación del modelo se realizó siguiendo una metodología de dos etapas, la primera etapa se refiere a la medición cuantitativa basada en prototipos y la segunda a una valoración cualitativa del modelo.

Cuantitativa. La evaluación cuantitativa se realizó en dos fases:

- *Evaluación controlada*. Se determinó un conjunto de experimentos con base en los aspectos críticos que podrían afectar el desempeño del modelo implementado. Se identificaron como aspectos críticos: la cantidad de etapas en un flujo de trabajo, el tamaño de las fuentes de datos que procesará un flujo de trabajo, cantidad de puntos transversales, el tiempo de construcción del DAG_{TP} en base a la cantidad de EP, tiempo de acoplamiento de los TPS con los puntos transversales.

- *Estudio de caso*. Se tomaron los resultados de la evaluación controlada y se validó la solución

mediante un estudio de caso, enfocándose en las características que más resaltaron en la evaluación controlada. Se realizaron tres estudios de caso tomando los EP de MERRA, Landsat8, REDMET y RAMA.

Cualitativa. Esta evaluación se realizó sobre las propiedades del modelo propuesto con respecto a las soluciones encontradas en el estado del arte. Se realizó una comparativa con respecto a los trabajos que atacan parcialmente el problema planteado. Adicionalmente, se adapto el modelo transversal a una comparativa cualitativa entre motores de flujos de trabajo encontrados en la literatura. Ambas comparativas cualitativas se pueden encontrar en el apartado de discusión de la Sección 3.

5.2.1 Métricas para la evaluación

Las métricas utilizadas para la evaluación del prototipo se describen a continuación:

- Tiempo de construcción (T_B): Corresponde al tiempo que toma la construcción de un DAG_{TP}. El proceso de construcción es tomado en cuenta desde la declaración de cada uno de los elementos en la aplicación y la detección de dependencias.

- Tiempo de validación (T_V): Es el tiempo que toma la validación y detección de ciclos dentro del DAG_{TP}, ya sea desde dependencias propias dentro del mismo grafo o TPPs.

- Tiempo de espera (T_I). Se refiere al tiempo que debe esperar una EP antes de ser ejecutada. Este tiempo se puede dar cuando una tarea tiene un TPP. El tiempo de espera también abarca los tiempos de descarga de datos provenientes desde otro nodo, es decir, el transporte los insumos necesarios para que una tarea pueda ser ejecutada.

- Tiempo de ejecución (T_E): El tiempo de ejecución de un flujo de trabajo está dado por el tiempo que toma la ejecución de todas las tareas de una EP, sin tomar los procesos previos a la ejecución, como lo son el tiempo de construcción y el tiempo de validación (pero si toma los tiempos de espera).

- Tiempo de respuesta de una solución (T_R): Se entiende por tiempo de respuesta al tiempo que transcurre desde la ejecución de la solución, hasta que el usuario recibe los resultados. El tiempo de respuesta de un DAG_{TP} está dado por:

$$T_R = T_B + T_V + T_E + T_I \tag{5.1}$$

- Tiempos de TPS: El tiempo de un TPS se compone de dos elementos:

  1. Tiempo de extracción (T_{EXT}): Refiriéndose al tiempo de extracción de los datos desde las fuentes señaladas y el proceso de unión de datos (en caso de ser dos fuentes).
  2. Tiempo de Procesamiento (T_{TPS}): Se refiere al tiempo de transformación de los datos a través del TPS.

5.3 Escenario 1: Evaluación controlada

En esta sección se describen las pruebas realizadas con un prototipo del modelo de acoplamiento transversal, el cual permite validar los aspectos críticos del modelo, así como rectificar su funcionalidad.

El objetivo principal de los experimentos realizados en este escenario consisten en evaluar la eficiencia del modelo mediante pruebas de rendimiento.

Se tomó como objeto de estudio MERRA-Crawler, realizando diferentes soluciones de DAG_{TP} a través de dependencias transversales entre múltiples instancias de una misma EP. Para este escenario se realizaron cuatro experimentos:

1. Aumentando la cantidad de TPS (conexión transversal independiente).
2. Aumentando la cantidad de EP con conexiones dependientes síncronas.
3. Aumentando la cantidad de EP con conexiones dependientes asíncronas.

4. Aumentando la cantidad de datos de entrada.

5.3.1 Experimento 1: Aumentando la cantidad de TPS

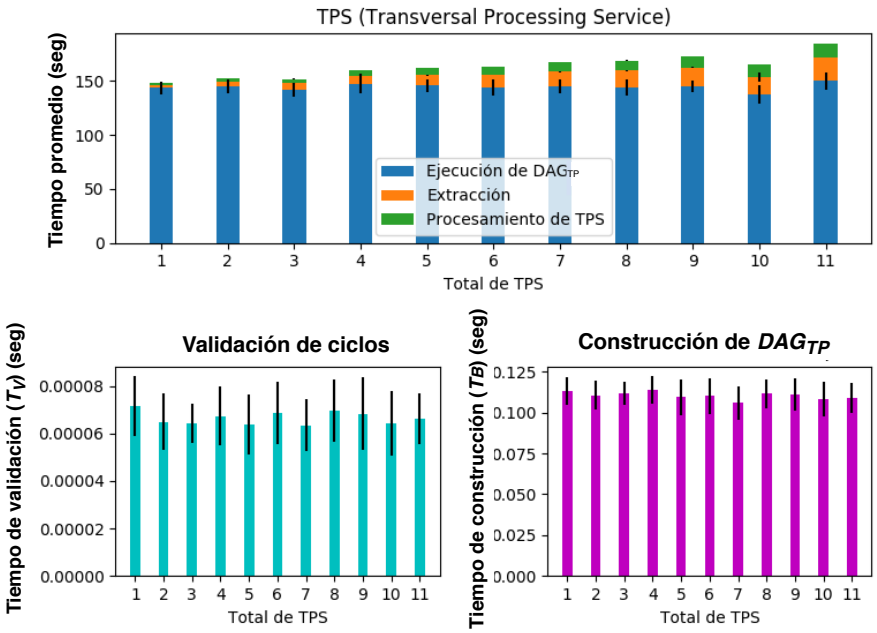


Figura 5.4: Impacto de TPS en los flujos de trabajo.

La realización de este experimento fue bajo las siguientes condiciones:

- Se generó un script de meta-flujo construido basándose en dos instancias de MERRA-Crawler.
- La ejecución de las tareas se realizó en un mismo recurso de cómputo (compute11).
- Se utilizaron dos instancias de la misma EP conectados a través de TPS. Ambas instancias tienen los mismos datos de entrada.

Los resultados obtenidos muestran que no existe un aumento en los tiempos de los flujos de trabajo con respecto al aumento aumento de TPS. La Figura 5.4 muestra el aumento en los tiempos de servicio (eje vertical) que producen los TPS que fueron evaluados en cada experimento (eje

horizontal). Como se puede observar, el tiempo de procesamiento y extracción de los TPS incrementa en medida que la cantidad de TPS aumentan, sin embargo, esto es debido a que el procesamiento de estos TPS va de forma secuencial, es decir, uno de tras de otro. En el caso de los procesos de **validación de servicios** y **construcción de** DAG_{TPS}, su costo no se ve afectado significativamente por el número de TPS, lo cual implica que estos procesos no representan la generación de una posible sobrecarga para las EP creadas por el modelo transversal propuesto.

5.3.2 Experimento 2: Aumentando la cantidad de EP con conexiones dependientes síncronas

Con el fin de conocer el impacto que tienen las conexiones transversales entre EP, se realizó un experimento variando la cantidad de dependencias transversales, y adicionalmente, la instancias. Para este experimento se aumentaron las instancias de MERRA-Crawler. Por cada nueva instancia del la EP, se agregaron i dependencias que representan el número de instancias antecesoras, es decir, la i-esima instancia de la solución generada contó con i-1 instancias. La dependencia, en todos los casos, va desde la tarea *M-Acq* de los flujos de trabajo anteriores, hacia las tareas *M-Int* sucesoras. En la Figura 5.5 se observa de manera gráfica la estructura de este experimento, donde, por ejemplo, la instancia $WF1$ al no tener instancias antecesoras, no cuenta con ninguna dependencia, mientras que la instancia $WF3$, tiene dos dependencias provenientes de $WF1$ y $WF2$. Las diferentes instancias son definidas, construidas y ejecutadas dentro de un mismo script de ejecución, formando una solución DAG_{TP} construida a partir de todas las instancias, por lo cual toda la solución es ejecutada como un solo flujo de trabajo.

Así mismo, la carga de trabajo para las tareas *M-Acq* fue repartida entre el número de instancias. Para este experimento se descargaron 31 archivos (correspondientes al mes de enero del 2019, el cual cuenta con 31 días) y se dividió la carga entre el total de instancias. Por lo cual, por ejemplo, en el experimento con dos instancias, a cada tarea de *M-Acq* se le asignó procesar 15 y 16 archivos

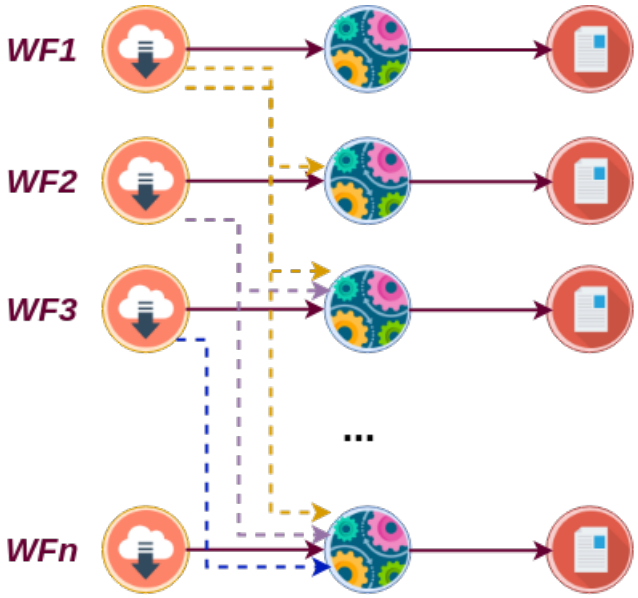


Figura 5.5: Estructura DAG_{TP} aumentando el número de instancias de MERRA-Crawler.

respectivamente.

En la Figura 5.6 se muestran los tiempos de ejecución, construcción y validación de soluciones en segundos (eje vertical) para un total de instancias que van desde la 1 a la 12 (eje horizontal). Al aumentar el número de instancias a la solución DAG_{TP}, T_E disminuye en forma exponencial. Esto es debido a la división de trabajo entre las etapas de *M-Acq*, puesto que en lugar de descargar 31 archivos de forma secuencial, se procesan $\frac{31}{n}$ de forma paralela, donde n es el número total de instancias.

Por otro lado, se observa un crecimiento lineal en T_B, y un aparente crecimiento exponencial en T_V. Para corroborar esta observación, se amplió la experimentación midiendo únicamente T_B y T_V para soluciones desde 1 a 100 instancias a intervalos de 10. Los resultados se muestran en la Figura 5.7, donde se puede observar de manera más clara el comportamiento exponencial en T_V al aumentar el número de instancias (eje horizontal), así como el comportamiento lineal presentado en T_B.

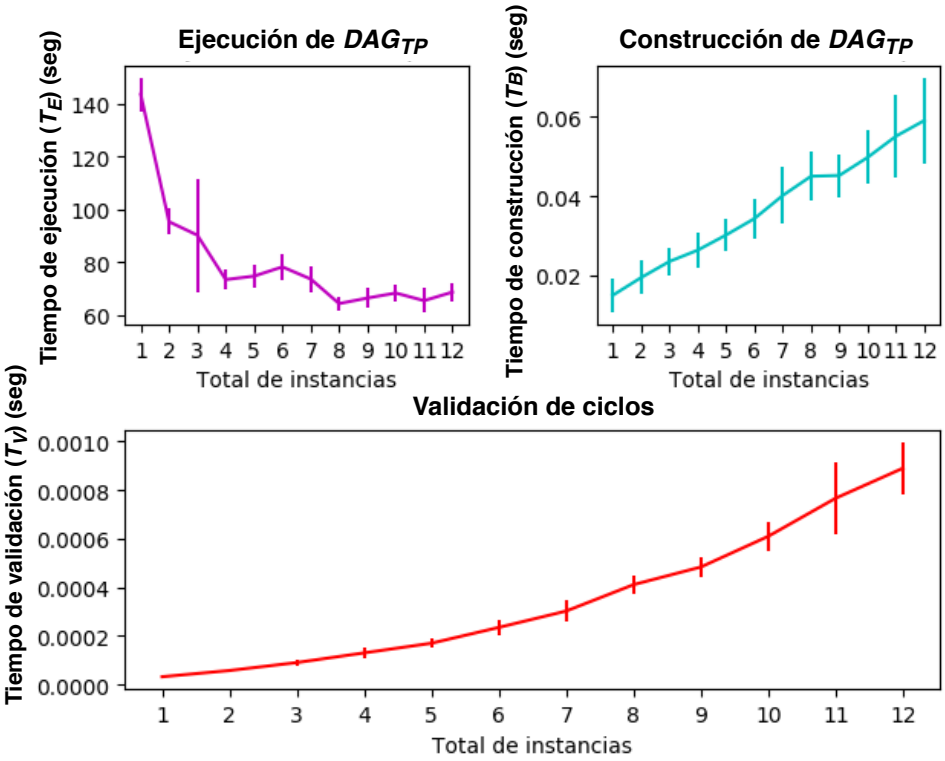


Figura 5.6: Tiempos de ejecución, construcción, y validación del experimento 2.

