Prueba de Concepto de Correlacionador para VLBI basado en Spark

Autor: Antonio José Vázquez Álvarez.

Directores: Agustín Carlos Caminero Herráez,
Rafael Pastor Vargas.

TRABAJO DE FIN DE MÁSTER DEL
MÁSTER UNIVERSITARIO EN INGENIERÍA Y CIENCIA DE DATOS

17 de junio de 2021
Convocatoria de junio de 2021

A mis padres.

Agradecimientos

En estas líneas quisiera dar las gracias a mis padres por su apoyo y ejemplo, a Paloma por su paciencia por todas las horas que he estado ausente, a mis tutores por sus revisiones y ayudas con las gestiones, y al resto de profesores del máster y a la UNED por los conocimientos adquiridos. Sin todos ellos este trabajo no hubiera sido posible.

Resumen

En este TFM se presenta el desarrollo de tipo prueba de concepto de un sistema basado en Apache Spark, a partir de un desarrollo en Hadoop, para procesado de datos astronómicos mediante la técnica de correlación para interferometría de base muy larga (VLBI). Esta técnica, que consiste en la sincronización y combinación coherente de señales de una red de radiotelescopios apuntando a la misma fuente de radiación, tiene aplicaciones en la toma de imagénes astronómicas y en geodesia.

Los sistemas software de procesado para correlación en VLBI utilizados en la actualidad están diseñados para ser ejecutados en clústeres dedicados, lo que ha dado lugar a desarrollos alternativos, entre los que se incluye un sistema basado en Apache Hadoop. Gracias a los avances en los últimos años en el ecosistema Hadoop, y en concreto en el framework de procesado paralelo Apache Spark, es posible incrementar el rendimiento de los sistemas mediante su adaptación para correr en estos nuevos frameworks.

En este TFM, además de describirse la planificación e implementación del sistema propuesto, se presentan las pruebas y resultados del despliegue preliminar del sistema en un servicio en la nube.

Palabras Clave

Astronomía, Interferometría de Base muy Larga (VLBI), Computación de Alto Rendimiento, Apache Hadoop, Apache Spark.

Índice

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

En esta sección se describen los objetivos de este Trabajo de Fin de Máster, se presenta una planificación del mismo, y se detallan las herramientas que se utilizarán para su realización.

Antes de entrar en materia se explicarán de manera resumida algunos conceptos básicos. El lector experto en correlacionadores para interferometría de base muy larga puede ir directamente a la Sección 1.2.

1.1. Ámbito

La interferometría es una disciplina que consiste en la combinación de ondas recibidas en múltiples antenas para conseguir más resolución[1]. Interferometría viene de interferencia, que es la combinación constructiva o destructiva de señales. Aunque se utiliza en múltiples áreas, en esta tesis nos centraremos en su aplicación en radioastronomía.

La radioastronomía es la parte de la astronomía que estudia la radiación electromagnética que emiten los cuerpos celestes. En contraste con los telescopios ópticos, que trabajan en el espectro visible (utilizan espejos en la superficie reflectora), los radiotelescopios trabajan en el espectro radio, típicamente en microondas (utilizando líneas conductoras en la superficie reflectora). Los telescopios ópticos y los radiotelescopios tienen en común que a mayor superficie de captación de radiación, mayor relación señal a ruido, y mayor resolución se puede conseguir en los objetos observados.

Lógicamente existen límites físicos estructurales sobre las superficies de captación que se pueden construir. Una idea para atajar estos problemas de viabilidad es la combinación de señales tomadas por varios de ellos mediante interferometría. En esta situación, los telescopios ópticos deben estar conectados físicamente por líneas de retardo medidas con mucha precisión, lo que dificulta esta técnica. Sin embargo, con radiotelescopios, es posible separarlos tanto como permite el tamaño de la Tierra, colocándolos en puntos opuestos del planeta, grabando las señales que reciben digitalmente, y procesándolas a posteriori. Esta técnica se conoce como interferometría de base muy larga, y es la responsable de que en los últimos años se hayan conseguido tomar imágenes de un agujero negro.

El procesado para la combinación de estas señales se conoce como correlación, y la realiza un sistema denominado correlacionador. Estos correlacionadores inicialmente eran dispositivos hardware hechos a medida, en los últimos años se han desarrollado varios sistemas software

[1] A lo largo de esta tesis se consideran como sinónimos a efectos del procesado los términos antena, estación, telescopio, radiotelescopio y receptor.

hechos para correr en clústeres dedicados, y en la actualidad se está intentando realizar este procesado en servicios en la nube.

Esta evolución se puede ver en muchos sistemas de alto rendimiento, y aunque cada tecnología tiene su nicho, la tendencia ha pasado de diseños de bajo nivel en infraestructura dedicada, generalmente con escalado vertical, a lenguajes de más alto nivel en infraestructura virtualizada con escalado horizontal. Es decir, se ha pasado de basar la mejora de rendimiento en aumentar la capacidad de las máquinas a añadir más máquinas. Este paso implica una reducción en costes pero también conlleva retos a la hora de diseñar la solución. En este sentido, sistemas de procesado paralelo como Apache Hadoop (inicialmente una tecnología para la realización de procesado de datos masivos mediante el paradigma map-reduce, actualmente un ecosistema de procesado paralelo) o Apache Spark (mucho más rápido, orientado a trabajar en memoria, y parte de este ecosistema) han ganado protagonismo en los últimos años.

Esta tesis presenta el diseño, implementación, y prueba preliminar en un servicio en la nube de un correlacionador basado en Spark. El sistema propuesto está basado en un sistema sobre Hadoop diseñado e implementado hace años por el autor de esta tesis en un centro puntero en correlación para interferometría de base muy larga. El trabajo que aquí se presenta no es una mera migración de tecnologías, si no que va más allá, buscando mejorar el correlacionador con contribuciones en rendimiento, sencillez de uso, arquitectura, fiabilidad, etc.

1.2. Objetivos

El objetivo del proyecto realizado para este TFM es el desarrollo de tipo prueba de concepto de un sistema de procesado de datos astronómicos en Apache Spark, en concreto de un correlacionador para interferometría de base muy larga (más conocido por sus siglas en inglés VLBI, Very Long Baseline Interferometry). Para ello se plantea la reutilización de un proyecto existente, desarrollado por este mismo autor entre 2015 y 2017 en un postdoctorado en el observatorio MIT Haystack, basado en Hadoop 2 y en Python 2.7.

El objetivo principal es el desarrollo de un correlacionador para VLBI basado en Spark 3 y en Python 3.8, que se puede desglosar en varios sub-objetivos:

- Diseño:
  - OBJ-DES-01: Comparación con soluciones existentes.
  - OBJ-DES-02: Diseño del sistema.
- Desarrollo:
  - OBJ-DES-03: Actualización de tecnologías del proyecto reutilizado.
  - OBJ-DES-04: Creación de entorno controlado de pruebas.
  - OBJ-DES-05: Introducción de mejoras.
- Prueba de concepto de correlación en Spark:
  - OBJ-DES-06: Correlación con datos de prueba en Spark en local.
- Correlación en la nube:
  - OBJ-DES-07: Correlación con datos de prueba en Spark en la nube.
  - OBJ-DES-08: Pruebas de escalabilidad en la nube.
- Validación:

- OBJ-DES-09: Validación de resultados.

Los objetivos educacionales son los siguientes:

- OBJ-EDU-01: Familiarización y aumento de experiencia con el entorno de procesado de Apache Spark.
- OBJ-EDU-02: Familiarización y aumento de experiencia con un proveedor de servicios en la nube para ejecución de tareas de procesado.

1.3. Estudio de Viabilidad

Este proyecto plantea varios retos:

- Reutilización de un proyecto de hace cuatro años: aunque este proyecto fue realizado por este mismo autor, su experiencia en Python en el momento era más limitada, por lo que el proyecto podría requerir una fase de refactorización para facilitar su reutilización.
- Envejecimiento de dependencias: relacionado con el anterior elemento está el tema de la tecnología utilizada, en concreto Hadoop 2 y Python 2.7. Aunque en su momento se intentó tener en cuenta este tema a futuro, la migración de Python 2 a Python 3 afecta a librerías que incluyen la lectura de ficheros binarios, algo que podría complicar el desarrollo.
- Cambio de paradigma de programación: los sistemas preparados para funcionar siguiendo un paradigma MapReduce están fuertemente anclados en el mismo, dado que este paradigma se tiene en cuenta a la hora de diseñar el sistema (e.g.: separación de la funcionalidad entre mapper y reducer [1]).

Por otro lado, existen varios factores positivos a tener en cuenta:

- El hecho de reutilizar un proyecto desarrollado por el mismo autor facilita la toma de contacto con el código a reutilizar.
- En su momento se prepararó un dataset pequeño con datos reales y con resultados de control para poder hacer pruebas en el futuro comprobando que todo sigue funcionando correctamente, así como una herramienta para generar datos de manera controlada, lo que facilitaría realizar pruebas con grandes datos si no se dispone de datos reales.
- El autor cuenta con varios años de experiencia extra tanto en gestión de proyectos como en desarrollo de sistemas en Python desde el desarrollo del anterior proyecto, lo que debería resultar en un uso más eficiente del tiempo dedicado al mismo.

1.4. Planificación

El desarrollo de este TFM se divide en varias épicas o bloques de trabajo que se describen a continuación, incluyendo sus tiempos estimados [2]. Cabe destacar que estos bloques no se desarrollan uno tras otro, si no que existe solape entre los mismos, tal y como se describe en la Figura 1.1.

[2]Para los tiempos estimados se utilizan equivalencias como si el proyecto se estuviera realizando a tiempo completo, es decir semanas de 5 días con jornadas de trabajo de 8 horas, dado que resulta más sencillo estimar tiempos de esta forma; aunque en el realidad el trabajo se realizará en el tiempo libre del estudiante por estar compaginando el desarrollo del máster con su trabajo.

- Revisión de estado del arte: 4 días.
  - Correlación para VLBI: revisión de avances en este campo en los últimos años.
  - Spark y alternativas: justificación de la migración de Hadoop a Spark y no a otra tecnología.
- Diseño: 4 días.
  - Arquitectura: división del sistema en bloques y consideración de elementos adicionales a Spark.
  - Generación del diagrama acíclico describiendo el procesado.
- Refactorización: 2 semanas.
  - Hacer código reutilizable: "wrapping" de ciertas funciones y refactorizaciones necesarias para hacer el código reutilizable.
  - Migración de Python 2 a Python 3: estudio del alcance y viabilidad de esta migración.
- Desarrollo del correlacionador: 4 semanas.
  - Prueba de concepto de procesado: pruebas preliminares en Spark para relizar un procesado representativo de lo que se pretende implementar.
  - Correlación básica: implementación básica que permita probar la funcionalidad completa.
  - Refactorización: revisión de la calidad del código generado.
  - Documentación de los detalles de la implementación.
- Prueba de concepto en la nube: 1 semana.
  - Configuración para la nube.
  - Generación de datos de prueba si no existe acceso a datos reales.
  - Pruebas preliminares de escalabilidad.
- Escribir memoria: 1 semana.
  - Documentación del estado del arte.
  - Documentación del diseño.
  - Documentación de la implementación.
  - Documentación de los resultados obtenidos.

Cabe destacar que aunque la tarea de escribir la memoria en la planificación se expande durante 3 meses, la dedicación es la indicada (1 semana) distribuyendo este tiempo durante ese período.

Se proponen además los siguientes hitos de revisión interna durante el desarrollo del proyecto:

- Hito 1: Documentación del estado del arte y prueba de concepto del procesado, aproximadamente un mes después de iniciar el proyecto.
- Hito 2: Documentacićon del diseño y correlación básica, aproximadamente un mes después del hito anterior

- Hito 3: Documentación de la implementación y prueba preliminar en la nube, aproximadamente un mes después del hito anterior
- Hito 4: Documentación de resultados, dos semanas después del hito anterior.

El objetivo de estos hitos es facilitar las tareas de desarrollo y de revisión diviendo el desarrollo del proyecto en 4 fases.

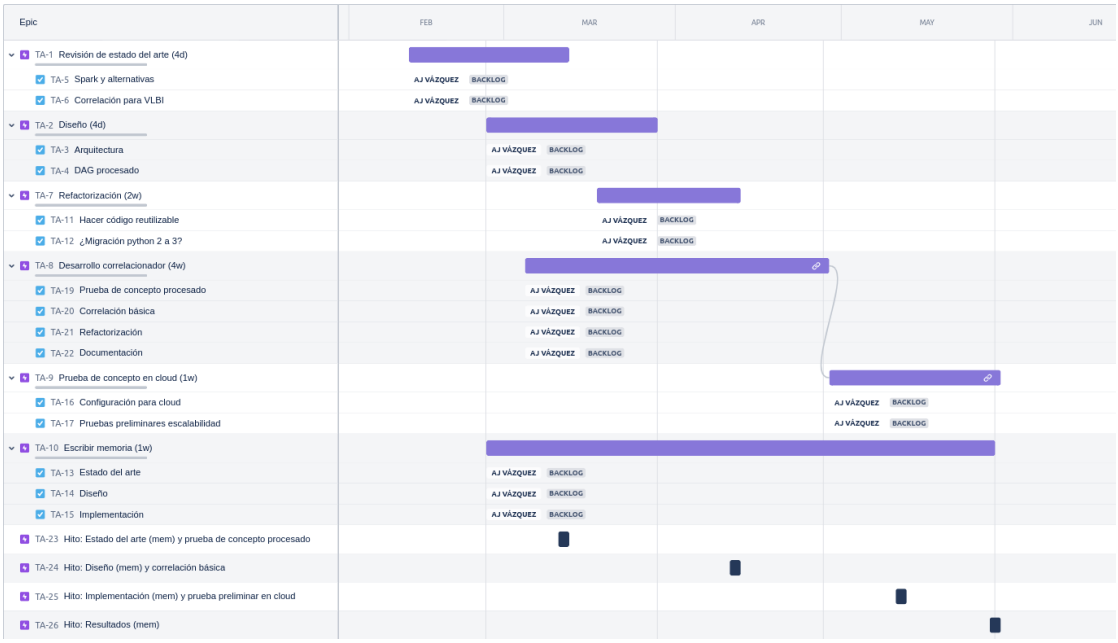


Figura 1.1: Planificación del proyecto.

La suma de los tiempos estimados hace un total de 48 días, que equivalen a 384 horas de trabajo. Teniendo en cuenta que se trata de una estimación y de que está dentro de unos márgenes cercano al número de horas recomendado en el programa de la asignatura (300 horas o 12 créditos ECTS), se considera que el alcance del proyecto es razonable.

El desarrollo del TFM se plantea para el segundo cuadrimestre de 2021, empezando desde mediados de febrero planteando como objetivo su presentación en la convocatoria de junio, es decir, un total de aproximadamente 4 meses. Considerando las 384 horas estimadas, se trata de una carga de trabajo de aproximadamente 24 horas por semana, que se podrían distribuir en 8 horas sábado, 8 horas domingo, y 2 horas cada día de lunes a jueves; siempre teniendo en cuenta que se trata de una estimación. Éste sería el peor caso; para el número de horas recomendadas (300 h) correspondería con algo menos de 19 horas por semana, que se podrían distribuir en 8 horas sábado, 8 horas domingo, y tres horas a repartir el resto de los días, por lo que se considera factible.

1.5. Herramientas

Para la realización de este TFM se han utilizado un ordenador portátil Intel x64 i7-3632QM 2.2GHz x8 y las siguientes herramientas software:

- Sistema operativo: Ubuntu 20.10 [3].
- Control de versiones: git [4].
- Documentación de la memoria: TexMaker [5].
- Lenguaje de programación: Python 3.8 [6].
- Motor de computación en paralelo: Apache Spark [7].
- Edición de imágenes: GIMP [8].
- Generación de diagramas: Dia [9].

Se han utilizado también los siguientes servicios en la nube:

- Gestión del proyecto: Jira [10].
- Alojamiento del proyecto y de la memoria: GitHub [11].
- Procesamiento en la nube: Amazon EMR [12].

En cuanto a presupuesto, para personal no se consideraron gastos de mano de obra; y para los servicios en la nube, teniendo en cuenta que se cuenta con los créditos proporcionados por la UNED para el servicio de AWS (de la asignatura de "Infraestructuras Computacionales para el Procesamiento de Datos Masivos"), no se considera necesario realizar gastos adicionales.

En cualquier caso, a continuación se presenta una estimación de costes si hubiera que plantear el proyecto en otras condiciones, supongamos para una empresa. Para los gastos en computación, es estiman 60 horas de uso de un clúster EMR de 6 nodos, lo que resulta en menos de 100 € [13]; y para el ordenador que se utilizará como entorno local se estiman aproximadamente 1000 €. Para mano de obra, se considera una tasa bruta aproximada de 20 € la hora [14], lo que resultaría en un total de 20 €/h·300h = 6000 €. Así, el presupuesto aproximado sería de 7100 €. Si hubiera que tener en cuenta la formación y experiencia del personal el coste de mano de obra debería sería mayor, y si hubiera que subcontratar el servicio los costes ascenderían todavía más.

1.6. Estructura del Documento

Este docuemento está dividido en los siguientes capítulos:

- Introducción: este capítulo, en el que se describen los objetivos y alcance de este TFM.
- Revisión de Estado del Arte: en este capítulo se hace una revisión de los avances en el campo de la correlación para interferometría de base muy larga (VLBI) en los últimos años.
- Diseño: en este capítulo se presenta el diseño de la solución desarrollada.
- Implementación: aquí se proporcionan los detalles de la implementación y se detalla el trabajo realizado.
- Prueba de Concepto en la Nube: en este capítulo se describen las pruebas de escalabidad realizadas en un servicio de computación en la nube.
- Evaluación: en este capítulo se analizan los resultados obtenidos.
- Conclusiones: finalmente se presentan las conclusiones de este proyecto, así como las líneas de trabajo futuro.

Capítulo 2

Revisión de Estado del Arte

En este capítulo se presenta una revisión de literatura relativa a la correlación de señales en interferometría de base muy larga, para evaluar la relevancia y contribuciones de este TFM.

2.1. Correlación para VLBI

La interferometría de base muy larga es una disciplina de la radioastronomía que consiste en la combinación, mediante técnicas complejas de procesado de señal, de las señales recibidas por dos radiotelescopios geográficamente separados apuntando hacia el mismo objetivo.

2.1.1. Conceptos Básicos

La idea principal detrás de esta técnica es la combinación de señales recibidas por radiotelescopios lejanos geográficamente apuntando a la misma fuente. Aunque los niveles de estas señales están muy por debajo del nivel de ruido, estas contribuciones de ruido en los distintos radiotelescopios no están correlacionadas (debido a que son recibidos en localizaciones separadas); por lo que las señales, que sí están correlacionadas pues provienen de la misma fuente, pueden ser combinadas. Así la correlación y acumulación durante diferentes ventanas de tiempo consigue levantar la señal por encima del nivel de ruido.

El procesado es complejo dado que requiere tener en cuenta el efecto de rotación de la Tierra durante la recepción de las señales, así como otros efectos relativistas [15]. Para la caracterización de los retardos que se debe aplicar a las señales para alinearlas se utilizan modelos polinómicos de retardo generados con el software Calc [16] de la NASA. Para complicar más el procesado, las señales sufren efectos no lineales en su transmisión desde la antena hasta los receptores, que son corregidos mediante sistemas de calibración de fase [17].

La correlación para VLBI es por tanto un área de difícil acceso ya que es necesario reunir conocimientos que permitan desenvolverse con soltura en radioastronomía, procesado de señal, y más recientemente en computación de alto rendimiento.

2.1.2. Aplicaciones

Esta técnica tiene aplicaciones principalmente en la obtención de imágenes y también en geodesia. En las siguientes líneas se intenta explicar a grandes rasgos en qué consiste cada una de estas áreas, y cuál es el papel de la correlación en cada una de ellas.

La toma de imágenes en astronomía ocurre en varias bandas del espectro electromagnético además de en el espectro visible, incluyendo rayos-X, el infrarrojo, o las microondas. La interferometría trabaja en el espectro de las microondas. La generación de la imagen se basa en el concepto del muestreo de la transformada de Fourier en dos dimensiones. Cada par de antenas capta un patrón de interferencias que se corresponde con un punto en la transformada en 2D de la imagen. Así, cuantas más antenas estén disponibles para ser combinadas, más patrones estarán disponibles para ser combinados. En la referencia [18] se dan unas nociones básicas sobre la composición de una imagen como la suma de las contribuciones de cada frecuencia espacial. Uno de los ejemplos más recientes en este campo es el de las imágenes del agujero negro M87 obtenidas hace unos pocos años [19], en 2017 a través del Event Horizon Telescope (EHT) [20]. El EHT es una red de radiotelescopios distribuidos en múltiples localizaciones del mundo elegidas cuidadosamente [21], generalmente zonas secas de gran altitud donde la atmósfera es menos densa.

La geodesia estudia la forma geométrica, rotación, orientación, y campo gravitatorio de la Tierra. El objetivo en este caso es el cálculo de manera muy precisa del retardo de las señales recibidas en los dos radiotelescopios apuntando hacia una fuente extragaláctica, mediante la correlación de estas señales. Esta técnica permite medir distancias con precisiones de medio milímetro [22]. En este área, la interferometría de base muy larga está detrás de hallazgos tan importantes como la confirmación de la tectónica de placas, con mediciones que se remontan a hace 50 años; la primera observación tuvo lugar en 1968 combinando los radiotelescopios de MIT Haystack (Estados Unidos) y de Onsala (Suecia) [23]. En la actualidad sigue siendo una técnica puntera utilizada en proyectos como el VLBI Global Observing System (VGOS) [24], con radiotelescopios que están siendo desplegados en la actualidad (2020).

2.1.3. Evolución

Los primeros correlacionadores estaban basados en hardware, y funcionaban como sistemas de streaming. Las señales de los radiotelescopios eran grabadas en sistemas de almacenamiento de alta capacidad y posteriormente enviadas al centro donde se hallaba el correlacionador, donde eran procesadas leyendo los datos de los distintos radiotelescopios simultáneamente, de manera secuencial. La salida de un correlacionador se conoce como "visibilidades", y son las componentes de la transformada de Fourier del resultado de la correlación para cada par de radiotelescopios, para cada ventana de acumulación y para cada canal (o banda de frecuencias). Tradicionalmente existen dos aproximaciones para la implementación, FX y XF, que hacen referencia al orden en que se realiza el procesado de la correlación (F hace referencia a la transformada de Fourier, y X al producto de cada par de señales).

Hasta finales de los 2000, las implementaciones estaban basadas principalmente en ASICs (Application Specific Integrated Circuits) y FPGAs (Field Programmable Gate Arrays) [25], y es entonces cuando tomaron protagonismo los correlacionadores por software. En 2007 se presenta DiFX (Distributed FX correlator) [25], una implementación que a día de hoy, tras varios cambios de version, es el estándar de facto en correlación para VLBI [26]. Inicialmente concebido para correr en clústeres Beowulf [27], actualmente se ejecuta en clústeres dedicados en los centros de investigación punteros en VLBI, como el observatorio MIT Haystack (Estados Unidos) [26] o el Instituto Max Planck de Radio Astronomía (Alemania) [28], entre otros. En la referencia [29] se incluye una lista con especificaciones de los clústeres de los usuarios de este sistema. Existen otros correlacionadores software con menor repercusión como SFXC (Super FX Correlator) [30], de 2015; y existen otras vías de desarrollo basadas en GPUs, pero con menor importancia que las anteriores, como xGPU [32], de 2011. Una de las características más interesantes de xGPU es que es código libre (licencia GPL). Existen otras alternativas para clústeres con GPUs como RASFX

[31]. Cabe destacar que RASFX, al igual que SFXC, puede generar resultados compatibles con la cadena estándar de procesado, sin embargo no se han encontrado datos sobre su licencia.