5.3.3 Experimento 3: Conexiones dependientes asíncronas

Para este experimento fue utilizada la misma estructura de flujos de trabajo que en el experimento número dos, con la diferencia que cada instancia es ejecutada en scripts por separado dentro del mismo computador. La coordinación de dependencias es realizada a través del monitor de registro, el cual registra el momento en el que cada actividad termina su procesamiento. Dado que los flujos tienen dependencias provenientes de otros, estos deben esperar a que terminen las tareas de las que dependen. Para fines de este experimento, cada script es ejecutado uno tras otro con un segundo de diferencia y en orden ascendente de la cantidad de dependencias. Esto último es debido a que, si la instancia no detecta la dependencia transversal registrada en el servicio de monitores, esta es ignorada. De esta manera, un segundo de diferencia asegura que cada flujo de trabajo sea registrado

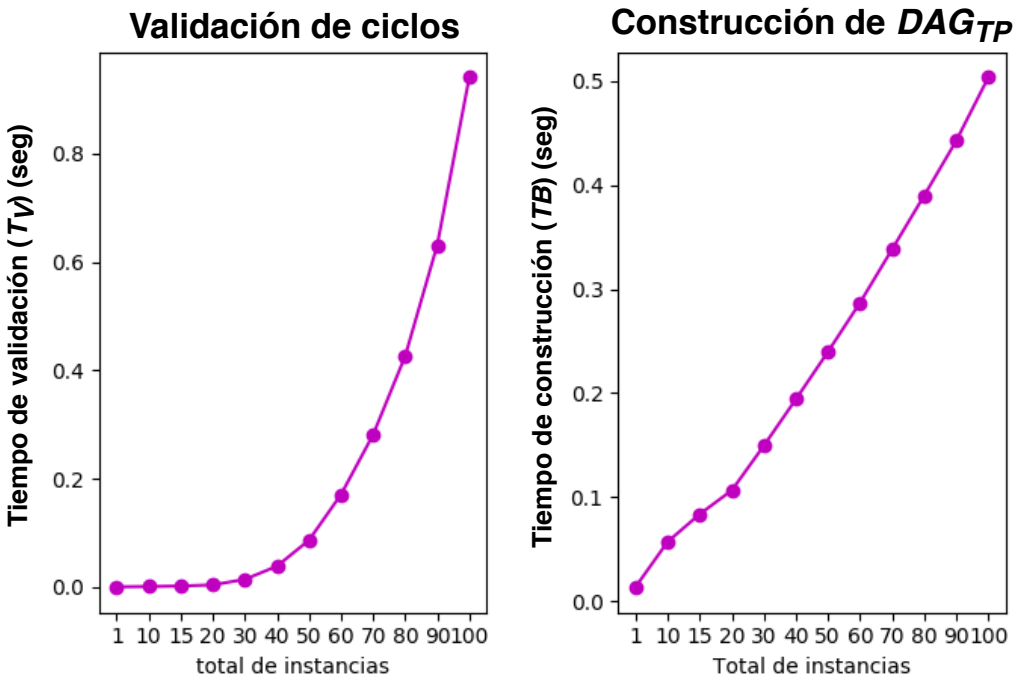


Figura 5.7: Tiempos de construcción, y validación extendidos.

en el servicio de monitores antes de ser requerida por otra instancia.

Los resultados de la experimentación mostraron diversos comportamientos entre los flujos de trabajo. En la Figura 5.8 se muestran los tiempos de ejecución, construcción y validación para la primera y última instancia (eje vertical) en una serie de experimentos que van de un total de dos instancias a doce (eje horizontal). Primeramente, se encontró que el primer flujo de trabajo ve afectado sus tiempos T_B y T_V. Bajo estas condiciones, cada flujo de trabajo se encuentra separado de los demás, por tanto, el primer flujo no depende de ningún otro flujo de trabajo en ningún caso, dando como resultado siempre el mismo comportamiento. Ocurre el caso contrario en el último flujo de trabajo, donde se observa un claro incremento en T_B y T_V; debido a que, en todos los casos vistos, este flujo de trabajo cuenta con n-1 dependencias.

Por otro lado, para ambas instancias muestra un decremento de T_E. Para el primer flujo de trabajo, el tiempo decrece de forma exponencial a causa de la división de trabajo, por lo cual, entre mayor cantidad de flujos de trabajo, a estos le corresponde menos trabajo a procesar. El último flujo

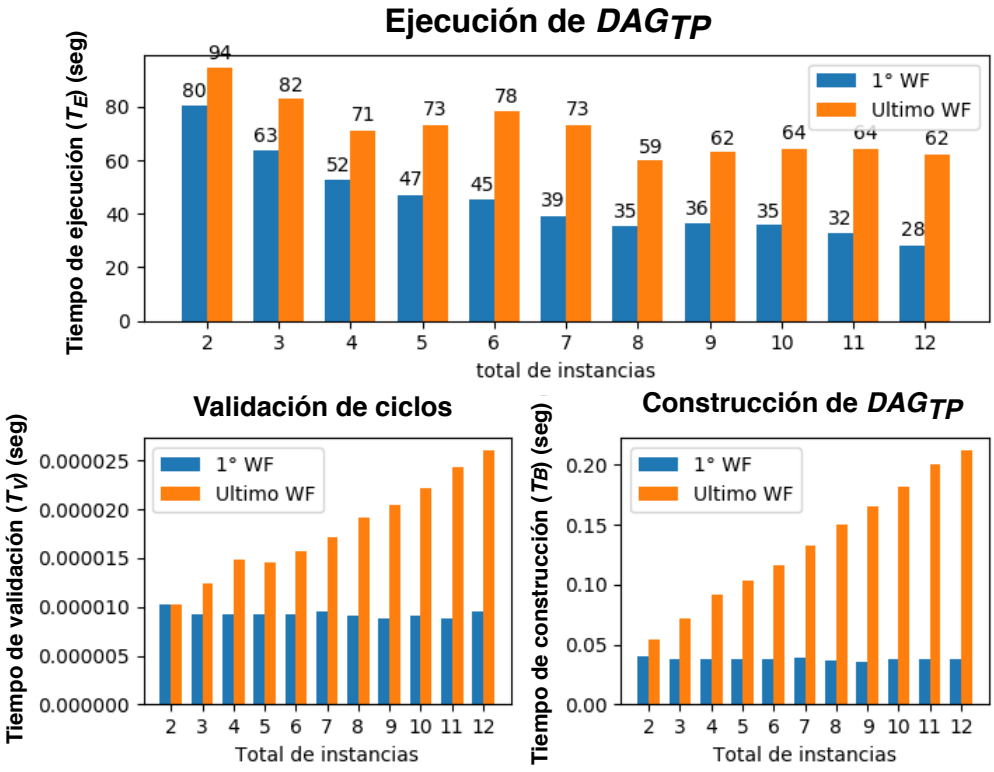


Figura 5.8: Tiempos de ejecución, construcción, y validación del experimento 3.

de trabajo por otro lado, mostró un decremento en los tiempos pero en una proporción diferente. Este decremento es explicado por dos motivos: El primer motivo es el total de dependencias con las que se cuenta, dado que si bien el procesamiento de datos es dividido entre el total de instancias, a la tarea *M-Int* del último flujo de trabajo llegan todos los datos procesados por las tareas *M-Acq* de las instancias anteriores, por lo cual, procesa el total de los datos. La segunda razón es debido al tiempo de ejecución, puesto que existe un tiempo de espera entre las dependencias transversales antes de continuar con su procesamiento.

5.3.4 Experimento 4: Aumentando la cantidad de datos de entrada

El propósito de este experimento es medir el impacto de aumentar la cantidad de datos a procesar por una solución DAG_{TP}. De la mano con esto, se busca medir el impacto que tiene la reutilización de

datos previamente producidos por otro flujo de trabajo. Para lograr esto, se estableció una instancia ($WF1$) independiente con una carga de trabajo de 31, 90 y 181 días de descarga. Posteriormente, se ejecutó un flujo de trabajo ($WF2$) compuesto por únicamente las tareas de *M-Int* y *M-F&I*, donde *M-Int* tiene una dependencia transversal desde *M-Acq* de $WF1$. La ejecución de $WF2$ no tiene tiempo de espera, pues se asume que el procesamiento de $WF1$ ha terminado, por lo cual, se toman solamente T_E sin T_I. El experimento se llevó a cabo en dos escenarios: En el primer escenario, WF2 es ejecutado en la misma infraestructura que $WF1$, por lo cual no hay un transporte de datos a través la red. Mientras que, para el segundo escenario, $WF2$ es ejecutado en un computador diferente al de $WF1$ dentro de la misma red local, para lo cual es utilizado el servicio FTP y la correspondiente API para el transporte de datos.

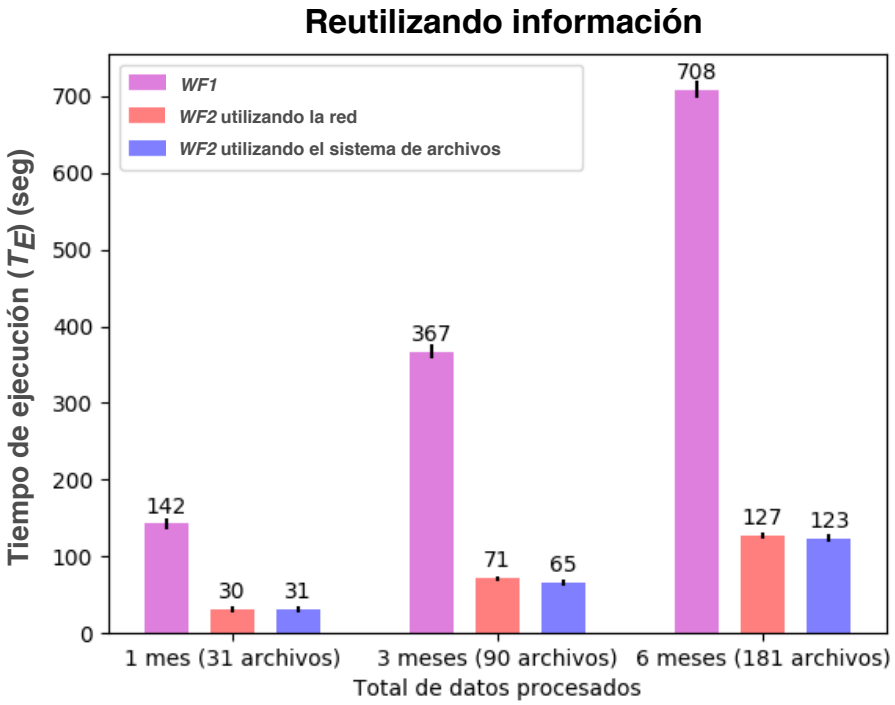


Figura 5.9: Tiempos al reutilizar información procesada.