En cuanto a detalles de implementación, cabe destacar que aunque el procesado de estos datos tradicionalmente se trata "por lotes" (es decir, una correlación se considera terminada cuando se han generado todos los resultados), correlacionadores como DiFX o SFXC están implementados como sistemas de streaming (inspirados en los correlacionadores hardware iniciales), y generan los resultados incrementalmente. La arquitectura típica consiste en varios nodos leyendo datos enviando duplicados de los mismos a nodos que hacen las correlaciones, todo ello orquestado por un nodo de control que finalmente recoje los resultados.

Cabe destacar (principalmente por ser el correlacionador tomado como referencia en este TFM) que en 2017 MIT Haystack libera el proyecto desarrollado por el autor de este TFM, CorrelX [35]. Este proyecto nacía con tres objetivos: "escalabilidad, flexibilidad, y simplicidad" [1]. La implementación está basada en Hadoop streaming con Python, y se realizaron pruebas en un clúster compartido sobre datos reales que demostraron que el procesado era escalable. Aunque el proyecto se documentó en detalle, y abría la puerta a la entrada de desarrolladores en Python y a la ejecución en clústeres en la nube, el proyecto no ha tenido modificaciones desde la fecha de liberación. Se espera que este TFM sirva de empujón en el desarrollo de este correlacionador a través de una versión mejorada del mismo.

Para la implementación de CorrelX se tomó como aproximación inicial un procesado por lotes, por lo que Hadoop pareció un buen candidato, pero a la finalización del desarrollo de la prueba de concepto se consideró que frameworks como Spark podrían aportar mejoras en el sistema [1]. En cuanto a licencia, este software fue liberado con licencia MIT, lo que facilita su reutilización tanto para proyectos comerciales como de investigación.

Posteriormente en la Sección 2.3 se presenta una tabla resumiendo las características principales de cada uno de estos sistemas, donde se añade además el sistema propuesto a modo de comparativa.

2.2. Motores de Procesado Paralelo

La elección del motor de procesado es muy importante a la hora de diseñar un sistema de este tipo. Mientras que aproximaciones tradicionales orientadas al clúster dedicado se basaban en exprimir al máximo los recursos disponibles, los frameworks de paralelización en los últimos años han apostado por bajar un poco la optimización de recursos a cambio de conseguir mayor escalabilidad y trabajar con lenguajes de programación de más alto nivel.

2.2.1. Aproximación tradicional: C++ y MPI

Hace unos años la elección por defecto para desarrollar sistemas de alto rendimiento era C++ y librerías de paso de mensajes (MPI) [33] para la comunicación entre los nodos y OpenMP en sistemas de memoria compartida [34]. Ésta (C++ y MPI) es de hecho la elección para los correlacionadores considerados actualmente estándar de facto en VLBI. Si bien estos desarrollos exprimen al máximo las máquinas (y son la elección preferida para dispositivos "embebidos"), estos desarrollos presentan problemas a la hora de escalar, tanto en distribución de carga como en complejidad.

Así, en el caso concreto de los correlacionadores, el diseño de un clúster para correlación requiere especial cuidado en el dimensionado de la capacidad de las conexiones entre los nodos. Este tipo de diseños complica así en principio la ejecución en clústeres compartidos, y dificultan potenciales contribuciones por la curva de aprendizaje requerida para poder contribuír a la base de código.

No obstante, existen corrientes de investigación trabajando en la dirección de ejecutar estas correlaciones en la nube como la presentada en la referencia [36], donde se evalúan costes de esta aproximación en Google Cloud Platform (GCP) con máquinas de tipo "n1-highmem-16" a "n1-megamem-96", con costes por nodo entre los $0.94 y los $10.67 por hora respectivamente, resultando en costes de miles de euros por la ejecución de cada experimento. Estos estudios se centran en escalabilidad, pero siguen sin facilitar la apertura a la aplicación de técnicas de ciencia de datos como el aprendizaje automático a este procesado.

2.2.2. Tecnologías para Ciencia de Datos: de Hadoop a Spark

Apache Hadoop [38] es un framework (en la actualidad se ha convertido en un ecosistema) de procesamiento distribuido que trabaja sobre el paradigma MapReduce. Un programa diseñado con este paradigma se divide en dos partes, la primera "map", en la que los datos de entrada se dividen en bloques y cada uno de ellos es procesado por un proceso de este tipo; y posteriormente, tras una reordenación de los resultados por clave, estos se distribuyen en otros procesos "reduce" que (valga la redundancia) reducen los datos.

Apache Spark [7] es un framework de computación en paralelo que trabaja sobre el ecosistema de Hadoop (no lo reemplaza sino que lo mejora [39]), y está optimizado para trabajar en memoria. Apache Spark es inherentemente más rápido que Hadoop (100 veces más rápido en memoria [7, 40, 41]), incorpora funcionalidades adicionales de Streaming además de trabajar por lotes, y presenta importantes optimizaciones en cuanto al procesado. Internamente trabaja con un "dataset distribuído resiliente" (RDD), con lo que es tolerante a fallos, y únicamente ejecuta los cálculos que necesita, distribuyéndolos de la mejor manera posible entre los nodos del clúster.

Además de estas ventajas, Spark presenta APIs para múltiples lenguajes de programación de alto nivel, así como librerías específicas de aprendizaje automático; y es posible ejecutar tareas sobre servicios gestionados fácilmente. Esto último facilita la separación en capas de estos sistemas, facilitando la especialización de los desarrolladores a las distintas áreas de interés: infraestructura, rendimiento, ciencia e investigación.

2.2.3. Tendencias

A continuación se muestra la evolución del interés en las tecnologías citadas a lo largo del tiempo. En la Figura 2.1 se muestran datos de Google Trends sobre búsquedas en varios motores de paralelización [42], y sobre varios lenguajes de programación utilizados en computación de alto rendimiento [43]. Sobre estos datos es posible comprobar que en la década de los 2000 tenían gran protagonismo desarrollos basados en C++ y librerías de memoria compartida o distribuida, mientras que en la actualidad destacan desarrollos en Python sobre frameworks de paralelización en la nube.

2.3. Contribuciones de este Proyecto

A continuación se describen las contribuciones de este proyecto en comparación con las soluciones existentes de software para correlación en VLBI:

- Arquitectura por capas: en línea con su predecesor (CorrelX), la arquitectura por capas del desarrollo propuesto facilita la entrada al código a desarrolladores interesados únicamente en ciertas áreas de la correlación para VLBI. El hecho de que este proyecto se considere viable confirma las ventajas de esta arquitectura (debido a que las capas que se pretende atacar en este proyecto son las de infraestructura y rendimiento).

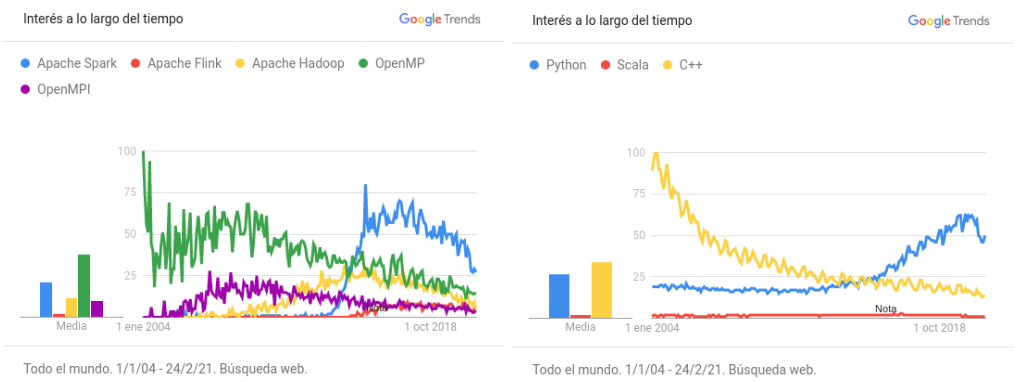


Figura 2.1: Tendencias en motores de paralelización y lenguajes de programación en los últimos años. Fuente de datos: Google Trends (www.google.com/trends).

- Rendimiento: debido al incremento de rendimiento inherente a Spark en comparación con Hadoop, se espera conseguir una mejora de rendimiento en comparación con el correlacionador CorrelX.
- Curva de aprendizaje: el cambio de Hadoop a Spark baja un poco la curva de entrada para poder contribuir al proyecto CorrelX por parte de desarrolladores recién llegados. Este proyecto desacopla un poco la implementación del paradigma MapReduce.
- Licencia: también en línea con su predecesor (CorrelX), el desarrollo propuesto será liberado con licencia MIT, lo que proporciona facilidades burocráticas de cara a reutilizaciones futuras en comparación con otros correlacionadores con licencias más restrictivas.

Se presentan a continuación en la Tabla 2.1 a modo de resumen las características de los correlacionadores de mayor uso descritos en párrafos anteriores, donde M. P. hace referencia al motor de paralelización.

Para facilitar la comprensión de las contribuciones presentadas en este TFM, en la Figura 2.2 se representan en un gráfico de radar las características de dos correlacionadores que tomaremos como referencia, DiFX (por ser el estándar en correlación para VLBI), y CorrelX (por ser el sistema legado del cual se hereda código), junto con las del sistema propuesto. Las características consideradas son las siguientes:

- *Arquitectura*: se proponen tres clases, compleja, sencilla, y más sencilla. Se considera que la arquitectura de DiFX es compleja dado que hay varios roles posibles para los nodos que deben ser configurados manualmente [25], en CorrelX esto se simplifica dado que todos los nodos son polivalentes, y en el sistema propuesto se simplifica (esta simplificación se presenta en la Sección 3.1). Este parámetro es importante, no sólo para el acceso de nuevos desarrolladores al proyecto, sino también para el mantenimiento y mejora del mismo.
- *Desarrollo*: en este caso se diferencia entre inactivo y activo, considerando un proyecto inactivo si la fecha indicada en el repositorio donde se hospeda es anterior a un año de la fecha de escritura de esta tesis (momento en que se consultaron estos repositorios), y activo en caso de que esa fecha sea reciente, o en su defecto existan artículos publicados con referencias a desarrollos o resultados recientes. Este parámetro es importante de cara a la necesidad de actualización del código una vez accedido al proyecto, y parte de las

Tabla 2.1: Características más importantes de los correlacionadores.

	Inicio	*M. P.*	*Lenguaje*	*Licencia*	*Ventajas*	*Desventajas*
DiFX	2007	MPI	C++	?	Rendimiento Estándar de facto Desarrollo activo	Arquitectura compleja Orientado a clúster dedicado Comunidad limitada
xGPU	2011	GPU	CUDA	MIT	-	Requiere hardware específico Sin desarrollo desde 2018
RASFX	2014	GPU	?	?	Desarrollo activo Salida compatible con estándar	Requiere hardware específico
SFXC	2015	MPI	C++	GPL	Salida compatible con estándar	Sin desarrollo desde 2017
CorrelX	2017	Hadoop	Python	MIT	Arquitectura sencilla Orientado a datos masivos E/S compatible con estándar	Prueba de concepto Sin desarrollo desde 2017
Sistema propuesto	2021	Spark	Python	MIT	Arquitectura sencilla Orientado a datos masivos E/S compatible con estándar Rendimiento	Prueba de concepto

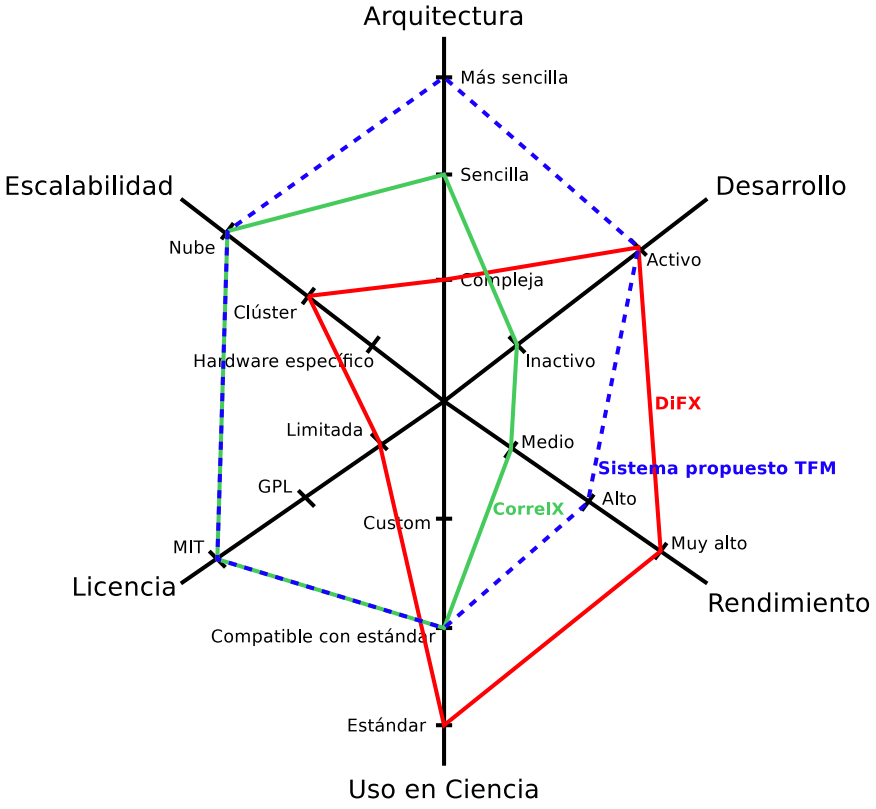


Figura 2.2: Comparación cualitativa de DiFX (rojo), CorrelX (verde), y el sistema propuesto (azul) con respecto a varias características.

contribuciones relativas a la mejora y actualización del código heredado de CorrelX se describen en el Capítulo 4.

- *Rendimiento*: posiblemente el indicador más importante de todos de cara al usuario final. En este caso se consideran tres clases: medio, alto, y muy alto. La separación inicial parte de prototipo por el rendimiento de CorrelX, cuyo estado es el de prueba de concepto, muy alto por el de DiFX que es la referencia en el sector, y se añade alto pues uno de los objetivos del sistema propuesto es recortar distancias con DiFX en este parámetro. Estos resultados se muestran en el Capítulo 5.

- *Uso en Ciencia*: se consideran las clases "custom" para desarrollos independientes, compatible con el estándar para aquellos que pueden reemplazar por completo a DiFX en la cadena de procesado, como es el caso de CorrelX, y estándar para DiFX por ser el estándar de facto en Ciencia para procesado de datos. Dado que el sistema propuesto hereda las herramientas y formatos de CorrelX, se sigue manteniendo compatibilidad con el estándar. Este parámetro también es muy importante para el usuario final, ya que es importante poder poder utilizar las herramientas de postprocesado existentes, pues son las que proporcionan al usuario final el resultado que busca.

- *Licencia*: en este caso se considera que código libre es mejor que código privativo pues favorece su reutilización (especialmente en entornos abiertos de investigación), y también que licencias como MIT mejoran la GPL al facilitar el uso comercial del software licenciado. De nuevo, ésta clasificación también podría considerarse subjetiva. Para mantener un criterio objetivo, se considerará que menos restricciones sobre la licencia es mejor que más restricciones. Mientras que CorrelX tiene licencia MIT y el código está disponible en github, el código de DiFX no es accesible de manera inmediata.

- *Escalabilidad*: hace referencia al entorno para el cual nació el correlacionador, así se considera que orientado a la nube es mejor que a un clúster dedicado, que a su vez es mejor que orientado a hardware específico como GPUs. Se considera que el ecosistema Hadoop está orientado a la nube (CorrelX, y por tanto el sistema propuesto también al trabajar sobre Spark, que forma parte de este ecosistema), mientras que teconologías como MPI estarían orientadas a clústeres dedicados (DiFX) [25]. Esto a pesar de que hay publicaciones recientes [36] relativas a la migración de DiFX a la nube, pero las pruebas presentadas en esta referencia consideran un único nodo, con pruebas de escalabilidad vertical (número de núcleos). Así, posiblemente sería más correcto hablar aquí de escalabilidad horizontal, que consideraría el incremento del número de máquinas). En la referencia [45] se presentan resultados de escalabilidad para el sitema legado, escalando horizontalmente para más de 100 nodos.

Cabe destacar de nuevo que muchas de estas clasificaciones podrían considerarse subjetivas. El objetivo que se persigue en este apartado no es conseguir una taxonomía exacta de estos correlacionadores, sino facilitar al lector la comprensión de las diferencias entre estos sistemas, y describir las contribuciones presentadas en este TFM.

De esta representación podemos extraer varias conclusiones:

- DiFX nació orientado a rendmiento, mientras que CorrelX cuidó aspectos orientados a desarrollo que facilitaran el crecimiento del proyecto.

- El sistema propuesto no es una mera migración de motor de paralelización (de Hadoop a Spark), si no que busca simplificar la arquitectura, activar el desarrollo, y recortar distancias con DiFX en cuanto a rendimiento.

2.4. Resumen

En este capítulo se han introducido varios conceptos básicos relativos a la correlación para interferometría de base muy larga, se han comparado las soluciones existentes en correlación, y se han presentado varias alternativas para motores de procesado en paralelo. Estos conceptos básicos serán de utilidad para comprender el diseño presentado en el siguiente capítulo.

Capítulo 3

Diseño

En este capítulo se presenta el diseño del sistema propuesto.

3.1. Arquitectura del Sistema

Para la arquitectura del sistema se consideraron principalmente dos opciones:

- *Procesado por lotes*. Con esta aproximación la idea es mantener el procesado heredado por lotes, sujeto a las adaptaciones necesarias para poder correr estas tareas en Spark, tenidendo en cuenta aspectos como el correcto particionado de los datos para facilitar el balanceo de carga. En cuanto a la implementación, se consideraría el encapsulado de las partes del procesado en unas clases ejecutoras o "workers".
- *Streaming*. Se estudió en detalle la posibilidad de montar una arquitectura basada en streaming con una cola de mensajes entre los nodos lectores de los datos y los que realizan la correlación de bajo nivel. En cuanto a implementación, implicaría el uso de Spark Streaming [46] recibiendo streams de procesos leyendo los ficheros binarios de entrada a través de Apache Kafka [47]. Esta arquitectura también parece una opción natural si recordamos el origen de los correlacionadores software a partir de sistemas hardware de procesado de streams (Sección 2.1.3).

Dado que el procesado heredado es por lotes, esta parece la opción más clara para iniciar el desarrollo. En cualquier caso, en las siguientes líneas se estudian los pros y contras de cada una de las opciones, y finalmente se justifica la opción elegida. Esta discusión pretende facilitar líneas futuras de implementación.

1. La opción de streaming podría ser interesante de cara al procesado en tiempo real (es una línea activa de investigación [48]), o incluso la incorporación del postprocesado de manera iterativa según se van generando resultados. Este procesado iterativo parece que surge de manera natural con la opción de streaming debido a la relación más directa entre los tiempos con los que están etiquetados los datos y los tiempos de procesado de los mismos.

   Sin embargo, este postprocesado iterativo es posible conseguirlo también con el procesado por lotes, dado que la última etapa puede realizarse de manera paralela (los datos de salida de la etapa de reducción no necesitan ser combinados en su conjunto para ser procesados), con lo que bastaría con planificar de una manera concreta el particionado para así conseguir el orden de ejecución de tareas deseado. El procesado conjunto se realizaría de manera

incremental, cubriendo una ventana de tiempo que se iría ampliando conforme se reciben los resultados correspondientes a subsiguientes ventanas de acumulación.

2. Por otra parte, en streaming generalmente se busca un compromiso entre latencia y precisión [49]. Así, dado que en estos procesados tradicionalmente se intenta garantizar que se procesan todos los datos, el procesado en streaming podría no resultar válido, sin embargo su uso podría ser interesante para una evaluación en tiempo real de la toma de datos, a modo de monitorización. Esto permitiría detectar problemas en tiempo real en la toma de datos, y no posteriormente durante el procesado.

   En cuanto a la reducción de precisión, aunque el procesado aproximado para VLBI es una línea de investigación que sería interesante abrir (es llamativo el hecho de que es posible reconstruir información de la señal de origen muestreándola muy por debajo del nivel de ruido con muestras de tan sólo uno o dos bits), para una primera aproximación se considera necesario buscar una implementación de referencia que una vez rodada pueda ser adaptada a este tipo de escenarios.

3. Siguiendo con la arquitectura en streaming, a priori se consideraría únicamente la arquitectura Kappa (únicamente rama en tiempo real) por simplicidad frente a la Lambda [49]; de considerar la segunda el camino sería comenzar con la arquitectura por lotes y posteriormente investigar sobre procesado aproximado para disponer de resultados antes. En cualquier caso, la correlación es un procesado que tradicionalmente se realiza offline, por lo que desde este punto de vista tiene más sentido comenzar con un procesado por lotes.

4. Otro de los problemas de la arquitectura en streaming es la complejidad, ya que requiere infraestructura adicional para el paso de mensajes, y una gestión manual de los procesos productores, algo que no encaja con uno de los puntos clave de la filosofía del correlacionador inicial: simplicidad [1].

La arquitectura elegida es por tanto la opción de *procesado por lotes*. Es interesante el hecho de que utilizar Spark como motor de procesado abre la puerta a incorporar en el futuro soluciones basadas en streaming. Esta mejora se deja como línea futura, sobre la que se dejan planteadas las recomendaciones indicadas en la Sección 2.1.3.

3.2. Organización del Código

Parte importante de las contribuciones de este proyecto es la reducción de la curva de aprendizaje para potenciales desarrolladores al proyecto, con lo que se ha dedicado esfuerzo a la refactorización del proyecto para mejorar su arquitectura.

3.2.1. Estructura General

Las arquitecturas de CorrelX y del sistema propuesto se describen brevemente en la Figura 3.1. Se observa una separación marcada en capas, lo que a priori facilita la separación de responsabilidades. A pesar de esta separación, para la organización inicial del código en CorrelX se había optado por incluir todos los ficheros en la misma carpeta `src`, principalmente para facilitar su ejecución en Hadoop Streaming, donde es necesario enviar todos los archivos requeridos para ejecutar la aplicación como parte de la tarea MapReduce enviada al clúster Hadoop. Además se incluían importaciones generales, que es una mala práctica, y de hecho dificulta el mantenimiento (esto se ha solucionado en el sistema propuesto en esta tesis). La nueva organización del código, más cercana a la arquitectura por capas se muestra en la Figura 3.2.

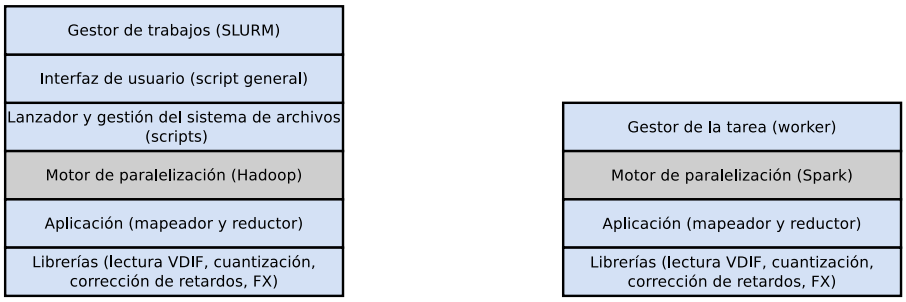


Figura 3.1: Arquitectura por capas: (a) CorrelX y (b) sistema propuesto.