Los resultados del experimento muestran el impacto que tiene la tarea de *M-Acq* con respecto al resto. Los resultados de muestran en la Figura 5.9, donde, en el eje vertical se muestran los tiempos

de ejecución en segundos de la solución tradicional (sin utilizar el modelo transversal ($WF1$), es decir, como un único flujo de trabajo), con el modelo transversal distribuido (usando la red) y con transversal en un solo recurso de cómputo (usando el sistema de archivos), para el procesamiento de 31 90, y 181 archivos (eje horizontal). La primera observación de los resultados es el impacto generado por la tarea *M-Acq* con respecto al resto de tareas, siendo esta la que aumenta en mayor medida el tiempo de las soluciones. Los tiempos obtenidos por $WF2$ no toman en cuenta T_I de *WF1*, lo cual se refleja solamente en el tiempo de las tareas *M-Int* y *M-F&I* de $WF2$, ahorrando alrededor de un $80\,\%$ en el tiempo, correspondiente a la tarea de *M-Acq*.

5.4 Escenario 2: Evaluación del modelo aplicado a diferentes motores de flujos de trabajo

En esta sección se describen las pruebas realizadas al prototipo del modelo utilizando diferentes motores de flujos de trabajo, y aplicando TPS a los flujos generados.

El objetivo principal del conjunto de experimentos es demostrar que el modelo puede ser adaptable a los motores de flujos de trabajo encontrados en la literatura, evaluando así la factibilidad del modelo.

Para la realización de los experimentos se utilizó como caso de prueba la tubería de MERRA, generando una solución a partir de múltiples instancias.

Se implementaron 33 instancias de MERRA. Una de las instancias realiza la adquisición de datos meteorológicos correspondientes a siete días (siete archivos), posteriormente, se realiza un proceso de interpolación (M-Int) en un total de 302,099 puntos geográficos, correspondientes a la cantidad de localidades total de México. Finalmente, los resultados son procesados por la etapa de *M-F&I*. La lista de puntos a procesar es generada por una tarea adicional llamada *M-Split*. *M-Split* toma como entrada un documento proporcionado por el INEGI[3] del cual se extrae un identificador y coordenadas geográficas de cada localidad de México, generando así la lista de localidades. Adicionalmente, se

[3]https://www.inegi.org.mx/app/ageeml/

crean sublistas por cada uno de los estados de la república, es decir, se generan 32 sublistas que contienen las respectivas localidades por estado.

Como parte del experimento, se desplegaron las 32 instancias restantes con dependencias transversales de las tareas *M-Acq* y *M-Split* de la instancia máster (aquella que procesa todas las localidades). Cada instancia realiza únicamente el proceso de interpolación tomando la lista correspondiente a uno de los 32 estados de la república (generados por *M-Split*). En resumen, se desplegaron 32 instancias donde cada una procesa la cantidad correspondiente de localidades a un estado de la república (Figura 5.10).

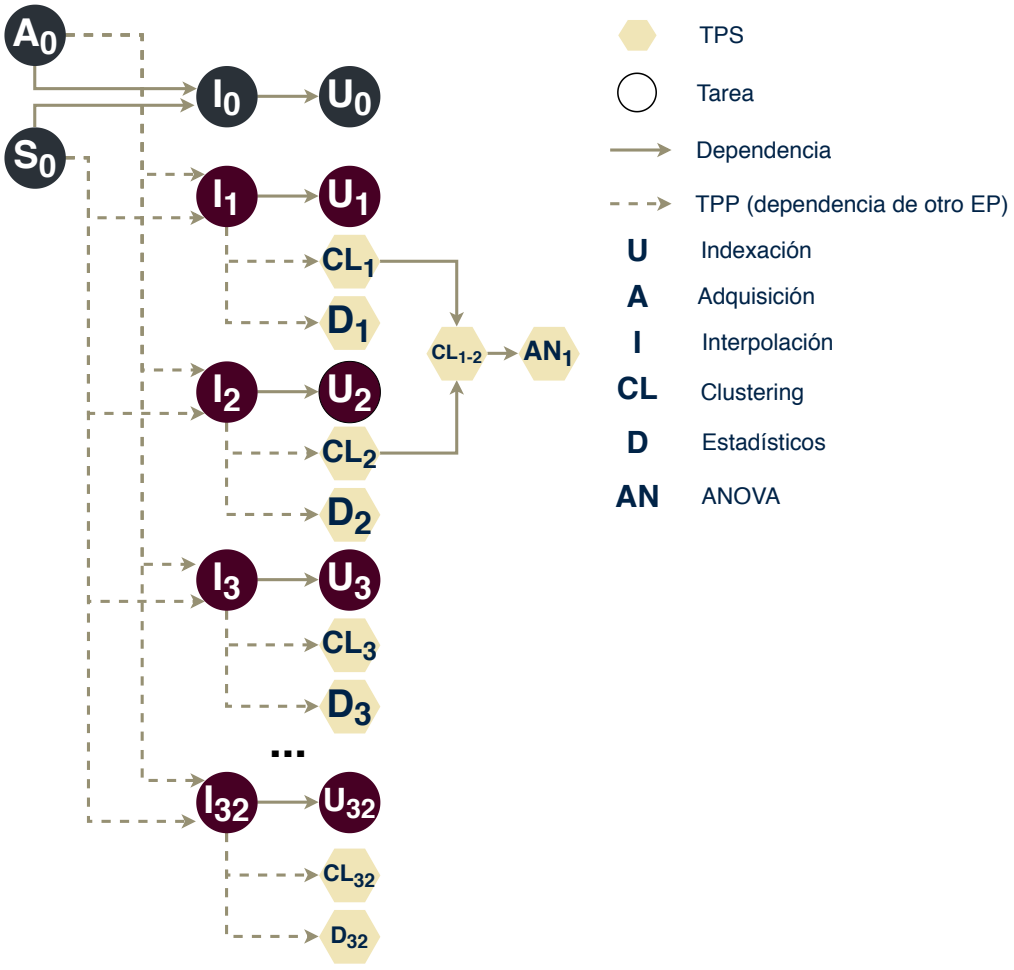


Figura 5.10: Instancias de MERRA procesando localidades de México, correspondientes a los estados de la república.

Adicionalmente, a cada instancia se le aplicaron dos TPS, generación de una descriptiva estadística y la aplicación del algoritmo de clustering K-means con tres grupos. Esto con el propósito de corroborar la funcionalidad y factibilidad de los TPS.

Los experimentos fueron ejecutados utilizando dos motores diferentes: DagOn* y Makeflow.

5.4.1 Experimento 1: Tiempo de procesamiento con una sola instancia

Se obtuvo el tiempo de procesamiento utilizando ambos motores y procesando todo el conjunto de puntos utilizando una única instancia. Para este experimento se realizaron 32 repeticiones en la ejecución de la solución tanto para DagOn* como para Makeflow.

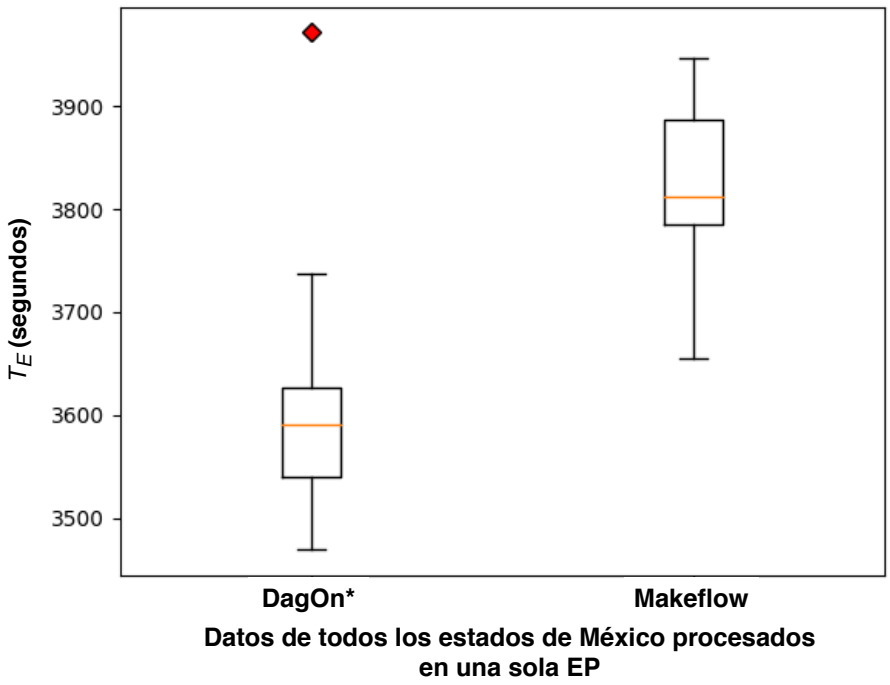


Figura 5.11: Instancias de MERRA procesando localidades de México, correspondientes a los estados de la república.

Como se puede observar en la Figura 5.11, los resultados muestran que existe una diferencia en

los tiempos de ejecución (eje vertical) entre DagOn* y Makeflow (eje horizontal), siendo DagOn* el que cuenta con un menor tiempo de procesamiento. Sin embargo, la diferencia en tiempos de respuesta entre motores es propia de las diferencias de cada uno, puesto que los experimentos fueron ejecutados en los mismos recursos y aplicando las mismas tareas. Lo que muestra este experimento básicamente que este modelo de acoplamiento transversal puede ser utilizado por diferentes motores de flujo de trabajo para acoplar múltiples flujos de trabajo y para extraer información a diferentes puntos de la solución final, la cual además se construye como un servicio, lo cual nos es provisto por los motores estudiados.

5.4.2 Experimento 2: Instancias por estado

Se generaron un total de 32 instancias de procesamiento con dependencias transversales de la tarea de *M-Acq* y *M-Split* desde el flujo de trabajo descrito en el experimento anterior. De esta manera, cada instancia procesa los datos de las localidades de un estado. Los resultados obtenidos mostraron una reducción en los tiempos de ejecución de la solución con respecto al procesamiento mediante una sola instancia. Esta reducción es esperada debido a que el procesar la misma cantidad de datos entre un mayor número de instancias en paralelo tiende a ser rápido, como fue el caso de este experimento. Los tiempos mostrados en las Figuras 5.12 y 5.13 corresponden a los tiempos de ejecución para cada instancia (eje vertical) obtenidos por Makeflow y DagOn* respectivamente, procesando datos de cada uno de los treinta y dos estados de la república mexicana (eje horizontal). Se observan sutiles diferencias entre los tiempos de procesamiento de DagOn* y Makeflow. Sin embargo, el comportamiento de cada instancia es similar, lo cual es proporcional a la cantidad de localidades con las que cada estado cuenta y debe ser procesada.

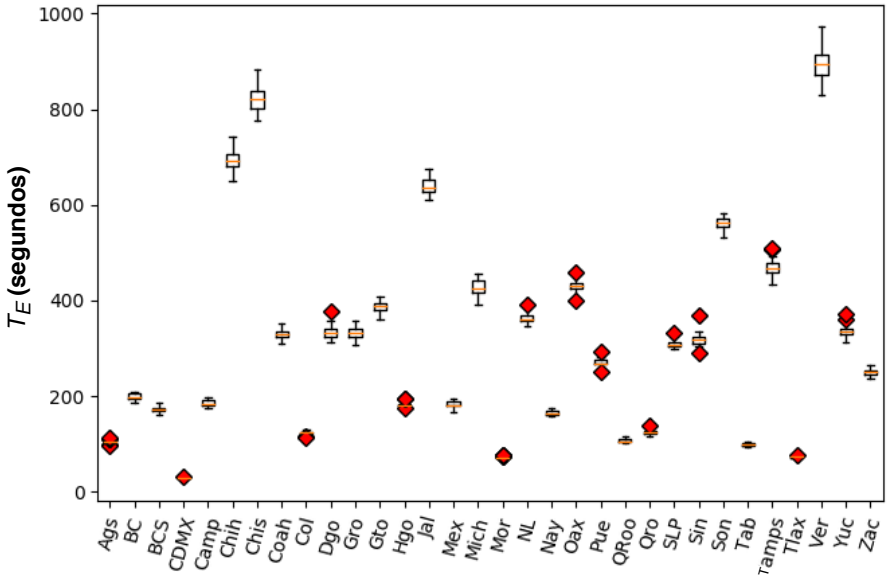


Figura 5.12: 32 Instancias de MERRA procesando localidades de cada estado, utilizando Makeflow.

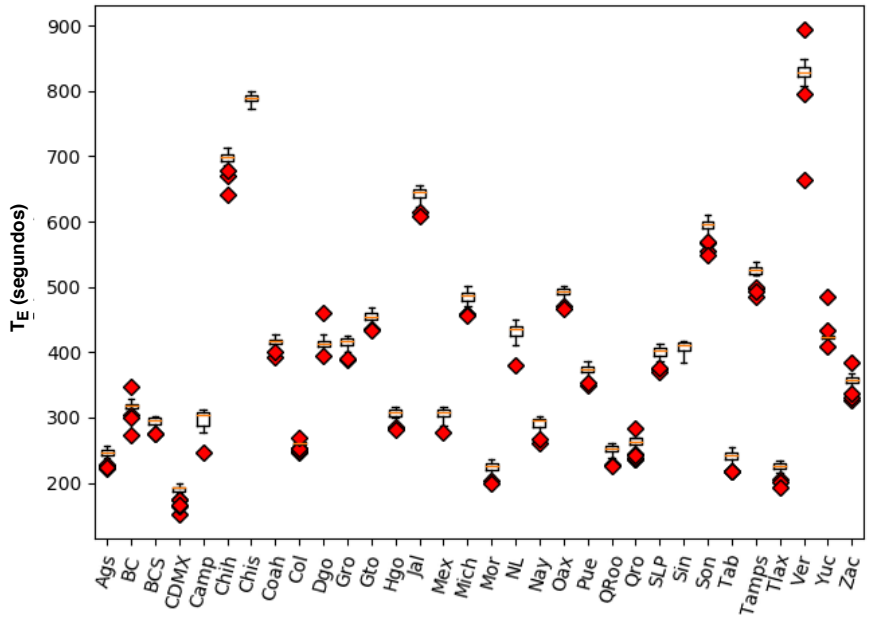


Figura 5.13: 32 Instancias de MERRA procesando localidades de cada estado, utilizando DagOn*.

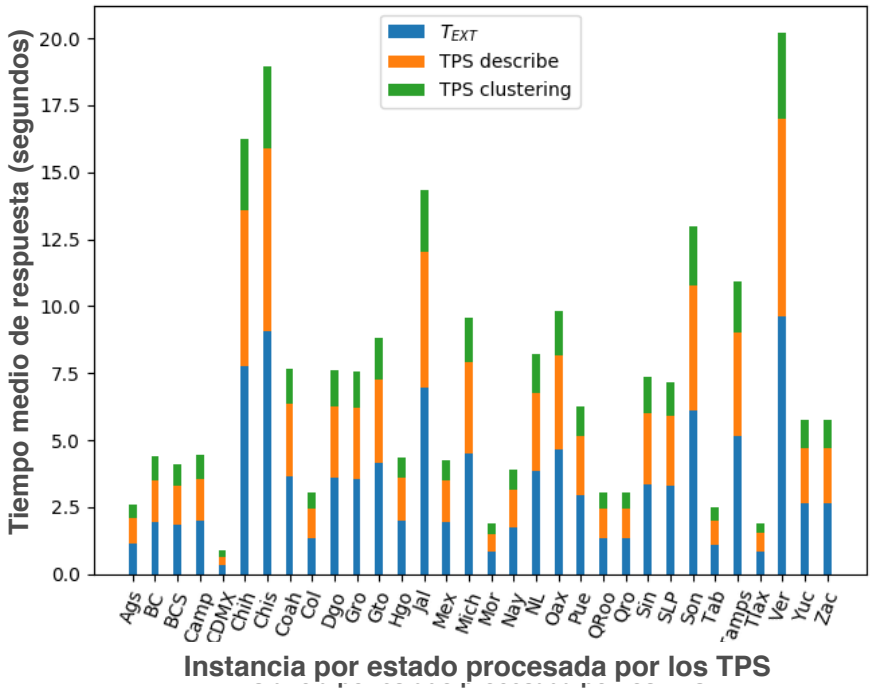


Figura 5.14: Tiempos correspondientes a los TPS por instancia, utilizando Makeflow.

5.4.3 Experimento 3: Aplicando TPS por instancia

Se aplicaron dos TPS distintos a los resultados obtenidos por cada una de las 32 instancias ejecutadas, tomando únicamente los datos producidos por la etapa *M-Int*, por lo cual, los datos resultantes son proporcionales a la cantidad de localidades procesadas. Dentro de los resultados (Figura 5.14 para Makeflow y Figura 5.15 para DagOn*) se observaron tres tiempos distintos: El tiempo de extracción de los datos (T_{EXT}), el tiempo del TPS de descripción estadística (describe) y el tiempo del TPS de agrupamiento (clustering).

Los resultados obtenidos muestran una relación directa con la cantidad de datos producidos por la tarea M-Int de cada instancia, donde estados como Veracruz (Ver) el cual cuenta con una mayor cantidad de localidades a procesar (como se observó en los resultados del experimento No. 2), cuenta con el mayor tiempo de procesamiento, desde la extracción de los datos y los TPS aplicados. Así

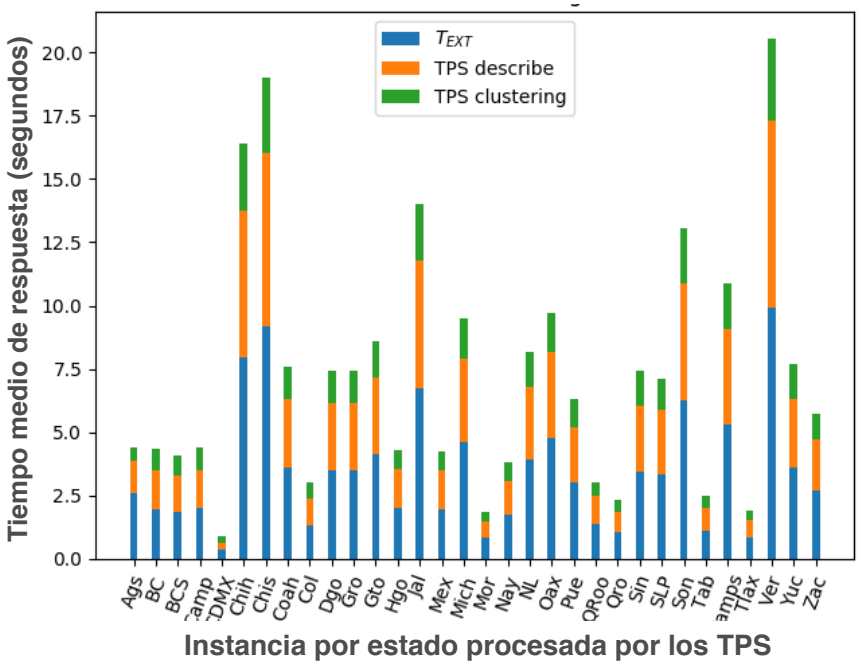


Figura 5.15: Tiempos correspondientes a los TPS por instancia, utilizando DagOn*.

mismo, existen sutiles diferencias entre los resultados obtenidos por ambos motores, sin embargo, el módulo de TPS se encuentra totalmente separado de las funcionalidades de los motores, por lo cual, las diferencias encontradas (como en las instancias *Ags* y *Yuc*) se deben a datos atípicos provocados por latencia en el transporte de datos, esto se sostiene debido a que son los tiempos de extracción de datos los que varían.

5.5 Escenario 3: Modularización de soluciones y manejo de datos no estructurados

En esta sección, se describen los experimentos realizados para ejemplificar la modularización de estructuras de procesamiento utilizando el modelo transversal. El objetivo principal de este conjunto de experimentos es evidenciar la posibilidad de modularizar estructuras de procesamiento en sub-

soluciones con objetivos específicos, así como denotar el impacto que tiene esto sobre la solución general.

Para este conjunto de experimentos se tomó como estudio de caso el flujo de trabajo Landsat8, descrito en la Sección 5.1. El primer flujo de trabajo, renombrado como correcciones-Landsat8 (CL8), consiste en transformar imágenes y aplicar correcciones para generar nuevos productos (Figura 5.16). El segundo flujo de trabajo renombrado como procesamiento-Landsat8 (PL8) se centra en la indexación de productos, haciendo una reducción (bajando la resolución de la imagen) para una posible publicación en un portal web (Figura 5.17).

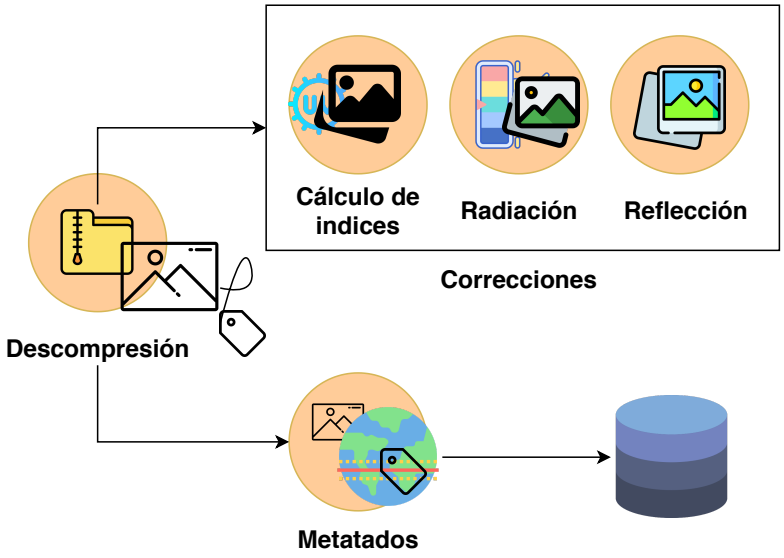


Figura 5.16: Estructura general de CL8. Se separaron las tareas de descompresión, metadata y correcciones de TIFF2JPG.

Con los flujos de trabajo separados, se definió un TPP en la tarea *Correcciones* y se acopló mediante un TPP desde la tarea *TIFF2JPG*. En este contexto, *TIFF2JPG* toma los índices producidos por las *Correcciones* para realizar su procesamiento. Asimismo, se definieron un conjunto de extractores para indexar cada uno de los índices reducidos. La estructura final del experimento se muestra en la Figura 5.18

La ejecución del experimento se realizó con base en las siguientes características:

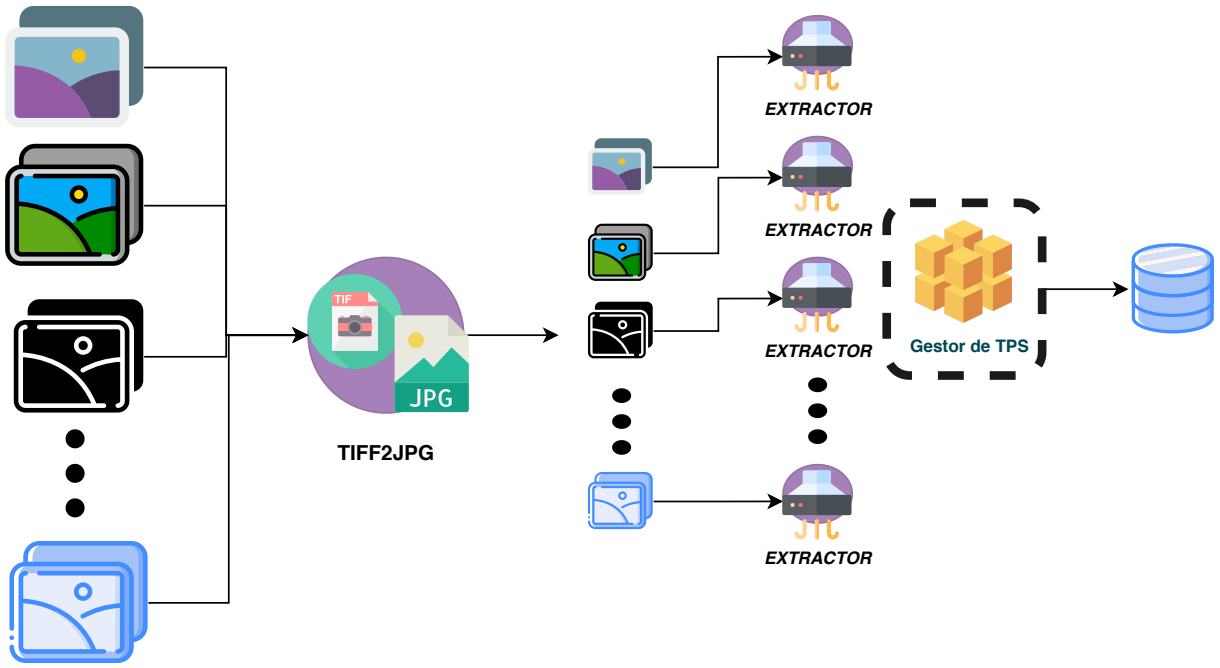


Figura 5.17: Estructura del flujo de trabajo PL8.