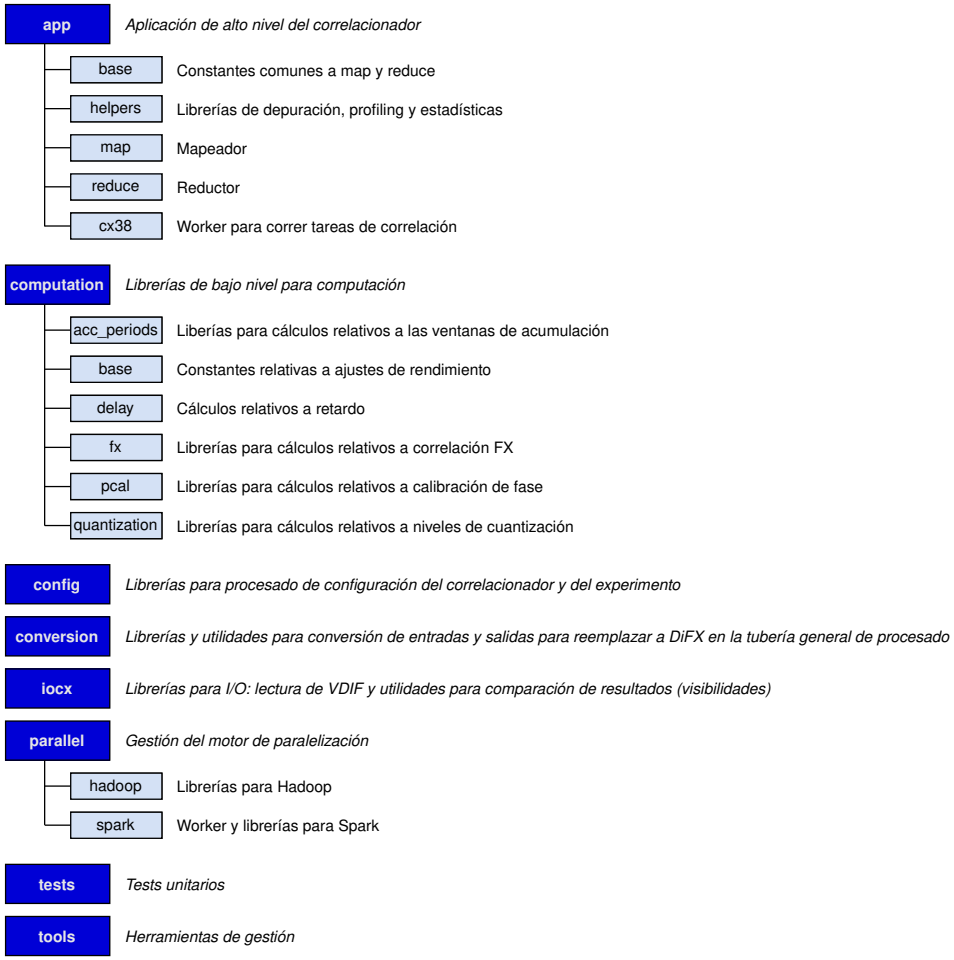


Figura 3.2: Organización del código en el sistema propuesto.

Se ha intentado mantener compatibilidad hacia atrás con la estructura antigua (aunque en capas diferenciadas, con una distribución plana de ficheros), para mantener la posibilidad de realizar pruebas contra la infraestructura de paralelización de referencia (Hadoop). Esto se ha realizado básicamente reemplazando la carpeta con las fuentes por enlaces simbólicos a los ficheros de la nueva arquitectura, y manteniendo dos casos de importación al inicio de todas las fuentes, dependendiendo de si corren con la arquitectura antigua o la nueva. Esta decisión de diseño se ha pensado bastante, ya que mantener compatibilidad hacia atrás es útil en esta etapa de prueba de concepto pues facilita la ejecución de tareas utilizando la infraestructura de Hadoop, pero introduce complejidad al ser necesario mantener esta lógica extra. Así, a largo plazo posiblemente habría que primar la simplicidad del proyecto y quedarse únicamente con la nueva arquitectura y con el nuevo motor de paralelización.

3.2.2. Componentes

En cuanto a la organización de clases, se plantea un script general para el procesado de los argumentos, que crea una instancia con la clase del correlacionador. Esta clase heredará de otra (que contine los wrappers para gestionar la correlación), y contiene los métodos necesarios para gestionar el procesado en Spark. Así, el código pasa de una organización plana a estar distribuído en los siguientes módulos:

- *App*: incluye la infraestructura de paralelización, es decir, toda la funcionalidad relativa a mapeador y reductor, convenciones y definiciones de interfaz entre los mismos. Este es un módulo clave en este sistema y también uno de los más complejos. Los conocimientos necesarios para poder contribuir en este módulo tienen que ver con el procesado de señal de alto nivel, alineación de streams, y computación de alto nivel.

- *Computation*: incluye los cálculos de bajo nivel para la realización de la correlación, cálculo de retardos, calibración de fase, cuantización, etc. Este es otro módulo clave del sistema ya que interviene en la precisión del proceasdo. Para poder contribuir en este módulo son necesarios conocimientos específicos de procesado de señal para radioastronomía.

- *Config*: agrupa diversas librerías de configuración. La aproximación tomada en su definición (está definido de manera flexible para que el usuario pueda agregar parámetros concretos de Spark fácilmente) facilita la entrada para la contribución en este módulo.

- *Conversion*: incluye varias herramientas de conversión de formatos de ficheros para su integración en la tubería de procesado de DiFX. Para poder contribuir en este módulo es necesario un conocimiento de bajo nivel de la cadena de procesado de DiFX.

- *IOcx*: librerías de entrada/salida, importantes para el procesado de los ficheros binarios de entrada. Para la contribución en este módulo son necesarios conocimientos de bajo nivel de lectura de binarios, y familiaridad con los estándares de codificación de señales en radioastronomía. Este módulo es importante pues forma parte del primer eslabón de la cadena de procesado, siendo un factor crítico para el rendimiento.

- *Parallel*: liberías de gestión del motor de paralelización. Para la contribución en este módulo es necerio conocimiento en detalle del motor de paralelización. En el sistema legado era Hadoop, por lo que el módulo correspondiente incluía varias utilidades para la gestión de ficheros y la preparación de la tarea para ser enviada al clúster. En el caso del sistema propuesto, sigue siendo necesario un conocimiento detallado del motor de paralelización (en este caso Spark), pero Spark pone las cosas más fáciles que Hadoop a la hora de interactuar

con él, de manera que la parte del módulo correspondiente al mismo se simplifica con respecto a la de Hadoop.

- *Test*: infraestructura de pruebas. La generación y mantenimiento de una infraestructura de pruebas es clave en el desarrollo de sistemas software, especialmente aquellos complejos. Aunque en esta tesis se plantean unos tests básicos, queda planteado como línea futura el desarrollo de tests unitarios concretos para cada módulo.
- *Tools*: herramientas generales, actualmente dedicadas al mantenimiento de la compatibilidad hacia atrás.

El diseño de alto nivel de cómo estos módulos interactúan se describe brevemente a continuación:

- El módulo *app* incluirá una clase worker sobre el que se definen unos métodos para el procesado de los ficheros de configuración y la realización de las etapas principales del procesado. Esta clase facilita así el acceso a las acciones de procesado. El esfuerzo principal en este módulo es por tanto el de refactorización.
- El módulo *parallelization* incluirá otra clase worker, que heredará de la clase anterior para acceder a estos métodos de procesado, y sobre ella se añadirán unos métodos para la realización de las distintas etapas del procesado en Spark. El esfuerzo en este caso es de desarrollo con una componente importante de depuración.

Este diseño se resume en la Figura 3.3. Cabe destacar que únicamente se incluyen los módulos más importantes para simplificar el diagrama (principalmente los de aplicación y paralelización), debido a que algunos de estos módulos se utilizan fuera del procesado estrictamente relacionado con la correlación, como pueden ser librerías de conversión de formato, generación de representaciones visuales, etc.

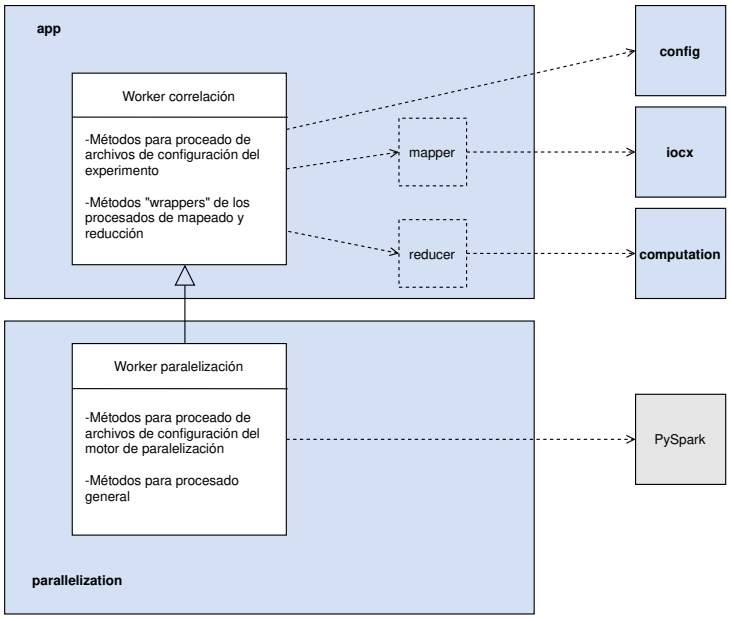


Figura 3.3: Diseño de alto nivel de las clases principales del sistema propuesto.

3.2.3. Discusión

La decisión de herencia en lugar de composición va unida a la filosofía seguida de considerar el correlacionador que corre en Spark como un caso concreto de la manera en que se ejecuta el procesado, y no como algo externo al mismo. Así el worker de correlación (que encapsula la funcionalidad del correlacionador), simplemente define unos atributos relativos a particionado y unos métodos de inicialización, además de los métodos de mapear y de reducir. Podría ser interesante definir otro método en la clase del correlacionador con la misma firma que el definido en la clase de paralelización para lanzar el procesado en "tubería", reemplazando así la implementación legada, pero por simplicidad se deja como línea futura. Este paso sería necesario para eliminar la dependencia del la cadena de procesado en "tubería" legada. Por otra parte, se ha utilizado la composición para definir clases que encapsulan la configuración del correlacionador y la del experimento respectivamente, y que son utilizadas por la clase del correlacionador.

En cuanto a la visión de conjunto, el desarrollo legado está basado principalmente en funciones. Esta decisión se tomó inicialmente por sencillez, pero a largo plazo sería beneficioso un replanteo de la arquitectura de la aplicación (módulo *app*) basado en clases. Por restricciones de tiempo y dado que no entra dentro del alcance del proyecto, esta refactorización se deja también como línea futura.

3.3. Etapas de Procesado

En este apartado se explica en detalle el procesado que realiza la aplicación orientada a la computación en paralelo, para entender qué tareas hay que realizar, en qué orden, cuáles de ellas son paralelizables, etc. Así, se darán unas nociones sobre el procesado de bajo nivel, pero sin perder el foco en la computación en paralelo en Spark.

El procesado se compone de las siguientes etapas:

- *Lectura de ficheros de entrada.* Este paso puede ser realizado en paralelo mediante el método `binaryFiles()` del contexto de Spark. Normalmente se dispone de tantos ficheros de entrada como estaciones participan en el experimento. Sería posible aumentar el grado de paralelización de esta etapa dividiendo los archivos de entrada en bloques, ya que el formato de estos ficheros (principalmente VDIF [44]) contiene metadatos que permiten su división.

- *Procesado de los datos de entrada.* Este paso también será realizado en paralelo por Spark, cada ejecutor tomará como entrada los datos leídos de un fichero y generará registros de salida. Así, habrá tantas tareas de este tipo como ficheros de entrada (o bloques de entrada, si estos ficheros de entrada están particionados). Estas tareas se corresponden con `msvf`, el "mapper" de CorrelX con las modificaciones necesarias para ser integrado en esta nueva cadena de procesado. Cada registro de salida contiene una clave que identifica a la partición donde se realizará el procesado de reducción, metadatos con información relativa a la posición de las muestras dentro del stream, y por supuesto, las muestras.

- *Repartición y ordenación de los registros generados.* En este punto, se tienen los datos repartidos en tantas particiones como ficheros de entrada (tantas como procesos se lanzaron en la etapa anterior). Esta tarea es de transición, y consiste en la repartición de los datos en función de su clave, de manera que los datos que corresponden al mismo canal y a la misma ventana de acumulación van a la misma partición. Además, los datos deben ser ordenados dentro de la misma partición para que en el paso de reducción se reciban las muestras de manera ordenada.

- *Reducción de los datos de cada partición.* En este punto ya se dispone de tantas particiones como claves de particionado (básicamente habrá una partición por canal y ventana de acumulación), con lo que pueden iniciarse las tareas de reducción. Estas tareas corresponden con el módulo `rsvf`, el "reducer" de CorrelX, con las modificaciones necesarias para su integración. Este paso es el que tiene el mayor peso del procesado, y es donde debería dedicarse esfuerzo en optimización.

- *Escritura de los resultados en disco.* Tras la terminación de cada una de las tareas, sus datos son escritos en disco. Aunque sería posible combinar todos los datos en el driver de Spark, dado que en este punto no es necesario realizar más procesado, se considera que ésta es la mejor opción por escalabilidad (de hecho se llegó a esta opción, junto con la codificación de los resultados de salida, como se explicará posteriormente). Esta escritura distribuida ha tenido que indicarse explícitamente en la implementación.

- *Unión de los resultados.* Al final del procesado se incorpora una tarea de unión de los datos en disco en un único fichero por compatibilidad hacia atrás.

Es fácil observar el paralelismo entre estos pasos y el paradigma mapreduce, ya que se ven claramente dos etapas, una de mapeo y otra de reducción, y también una intermedia de repartición que debe ser cuidada para evitar problemas de rendimiento. Estos cuidados tienen que ver con una definición adecuada de las claves de los registros (incluida en el sistema legado) y una asignación adecuada de particiones basadas en estas claves (añadida en la implementación), así como una configuración correcta de Spark.

Estos pasos se muestran en la Figura 3.4, en la cual se representan aquellos pasos que corresponden con tareas paralelizables con bloques superpuestos.

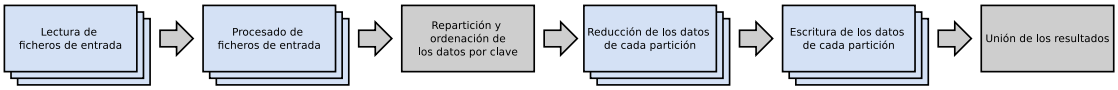


Figura 3.4: Diagrama de procesado.

3.4. Plan de Pruebas

La depuración de software de computación en paralelo no es para nada trivial, y es necesaria la comparación de resultados con unos datos de referencia. Así, para poder iniciar el desarrollo de este proyecto, el primer paso de cara a la implementación es generar una infraestructura de tests unitarios que permita validar rápidamente las modificaciones que se aplicarán al software. Aunque es de sobra conocido el principio básico de Dijkstra de que "los tests pueden probar la presencia de bugs pero no su ausencia" [51], su uso es muy importante para incrementar la fiabilidad del sistema y evitar regresiones.

El proyecto de referencia no dispone de tests unitarios, aunque sí de herramientas para comparar resultados de una correlación, de datos de entrada de prueba y de resultados de referencia (asociados a estos datos de entrada), así que se utilizarán éstos para preparar la infraestructura indicada. Así, se propone la siguiente lista de pruebas:

- Tests de referencia:

    - Test básico de referencia. Todos los tests dedicados al desarrollo se basan en la comparación de resultados de salida de la realización de un procesado con resultados de

referencia mediante una herramienta del sistema legado. Por tanto, es importante disponer de tests que ayuden a detectar problemas específicamente en esta herramienta de comparación.

- Tests de desarrollo:
  - Test de cadena de referencia (tubería), arquitectura antigua. Esta prueba consiste en la realización de una correlación completa con el software dispuesto según la arquitectura de referencia, lo que facilita la detección de problemas en la compatibilidad hacia atrás. Es decir, su definición toma como referencia la salida que genera el proyecto legado, de manera que si existe un error que afecta a la realización de este test, pueden descartarse problemas relativos a la infraestructura de paralelización. [1]
  - Test de cadena de referencia (tubería), arquitectura nueva. Sigue probando el procesado tipo tubería, pero esta vez utilizando la nueva arquitectura. Este test es importante porque la nueva arquitectura es la utilizado en el procesado con Spark, así errores en este test pero no en el de la arquitectura plana indicarían problablemente un problema relacionado con la gestión del código.
  - Test de cadena con paralelización en Spark. Este test probaría el procesado para los mismos datos de entrada que el resto de tests, y comprobaría que los resultados son muy parecidos, es decir, que comprueba que el error que proporciona una función de comparación entre ambos resultados es despreciable.

3.5. Resumen

En este capítulo se ha descrito en detalle el diseño del sistema presentado en este TFM, y se han justificado varias decisiones de diseño. Además se ha presentado un plan pruebas que permitirá soportar la implementación, presentada en el siguiente capítulo, con unas garantías básicas de fiabilidad.

[1] Además se añadió una variante de este test manteniendo la ordenación antigua, pero finalmente se optó por mejorar la herramienta de comparación de resultados para ignorar el orden en que estos aparecen (ya que no es relevante y no afecta a los mismos).

Capítulo 4

Implementación

En este capítulo se describe en detalle la implementación. Dado que el código del proyecto está gestionado mediante un repositorio `git`, se tendrá en cuenta el histórico de desarrollo detallando la correspondencia entre la implementación y los "commits" del mismo. [1]

Además se describe una ejecución paso a paso de una correlación en una instancia en local. En el siguiente capítulo se dará el salto a la nube.

4.1. Migración de Python 2 a Python 3

La migración de Python 2 a 3, más allá de actualizar las versiones de las dependencias y añadirlas en un archivo `requierements.txt`, incluye la modificación de la librería de lectura de ficheros binarios. Anteriormente se utilizaba la función `fromfile()` de numpy, que se reemplazó por `frombuffer()`. Además se añadió la gestión de una excepción para detectar el fin del stream de lectura. Estos cambios se recogen principalmente en el commit id 9d1d6c14.

4.2. Tests Unitarios

Los tests desarrollados están destinados a garantizar que la precisión del sistema propuesto es la misma que la del sistema legado. Para su desarrollo se siguió la estrategia descrita en la Sección 3.4:

- Test de referencia: distintas arquitecturas (commit id dfc8a81f).
- Tests de desarrollo: ordenación numérica (commit id fb078b9a), Spark y estructura de código legada (commit id cc16b87e).

4.3. Refactorización

El trabajo de refactorización tiene que ver principalmente con la organización del código siguiendo la estructura presentada en la sección 3.2.2, (commit id be943374). Como se indicó en esa sección, mapeador y reductor se envolvieron con "wrappers" (commit id b2c6d43a), y se encapsularon las configuraciones del experimento y del correlacionador en objetos para facilitar

[1] Posiblemente hubiera sido más correcto utilizar el término en español confirmación o consolidación, pero dado el uso extensivo de esta herramienta en inglés se mantendrá el término original "commit".

su gestión (commit id a76c9710). Además, se realizó una limpieza de llamadas a la librería `imp`, utilizada en el pasado debido al desarrollo del proyecto en IPython Notebooks, más recientemente conocido como Jupyter Notebooks [53] (commit id 7db768d4).

4.4. Corrección de Bugs

Durante el desarrollo se trabajó con varios datasets [2] distintos al proporcionado en el repositorio. Esto permitió la detección y corrección de varios bugs importantes:

- Se corrigió un problema que afectaba a ciertos casos en el manejo de muestras reales. La librería de lectura de ficheros binarios es capaz de decodificar muestras reales y complejas, y el caso de muestras reales para ciertas combinaciones de parámetros fallaba. Esto se solucionó en el commit id 30897468.
- Se detectó un bug en la función de reducción que afectaba al alineado de streams en ciertos casos dependiendo del retardo entre estaciones, que tenía que ver con la gestión de las muestras. Este bug se solucionó en el commit id 9a50502d.
- Se corrigió un problema de gestión del flujo de muestras en la función de reducción que afectaba a la última ventana de acumulación (commit id 1f3f8f45).
- Existía también otro bug conocido que afectaba a casos en los que las ventanas de acumulación no eran múltiplo de un segundo. Estaba relacionada con un tema de precisión y codificación, y se solucionó en el commit id 81c521d6.

Se corrigieron también otros bugs menores, como uno relacionado con la importación de la librería de "profiling" [3] (commit id 4665524f).

4.5. Mejoras Funcionales

Además de la corrección de bugs, hay varias mejoras que afectan directamente al proyecto legado:

- Se redujeron los tiempos de ejecución de la correlación en modo tubería, es decir, realizando el procesado de manera lineal mediante "pipes". Esto se consiguió simplificando la distribución de ficheros del proyecto legado en los commit ids 25c3fceb y 23e4a01a4.
- Se añadió una opción a la herramienta de comparación de resultados para no tener en cuenta el orden de geneneración de resultados, que como se explicó anteriormente no es relevante en el resultado (commit id 0f297c69). Esta mejora facilita la comparación de resultados generados de manera distribuida.
- Se añadió información en la herramienta de lectura y generación de estadísticas de ficheros de entrada en formato VDIF (commit id 23ad396f), y se añadió soporte para la generación básica de muestras complejas (commit id f177d770), ya que anteriormente sólo se soportaba la generación de muestras reales. Esta modificación era necesaria para poder generar los datos de prueba que se utilizaron para las pruebas descritas en el Capítulo 5.

[2] Se opta por el término en inglés dado su uso extensivo en este tipo de aplicaciones, haciendo referencia a un conjunto de ficheros de entrada y archivos de configuración del experimento.

[3] También en este caso se utiliza este término en inglés por ser de uso extensivo, y hace referencia al anális de rendimiento de una aplicación mediante la generación de estadísticas o grafos de llamadas entre funciones con datos detallados de tiempo y número de llamadas.

4.6. Empaquetado

El empaquetado es la preparación del software para su distribución, siguiendo unos procedimientos que faciliten la instalación y configuración del mismo. Existen varias opciones, por ejemplo:

- Distribuir la estructura de archivos .py. Esta fue la opción elegida en el proyecto legado [1]. Es la opción más sencilla y es válida en muchos casos, pero pensando en dar el salto a un proveedor de servicios en la nube habría que buscar una opción más portable.
- Preparar un contenedor con una instalación completa del paquete y sus dependencias. Ésta es la opción más portable pero también la más compleja, y de hecho no se adapta necesariamente bien a la distribución a un servicio de ejecución de trabajos del ecosistema Hadoop en la nube.
- Preparar un paquete Python. Existen utilidades que es posible configurar para generar un fichero instalable en un entorno virtual de Python. Ésta es la opción preferida para algunos proveedores de servicios en la nube, que permiten añadir un script de aprovisionado donde se pueden instalar dependencias.

Para este proyecto se optó por la última opción (commit id 4a508fa9) por ser la más adecuada al entorno de procesado en la nube, como se verá en el siguiente capítulo.

La organización propuesta para el mismo se basa en un par de ficheros: `setup.py` (script utilizado para la generación del paquete que se desplegará en el servidor) y `MANIFEST.in` (en el que se detallan los ficheros que no corresponden con código de los módulos). El empaquetado elegido facilita la distribución del software (reduciendo ésta a un par de pasos, copia segura al servidor mediante la herramienta `scp` e instalación del mismo mediante la herramienta `pip`), y facilita el control de versiones. Se simplifica también la ejecución del correlacionador y las herramientas asociadas, ya que en el `setup.py` se definen unos aliases que será posible ejecutar directamente una vez instalado el paquete en el entorno virtual.

4.7. Correlación con Spark

A continuación se detallan los commits más importantes durante el desarrollo de la librería para realizar la correlación con Spark.