- 23 imágenes (m) fueron procesadas. El flujo de trabajo PL8 toma los índices producidos por la tarea **Correcciones** en CL8 (siete índices).
- Hay un extractor para cada imagen de índice.
- Dado el alto uso de memoria de algunas tareas (debido al tamaño de las imágenes), la ejecución paralela de las tareas por parte del motor de flujos de trabajo tuvo que limitarse. Para el experimento se utilizaron dos trabajos simultáneos por flujo de trabajo (a excepción de los extractores, estos siempre se ejecutan en paralelo).
- Para la gestión y ejecución de tareas, se utilizó el motor de flujo de trabajo DagOn*.

Se obtuvo una comparativa de los tiempos de ejecución para cada uno de los flujos de trabajo. En la Figura 5.19 se muestran los tiempos de ejecución en minutos (eje vertical) para cada uno de los flujos de trabajo diseñados. En este caso, CL8 tiende a tardar significativamente más tiempo que PL8. Así mismo, se realizó la comparativa entre las dos posibles alternativas al generar una única solución.

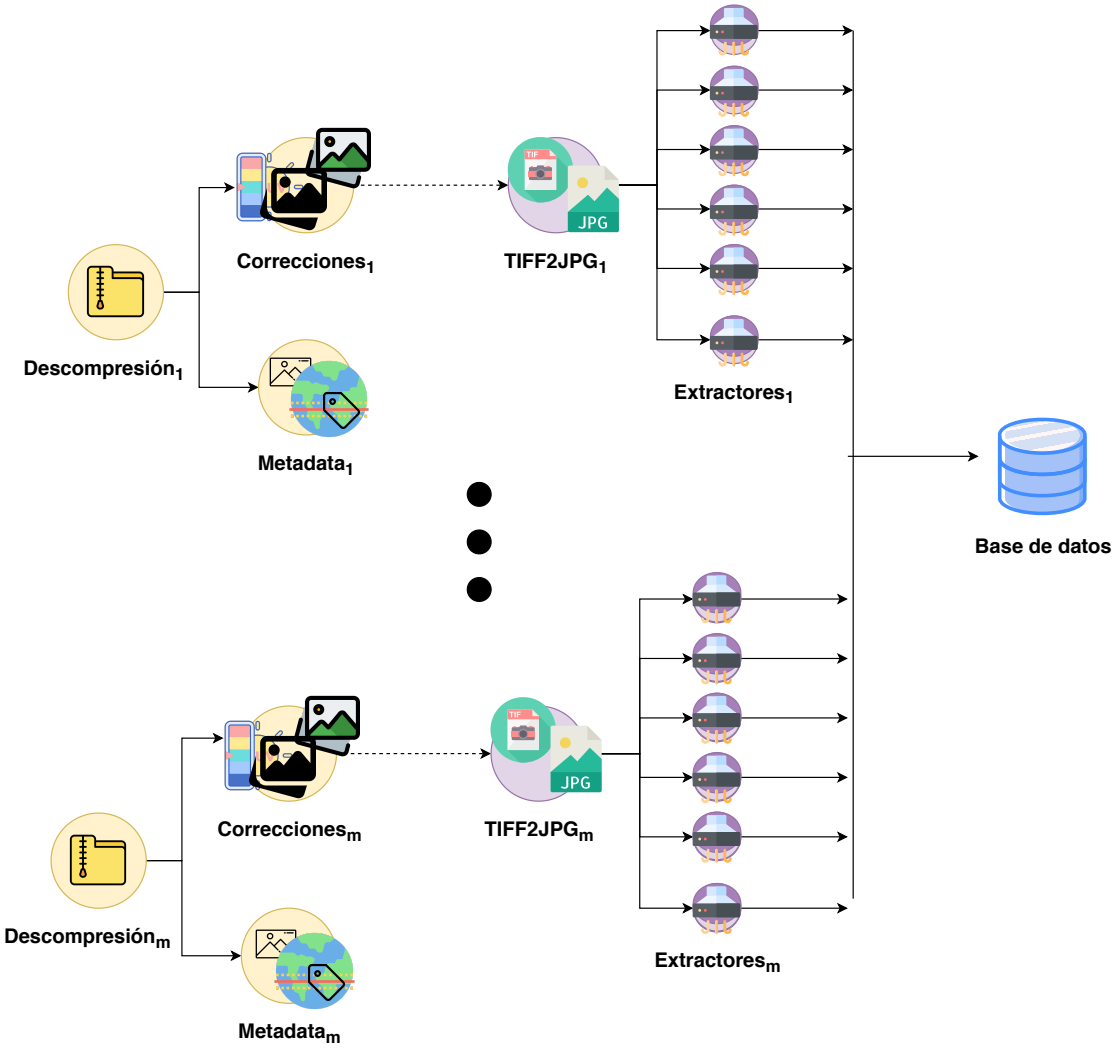


Figura 5.18: Estructura final del DAG_{TP} conformado por CL8 & PL8.

Por un lado, se tiene a todas las tareas ejecutadas como única solución y por otro lado, se tiene a dos soluciones coordinadas como una sola a través del modelo propuesto. Contrario a lo esperado, se obtuvo una reducción en el tiempo de ejecución de aproximadamente tres minutos en el caso de CL8 y PL8 unidos por el modelo transversal (por TPP) en comparativa con una solución construida de manera tradicional (Figura 5.20). Si bien la reducción en el tiempo es debido a los modelos de ejecución (dado que CL8 y PL8 son ejecutados de manera independiente), no es el objetivo de este trabajo mejorar el rendimiento de las soluciones en base a tiempo de ejecución. No obstante, la capacidad de gestionar esta diversidad de modelos de ejecución en una única solución si lo es. En este

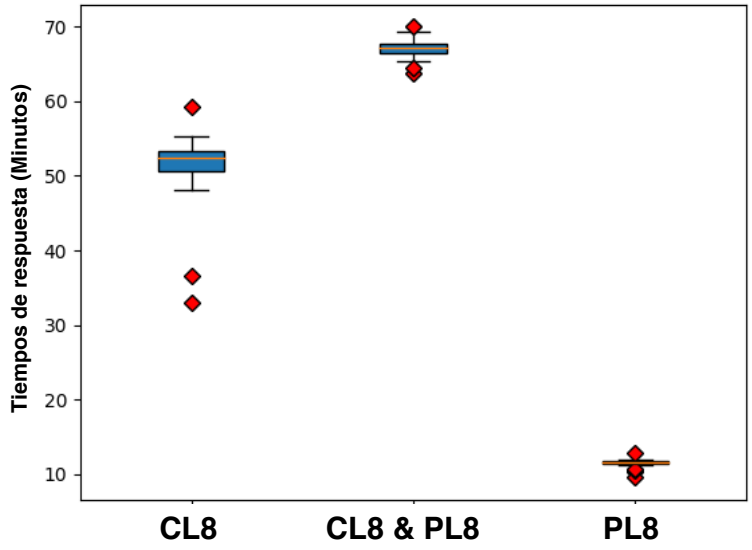


Figura 5.19: Tiempos de ejecución por cada EP.

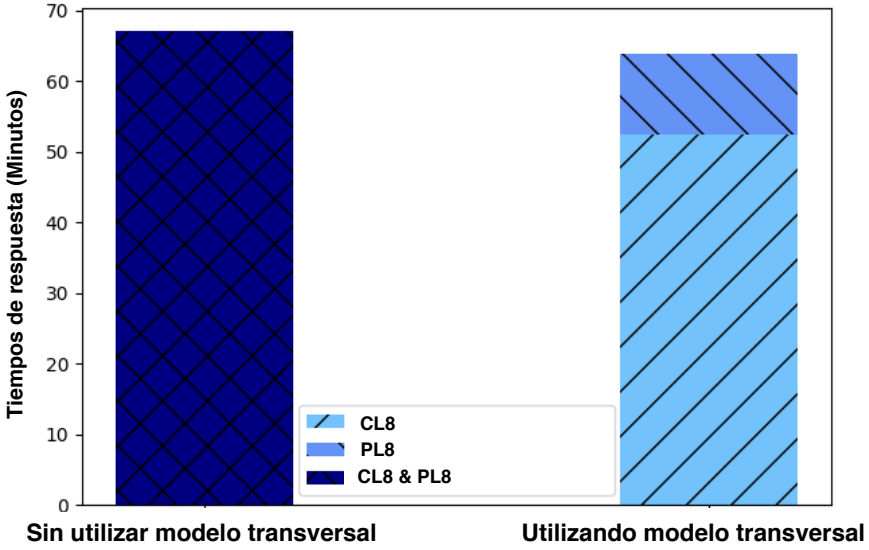


Figura 5.20: Comparación del tiempo de ejecución entre una solución implementada con el modelo transversal y una solución implementada de manera tradicional (sin el modelo transversal).

sentido, se puede notar en la Figura 5.20, que con el método tradicional es imposible diferenciar donde comienza una EP y termina la otra, mientras que utilizando el modelo de acoplamiento transversal, es posible, dado que las soluciones generadas no son únicamente cajas negras. Finalmente, es importante notar que las soluciones creadas con el modelo transversal son reutilizables y se pueden acoplar con soluciones existentes.

5.6 Escenario 4: Generación de estructuras de procesamiento como servicio mediante TPS

En esta sección, se describe un estudio de caso para la creación de una solución automatizada de principio a fin conformada por tres estructuras de procesamiento y la creación de estructuras utilizando TPS.

Para este estudio de caso, se tuvieron en cuenta tres tuberías de procesamiento existentes, que procesan tres fuentes de datos diferentes: RAMA, REDMET y MERRA. Las características de cada una de estas tuberías se describieron previamente.

El objetivo de este estudio de caso es unificar, a través del modelo propuesto, las diferentes fuentes de datos en una sola fuente, permitiendo que las tres tuberías de procesamiento converjan en un sumidero de datos, de tal manera que sean útiles para enriquecer el estudio de datos o simplemente para cruzar información. Lo anterior es posible debido a que las fuentes en cuestión contienen parámetros de altitud y latitud, lo cual hace que todas tengan registros comparables. Es importante notar que, en este tipo de escenario, cualquier fuente de datos incluirá estos campos, por tanto, el escenario es extensible a la mayoría de las fuentes de datos relacionadas con estudios sobre la tierra (clima, topografía, suelos, hidrológicos, etc).

La tubería MERRA realiza un proceso de interpolación de datos mediante el cual, tomando los datos MERRA como entrada, infiere la información meteorológica en diferentes puntos geográficos proporcionados. Para este escenario se utilizó como entrada la lista de coordenadas geográficas

pertenecientes a las estaciones RAMA y REDMET, siendo de esta manera posible unir los datos de las distintas fuentes mediante las variables geográficas de latitud y longitud (ver Figura 5.1).

Por otro lado, se tienen las tuberías de procesamiento de RAMA y REDMET. Estas tuberías realizan únicamente la adquisición de datos de la fuente primaria, sin embargo, estos datos se consideran sucios, pues contienen valores atípicos y están organizados en una estructura no aceptable para el análisis de datos (Figura 5.3). Para resolver este problema, sin desarrollar nuevas aplicaciones o rediseñar la tubería existente, se utilizaron múltiples TPS para construir un flujo de trabajo de post-procesamiento de datos (Figura 5.21).

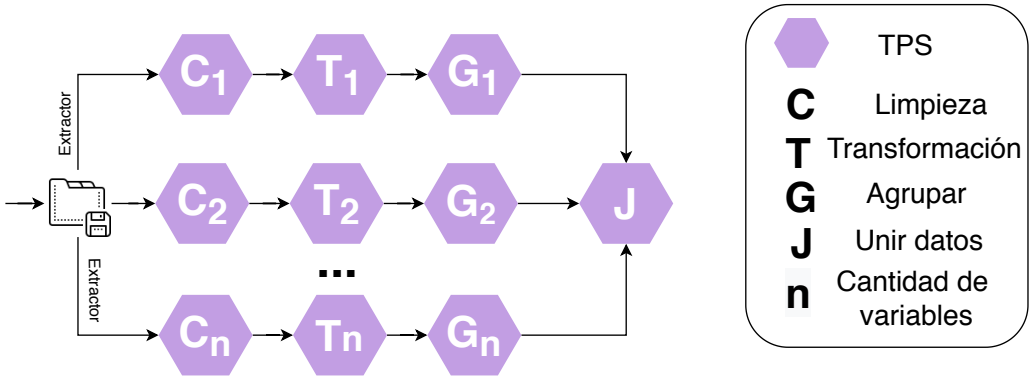


Figura 5.21: Flujo de trabajo de post-procesamiento para RAMA y REDMET construido mediante TPS encadenados.

El flujo de trabajo de post-procesamiento consta de tres tareas:

- **Limpieza (C)**: Realiza un proceso de limpieza de los datos, eliminando o reemplazando los datos faltantes y/o atípicos.

- **Transformación (T)**: Transforma la estructura de la tabla de datos. En este caso, las columnas pasan a ser registros. Este proceso se realiza debido a que cada columna es una estación sensora y cada registro es el dato puntual de cada hora. La transformación de columnas a registros mejora el manejo de datos así como el acoplamiento posterior con los datos de MERRA.

- **Agrupación (G)**: Agrupa un conjunto de registros y los transforma en uno, obteniendo la media, la mediana o la moda. Para este caso, los registros de la misma antena y el mismo día (pero diferente tiempo de detección) se agruparon en uno usando la media. Este proceso se realizó con el fin de reducir la cantidad de registros, además de obtener una media representativa por día para cada variable.

Se aplicó la tubería de procesamiento de tres tareas a cada una de las variables producidas por REDMET y RAMA, para finalmente unificarlas (*join data*) en una sola tabla de *Contaminantes* (RAMA) y *Temperaturas* (REDMET). Finalmente, las estructuras de procesamiento se unen utilizando el prototipo desarrollado. Se definió un TPP desde la tarea *Organizador* a una instancia del flujo de trabajo de post-procesamiento diseñado (la definición se hizo tanto para RAMA como para REDMET). Posteriormente, y como parte de un análisis de los datos resultantes, se utilizaron diferentes TPS para obtener información útil de la unión de los datos, además de aplicar diferentes procesos con el fin de encontrar patrones en los datos (Figura 5.22).

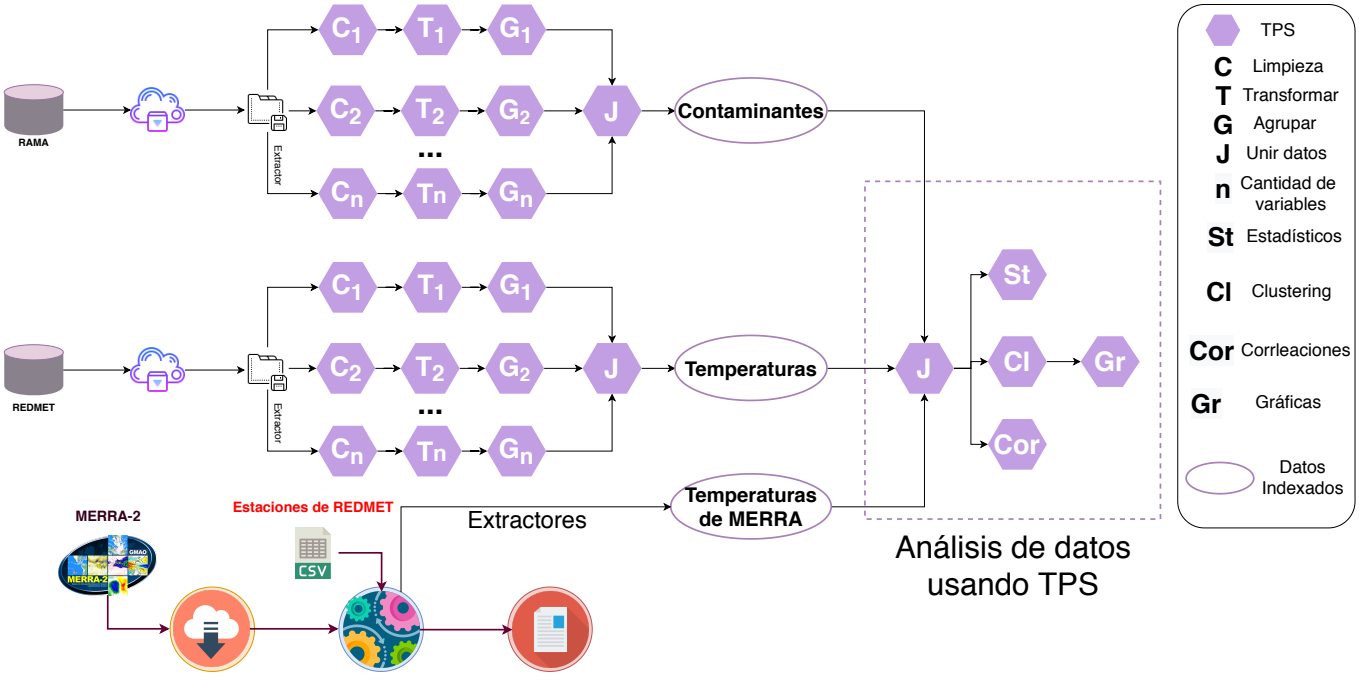


Figura 5.22: Estructura completa de la solución.

En la Figura 5.23 se muestran los tiempos de ejecución en minutos (eje vertical) de cada una de las tuberías (eje horizontal). La tubería de MERRA es la que conlleva más tiempo de procesamiento, triplicando el tiempo de respuesta de RAMA junto con su flujo de trabajo de procesamiento posterior correspondiente. Al ser RAMA y REDMET independientes de la tubería de MERRA, es posible que estas puedan entregar sus resultados finales al usuario sin necesidad de esperar a que el resto de las instancias de la misma solución (en este caso MERRA) terminen.

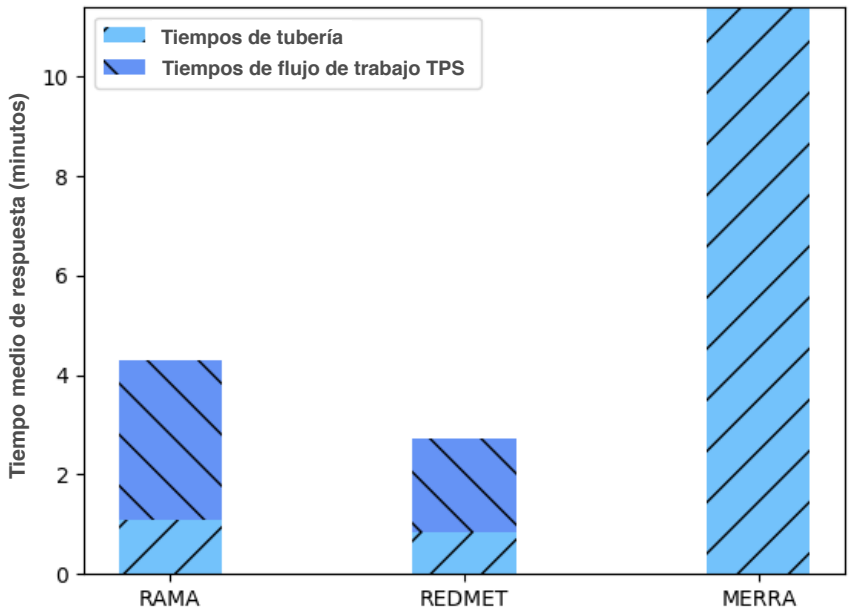


Figura 5.23: Tiempos de respuesta de las soluciones.

Por otro lado, se realizó un análisis de datos en un rango de fechas de 2016-2020. Al vincular las tres fuentes de datos, es posible contrastar, por ejemplo, las temperaturas obtenidas por los sensores REDMET con las obtenidas por el re-análisis MERRA, para cada estación y cada día de detección. Usando TPS, se creó una nueva variable llamada DIFF que es la diferencia de temperatura MERRA-REDMET para cada día y cada estación.

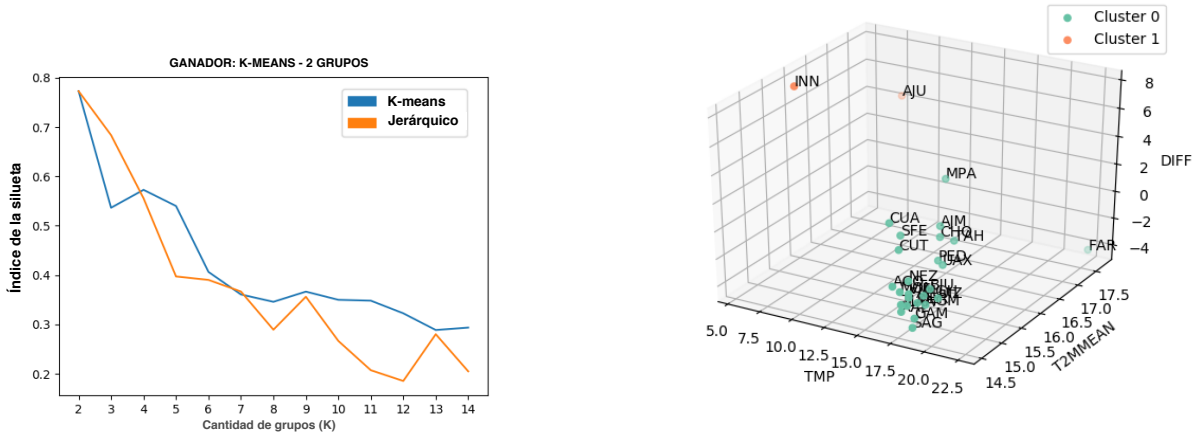
El modelo permite que los datos producidos en cualquier etapa se reutilicen para cualquier tipo

de procesamiento, ya sea por los TPS o aplicaciones (o servicios) externas. Usando un TPS, se utilizaron dos algoritmos de agrupamiento (K-means y jerárquico) en comparación a través del índice de agrupación de silueta [46], con el objetivo de observar las diferentes agrupaciones presentes en estaciones de detección de datos en función de los datos que obtienen. El agrupamiento se dividió en dos partes:

- Generando grupos basados en la temperatura de MERRA, temperatura de REDMET y DIFF (temperatura MERRA - temperatura REDMET).
- Grupos generadores basados en todos los contaminantes y partículas del aire.

Para el primer caso, se obtuvieron un total de dos grupos, elegidos en base a la calificación otorgada por el índice de silueta. Como se puede ver en la Figura 5.24(a), el rendimiento de los dos algoritmos se midió a través de diferentes valores de K, donde K-means presento el mejor rendimiento con dos grupos. La diferencia resulta ser muy sutil para ser captada mediante la gráfica, sin embargo, el TPS devuelve (en el titulo de la gráfica en este caso) el algoritmo ganador. Finalmente, los resultados obtenidos son presentados en la Figura 5.24(b), siendo el grupo 0 donde el valor de la diferencia entre DIFF (diferencia de temperaturas) es menor o igual a cero, mientras que el grupo 1 está compuesto únicamente por las estaciones *INN* y *AJU* donde se presentan los valores DIFF positivos.

Para el caso número dos, se aplicaron los mismos algoritmos de agrupamiento con la diferencia de que las variables para generar la agrupación fueron los contaminantes obtenidos por RAMA. Sin embargo, para obtener un resultado que pueda entenderse gráficamente y sin alterar los datos en gran medida, se optó por la selección de variables mediante el análisis de componentes principales (PCA). PCA también se proporciona como un servicio por el TPS a cargo del clustering, por lo que solo es necesario especificar los parámetros para que el servicio lleve a cabo este proceso, ya sea seleccionando un rango de variación (es decir, seleccionando una serie de componentes que mantienen una variación especificada) o seleccione el número especificado de componentes. Para este caso, se



(a) Silueta

(b) Estaciones

Figura 5.24: Resultados de clustering para datos de temperaturas. a) corresponde al rendimiento de los algoritmos con el índice de silueta. b) Resultados de clustering sobre temperaturas de REDMET (TMP), MERRA (T2MMEAN) y su diferencial (DIFF).

eligió seleccionar 3 componentes principales entre el total de variables contaminantes, manteniendo así una varianza en los datos superior al 90 %.

A diferencia del primer caso, el algoritmo con el rendimiento más alto fue el jerárquico (agglomerative) con un total de dos grupos (Figura 5.25(a)). Otra diferencia notable es que las estaciones obtenidas en el primer caso no aparecen en el mismo grupo obtenido en el segundo caso (Figura 5.25(b)), por lo cual se asume que no hay una relación entre las variables de temperatura y contaminantes que permita un agrupamiento similar en las estaciones.

Otro tipo de aplicación que se puede realizar es el análisis predictivo. Mediante el uso de datos meteorológicos obtenidos por las estaciones y la verificación por los datos de re-análisis MERRA, es posible utilizar canales de aprendizaje automático para predecir los valores de contaminantes.