- Inicialmente se preparó un script sencillo que realizara el procesado básico (commit id 2feaca07).
- Posteriormente se organizó un poco el código (commit id 570fad1d) y se simplificó la configuración del correlacionador (53229f305). Se consiguió una reducción considerable en el archivo de configuración del correlacionador (comparado con el legado, basado en Hadoop).
- Se añadió también soporte para permitir añadir parámetros de configuración de Spark en el archivo de configuración del correlacionador, en una sección destinada a tal efecto. Esto facilita la gestión del correlacionador y evita tener que modificar múltiples archivos de configuración para configurar aspectos del correlacionador y de Spark.
- Para facilitar la depuración, se añadió la posibilidad de realizar comprobaciones relativas a los datos de entrada y la configuración del experimento (frecuencia de muestreo, número de muestras por paquete, tipos de muestras), permitiendo así detectar errores de configuración.

Esto resulta útil durante el desarrollo, donde es interesante estar seguros de que los datos de prueba están bien generados, pero también en producción para detectar errores humanos en la configuración antes de realizar el procesado (commit id 7f169a8f).

- Tras incrementar el tamaño de los datos de prueba, se detectaron algunos problemas en el particionador. Estos se corrigieron en el commit 8cbd2b9e, donde se garantiza que las particiones con los datos están bien repartidas (basándose principalmente en ventana de acumulación y canal). Además se añadió un modo de depuración para mostrar metadatos sobre las particiones creadas.

- Se continuó incrementando el tamaño de los archivos de entrada, y se detectó un problema de rendimiento relacionado con el formato de escritura del correlacionador legado. Al tratarse de un prototipo normalmente se trabajaba con datos pequeños, por lo que el formato de salida era texto plano con los resultados. Se solucionó codificando la salida en base64 (commit id 7b31e2d5). Aunque es posible una solución más eficiente, esto se hizo así por simplicidad, para mantener los formatos de salida como tipo texto. Se plantea como línea futura incorporar una gestión más eficiente de los datos en formato binario.

- También relacionado con este incremento en el volumen de datos y con el formato de salida, se detectó otro problema en la agrupación de los resultados que tenía que ver la unión de los resultados en un único fichero. Este paso no es estrictamente necesario (dado que la unión es una mera concatenación de líneas de ficheros de texto), así que optó por incorporar la escritura a disco en cada uno de los ejecutores de la función reductora (commit id c3a6ebb5). Tras ejecutar esta solución en la nube se vio que no era posible al correr los ejecutores en distintas máquinas, por lo que finalmente se optó por una solución más canónica utilizando la función de Spark `saveAsTextFile` (commit id d3e8c13e), que además permite escribir directamente al sistema de AWS Simple Storage Service (S3) como se verá en el siguiente capítulo.

- Se detectó un problema relacionado con el cálculo del número de particiones que se solucionó en el commit id b90a51580, y se depuró la distribución de carga (commit ids 87c72741 y 6e367c0) para procesar datasets grandes.

- Además se añadió la opción de mantener Spark lanzado durante un tiempo configurable para poder revisar los resultados relativos a rendimiento en la interfaz web (commit id ac05e0b1). Esta opción no es necesaria en una solución de procesado en la nube donde este servicio está corriendo tras iniciar las instancias asociadas, pero es útil para depuración en ejecuciones en una instancia local.

4.8. Pruebas Preliminares en Entorno Local

En esta sección se presentan los diagramas generados por Spark ejecutando una tarea de correlación del sistema propuesto. Esto permitirá al lector ver de manera más gráfica el procesado descrito anteriormente en la Sección 3.3.

Dado que este procesado preliminar se ejecutará en un ordenador portátil (Sección 1.5), para los datos de prueba se utilizará una versión reducida (los archivos de entrada pesan 1/100 de los originales) de los datos utilizados para las pruebas en la nube, cuyos detalles se describen posteriormente en la Sección 5.2. Sus características se resumen a continuación:

- Estaciones: 2 estaciones con 2 polarizaciones por estación.

- Datos: 20 s muestreados a 20 MHz (1/100 de la original para conseguir la reducción de 20 GB a 200 MB por fichero), formato VDIF. Muestras reales, 2 bits/muestra, 1 canal.
- Correlación: 0.512 s/ventana, FFT de 262144 componentes.

Estos datos se generaron lanzando el script `python examples/test_dataset_test/gen_test_file.py`, cambiando la constante `SUB_TEST=True`. Estos datos de prueba, incluidos los archivos de configuración, están en la carpeta `examples/test_dataset_test/sub`.

Con estos datos, de cara a la paralelización, es posible extraer las siguientes conclusiones:

- Número de tareas de mapeo: 2, es decir, tantas como ficheros (posteriormente se verá que es posible aumentar esta cifra dividiendo en partes los ficheros de entrada).
- Número de tareas de reducción: 40, es decir, dado que sólo hay un canal, la función cielo de 20 s / 0.512 s.

Se proporciona también un script para lanzar la correlación en script de estos datos, `bash examples/run_example_test_sub_spark.sh`, donde se añade un tiempo extra al final de la ejecución para acceder a la interfaz web, que por defecto se lanza en http://localhost:4040/.

A continuación se muestra el contenido del archivo de configuración del correlacionador:

```
[Experiment]
Experiment folder: /home/aj/work/cx\_git/CorrelX/examples/test_dataset_test/sub
Spark input files: file:///home/aj/work/cx_git/CorrelX/examples/test_dataset_test/sub/
    ↪ media/test*.vt

[Files]
Output directory:  /home/aj/work/cx_git/CorrelX/output

[Spark]
spark.master: local[4]
spark.executor.cores: 1
spark.cores.max: 8
spark.driver.memory: 2g
spark.executor.memory: 4g
spark.driver.maxResultSize: 100g
```

Se puede comprobar que la configuración es muy sencilla, únicamente es necesario especificar la ubicación del directorio con los ficheros de configuración del experimento, cuyo formato está heredado de CorrelX [1], el patrón con la ubicación de los ficheros de lectura, y la ruta para la salida.

Además, se permite incluir una lista de parámetros de Spark. En este caso se indica que la ejecución correrá localmente en la propia máquina utilizando 4 núcleos del procesador, y se establecen algunos límites de procesado y de memoria. Estos valores de configuración se alcanzaron de manera iterativa.

A continuación se muestran algunas capturas tras la ejecución. En la Figura 4.1 se muestra la tarea ejecutada, y en la Figura 4.2 sus etapas: procesado de ficheros de entrada (etapa 0), repartición y reordenación (etapa 1), y reducción (etapa 2).

Como puede verse en la Figura 4.2, el mayor tiempo de ejecución recae en la última etapa, que es la que involucra los cómputos de correlación.

En las Figuras 4.3, 4.4 y 4.5 se muestran los grafos y datos asociados a cada una de estas etapas. En la Figura 4.3 puede comprobarse la ejecución de dos procesos, uno para cada uno de los ficheros, cada uno de ellos de aproximadamente 200 MB. El reparticionado es muy rápido

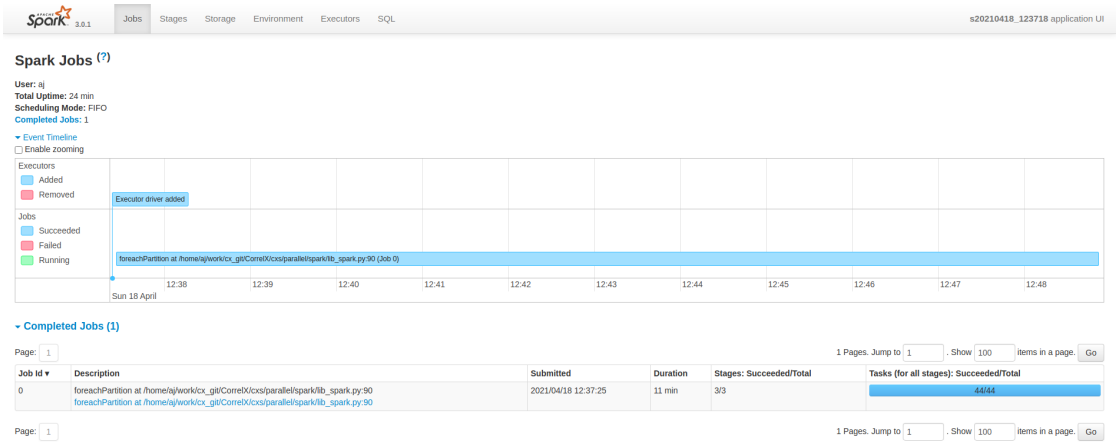


Figura 4.1: Ejecución: tareas.

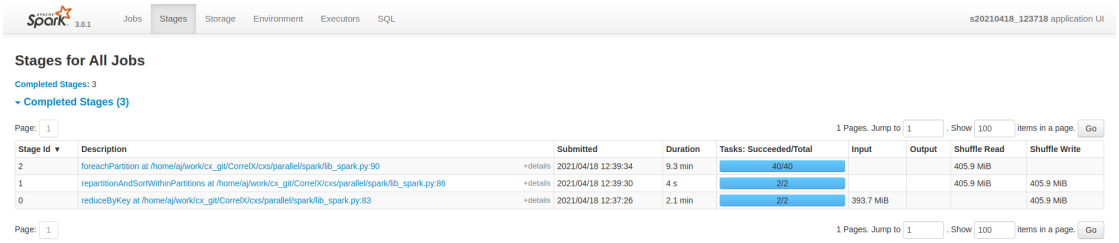


Figura 4.2: Ejecución: etapas.

en comparación con el resto de etapas (Figura 4.4). Finalmente, en la Figura 4.5 se muestra la ejecución de la etapa de reducción, donde se puede comprobar que como máximo se ejecutan cuatro tareas simultáneamente (pues anteriormente el número de núcleos se limitó a 4), y se puede comprobar también que se lanzan 40 subtareas.

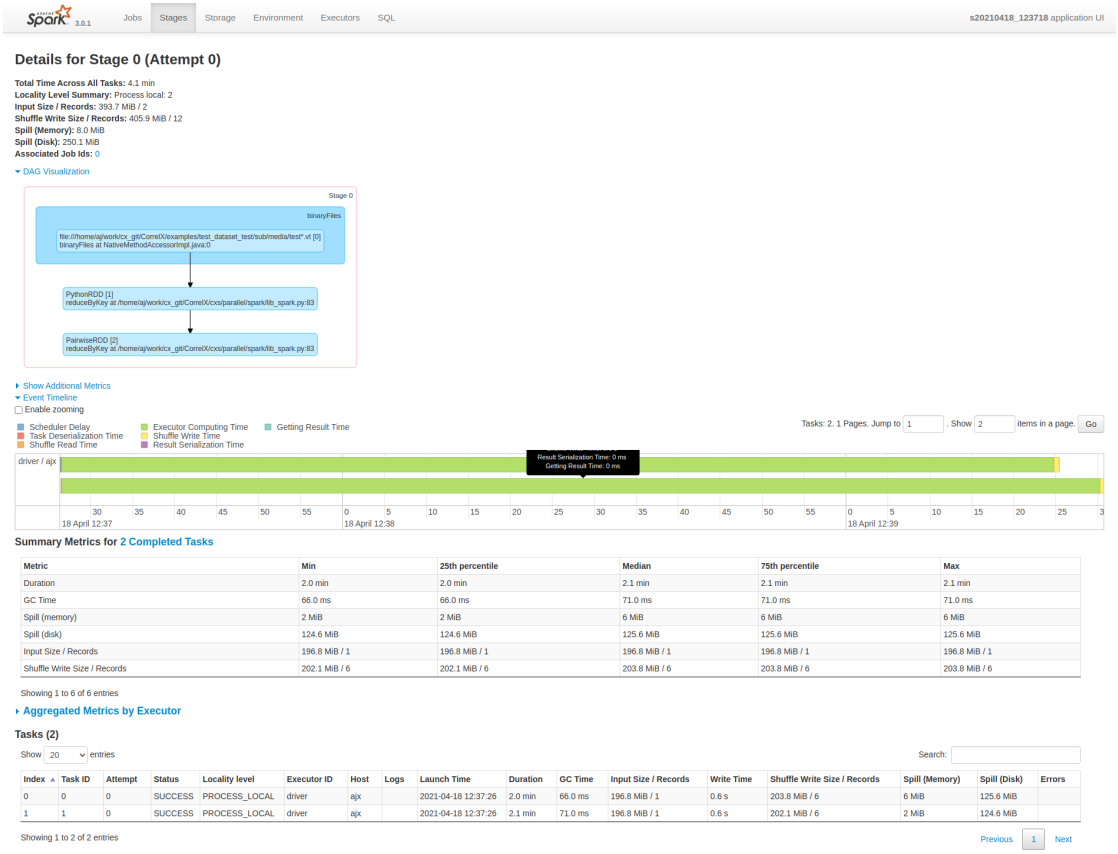


Figura 4.3: Ejecución: etapa 0, procesado de ficheros.

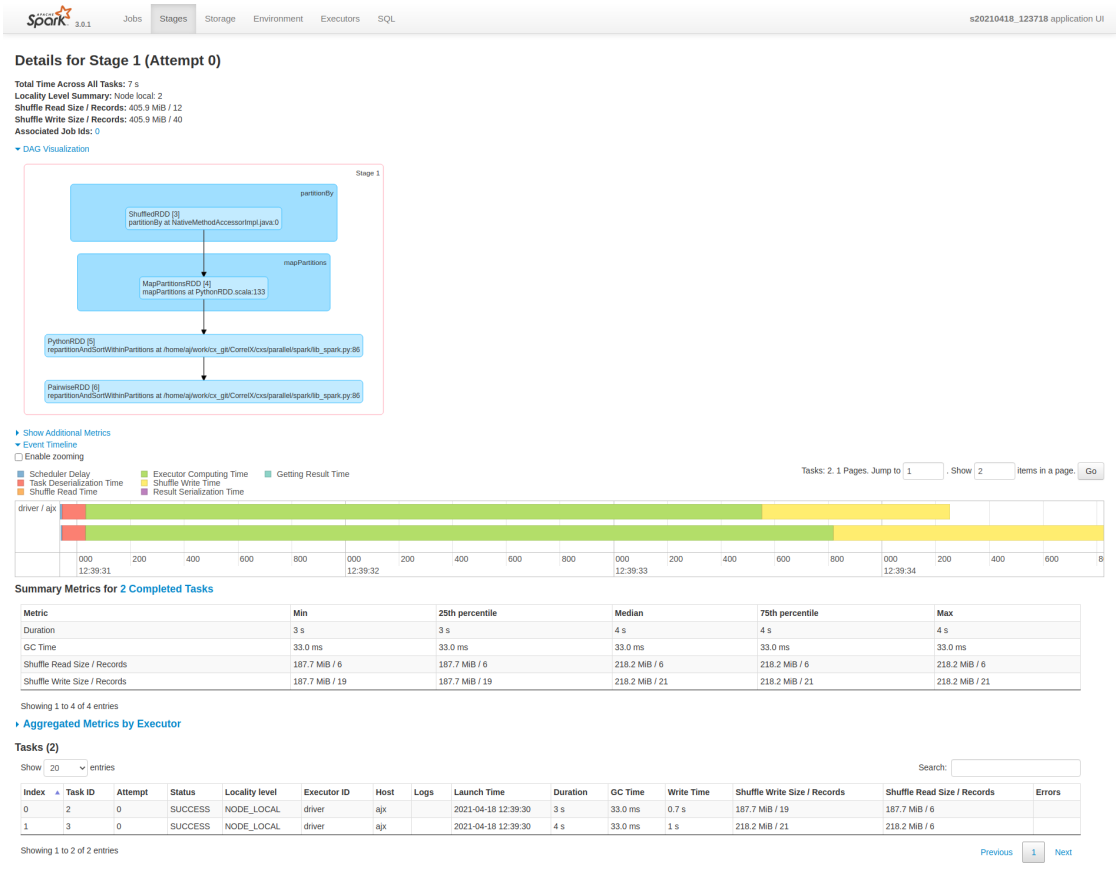


Figura 4.4: Ejecución: etapa 1, reparticionado.

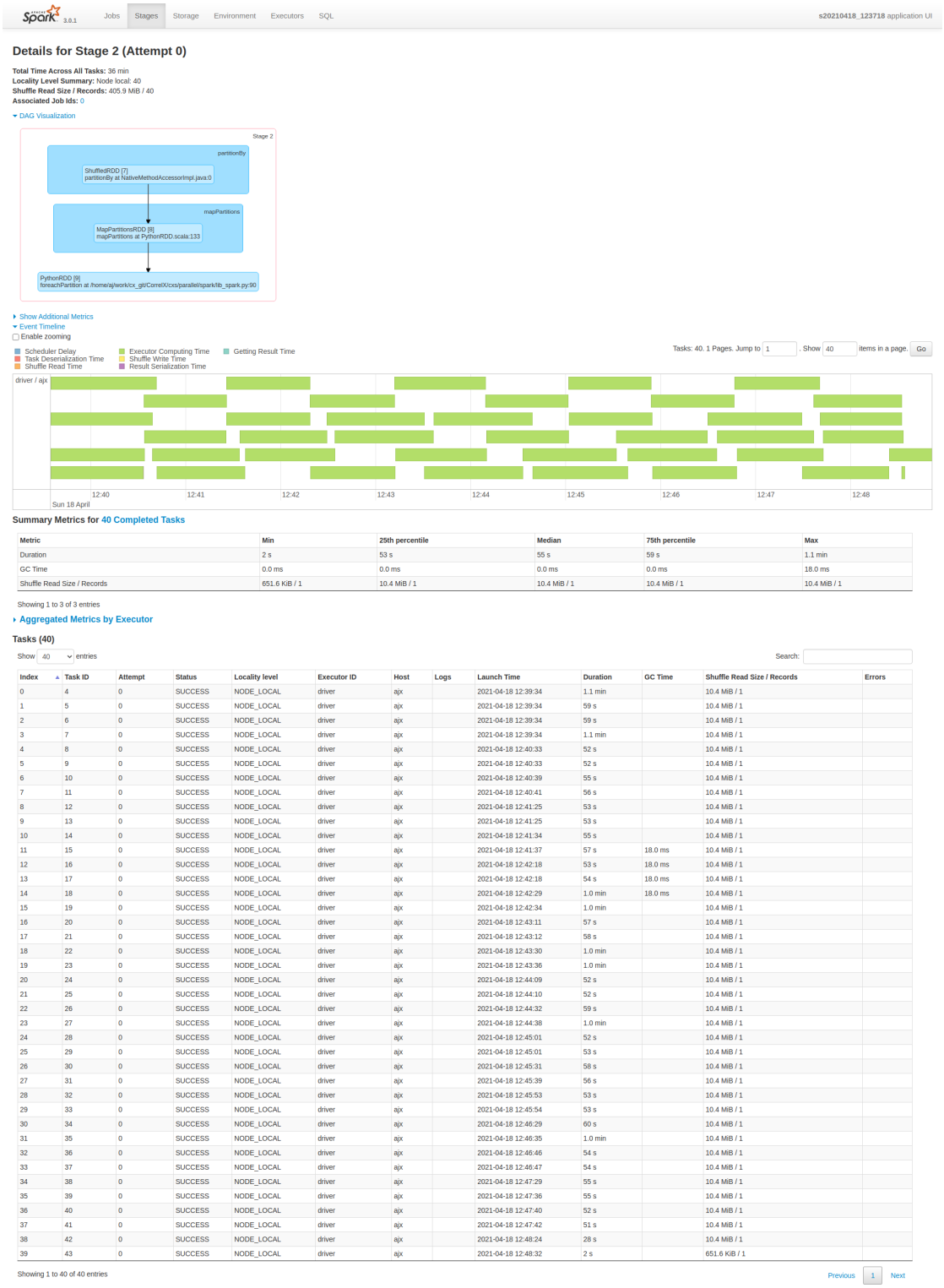


Figura 4.5: Ejecución: etapa 2, reducción.

En la Figura 4.6 se muestra el contenido de la sección de Entorno de la interfaz web, que presenta un resumen con la configuración de Spark, donde se puede comprobar que se han aplicado los parámetros de configuración especificados anteriormente.

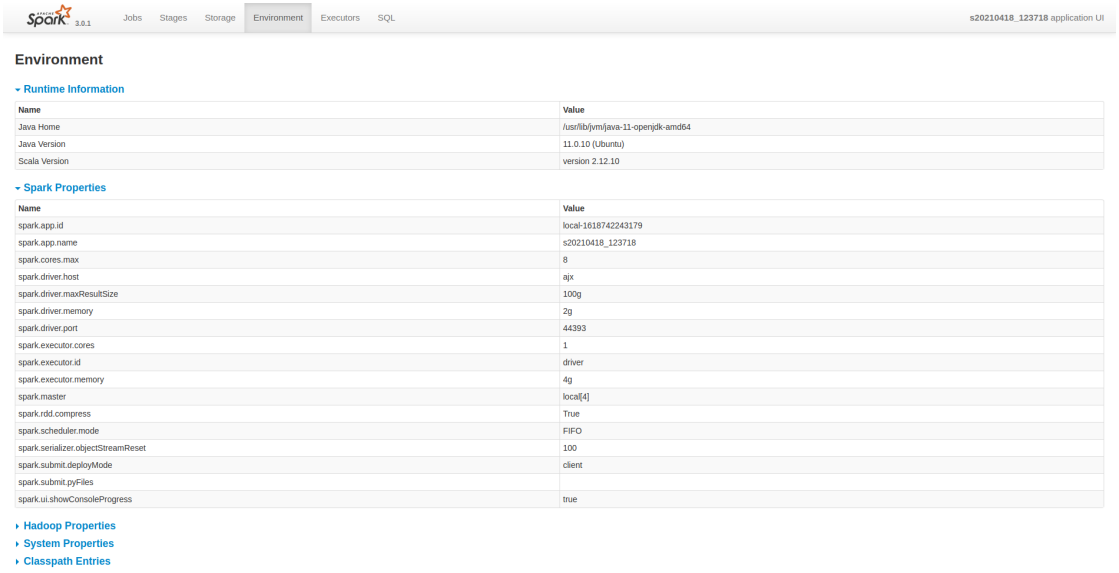


Figura 4.6: Ejecución: configuración.

A continuación se muestra el contenido del terminal donde se ejecutó lanzó la correlación:

```
(venv3) aj@ajx:~/work/cx_git/cx$ bash examples/run_example_test_sub_spark.sh
Partitions: 40 (accs: 40, channels: 1)
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/home/aj/
    ↪ work/tfm/spark-3.0.1-bin-hadoop2.7/jars/spark-unsafe_2.12-3.0.1.jar) to
    ↪ constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.
    ↪ Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective
    ↪ access operations
WARNING: All illegal access operations will be denied in a future release
21/04/18 12:37:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your
    ↪ platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(
    ↪ newLevel).
Elapsed: 693.3104610443115
Keeping Spark session open for 3600 minutes
```

Y a continuación se muestran los archivos de salida:

```
aj@ajx:~/work/cx_git/CorrelX/output/s20210418_123718$ ls -ltrh
total 2,1G
-rw-rw-r-- 1 aj aj 54M abr 18 12:40 sub__px-0.0-0.0-a.2.2.0-sxa19_dfcb90.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:40 sub__px-0.0-0.0-a.1.1.0-sxa19_b3c007.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:40 sub__px-0.0-0.0-a.3.3.0-sxa19_9cae07.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:40 sub__px-0.0-0.0-a.0.0.0-sxa19_e97bc5.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:41 sub__px-0.0-0.0-a.5.5.0-sxa19_fedf33.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:41 sub__px-0.0-0.0-a.4.4.0-sxa19_82a422.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:41 sub__px-0.0-0.0-a.6.6.0-sxa19_bfd63e.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:41 sub__px-0.0-0.0-a.7.7.0-sxa19_895b6f.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:42 sub__px-0.0-0.0-a.9.9.0-sxa19_2fb389.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:42 sub__px-0.0-0.0-a.8.8.0-sxa19_ebb61f.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:42 sub__px-0.0-0.0-a.10.10.0-sxa19_8019bb.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:42 sub__px-0.0-0.0-a.11.11.0-sxa19_1429ae.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:43 sub__px-0.0-0.0-a.12.12.0-sxa19_fe385c.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:43 sub__px-0.0-0.0-a.13.13.0-sxa19_b4a6b6.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:43 sub__px-0.0-0.0-a.14.14.0-sxa19_7beeb6.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:43 sub__px-0.0-0.0-a.15.15.0-sxa19_ad5fe6.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:44 sub__px-0.0-0.0-a.16.16.0-sxa19_8f2be1.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:44 sub__px-0.0-0.0-a.17.17.0-sxa19_8aefc6.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:44 sub__px-0.0-0.0-a.18.18.0-sxa19_0d3e07.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:44 sub__px-0.0-0.0-a.19.19.0-sxa19_d0b9a4.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:45 sub__px-0.0-0.0-a.20.20.0-sxa19_011116.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:45 sub__px-0.0-0.0-a.21.21.0-sxa19_18bf69.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:45 sub__px-0.0-0.0-a.22.22.0-sxa19_325b2d.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:45 sub__px-0.0-0.0-a.23.23.0-sxa19_8b3658.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:45 sub__px-0.0-0.0-a.24.24.0-sxa19_2f61c1.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:45 sub__px-0.0-0.0-a.25.25.0-sxa19_134ebd.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:46 sub__px-0.0-0.0-a.26.26.0-sxa19_ed240a.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:46 sub__px-0.0-0.0-a.27.27.0-sxa19_44c7e6.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:46 sub__px-0.0-0.0-a.28.28.0-sxa19_c25fdd.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:46 sub__px-0.0-0.0-a.29.29.0-sxa19_6b1955.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:47 sub__px-0.0-0.0-a.30.30.0-sxa19_459b0b.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:47 sub__px-0.0-0.0-a.31.31.0-sxa19_83bd1e.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:47 sub__px-0.0-0.0-a.32.32.0-sxa19_f5d57f.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:47 sub__px-0.0-0.0-a.33.33.0-sxa19_cba094.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:48 sub__px-0.0-0.0-a.34.34.0-sxa19_f6b5b9.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:48 sub__px-0.0-0.0-a.35.35.0-sxa19_f94441.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:48 sub__px-0.0-0.0-a.36.36.0-sxa19_36d282.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:48 sub__px-0.0-0.0-a.37.37.0-sxa19_023cef.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:48 sub__px-0.0-0.0-a.39.39.0-sxa1_fa017a.out
-rw-rw-r-- 1 aj aj 54M abr 18 12:48 sub__px-0.0-0.0-a.38.38.0-sxa19_53d943.out
```

A continuación se muestran dos extractos pequeños de dos ficheros de salida:

sub__px-0.0-0.0-a.0.0.0-sxa19_e97bc5.out fichero que corresponde con la ventana de acumulación 0, e incluye autocorrelaciones para todas las estaciones, así como todos los pares de correlaciones de las tuplas formadas por los pares estación-polarización.

```
px-0.0-0.0-a.0.0.0-sxa19 0.1 0 0.0 0.015594936964547366 -0.0 -0.0 -0.0 0.0 0.0 0.0 0.0
    ↪ 0.0 0.0 0.0 0.0 0.0 2000 20000000 2 10549898 r 0 0 0 3480400000.0 0.512 no
    ↪ 311898.0 b64_dp_1EbM3mGCsj4AAAAAAAAAAMSZVyVF8bg+AAAAAAAAAAB+rwrYxySx
```

```
[...]
[...]
```

sub__px-0.0-0.0-a.38.38.0-sxa19_53d943.out fichero que corresponde con la ventana de acumulación 38, donde aplican las mismas consideraciones que las explicadas para el archivo anterior.

```
px-0.0-0.0-a.38.38.0-sxa19 0.1 0 0.0 0.015592966232119839 -0.0 -0.0 -0.0 0.0 0.0 0.0 0.0
    ↪  0.0 0.0 0.0 0.0 0.0 2000 20000000 2 10551859 r 0 0 0 3480400000.0 0.512 no
    ↪ 312433.0 b64_dp_kT+5pCXitj4AAAAAAAAAAN/EK1h
[...]
[...]
```

4.9. Tests

Como se indicó en la Sección 4.2, se prepararon varios tests de dos tipos: (i) de referencia y (ii) de desarrollo. Los tests de referencia básicamente comparan resultados de referencia de la arquitectura antigua y la nueva, y los de desarrollo ejecutan un procesado y realizan una comparación con los resultados de referencia. Los pasos son los siguientes:

- Ejecución de la correlación (modo "tubería" o Spark).
- Comprobación de que se han generado resultados.
- Medición de error.
- Comprobación de que el error está debajo de un umbral predefinido.

La medición del error se calcula como la suma de los cuadrados de las diferencias de las componentes de las visibilidades para todas las ventanas de acumulación y para todos los canales. Para los tests de referencia se establece un umbral máximo de 10^{-10}, y para los tests de desarrollo de 10^{-20}. Nótese que para los tests de desarrollo es necesaria mayor precisión, ya que debe garantizarse que la diferencia en los resultados es mínima. Para los tests de referencia se permite menor precisión debido a que el archivo de referencia para la nueva arquitectura fue generado tras realizar los cambios de versiones, y de hecho únicamente estos cambios, para tener en cuenta diferencias de precisión derivadas de los mismos. Es en este punto, una vez se ha fijado el archivo de salida de referencia, donde ha comenzado el desarrollo del sistema propuesto, y donde las diferencias en la salida generada con la obtenida inicialmente deben ser mínimas.

A continuación se muestran los resultados de la ejecución de todos los tests.

```
aj@ajx:~/work/cx_git/cx$ source venv3/bin/activate
aj@ajx:~/work/cx_git/cx$ python -m unittest discover -s .
......
----------------------------------------------------------------------
Ran 6 tests in 39.749s

OK
```

Si se activa el modo verboso de los tests, se muestra toda la salida de las ejecuciones, así como las comparaciones de los errores. A continuación se muestra la salida mostrada, reducida a varias partes de interés, para un test de referencia, otro para el modo "tubería" y otro para el test con Spark:

Test de referencia Este test compara la salida de referencia de la arquitectura antigua (con Python2 como se indica en el nombre del archivo) con la salida de referencia conseguido con la arquitectura nueva (con Python3), ambos en modo "tubería".

```
[...]
 File 1: /home/aj/work/cx_git/CorrelX/cxs/tests/../../examples/test_dataset_vgos/
     ↪ example_output/OUT_s0_v0_python2_sort_legacy.out
 File 2: /home/aj/work/cx_git/CorrelX/cxs/tests/../../examples/test_dataset_vgos/
     ↪ example_output/OUT_s0_v0_python3_sort_legacy.out
px-0.0-0.0-a.0.0.0-sxa467 px-0.0-0.0-a.0.0.0-sxa467 -> 2.5128351192134167e-27
px-0.0-0.0-a.1.0.1-sxa467 px-0.0-0.0-a.1.0.1-sxa467 -> 7.33025824587479e-27
px-0.0-0.0-a.10.0.10-sxa467 px-0.0-0.0-a.10.0.10-sxa467 -> 1.6519829630343682e-26
[...]
px-1.1-1.1-a.7.0.7-sxa467 px-1.1-1.1-a.7.0.7-sxa467 -> 3.5157514938361396e-12
px-1.1-1.1-a.8.0.8-sxa467 px-1.1-1.1-a.8.0.8-sxa467 -> 3.515751493855578e-12
px-1.1-1.1-a.9.0.9-sxa467 px-1.1-1.1-a.9.0.9-sxa467 -> 3.515751493983345e-12
 Visibilities compared: 320
 Num. coefficients per vis.: 128
 Total error: 3.515751493983345e-12
.
```

Test en modo tubería En este test se ejecuta el modo "tubería" con la arquitectura nueva, y se compara con el archivo de referencia (el correspondiente para la arquitectura nueva del test anterior), por eso el error es nulo. Es decir, este test es en parte redundante con el anterior pero da más precisión sobre posibles cambios los resultados sobre la arquitectura nueva. Así, si hubiera una diferencia que pudiera pasar desapercibida en el test anterior, en este test se notaría.

```
[...]
Results\n#-------------------------------------------------------\n\nNote: 0 nodes and 0
    ↪ vcores correspond to pipeline execution\n\nShowing results...\n\nFile IO
    ↪ approximate times\n Type File IO. time [s]\n\nExecution times\n Type Exec. time [
    ↪ s]\n Pipeline 7.7047951221466064\n"
[...]
 File 1: /home/aj/work/cx_git/CorrelX/cxs/tests/../../examples/test_dataset_vgos/
     ↪ example_output/OUT_s0_v0_python3_sort_numeric.out
 File 2: /home/aj/work/cx_git/CorrelX/output/e20210531_192535/OUT_s0_v0.out
px-0.0-0.0-a.0.0.0-sxa467 px-0.0-0.0-a.0.0.0-sxa467 -> 0.0
px-0.0-0.0-a.1.0.1-sxa467 px-0.0-0.0-a.1.0.1-sxa467 -> 0.0
px-0.0-0.0-a.10.0.10-sxa467 px-0.0-0.0-a.10.0.10-sxa467 -> 0.0
[...]
px-1.1-1.1-a.7.0.7-sxa467 px-1.1-1.1-a.7.0.7-sxa467 -> 0.0
px-1.1-1.1-a.8.0.8-sxa467 px-1.1-1.1-a.8.0.8-sxa467 -> 0.0
px-1.1-1.1-a.9.0.9-sxa467 px-1.1-1.1-a.9.0.9-sxa467 -> 0.0
 Visibilities compared: 320
 Num. coefficients per vis.: 128
 Total error: 0.0
.
```

Test de Spark En este caso se compara la salida generada con Spark con la salida de referencia del primer test (modo "tubería" con la arquitectura nueva). Este test es muy importante, ya que muestra que el desarrollo consigue los mismos resultados que el sistema legado.

```
.Stdout:
b'Partitions: 96 (accs: 3, channels: 32)\nElapsed: 12.587505340576172\nDone.\n'
Stderr:
[...]
\n\r[Stage 0:>
(0 + 1) / 1]\r\r[Stage 2:====>
(8 + 8) / 96]\r\r[Stage 2:======>
(11 + 8) / 96]\r\r[Stage 2:=========>
(16 + 8) / 96]\r\r[Stage 2:===========>
(19 + 8) / 96]\r\r[Stage 2:=============>
(23 + 8) / 96]\r\r[Stage 2:=================>
(29 + 8) / 96]\r\r[Stage 2:=========================>
(44 + 9) / 96]\r\r[Stage 2:=============================================>
(75 + 8) / 96]\r\r \r'
 [...]
 File 1: /home/aj/work/cx_git/CorrelX/cxs/tests/../../examples/test_dataset_vgos/
     ↪ example_output/OUT_s0_v0_python3_sort_numeric.out
 File 2: /home/aj/work/cx_git/CorrelX/output/s20210531_192559/OUT_s0_v0.out
px-0.0-0.0-a.0.0.0-sxa467 px-0.0-0.0-a.0.0.0-sxa467 -> 0.0
px-0.0-0.0-a.1.0.1-sxa467 px-0.0-0.0-a.1.0.1-sxa467 -> 0.0
px-0.0-0.0-a.10.0.10-sxa467 px-0.0-0.0-a.10.0.10-sxa467 -> 0.0
[...]
px-1.1-1.1-a.7.0.7-sxa467 px-1.1-1.1-a.7.0.7-sxa467 -> 0.0
px-1.1-1.1-a.8.0.8-sxa467 px-1.1-1.1-a.8.0.8-sxa467 -> 0.0
px-1.1-1.1-a.9.0.9-sxa467 px-1.1-1.1-a.9.0.9-sxa467 -> 0.0
 Visibilities compared: 320
 Num. coefficients per vis.: 128
 Total error: 0.0
.
```

4.10. Resumen

En este capítuo se ha revisado en detalle la implementación, y se ha descrito paso a paso la ejecución en una máquina local. En el siguiente capítulo se dará el sato a la nube.

Capítulo 5

Prueba de Concepto en la Nube

En el capítulo anterior se ha descrito en detalle la implementación, y se han introducido también algunas mejoras y modificaciones requeridas para funcionar correctamente en la nube. En este capítulo se entrará en detalle en el entorno de procesamiento utilizado para las pruebas en la nube, se presentará su configuración, así como la del dataset utilizado para las pruebas, y se presentarán unas pruebas de escalabilidad.

5.1. Proveedores de Servicios de Computación

Existen varias soluciones para la migración de servicios a la nube. Las más conocidas son las siguientes:

- Amazon Web Services (AWS) [54].
- Microsoft Azure [55].
- Google Cloud Platform (GCP) [56].

Si comparamos las tres alternativas en cuanto a cuotas de mercado [57, 58], AWS lidera con un 32 %, seguido por Azure con un 19 %, y por GCP con un 7 %. En cuanto a coste, para la realización del proyecto se dispone de un cupón de créditos proporcionado por la UNED para la asignatura de infraestructuras computacionales dada en el máster. Estos motivos, unidos a la mayor familiaridad laboral del autor de este TFM con AWS, ha llevado a la decisión de utilizar AWS como el proveedor para ejecutar en la nube el proyecto presentado.

En términos más técnicos, se ha comprobado que todos los proveedores facilitaban la conexión con sus servicios de almacenamiento, lo que permitiría colocar los datos de entrada en un "bucket"[1] de ese servicio y al terminar la tarea de procesado escribir los resultados a otro "bucket" [59, 60, 61]. En términos de rendimiento, todos los proveedores facilitan bastantes opciones en cuanto al dimensionado de las instancias.

En cualquier caso, la aproximación al empaquetado descrita en la Sección 4.6 facilita migrar fácilmente los procedimientos descritos en este capítulo para AWS a otro servicio sin dificultades.

[1] Este término es de uso extendido por los distintos provedores de servicios en la nube, se refiere a un recurso de almacenamiento.

5.2. Línea Base

Para la realización de las pruebas de rendimiento se utilizará una línea base o referencia. Se ha buscado la referencia más reciente sobre resultados de correlación de DiFX, que como se ha visto en la Sección 2.1.3 puede considerarse el estándar de facto en correlación.

En el artículo [36] (de 2019) se presentan unas pruebas de escalabilidad en la Google Cloud Platform (GCP) utilizando datos generados en laboratorio para dichas pruebas. En esta sección se van a resumir sus características y resultados más importantes para utilizarlos como base con la que comparar los desarrollos del presente trabajo.

5.2.1. Dataset

Los datos de prueba tienen las siguientes características:

- Estaciones: entre 2 y 20 (y 2 polarizaciones por estación).
- Datos: 20 s muestreados a 2 GHz, formato VDIF. Muestras reales, 2 bits/muestra, 1 canal.
- Correlación: 0.512 s/ventana, FFT de 262144 componentes.

Cada archivo pesa aproximadamente 20 GB. Es interesante recalcar que la fuente original [36] respalda la utilización de datos generados en laboratorio (muestras de ruido) ya que no supone diferencias en cuanto a rendimiento de cara al procesado.

5.2.2. Configuración

Para la realización de las pruebas presentadas en la referencia [36] utilizan un clúster con una única máquina realizando el procesado, con un número de CPUs virtuales que va desde 16 hasta 96 (`n1-highmem-16`, `n1-highmem-32`, `n1-highmem-64`, `n1-highmem-96`). Las características de estas máquinas se pueden consultar en la referencia [62], y las más importantes se resumen a continuación:

- Procesador: Intel Xeon @ 2.50 GHz. Número de núcleos virtuales: desde 16 hasta 96.
- Memoria: 104 GB.
- Almacenamiento: 128 GB, SSD local.

5.2.3. Rendimiento

De cara a entender cómo algunos factores influyen en el rendimiento es suficiente con saber que la complejidad computacional de los cálculos de este procesado crecen de manera cuadrática con el número de estaciones, ya que es necesario realizar cálculos de todos los pares de estaciones entre sí. En cuanto a duración de la señal, número de muestras, frecuencia de muestreo, etc., estos parámetros afectan a los tiempos de computación de manera lineal (ya que una vez alienados los "streams" de datos no hay interacción entre los datos que corresponden con ventanas de acumulación distintas). No se entrará en detalles sobre cómo influyen otros parámetros como el número de componentes de la transformada de Fourier (FFT), ya que escapan a los objetivos de este TFM; basta con tener en cuenta que experimentos con tasas de este tipo distintas no son directamente comparables.

Para el cálculo del rendimiento se sigue la siguiente metodología. La idea es extraer de los resultados presentados en la fuente [36] el rendimiento relativo de cada máquina, en función del

número de estaciones. Esto permitirá obtener un valor de capacidad (medido por ejemplo en MB por segundo por CPU core), que será posible comparar con una ejecución en la nube del sistema propuesto. Dado que esta tasa elimina la dependencia de los parámetros temporales del dataset (e.g.: duración), sería posible comparar resultados para los dos datasets.

Así, de los resultados presentados en la Figura 2 de la referencia [36], extraemos las medidas de rendimiento aproximadas recogidas en la Tabla 5.1.

Tabla 5.1: Tasas de procesado por vCPU de DiFX según número de estaciones a partir de los datos de la referencia [36].

Número de estaciones	*Tasa por vCPU con 16 vCPU*	*Tasa por vCPU con 32 vCPU*	*Tasa por vCPU con 64 vCPU*	*Tasa por vCPU con 96 vCPU*	*Tasa por vCPU media*
2	1.28 MB/s	0.73 MB/s	0.64 MB/s	0.85 MB/s	**0.88 MB/s**
4	1.14 MB/s	0.73 MB/s	0.64 MB/s	0.85 MB/s	**0.84 MB/s**
8	-	-	0.73 MB/s	0.85 MB/s	**0.79 MB/s**

La tabla muestra las tasas de procesado relativo por CPU virtual para los distintos escenarios considerados en la referencia [36] para DiFX. Haciendo una cuenta rápida por ejemplo con el caso de 2 estaciones y 16 vCPUs, en la referencia se indica un tiempo de procesado de 2000 segundos aproximadamente. Si se divide el volumen total de datos de todas las estaciones (40 GB) por este tiempo (2000 s) y luego por el número de vCPUs (16) se obtiene la tasa relativa 1.28 MB/s. En la última columna se muestra la media de estas tasas, con lo que para estos casos se observa que la tasa de procesado de datos se mantiene relativamente constante en aproximadamente 0.8 MB/s por vCPU.

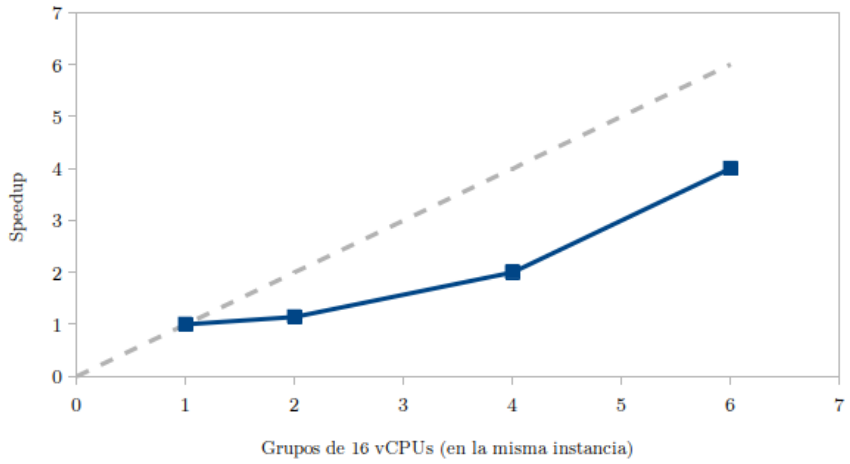


Figura 5.1: Speedup de DiFX frente a grupos de 16 vCPUs para 2 estaciones.

Es importante destacar que para experimentos con configuraciones similares el uso de tasas de procesado (volumen de datos dividido por tiempo empleado) permite en principio dejar fuera de la comparación el volumen de datos leído, lo que permitiría trabajar con ficheros más pequeños (grabaciones correspondientes a menos tiempo o muestreadas a una frecuencia más baja) como los presentados en la Sección 4.8[2].

[2]Dado que esta afirmación con respecto a rendimiento no se va a probar, se realizarán pruebas de escalabilidad

En la Figura 5.1 se representa el Speedup (tiempo total de referencia dividido por el tiempo total con el número de instancias). En este caso en vez de instancias se representa el número de grupos de vCPUs para la misma instancia). Idealmente esta curva debería ser una recta de pendiente unidad si multiplicar el número de instancias por N multiplica también el rendimiento por N. Este resultado ideal se muestra como una asíntota en el gráfico.

5.3. Preparación del Clúster

En esta sección se muestra la configuración de un cluster EMR para la ejecución de tareas. El proceso de creación del clúster se compone de cuatro pasos:

- Software y pasos (Figura 5.2).
- Hardware (Figura 5.3).
- Ajustes generales del clúster (Figura 5.4).
- Seguridad (Figura 5.5).

En la primera sección se selecciona la distribución (emr-6.3.0), así como las versiones de Hadoop (3.2.1) y Spark (3.3.1).

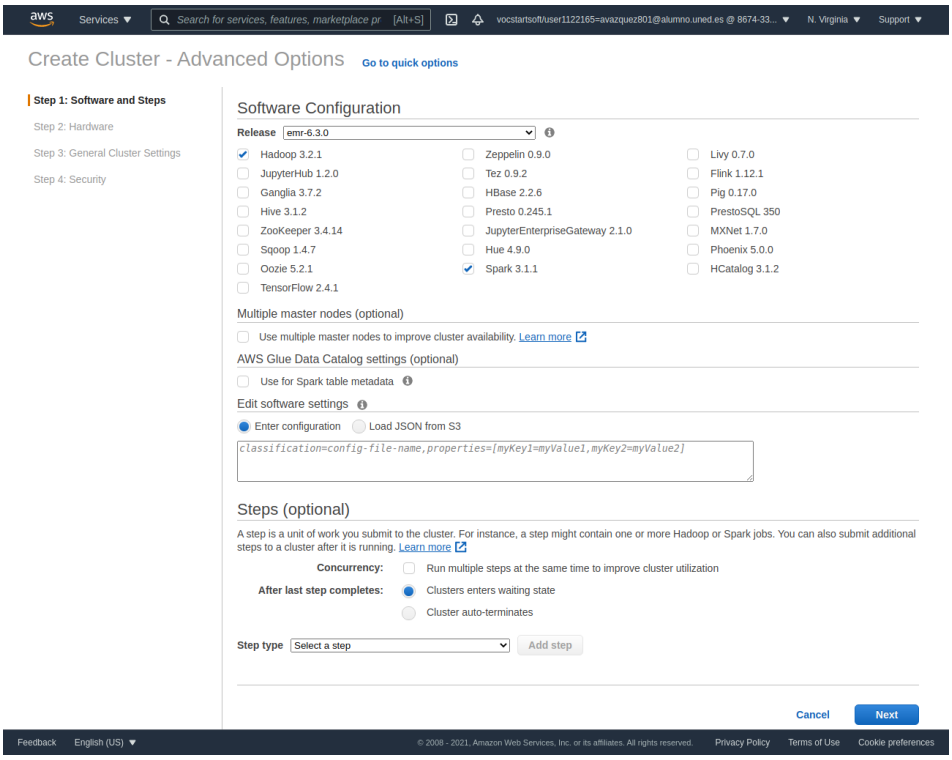


Figura 5.2: Configuración en la nube: Software.

con un dataset reducido pero equivalente a efectos de particionado (Secciones 5.4 y 5.5), y se realizarán también pruebas de rendimiento con el dataset completo, para tener una medida comparable (Sección 5.6).