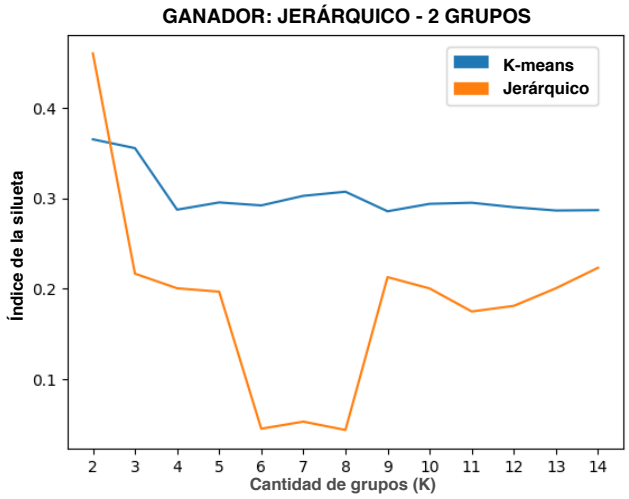
Para este ejemplo, se generó una red neuronal de perceptrón multicapa (MLP-NN) para el análisis predictivo de contaminantes por regresión. Del conjunto completo de datos, se utilizó una tercera parte como conjunto de datos de entrenamiento, tomando las temperaturas, diferencial, humedad, velocidad y dirección del viento y partículas en el aire (PM10 y PM25) de REDMET y MERRA como

parámetros de entrada. Una vez entrenada la red, se realizó se probó la red utilizando el conjunto de datos restantes y midiendo la calidad de los resultados con la métrica R2 [28] (R cuadrada, o en inglés R-squared), una medida estadística para conocer que tan cerca están los datos de la línea de regresión.

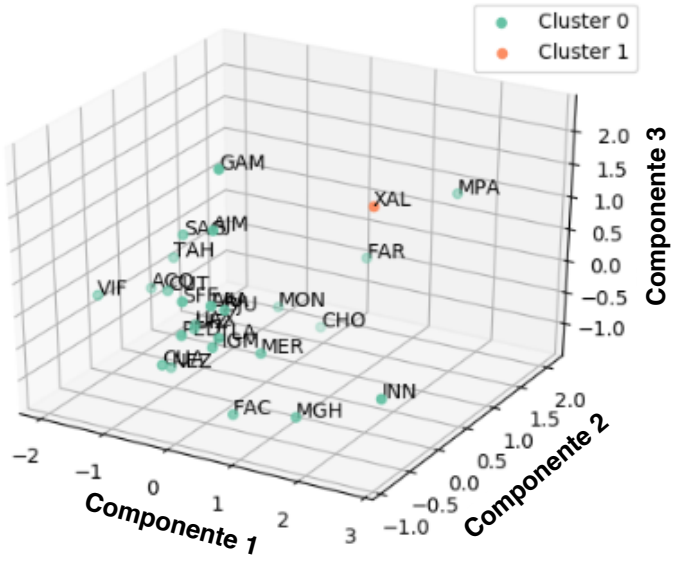
La Figura 5.26(a) muestra los resultados obtenidos al predecir las partículas de CO, que tienen una variación aceptable con respecto a los datos reales representados con una puntuación R2 de 0.92. Por otro lado, no se obtuvo un el mismo rendimiento al predecir partículas como NO (Figura 5.26(b)) y NO2 (Figura 5.26(c)), obteniendo puntajes R2 en un rango de 0.6 y 0.7.

Aunque es posible mejorar el rendimiento de la red configurando los parámetros, el número de neuronas, aumentando la cantidad de datos de entrenamiento o la selección de otro tipo de red neuronal, este no es el objetivo principal de este trabajo de tesis. El objetivo principal es mostrar las posibilidades de generar/utilizar servicios o aplicaciones que puedan consumir y aprovechar las características que ofrece el modelo transversal.

A través de los datos obtenidos mediante el procesamiento de las estructuras de procesamiento, los TPS y los datos indexados disponibles como servicio, fue posible generar una solución **completa y automatizada de principio a fin para la consulta y procesamiento bajo demanda de datos meteorológicos y contaminantes**. Esta solución puede ser consultada por diferentes usuarios que pueden aprovecharla utilizando los diferentes TPS expuestos (para transformación de datos, agrupamiento, limpieza de datos, ANOVA, gráficos) o por aplicaciones o servicios externos (como la red neuronal), que a su vez se pueden acoplar a la solución general creando nuevos TPS, o bien indexar los datos resultantes con el fin de que puedan ser consumidos bajo demanda.

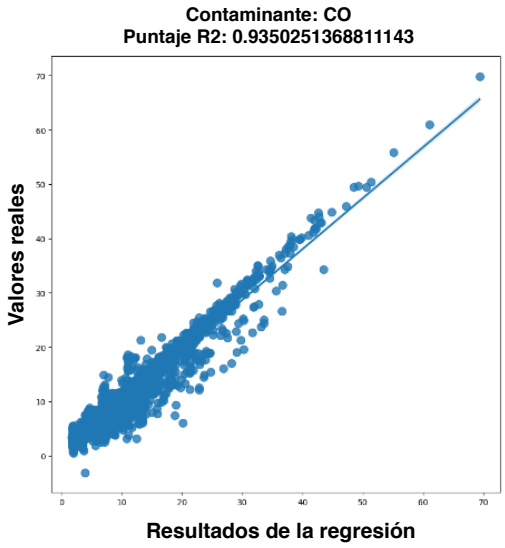


(a) Silueta

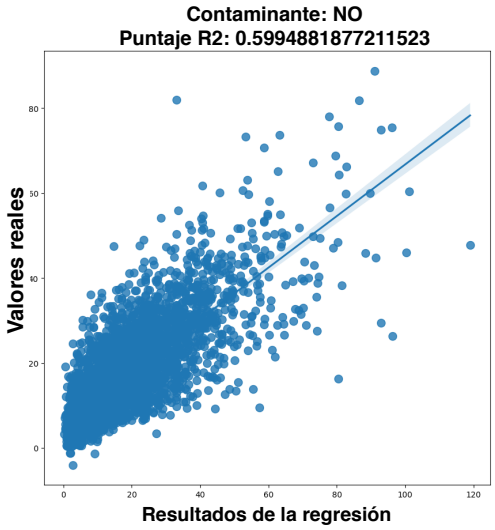


(b) Estaciones

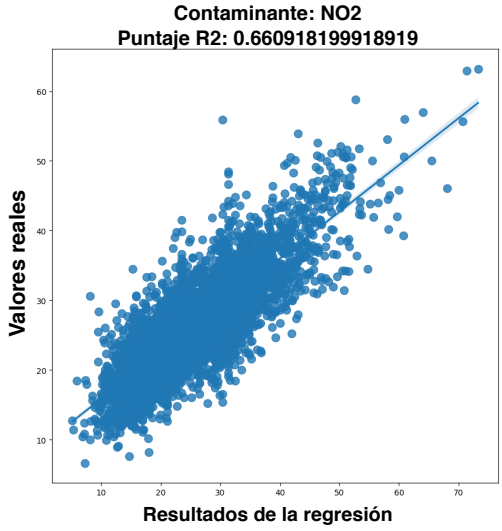
Figura 5.25: Resultados de clustering para datos de contaminantes.



(a) CO



(b) NO



(c) NO2

Figura 5.26: Resultados de la regresión. En la línea vertical se muestran los datos reales, mientras que en la línea horizontal se muestran los datos obtenidos por la red neuronal.

6

Conclusiones y trabajo futuro

A lo largo de este trabajo de tesis se presentó el diseño, implementación y evaluación de un modelo de acoplamiento transversal para estructuras de procesamiento científicas con el objetivo de construir nuevas soluciones como servicio a partir de estructuras ya existentes.

6.1 Conclusiones generales

Con base en los experimentos realizados y desde una perspectiva cualitativa el modelo exhibe las siguientes características:

- *Modularidad*. La capacidad de gestionar la ejecución de múltiples soluciones dependientes entre sí, con modelos tanto de programación como de ejecución distintos, permite que grandes estructuras de procesamiento puedan ser construidas en soluciones más pequeñas e interconectadas.

- *Heterogeneidad de ejecución.* El modelo permite que las estructuras de procesamiento que conforman una solución se ejecuten a diferentes tiempos, o más precisamente, en tiempos de ejecución diferentes y convenientes para el usuario.

- *Soluciones como servicio.* Tanto la estructura general de las soluciones generadas como los datos producidos son expuestos como un servicio. La generación de soluciones como servicio ha permitido dar cumplimiento al segundo objetivo específico.

- *Flexibilidad en modelos de programación y ejecución.* El modelo propuesto es capaz de adaptarse a los modelos de programación y ejecución de trabajos encontrados en la literatura (motores de flujos de trabajo o workflow engines). Esta característica es posible gracias a la abstracción de las estructuras a través de puntos transversales, siendo capaz de utilizarlos como un servicio para la ejecución de las tareas. La capacidad de gestionar estructuras de procesamiento definidas en múltiples modelos de programación existentes a permitido cumplir el primer objetivo específico.

- *Esquema de división de procesamiento en múltiples instancias de estructuras de procesamiento.* Este esquema surgió como una forma de división entre instancias de EP. El esquema promueve

la reutilización de datos previamente producidos con el objetivo de ahorrar esfuerzos y recursos de cómputo.

- *Esquema de gestión y acoplamiento de servicios.* Se obtuvo un esquema de gestión de servicios para la consolidación y publicación de los resultados producidos por las soluciones. Con este esquema se logró alcanzar el objetivo específico número tres.

Desde una perspectiva cuantitativa, la ejecución de soluciones construidas utilizando el modelo obtuvo un comportamiento eficiente en tiempos de ejecución con respecto a soluciones construidas sin el modelo. Si bien existe un impacto en los tiempos de ejecución, no es inviable la utilización del modelo transversal.

El método propuesto permite construir soluciones basadas en estructuras de procesamiento ya existentes, generando en su conjunto una solución en la forma de un meta-flujo de trabajo, que a su vez, puede ser utilizado para construir nuevas soluciones. Servicios de procesamiento transversal pueden ser utilizados para la consolidación de los resultados, siendo estos capaz de acoplarse, reutilizarse y ser coordinados con el fin de crear estructuras más complejas y automatizadas. El modelo es compatible con motores de flujos de trabajo, capaz de utilizarlos como servicio para la ejecución de tareas y dotándolos con la capacidad de gestionar múltiples estructuras previamente desplegadas. Con base a lo anterior y al cumplimiento de todos los objetivos específicos, se logra cumplir el objetivo general de este proyecto de tesis.

6.2 Contribuciones

1. Un modelo para la construcción de soluciones basadas en estructuras de procesamiento existentes que incluye:
    - Un prototipo para el acoplamiento de estructuras de procesamiento.
    - Un esquema de gestión y acoplamiento de microservicios.

- Un repositorio de soluciones definidas como DAG y desplegadas como servicio.
- Un monitor de ejecución de tareas adaptable a diferentes motores de flujos de trabajo.

6.3 Limitaciones

Durante el desarrollo del proyecto de tesis surgieron diversos obstáculos y dificultades, dando como resultado a diversas limitaciones:

- La unificación de modelos de programación es posible mediante la definición de reglas que otorguen al prototipo la suficiente información para realizar la ejecución de las tareas en un motor de flujos de trabajo de manera correcta. No obstante, el prototipo asume que el entorno de ejecución está previamente configurado, o en palabras más simples, se asume que el motor de flujos de trabajo está instalado, configurado y cuenta con todas sus dependencias.
- El modelo no se encarga de la detección de estructuras de procesamiento previamente desplegadas. Una solución debe primeramente ser registrada en el repositorio de DAG para poder ser utilizada como servicio por otras soluciones.
- La definición de dependencias transversales es hecha por el usuario.

6.4 Trabajo futuro

Se tienen en cuenta las siguientes características para agregar al modelo en futuros trabajos:

- Método para el descubrimiento de dependencias transversales de manera automática. Refiriéndose a la creación de un método que permita sugerir TPP posibles al usuario.
- Realizar la integración de más motores de flujos de trabajo. Definiendo las reglas y configuraciones correspondientes.

- Simplificar los métodos de ejecución de TPS. En otras palabras, crear un esquema para facilitar el desarrollo de nuevos TPS.

- Creación de un esquema para el descubrimiento y publicación de soluciones por parte de los usuarios. Para esto, se tomará como base el repositorio de DAG y se diseñara un esquema que permita a los usuarios conocer, publicar y subscribirse a soluciones existentes de manera intuitiva.

Bibliografía

[1] Abouelhoda, M., Alaa, S., and Ghanem, M. (2010). Meta-workflows: pattern-based interoperability between galaxy and taverna. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*, page 2. ACM.

[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] Arora, R. (2016). *Conquering Big Data with High Performance Computing*. Springer.

[4] Arshad, J., Terstyanszky, G., Kiss, T., Weingarten, N., and Taffoni, G. (2016). A formal approach to support interoperability in scientific meta-workflows. *Journal of Grid Computing*, 14(4):655–671.

[5] Atkinson, M., Gesing, S., Montagnat, J., and Taylor, I. (2017). Scientific workflows: Past, present and future.

[6] 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*.

[7] Badia, R. M., Conejero, J., Diaz, C., Ejarque, J., Lezzi, D., Lordan, F., Ramon-Cortes, C., and Sirvent, R. (2015). Comp superscalar, an interoperable programming framework. *SoftwareX*, 3:32–36.

[8] Bansal, S. K. (2014). Towards a semantic extract-transform-load (etl) framework for big data integration. In *2014 IEEE International Congress on Big Data*, pages 522–529. IEEE.

[9] Barker, A. and Van Hemert, J. (2007). Scientific workflow: a survey and research directions. In *International Conference on Parallel Processing and Applied Mathematics*, pages 746–753. Springer.

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

[11] de Sousa, N. T., Hasselbring, W., Weber, T., and Kranzlmüller, D. (2018). Designing a generic research data infrastructure architecture with continuous software engineering. In *Software Engineering Workshops 2018*, volume Vol-2066 of *CEUR Workshop Proceedings*, pages 85–88. CEUR-WS.org.

[12] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.-H., Vahi, K., and Livny, M. (2004). Pegasus: Mapping scientific workflows onto the grid. In *European Across Grids Conference*, pages 11–20. Springer.

[13] Deelman, E., Gil, Y., and Zemankova, M. (2006). Nsf workshop on the challenges of scientific workflows. *May*, pages 1–2.

[14] Di Francesco, P., Malavolta, I., and Lago, P. (2017). Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 21–30. IEEE.

[15] Dreher, M. and Peterka, T. (2017). Decaf: Decoupled dataflows for in situ high-performance workflows.

[16] Felter, W., Ferreira, A., Rajamony, R., and Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 171–172. IEEE.

[17] Fowler, M. and Lewis, J. (2014). Microservices a definition of this new architectural term. *URL: http://martinfowler.com/articles/microservices.html*.

[18] Garijo, D., Corcho, O., Gil, Y., Gutman, B. A., Dinov, I. D., Thompson, P., and Toga, A. W. (2014). Fragflow automated fragment detection in scientific workflows. In *2014 IEEE 10th International Conference on e-Science*, volume 1, pages 281–289. IEEE.

[19] Goecks, J., Nekrutenko, A., Taylor, J., Team, G., et al. (2010). Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11(8):R86.

[20] 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.

[21] Held, M. and Blochinger, W. (2009). Structured collaborative workflow design. *Future Generation Computer Systems*, 25(6):638–653.

[22] Hempelmann, N., Ehbrecht, C., Alvarez-Castro, C., Brockmann, P., Falk, W., Hoffmann, J., Kindermann, S., Koziol, B., Nangini, C., Radanovics, S., et al. (2018). Web processing service for climate impact and extreme weather event analyses. flyingpigeon (version 1.0). *Computers & Geosciences*, 110:65–72.

[23] Herres-Pawlis, S., Hoffmann, A., Balaskó, Á., Kacsuk, P., Birkenheuer, G., Brinkmann, A., De La Garza, L., Krüger, J., Gesing, S., Grunzke, R., et al. (2015). Quantum chemical meta-workflows in mosgrid. *Concurrency and Computation: Practice and Experience*, 27(2):344–357.

[24] Hollingsworth, D. (1995). The workflow reference model. *workflow management coalition TC00-1003*.

[25] Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P., and Oinn, T. (2006). Taverna: a tool for building and running workflows of services. *Nucleic acids research*, 34(suppl_2):W729–W732.

[26] Johanson, A., Flögel, S., Dullo, C., and Hasselbring, W. (2016). Oceantea: exploring ocean-derived climate data using microservices.

[27] Laster, B. (2018). *Jenkins 2: Up and Running: Evolve Your Deployment Pipeline for Next Generation Automation*. .º'Reilly Media, Inc."

[28] Lewis-Beck, M. S. and Skalaban, A. (1990). The r-squared: Some straight talk. *Political Analysis*, 2:153–171.

[29] Li, Z., Huang, Q., Carbone, G. J., and Hu, F. (2017). A high performance query analytical framework for supporting data-intensive climate studies. *Computers, Environment and Urban Systems*, 62:210–221.

[30] Li, Z., Huang, Q., Jiang, Y., and Hu, F. (2019). Sovas: a scalable online visual analytic system for big climate data analysis. *International Journal of Geographical Information Science*, 0(0):1–22.

[31] Li, Z., Yang, C., Jin, B., Yu, M., Liu, K., Sun, M., and Zhan, M. (2015). Enabling big geoscience data analytics with a cloud-based, mapreduce-enabled and service-oriented workflow framework. *PloS one*, 10(3):e0116781.

[32] Meidanis, J., Vossen, G., and Weske, M. (1996). Using workflow management in dna sequencing. In *Proceedings First IFCIS International Conference on Cooperative Information Systems*, pages 114–123.

[33] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2.

[34] Missier, P., Soiland-Reyes, S., Owen, S., Tan, W., Nenadic, A., Dunlop, I., Williams, A., Oinn, T., and Goble, C. (2010). Taverna, reloaded. In *International conference on scientific and statistical database management*, pages 471–481. Springer.

[35] Modeling, G. and Office, A. (2015). Global modeling and assimilation office (gmao). merra-2 statd_2d_slv_nx: 2d, daily, aggregated statistics, single-level, assimilation, single-level diagnostics v5.12.4, greenbelt, md, usa, goddard earth sciences data and information services center (ges disc).

[36] Montella, R., Di Luccio, D., and Kosta, S. (2018). 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.

[37] Montella, R., S.-G. D. D. D. L. D. K. S. G.-C. J. L. F. I. (2020). An efficient pattern-based approach for workflow supporting large-scale science: the dagonstar experience. In *Future Generation Computer Systems.*

[38] Morgan, R., Illidge, S., and Wilson-Wilde, L. (2019). Assessment of the potential investigative value of a decentralised rapid dna workflow for reference dna samples. *Forensic Science International*, 294:140 – 149.

[39] Muller, C. (2020). Historical background of big data in astro and geo context. In *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, pages 21–29. Elsevier.

[40] Naik, N. (2016). Building a virtual system of systems using docker swarm in multiple clouds. In *2016 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–3. IEEE.

[41] Navani, G., Evans, M. P., Dietrich, D. A., Allen, M. D., Moore, C. C., Hakimattar, L., Doyle, S. J., Bartel, W. C., Maher, K., Patel, V., et al. (2008). Computer system for providing a collaborative workflow environment. US Patent 7,448,046.

[42] Pankratius, V. and Stucky, W. (2005). A formal foundation for workflow composition, workflow view definition, and workflow normalization based on petri nets. In *Proceedings of the 2Nd Asia-Pacific Conference on Conceptual Modelling - Volume 43*, APCCM '05, pages 79–88, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

[43] Preston, B. L., Yuen, E. J., and Westaway, R. M. (2011). Putting vulnerability to climate change on the map: a review of approaches, benefits, and risks. *Sustainability Science*, 6(2):177–202.

[44] Puhlmann, F. and Weske, M. (2005). Using the π-calculus for formalizing workflow patterns. In *International Conference on Business Process Management*, pages 153–168. Springer.

[45] Richardson, C. (2018). Microservices patterns.

[46] Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

[47] Sánchez-Gallegos, D. D., Di Luccio, D., Gonzalez-Compean, J., and Montella, R. (2019). A microservice-based building block approach for scientific workflow engines: Processing large data volumes with dagonstar. In *2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pages 368–375. IEEE.

[48] Schnase, J. L., Duffy, D. Q., Tamkin, G. S., Nadeau, D., Thompson, J. H., Grieg, C. M., McInerney, M. A., and Webster, W. P. (2017). Merra analytic services: Meeting the big data challenges of climate science through cloud-enabled climate analytics-as-a-service. *Computers, Environment and Urban Systems*, 61:198–211.

[49] SEDEMA (2012a). Red automática de monitoreo atmosférico (rama), accessed: June 2020.

[50] SEDEMA (2012b). Red de meteorología y radiación solar (redmet), accessed: June 2020.

[51] Shad, R., Mesgari, M. S., Shad, A., et al. (2009). Predicting air pollution using fuzzy genetic linear membership kriging in gis. *Computers, environment and urban systems*, 33(6):472–481.

[52] Silva, V., De Oliveira, D., Valduriez, P., and Mattoso, M. (2016). Analyzing related raw data files through dataflows. *Concurrency and Computation: Practice and Experience*, 28(8):2528–2545.

[53] Silva, V., de Oliveira, D., Valduriez, P., and Mattoso, M. (2018). Dfanalyzer: runtime dataflow analysis of scientific applications using provenance. *Proceedings of the VLDB Endowment*, 11(12):2082–2085.

[54] Suthaharan, S. (2014). Big data classification: Problems and challenges in network intrusion prediction with machine learning. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):70–73.

[55] Taylor, I. J., Deelman, E., Gannon, D. B., Shields, M., et al. (2007). *Workflows for e-Science: scientific workflows for grids*, volume 1. Springer.

[56] Tejedor, E., Becerra, Y., Alomar, G., Queralt, A., Badia, R. M., Torres, J., Cortes, T., and Labarta, J. (2017). Pycompss: Parallel computational workflows in python. *The International Journal of High Performance Computing Applications*, 31(1):66–82.

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

[58] Thönes, J. (2015). Microservices. *IEEE software*, 32(1):116–116.

[59] Toro-Domínguez, D., Martorell-Marugán, J., López-Domínguez, R., García-Moreno, A., González-Rumayor, V., Alarcón-Riquelme, M. E., and Carmona-Sáez, P. (2018). Imageo: integrative gene expression meta-analysis from geo database. *Bioinformatics*, 35(5):880–882.

[60] Viggiato, M., Terra, R., Rocha, H., Valente, M. T., and Figueiredo, E. (2018). Microservices in practice: A survey study. *arXiv preprint arXiv:1808.04836*.

[61] Weisstein, E. W. (2003). Acyclic digraph.

[62] Wilde, M., Hategan, M., Wozniak, J. M., Clifford, B., Katz, D. S., and Foster, I. (2011). Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652.

[63] Woodman, S., Parastatidis, S., and Webber, J. (2007). Protocol-based integration using ssdl and π-calculus. In *Workflows for e-Science*, pages 227–243. Springer.

[64] Xia, J., Yang, C., and Li, Q. (2018). Using spatiotemporal patterns to optimize earth observation big data access: Novel approaches of indexing, service modeling and cloud computing. *Computers, Environment and Urban Systems*, 72:191–203.

[65] Yang, C., Yu, M., Hu, F., Jiang, Y., and Li, Y. (2017). Utilizing cloud computing to address big geospatial data challenges. *Computers, Environment and Urban Systems*, 61:120–128.

[66] Yildiz, O., Ejarque, J., Chan, H., Sankaranarayanan, S., Badia, R. M., and Peterka, T. (2019). Heterogeneous hierarchical workflow composition. *Computing in Science & Engineering*.

[67] Zeng, Q., Sun, S. X., Duan, H., Liu, C., and Wang, H. (2013). Cross-organizational collaborative workflow mining from a multi-source log. *Decision support systems*, 54(3):1280–1301.

[68] Zerger, A. and Smith, D. I. (2003). Impediments to using gis for real-time disaster decision support. *Computers, environment and urban systems*, 27(2):123–141.

[69] Zhang, L. (2006). Research on workflow patterns based on petri nets. In *2006 IEEE Conference on Robotics, Automation and Mechatronics*, pages 1–6. IEEE.

[70] Zhou, Z., Wen, J., Wang, Y., Xue, X., Hung, P. C., and Nguyen, L. D. (2020). Topic-based crossing-workflow fragment discovery. *Future Generation Computer Systems*.