En la segunda sección se configuran las instancias que compondrán el clúster, así como la red de interconexión. Para la realización de pruebas preliminares se ha trabajado con instancias de tipo `m4.large` por dos motivos: principalmente (i) por coste, ya que para depuración y pruebas iterativas interesa reducir al máximo los costes, y además (ii) para comprobar que es posible desplegar un clúster con máquinas modestas. Esto es importante ya que facilita la detección de problemas de distribución de carga, y porque facilita el acceso a la comunidad al proyecto. Si por el contrario los requisitos de entrada fueran altos, bien por las especificaciones mínimas de las instancias o por la necesidad de hardware específico como pueden ser GPUs (esta problemática se trató en la Sección 2.1.3), el proyecto perdería usuarios potenciales.

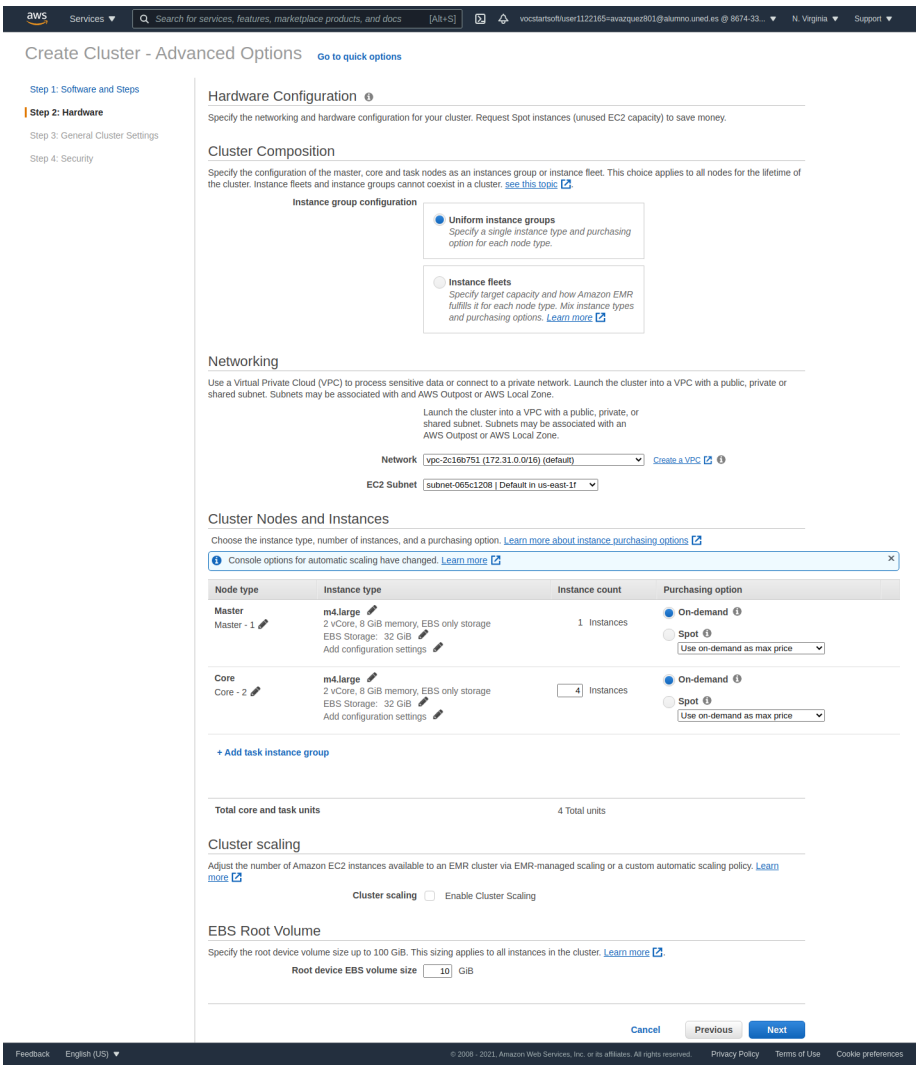


Figura 5.3: Configuración en la nube: Hardware.

En la tercera sección es posible configurar varios aspectos relativos a logging y a depuración, y también un aspecto importante, el aprovisionado personalizado. En la subsección "Bootstrap actions" es posible enlazar un script alojado en AWS S3 para que sea ejecutado en todas las máquinas una vez finalizado su aprovisionado. Esto permite gestionar las dependencias en las máquinas del clúster de manera sencilla.

Este script simplemente se trae la versión empaquetada del correlacionador a la máquina, crea un entorno virtual e instala el paquete (las dependencias están enlazadas en el propio paquete como se vio en la Sección 4.6).

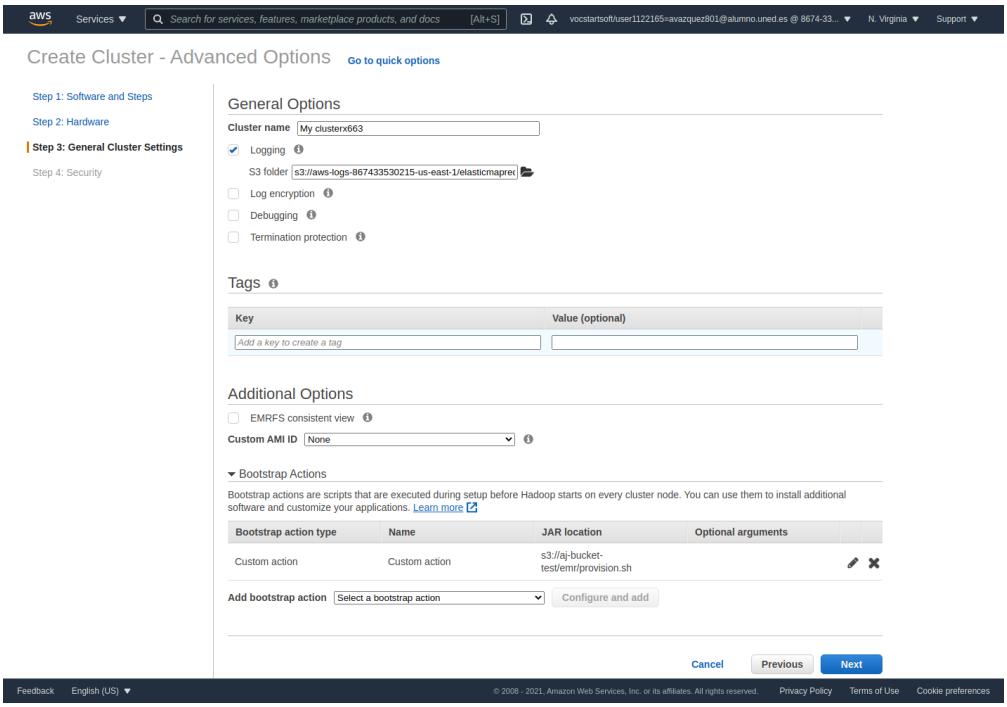


Figura 5.4: Configuración en la nube: Ajustes generales.

En la cuarta sección se configura la clave de acceso para poder conectarse a a través del protocolo SSH a las máquinas. Únicamente es necesario conectarse a la máquina principal o "master", aunque durante la depuración del sistema ocasionalmente fue necesario conectarse a los "cores".

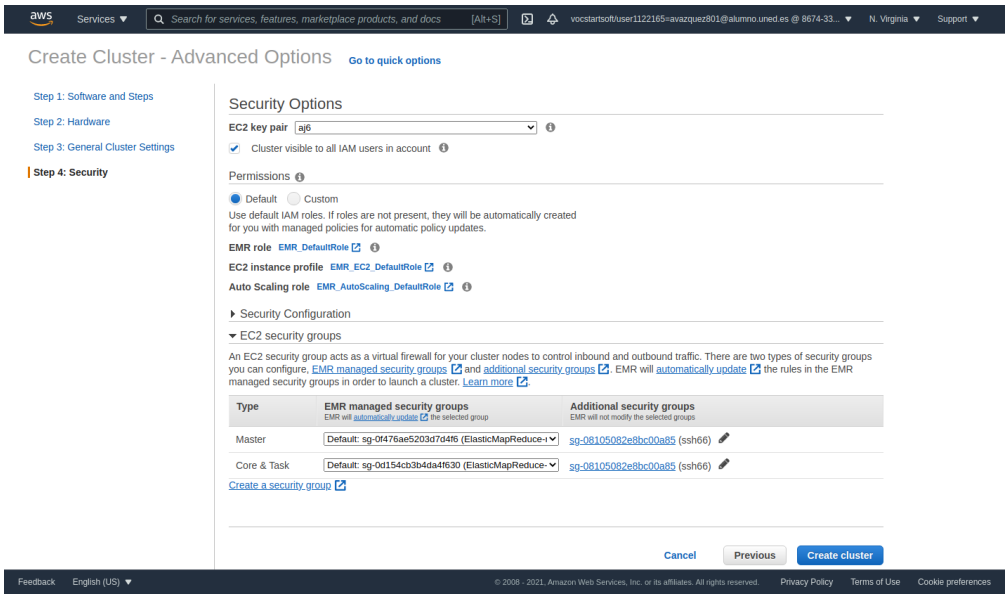


Figura 5.5: Configuración en la nube: Seguridad.

Una vez completada la configuración comenzará el aprovisionado del clúster. Este proceso puede llevar unos minutos, durante los cuales es posible comprobar el estado de configuración (Figura 5.6).

Al finalizar el aprovisionado es posible comprobar el estado general del clúster (Figura 5.7).

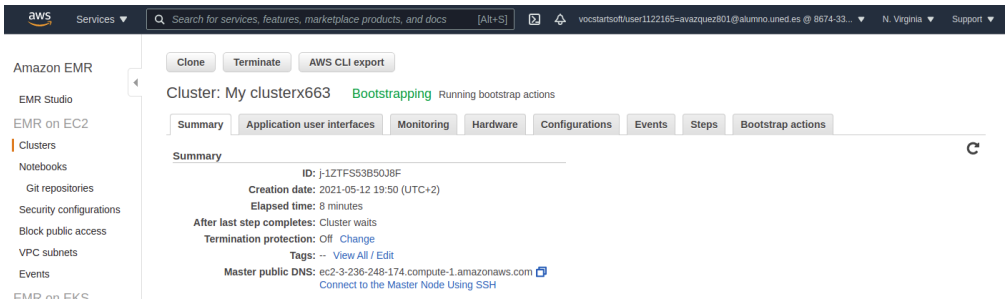


Figura 5.6: Configuración en la nube: Arranque.

Desde esta interfaz es posible comprobar tanto la configuración como el estado general del clúster, acceder a información de monitorización, etc. Destacan también unos enlaces directos a las interfaces web de Spark y de Yarn en la sección "Application user interfaces", donde es posible seguir el estado de la ejecución de las tareas del mismo modo que se hizo en la Sección 4.8.

A continuación se muestra el contenido del fichero de configuración.

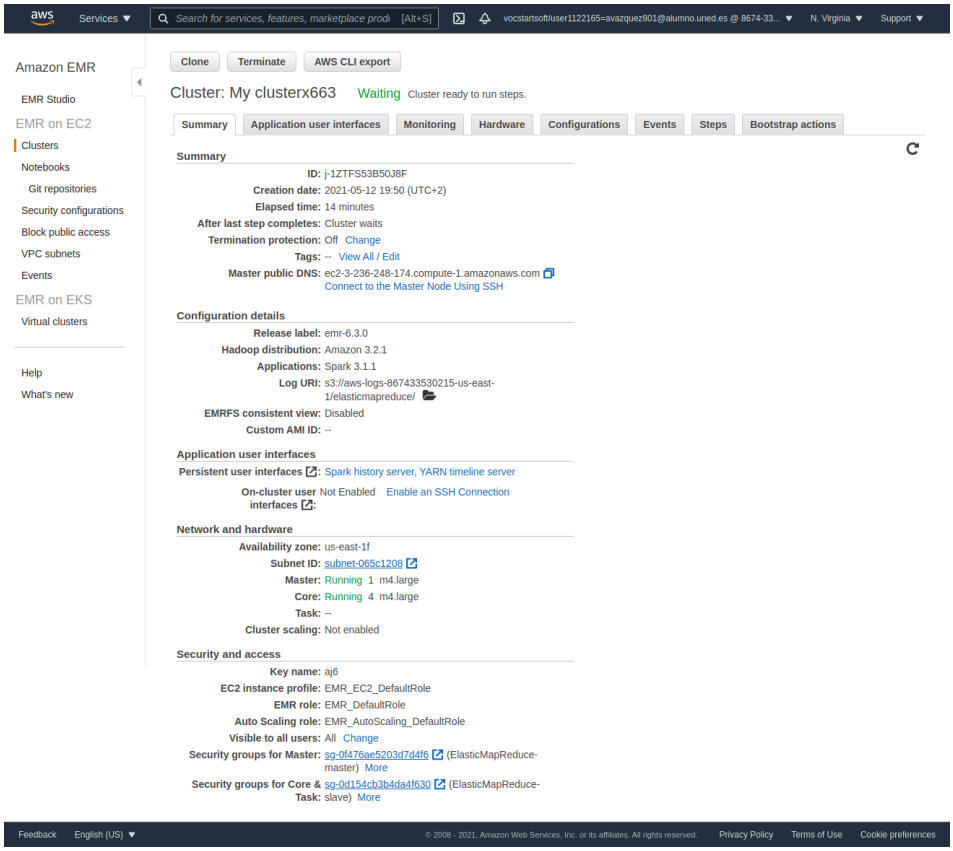


Figura 5.7: Configuración en la nube: Preparado.

```
[Experiment]
Experiment folder: /home/hadoop/exp2
Spark input files: s3://aj-bucket-test/emr/media/split_sub/*/*/test*.vt
Spark home: /usr/lib/spark

[Files]
Output directory: s3://aj-bucket-test/emr/output

[Spark]
spark.pyspark.python: /home/hadoop/venv/bin/python
spark.pyspark.driver: /home/hadoop/venv/bin/python
spark.executor.cores: 2
spark.driver.memory: 2g
spark.executor.memory: 4g
spark.driver.maxResultSize: 100g
```

La configuración es muy similar a la presentada en la Sección 4.8, pero cambiando las rutas de entrada y salida por ubicaciones en AWS S3, y cambiando el número de cores por ejecutor para que coincida con el número de cores virtuales de una instancia, ya que (durante pruebas preliminares se comprobó que daba mejores resultados que dejar el valor por defecto).

5.4. Pruebas Preliminares

En esta sección se describe brevemente la ejecución de una tarea de procesado en la nube. Tras lanzar una conexión por SSH, el sistema está preparado para ejecutar una correlación. A continuación se trae de AWS S3 un fichero comprimido con los datos de configuración de un experimento, se descomprimen en una carpeta, se activa el entorno virtual, y se lanza el proceso de correlación apuntando al fichero de configuración (Figura 5.8).

Figura 5.8: Ejecución en la nube: Lanzamiento de correlación.

Tras unos momentos dará comienzo la etapa de mapeo donde se leerán los ficheros de entrada. Como puede comprobarse se lanzan tantas tareas como ficheros de entrada (Figura 5.9).

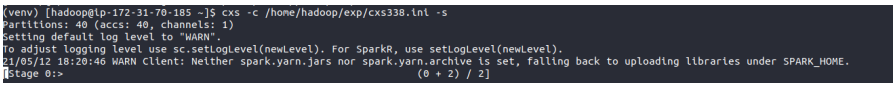


Figura 5.9: Ejecución en la nube: Lectura de ficheros de entrada.

Cuando la lectura se haya completado comenzará la redistribución de los registros generados, y comenzará la etapa de reducción. Como puede comprobarse se preparan tantas tareas como particiones se calcularon anteriomente basadas en la configuración, y puede verse que se ejecutan tantos procesos simultáneos como núcleos hay disponibles en el clúster (en este caso 4 nodos "core" con 4 núcleos por nodo "core" (Figura 5.10).

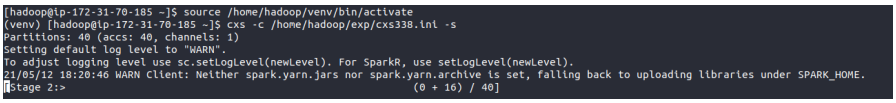


Figura 5.10: Ejecución en la nube: Cálculos de correlación (I).

A lo largo del proceso se irán completando tareas, como se muestra en la siguiente captura (Figura 5.11).

Y finalmente los archivos de salida generados se escribirán a un "bucket" en AWS S3 (Figura 5.12).

De igual modo que se hizo en la Sección 4.8, a continuación se muestran unas capturas de la interfaz web de gestión de trabajos de Spark.

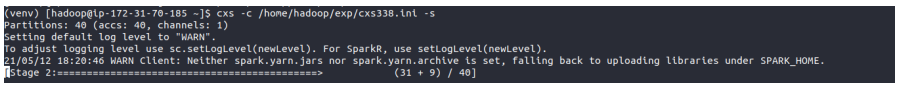


Figura 5.11: Ejecución en la nube: Cálculos de correlación (II).

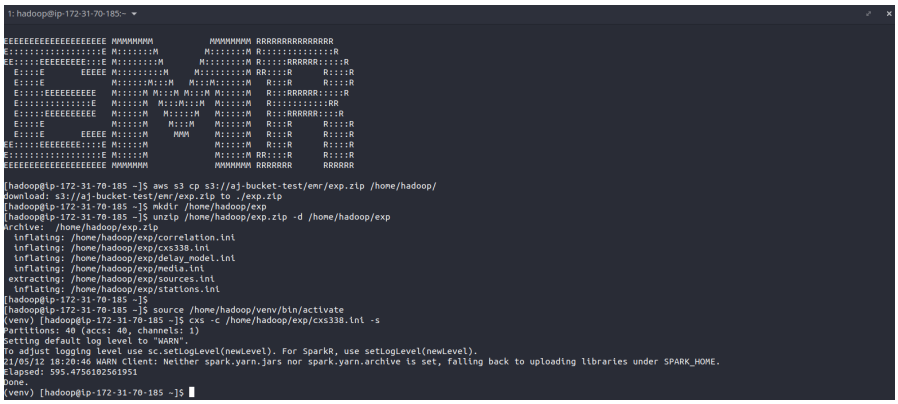


Figura 5.12: Ejecución en la nube: Proceso completado.

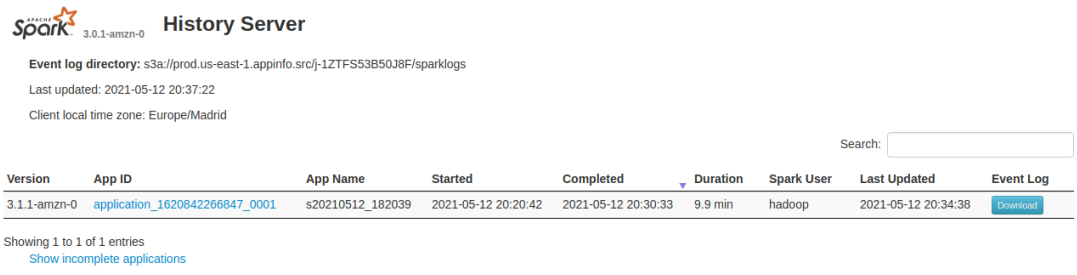


Figura 5.13: Interfaz de Spark en la nube: trabajos.

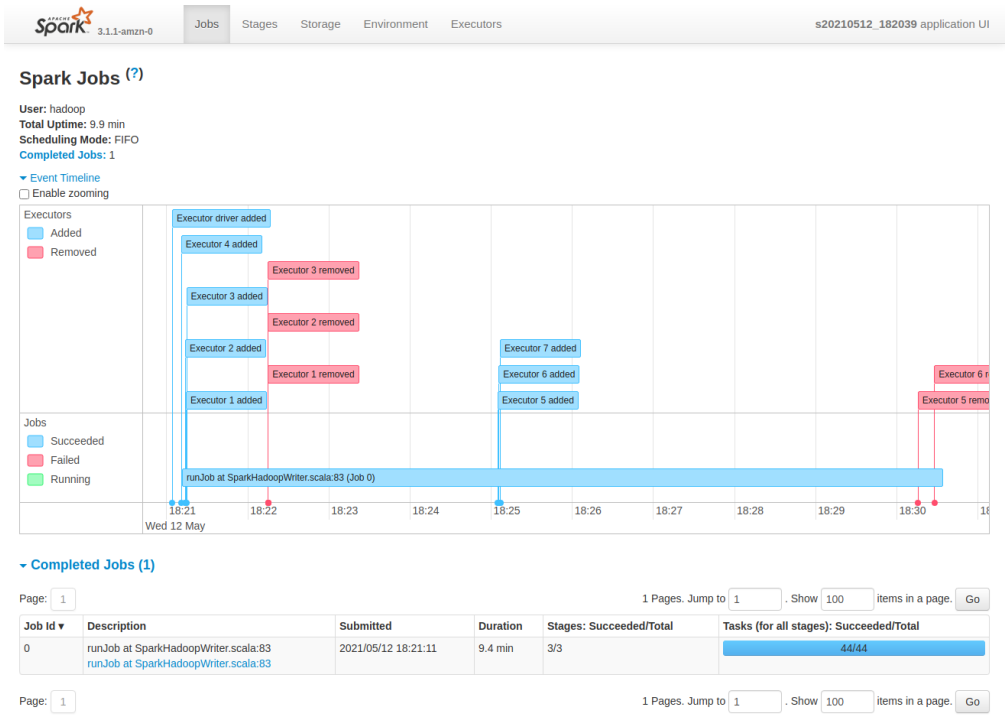


Figura 5.14: Interfaz de Spark en la nube: Resumen del trabajo.

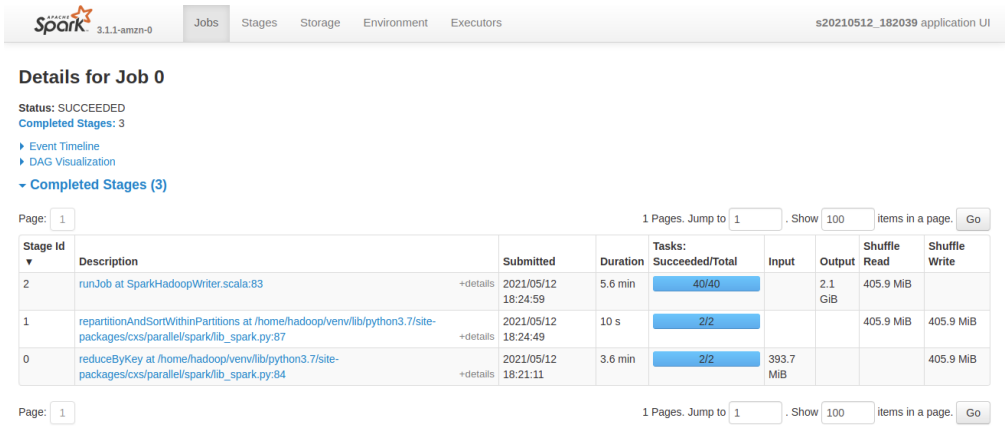


Figura 5.15: Interfaz de Spark en la nube: Etapas del trabajo.

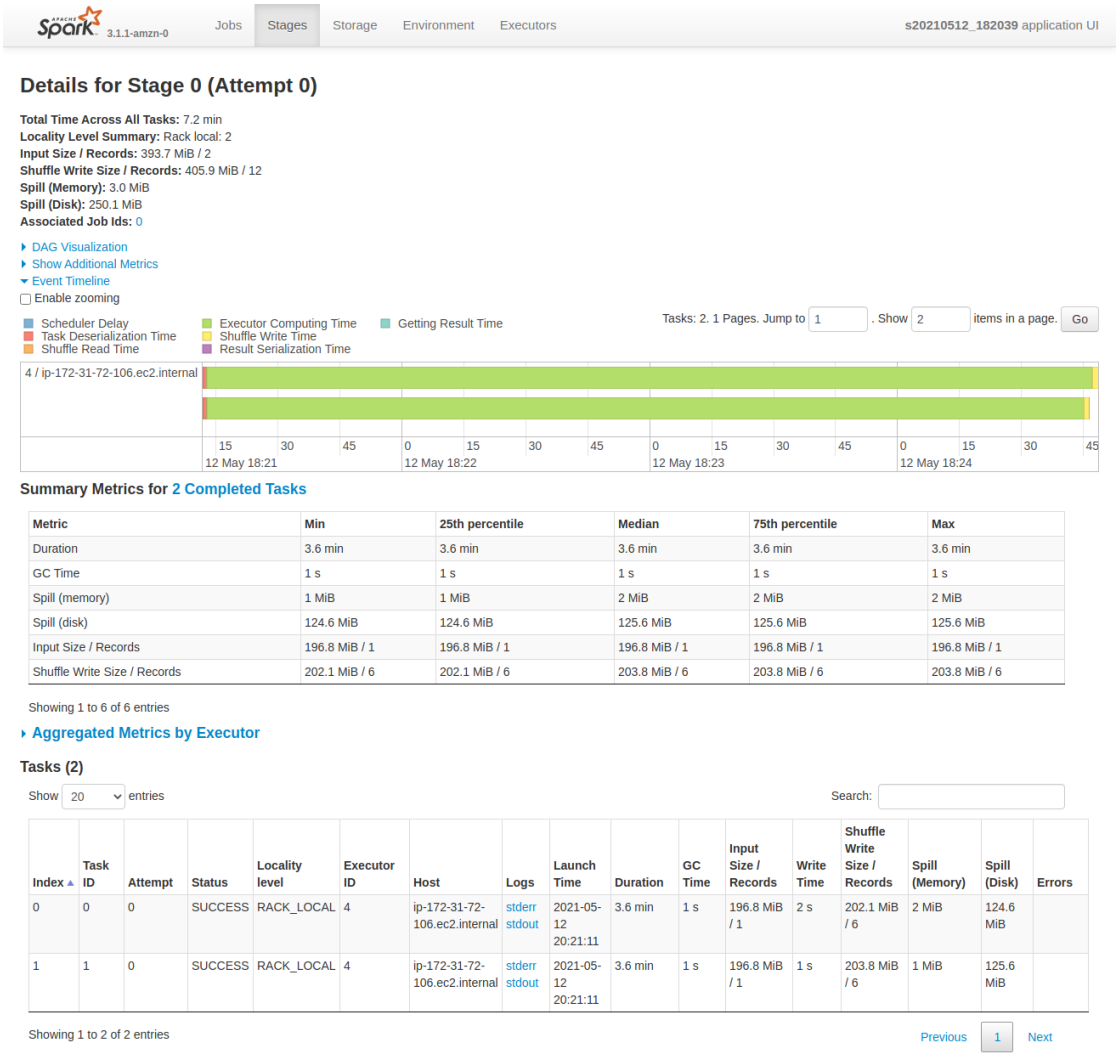


Figura 5.16: Interfaz de Spark en la nube: Etapa 0 (mapeado).

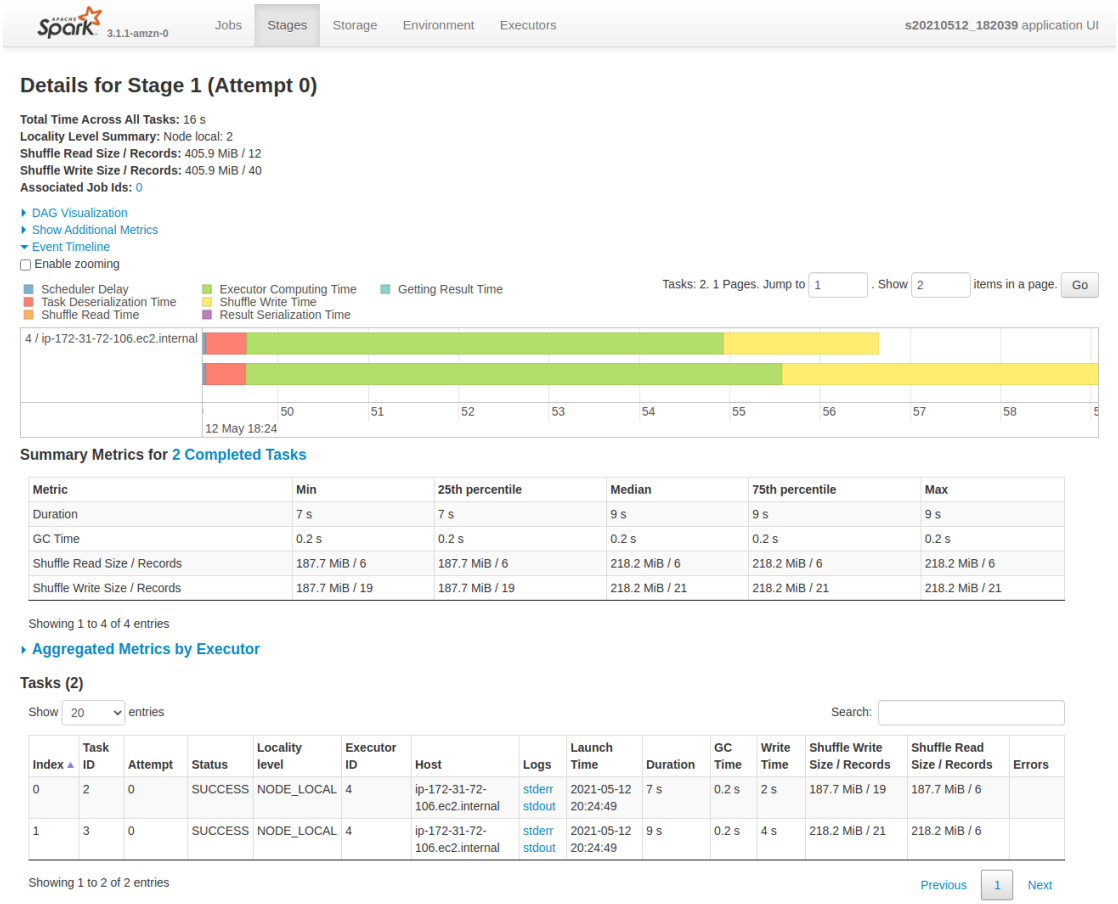
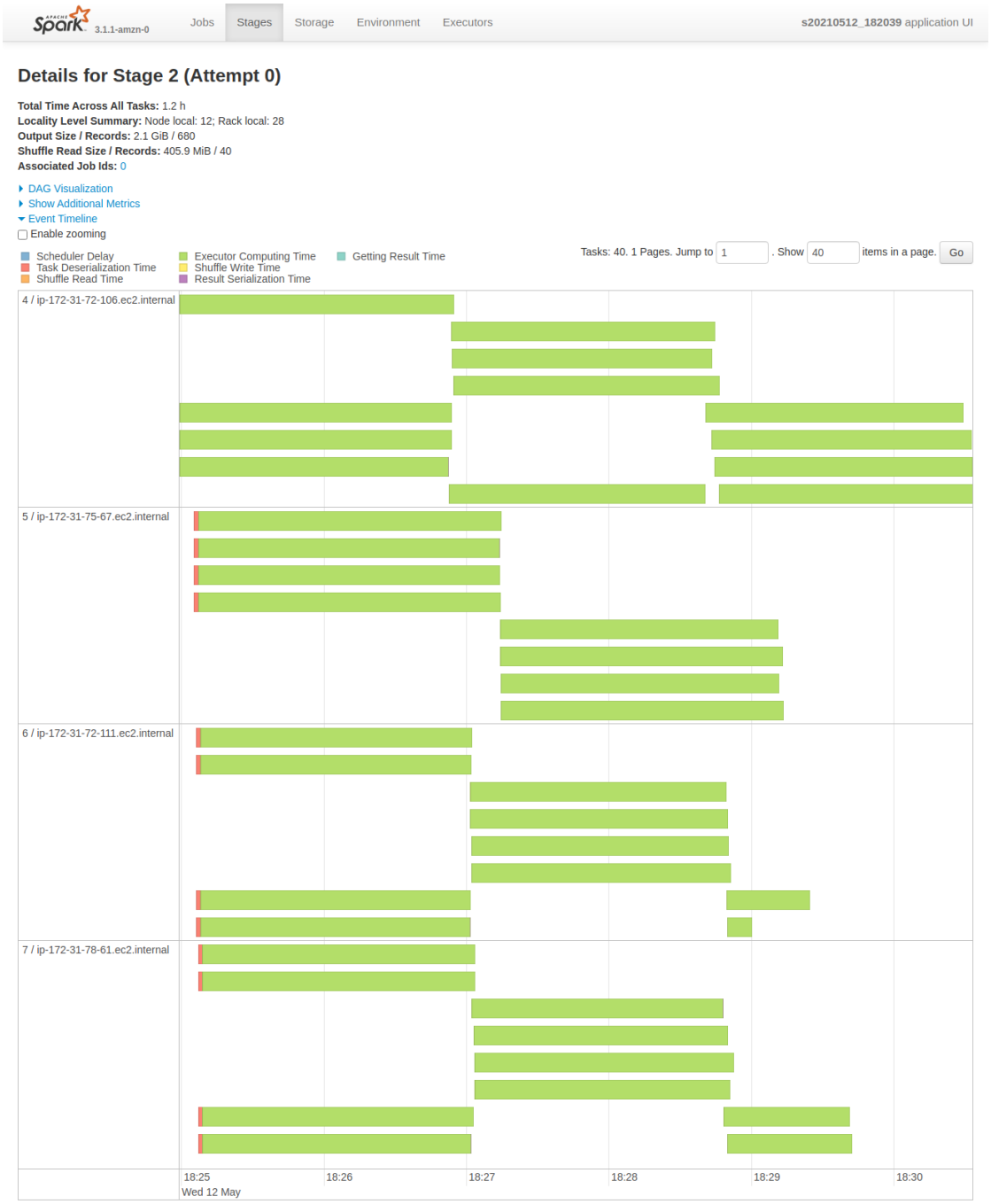


Figura 5.17: Interfaz de Spark en la nube: Etapa 1 (redistribución de datos).

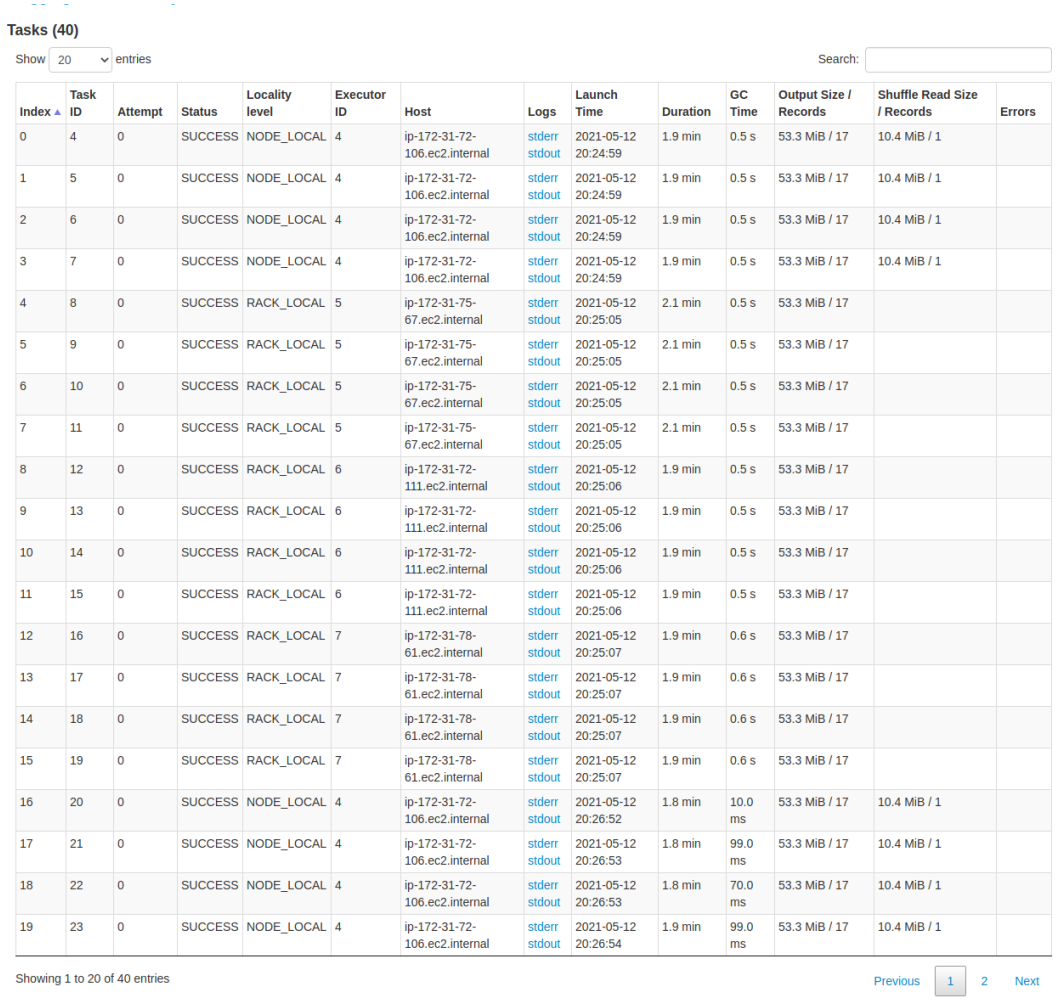


Summary Metrics for 40 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	10 s	1.8 min	1.9 min	1.9 min	2.1 min
GC Time	10.0 ms	75.0 ms	0.1 s	0.5 s	0.6 s
Output Size / Records	53.3 MiB / 17	53.3 MiB / 17	53.3 MiB / 17	53.3 MiB / 17	53.3 MiB / 17
Shuffle Read Size / Records	651.3 KiB / 1	10.4 MiB / 1	10.4 MiB / 1	10.4 MiB / 1	10.4 MiB / 1

Showing 1 to 4 of 4 entries

Figura 5.18: Interfaz de Spark en la nube: Etapa 2 (reducción, línea temporal).



Tasks (40)

Show 20 entries Search:

Index ▲	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Time	Output Size / Records	Shuffle Read Size / Records	Errors
0	4	0	SUCCESS	NODE_LOCAL	4	ip-172-31-72-106.ec2.internal	stderr stdout	2021-05-12 20:24:59	1.9 min	0.5 s	53.3 MiB / 17	10.4 MiB / 1	
1	5	0	SUCCESS	NODE_LOCAL	4	ip-172-31-72-106.ec2.internal	stderr stdout	2021-05-12 20:24:59	1.9 min	0.5 s	53.3 MiB / 17	10.4 MiB / 1	
2	6	0	SUCCESS	NODE_LOCAL	4	ip-172-31-72-106.ec2.internal	stderr stdout	2021-05-12 20:24:59	1.9 min	0.5 s	53.3 MiB / 17	10.4 MiB / 1	
3	7	0	SUCCESS	NODE_LOCAL	4	ip-172-31-72-106.ec2.internal	stderr stdout	2021-05-12 20:24:59	1.9 min	0.5 s	53.3 MiB / 17	10.4 MiB / 1	
4	8	0	SUCCESS	RACK_LOCAL	5	ip-172-31-75-67.ec2.internal	stderr stdout	2021-05-12 20:25:05	2.1 min	0.5 s	53.3 MiB / 17		
5	9	0	SUCCESS	RACK_LOCAL	5	ip-172-31-75-67.ec2.internal	stderr stdout	2021-05-12 20:25:05	2.1 min	0.5 s	53.3 MiB / 17		
6	10	0	SUCCESS	RACK_LOCAL	5	ip-172-31-75-67.ec2.internal	stderr stdout	2021-05-12 20:25:05	2.1 min	0.5 s	53.3 MiB / 17		
7	11	0	SUCCESS	RACK_LOCAL	5	ip-172-31-75-67.ec2.internal	stderr stdout	2021-05-12 20:25:05	2.1 min	0.5 s	53.3 MiB / 17		
8	12	0	SUCCESS	RACK_LOCAL	6	ip-172-31-72-111.ec2.internal	stderr stdout	2021-05-12 20:25:06	1.9 min	0.5 s	53.3 MiB / 17		
9	13	0	SUCCESS	RACK_LOCAL	6	ip-172-31-72-111.ec2.internal	stderr stdout	2021-05-12 20:25:06	1.9 min	0.5 s	53.3 MiB / 17		
10	14	0	SUCCESS	RACK_LOCAL	6	ip-172-31-72-111.ec2.internal	stderr stdout	2021-05-12 20:25:06	1.9 min	0.5 s	53.3 MiB / 17		
11	15	0	SUCCESS	RACK_LOCAL	6	ip-172-31-72-111.ec2.internal	stderr stdout	2021-05-12 20:25:06	1.9 min	0.5 s	53.3 MiB / 17		
12	16	0	SUCCESS	RACK_LOCAL	7	ip-172-31-78-61.ec2.internal	stderr stdout	2021-05-12 20:25:07	1.9 min	0.6 s	53.3 MiB / 17		
13	17	0	SUCCESS	RACK_LOCAL	7	ip-172-31-78-61.ec2.internal	stderr stdout	2021-05-12 20:25:07	1.9 min	0.6 s	53.3 MiB / 17		
14	18	0	SUCCESS	RACK_LOCAL	7	ip-172-31-78-61.ec2.internal	stderr stdout	2021-05-12 20:25:07	1.9 min	0.6 s	53.3 MiB / 17		
15	19	0	SUCCESS	RACK_LOCAL	7	ip-172-31-78-61.ec2.internal	stderr stdout	2021-05-12 20:25:07	1.9 min	0.6 s	53.3 MiB / 17		
16	20	0	SUCCESS	NODE_LOCAL	4	ip-172-31-72-106.ec2.internal	stderr stdout	2021-05-12 20:26:52	1.8 min	10.0 ms	53.3 MiB / 17	10.4 MiB / 1	
17	21	0	SUCCESS	NODE_LOCAL	4	ip-172-31-72-106.ec2.internal	stderr stdout	2021-05-12 20:26:53	1.8 min	99.0 ms	53.3 MiB / 17	10.4 MiB / 1	
18	22	0	SUCCESS	NODE_LOCAL	4	ip-172-31-72-106.ec2.internal	stderr stdout	2021-05-12 20:26:53	1.8 min	70.0 ms	53.3 MiB / 17	10.4 MiB / 1	
19	23	0	SUCCESS	NODE_LOCAL	4	ip-172-31-72-106.ec2.internal	stderr stdout	2021-05-12 20:26:54	1.9 min	99.0 ms	53.3 MiB / 17	10.4 MiB / 1	

Showing 1 to 20 of 40 entries Previous 1 2 Next

Figura 5.19: Interfaz de Spark en la nube: Etapa 2 (reducción, resultados).

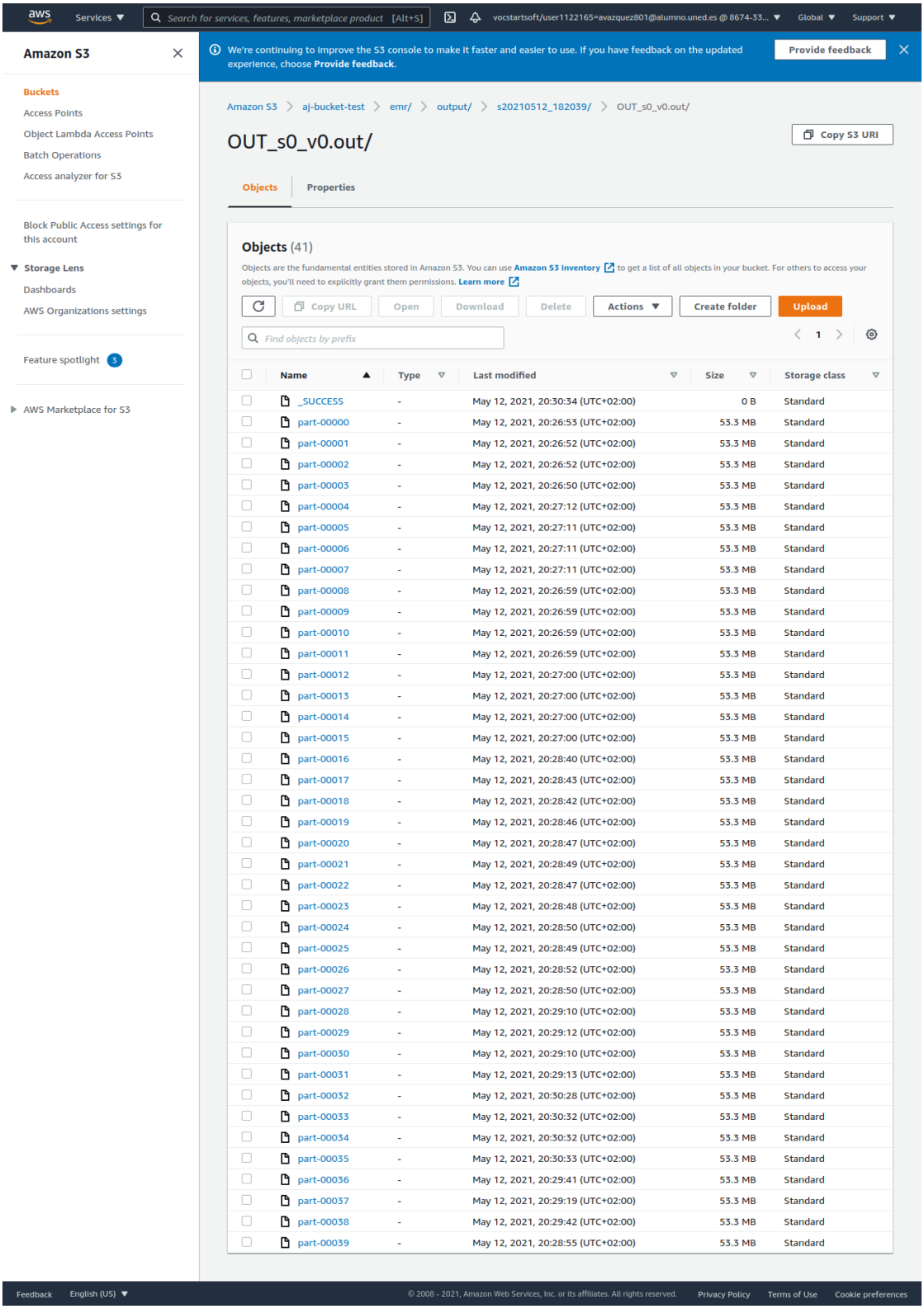


Figura 5.20: Resultados en S3 tras procesar el dataset reducido.

A continuación se muestran dos extractos pequeños de dos ficheros de salida:

part-00000 fichero que corresponde con la ventana de acumulación 0, e incluye autocorrelaciones para todas las estaciones, así como todos los pares de correlaciones de las tuplas formadas por los pares estación-polarización, por tanto con un total de 16 líneas de salida.

```
px-0.0-0.0-a.0.0.0-sxa19 0.1 0 0.0 0.015594936964547366 -0.0 -0.0 -0.0 0.0 0.0 0.0 0.0
    ↪ 0.0 0.0 0.0 0.0 0.0 2000 20000000 2 10549898 r 0 0 0 3480400000.0 0.512 no
    ↪ 311898.0 b64_dp_1EbM3mGCsj4AAAAAAAAAAMSZVyVF8bg+AAAAAAAAAAB+rwrYxySx [...]
[...]
```

part-00038 fichero que corresponde con la ventana de acumulación 38, donde aplican las mismas consideraciones que las explicadas para el archivo anterior.

```
px-0.0-0.0-a.38.38.0-sxa19 0.1 0 0.0 0.015592966232119839 -0.0 -0.0 -0.0 0.0 0.0 0.0 0.0
    ↪  0.0 0.0 0.0 0.0 0.0 2000 20000000 2 10551859 r 0 0 0 3480400000.0 0.512 no
    ↪ 312433.0 b64_dp_kT+5pCXitj4AAAAAAAAAAN/EK1h [...]
[...]
```

Como se indicó anteriormente, para este escenario se generan 40 ficheros de salida, uno por cada partición (en este caso un por cada ventana de acumulación).

Si se comparan estos resultados con los del final de la Sección 4.8 obtenidos ejecutando Spark en un entorno local, puede observarse que las fracciones mostradas coinciden. Si se miden estas diferencias con la herramienta de comparación para el fichero correspondiente a la primera ventana de acumulación se obtiene un error despreciable, tal y como se muestra en la Figura 5.21

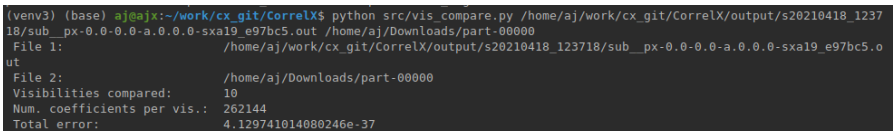


Figura 5.21: Comparación de resultados de ejecución con Spark en local y en la nube.

5.5. Pruebas de Escalabilidad con Dataset Reducido

El objetivo de estas pruebas es estudiar cómo afecta al rendimiento del sistema la variación del número de máquinas de clúster. Para cada iteración se sigue la misma metodología que en las pruebas preliminares:

1. Configuración y aprovisionado del clúster.
2. Copia de la configuración de la configuración del experimento al nodo principal.
3. Ejecución del procesado.
4. Registro del tiempo transcurrido.

En las siguientes subsecciones se describe el "dataset" utilizado, la configuración del clúster, y se describe una ejecución paso a paso.

5.5.1. Dataset

Se usarán los mismos datos que los de las pruebas preliminares (es decir, datos con similares características a los de la línea base pero reduciendo la frecuencia de muestreo en un factor de 100 para poder trabajar con ficheros de 200 MB en vez de 20 GB). Además, para mejorar el grado de paralelización en la etapa de lectura, los datos a procesar se subirán al "bucket" de S3 partidos en bloques. Este particionado inicial se realizó con un script bash básico que llama al comando `split` especificando como tamaño de los bloques un múltiplo de tamaño de trama VDIF. El script tarda unos pocos segundos en completar su ejecución. En concreto se configuró para generar bloques de 10000 tramas por bloque, siendo cada trama de 1034 bytes. Este tamaño de trama es fácilmente obtenible ejecutando la herramienta incluída en el proyecto legado `vdif_info`.

A modo de resumen, la única diferencia con la configuración presentada en la referencia [36] es la reducción de los datos de entrada de 20 GB a 200 MB por fichero, pero como se explicó la utilización de tasas de procesado permite hacer los resultados comparables.

5.5.2. Configuración

Para la configuración de las máquinas se ha elegido una configuración similar a la de la Sección 5.3, con máquinas `m4.large` (Figura 5.22). Las características detalladas de este tipo de máquinas pueden consultarse en el enlace [63], y se resumen a continuación:

- Procesador: Intel Xeon E5-2676 v3 de 2,4 GHz. Número de núcleos virtuales: 2.
- Memoria: 8 GB.
- Almacenamiento: 32 GB, sólo EBS (Elastic Block Storage, un servicio de almacenamiento de AWS), con un ancho de banda de red dedicado de 450 Mbps.

No se realizó ningún ajuste de rendimiento adicional, más allá de buscar una configuración inicial válida. El número de máquinas se varió entre 1 (2 vCores) y 8 (16 vCores) debido a las limitaciones del cupón utilizado para acceder a los servicios en la nube.

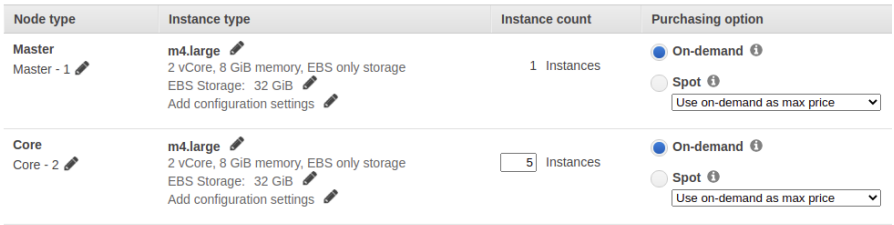


Figura 5.22: Instancias del clúster de AWS EMR.

5.5.3. Rendimiento

Los resultados obtenidos se muestran en la Tabla 5.2. En este caso la tasa de procesado media es 0.13 MB/s. En el siguiente capítulo se comparan estos resultados con los de la línea base de pruebas.

La estructura de la tabla es similar a la presentada en la Sección 5.2.3, pero cambiando el número de núcleos virtuales por las limitaciones comentadas anteriormente. Igual que para la

Tabla 5.2: Tasas de procesado por vCore del sistema presentado en este TFM según número de estaciones ejecutando el procesado en AWS EMR.

Número de estaciones	*Tasa por vCore, 2 vCores (1x2)*	*Tasa por vCore, 4 vCores (2x2)*	*Tasa por vCore, 8 vCores (4x2)*	*Tasa por vCore, 16 vCores (8x2)*	*Tasa por vCore, media*
2	0.16 MB/s	0.15 MB/s	0.13 MB/s	0.11 MB/s	**0.13 MB/s**
4	-	0.14 MB/s	0.14 MB/s	0.11 MB/s	**0.13 MB/s**

otra tabla, las tasas por vCore se calculan en cada caso dividiendo el volumen total de datos por el tiempo de ejecución y por el número de núcleos virtuales.

Como prueba adicional, se midió el "speedup" o mejora de rendimiento relativa a la introducción de nodos en el clúster en el mismo rango de número de máquinas considerado arriba, añadiendo además puntos intermedios. Así, esta métrica se calcula, para cada valor del eje de abscisas (número de máquinas en el clúster), como el tiempo de ejecución con un único nodo ejecutor dividido por el tiempo de ejecución con el número de nodos correspondiente.

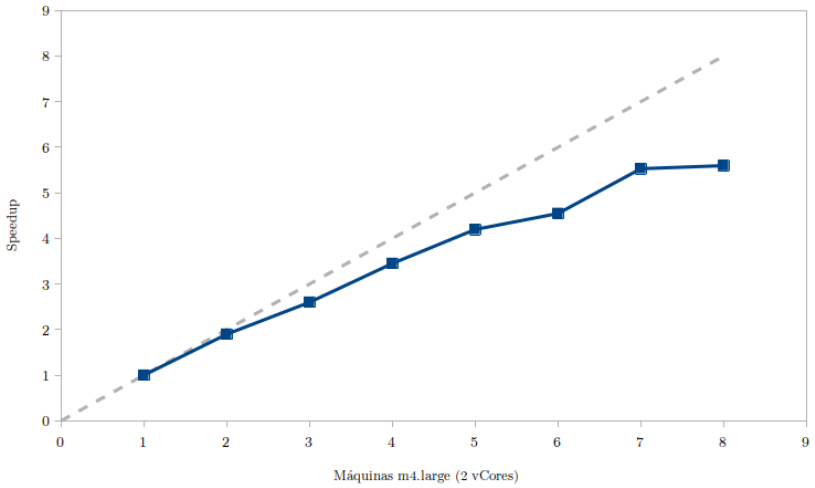


Figura 5.23: Speedup frente a número de instancias m4.large (2 vCores) en AWS EMR para 2 estaciones.

Idealmente la curva mostrada debería parecerse lo máximo posible a una recta de pendiente unidad, a continuación se explican las diferencias entre resultados ideales y los obtenidos. Dado que tanto en las fases de procesado de ficheros de entrada con en la de correlación existen 40 particiones, es posible razonar sobre el "speedup" teórico que es posible conseguir basado en el número de cores. Así, con una máquina, es decir, 2 vCores, es fácil ver que habrá $40/\lceil 2 \rceil = 20$ sub-etapas por fase (corresponde con el 1 en el eje de abscisas de la Figura 5.23). La función cielo es importante ya que el número de sub-etapas sólo puede ser entero, lo que pasará en la última sub-etapa es que no todos los vCores estarán realizando cálculos. Con dos máquinas habrá $40/\lceil 4 \rceil = 10$ sub-etapas por fase (la mitad de 20), con lo que la mejora esperada será de 1 a 2. Con tres máquinas $40/\lceil 6 \rceil \approx 6{,}66$, lo que resulta en un "speedup" de $20/\lceil 6{,}66 \rceil \approx 2{,}86$, y así sucesivamente. En la Figura 5.23 se observan valores parecidos para 7 y 8 máquinas. Si calculamos las mejoras teóricas para estos valores obtenemos $40/14 = 2{,}85$ y $40/16 = 2{,}5$. Ambos valores resultarán en 3 sub-etapas al aplicar la función cielo, lo que justifica que para 7 y 8 máquinas se

consigue una mejora de rendimiento aproximada de $20/3 \approx 6{,}66$ (en la medición un poco menor).

En el siguiente capítulo se añaden algunas consideraciones sobre estos resultados.

5.5.4. Validación

A modo de validación, aunque se ha preparado una infraestructura de tests para intentar garantizar que no hay diferencias entre los procesados en tubería y con Spark, se volverá a realizar esta comparación con los datos procesados en la nube. Dado que el procesado en modo "tubería" es lento, se ha realizado el procesado del primer segundo del dataset. Esto nos permitirá comparar la primera ventana de acumulación.

```
(venv27) (base) aj@ajx:~/work/cx_git/CorrelX$ bash examples/run_example_test_sub200_pipe.sh


#-----------------------------------------------------
#    Configuration
#-----------------------------------------------------

Checking if this node is master:
 Node ajx (192.168.1.134 172.17.0.1) is master

Overriding configuration file!
  Run pipeline:                                   1
  Run hadoop:                                     0

Config file updated: examples/test_dataset_test/sub/correlx.ini_mod_20210530_111744_ajx

Reading configuration file...
 FFT at mapper:                0
 Hadoop delays: [initialization = 5 s], [termination = 5 s], [HDFS = 5 s]
 Single run max. number of available nodes
 ---> VBox - Configured for running on VBox
 Temp dir overriden (default: /tmp)

Creating directories:
 /home/aj/work/cx_git/CorrelX/conf/ajx/
 /home/aj/work/cx_git/CorrelX/conf/ajx/etc_hadoop_ajx/
 /home/aj/work/cx_git/CorrelX/output/e20210530_111744/

Creating nodes file:
 /home/aj/work/cx_git/CorrelX/conf/ajx/nodes
  ajx
 Nodes file overwritten!

Processing *.ini files
 Checking maximum packet size for the hdfs block size...
 Max packet size: 1032
 Max num channels: 1
 Max num polarizations: 2
 Total frames: 400000
 Input files:   ['test_s-0.vt', 'test_s-1.vt']
 Processing (correlation).ini file
 FFT size:                     262144
 Accumulation time [s]:                0.512

 Recomputing delays:
  MJD: 57350
  seconds start: 68255
  tot steps: 40
  seconds per step: 0.512
  output file: /home/aj/work/cx_git/CorrelX/examples/test_dataset_test/sub/delays.ini
  (Silent output for delay libs)

  Using model from file
 Acc. periods: 40
```

Figura 5.24: Ejecución en modo "tubería" en local de la primera ventana de acumulación del dataset (primera parte).

La ejecución se muestra en las Figuras 5.28 y 5.25. La primera corresponde con pasos de configuración y preparación del comando enlazando etapas, y la segunda con la ejecución de este comando, donde además de mostrarse éste, se muestra también el tiempo correspondiente a la ejecución (que corresponde con 1/40 del total del dataset), ya que únicamente se lee el primer segundo. Dado que la ejecución de la correlación para 1/40 del total del dataset necesitó 1035 segundos, el procesado del dataset completo en modo "tubería" necesitaría $1035 \cdot 40 = 41400$ s, es decir, 11 horas y media.

Figura 5.25: Ejecución en modo "tubería" en local de la primera ventana de acumulación del dataset (segunda parte).

Para poder comparar los resultados, extraeremos del fichero resultante los resultados correspondientes a las primeras 17 líneas del fichero de salida.

```
px-0.0-0.0-a.0.0.0-sxa19 [...]
px-0.0-0.1-a.0.0.0-sxa19 [...]
px-0.0-1.0-a.0.0.0-sxa19 [...]
px-0.0-1.1-a.0.0.0-sxa19 [...]
px-0.1-0.1-a.0.0.0-sxa19 [...]
px-0.1-1.0-a.0.0.0-sxa19 [...]
px-0.1-1.1-a.0.0.0-sxa19 [...]
px-1.0-1.0-a.0.0.0-sxa19 [...]
px-1.0-1.1-a.0.0.0-sxa19 [...]
px-1.1-1.1-a.0.0.0-sxa19 [...]
zR kpa=px-A.A-A.A-a-0-0-0-,Adjusted stack=[0,0,0,0]
zR kpa=px-A.A-A.A-a-0-0-0-,Adjusted shifts=[]
zR st=0.0,lti_stats=[10549898,10238000,0,0]
zR st=0.1,lti_stats=[10549898,10238000,0,0]
zR st=1.0,lti_stats=[10548510,10236612,0,1]
zR st=1.1,lti_stats=[10548510,10236612,0,1]
zR Failed accs=1,dismissed accs=2,in=a311898.0
```

Estas líneas corresponden con las correlaciones de todos los pares estación-polarización como se explicó antes, incluyendo las autocorrelaciones para la primera ventana de acumulación (`a.0`). Además se incluyen varias líneas de depuración, que por motivos históricos (añadidos durante la etapa de desarrollo con Hadoop para que tuvieran la misma clave y en la ordenación se fueran al final) comienzan con "zR\t". Para realizar esta extracción simplemente se ejecuta el comando

"head" como se muestra en la Figura 5.26.

```
(venv27) (base) aj@ajx:~/work/cx_git/CorrelX$ head -n 17 /home/aj/work/cx_git/CorrelX/output/e20210530_111744/OUT
_s0_v0.out > /home/aj/work/cx_git/CorrelX/output/e20210530_111744/OUT_s0_v0_acc0.out
```

Figura 5.26: Extracción de primera ventana de acumulación de los resultados obtenidos en modo "tubería".

Tras extraer estos resultados, ya podemos comparar los resultados con los obtenidos en la Sección 5.4. Para ello se utiliza la herramienta de comparación de resultados incluida en el sistema legado, como se muestra en la Figura 5.27, donde puede observarse que el error acumulado es despreciable

```
(venv3) (base) aj@ajx:~/work/cx_git/CorrelX$ python src/vis_compare.py /home/aj/work/cx_git/CorrelX/output/e20210
530_111744/OUT_s0_v0_acc0.out /home/aj/Downloads/part-00000
 File 1:                    /home/aj/work/cx_git/CorrelX/output/e20210530_111744/OUT_s0_v0_acc0.out
 File 2:                    /home/aj/Downloads/part-00000
 Visibilities compared:     10
 Num. coefficients per vis.:  262144
 Total error:               2.959846024428836e-16
```

Figura 5.27: Comparación de resultados obtenidos en modo "tubería" y en la nube para la primera ventana de acumulación.

5.6. Pruebas de Escalabilidad con Dataset Completo

Vistos los resultados de pruebas anteriores, a continuación se realizarán pruebas con el dataset completo. Esto permitirá obtener resultados más realistas a efectos de rendimiento.

5.6.1. Dataset

Los datos son equivalentes a efectos de procesado a los presentados en la Sección 5.2.1, aunque únicamente para los casos de 2 y 4 estaciones:

- Estaciones: 2 (con 2 polarizaciones por estación).
- Datos: 20 s muestreados a 2 GHz, formato VDIF. Muestras reales, 2 bits/muestra, 1 canal.
- Correlación: 0.512 s/ventana, FFT de 262144 componentes.

Cada archivo pesa aproximadamente 20 GB. En cuanto al particionado, en este caso se trabajará con 200 particiones por archivo (en vez de 20, como en las Secciones 5.4 y 5.5). Así, para el caso de 2 estaciones, esto resultará en un total de 400 tareas de procesado de ficheros de entrada, e igual que para el dataset reducido, en 40 tareas de reducción.

5.6.2. Configuración

Tras varias pruebas se ha tenido que incrementar la capacidad de las máquinas un nivel con respecto a las utilizadas en la Sección 5.5.2, es decir, máquinas `m4.xlarge` (básicamente doblan el número de núcleos virtuales y la memoria RAM [63]). Además se incrementó el tamaño del disco duro a 100 GB.

Las características de estas máquinas se resumen a continuación:

- Procesador: Intel Xeon E5-2676 v3 de 2,4 GHz. Número de núcleos virtuales: 4.

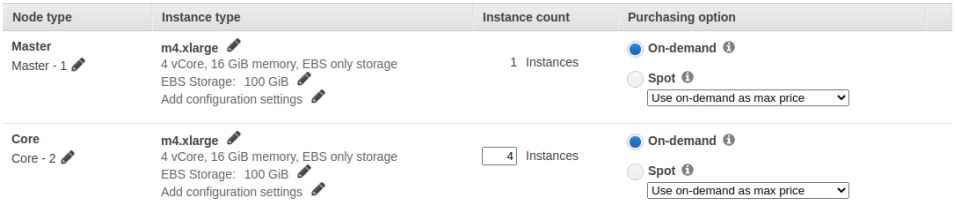


Figura 5.28: Configuración de las máquinas para el procesado del dataset completo.

- Memoria: 16 GB.
- Almacenamiento: 100 GB, sólo EBS (Elastic Block Storage, un servicio de almacenamiento de AWS), con un ancho de banda de red dedicado de 450 Mbps.

El número de máquinas se varió entre 2 (8 vCores) y 4 (16 vCores) debido a las limitaciones del cupón utilizado para acceder a los servicios en la nube, que impedían solicitar más máquinas de este tipo. En este caso fue necesario incrementar la memoria disponible para los ejecutores como se muestra a continuación:

```
[Experiment]
Experiment folder: /home/hadoop/exp2
Spark input files: s3://aj-bucket-test/emr/media/full/*/*/test*.vt
Spark home: /usr/lib/spark

[Files]
Output directory: s3://aj-bucket-test/emr/output

[Spark]
spark.pyspark.python: /home/hadoop/venv/bin/python
spark.pyspark.driver: /home/hadoop/venv/bin/python
spark.executor.cores: 2
spark.driver.memory: 8g
spark.executor.memory: 8g
spark.driver.maxResultSize: 100g
```

Es interesante destacar que el número de tareas simultáneas por nodo ejecutor en este caso viene limitado por memoria, y en este caso, aunque el número de núcleos por máquina es 4, realmente se está limitando a 2.

Aunque en este caso no se realizará validación, tanto por considerarse suficientes los resultados combinados de los tests presentados en la Sección 4.9 y la validación presentada en la Sección 5.5.4, así como por la dificultad para poder ejecutar en local este procesado, a continuación se revisará la estructura y metadatos de los resultados.

En la Figura 5.29 se muestran los ficheros de salida tras realizar el procesado.

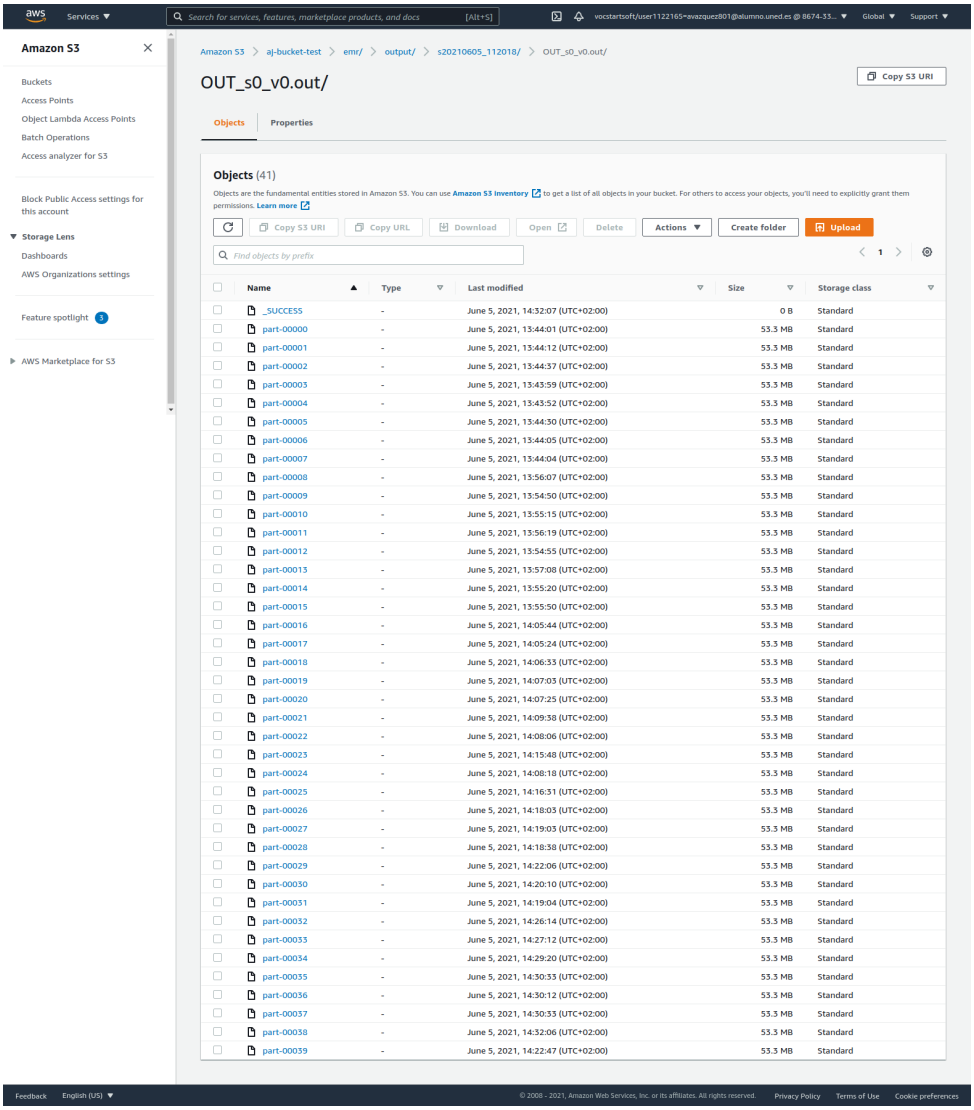


Figura 5.29: Resultados en S3 tras procesar el dataset completo.

A continuación se muestra la parte de metadatos correspondiente al archivo de salida `part-00000`.

```
px-0.0-0.0-a.0.0.0-sxa1952 [...]
px-0.0-0.1-a.0.0.0-sxa1952 [...]
px-0.0-1.0-a.0.0.0-sxa1952 [...]
px-0.0-1.1-a.0.0.0-sxa1952 [...]
px-0.1-0.1-a.0.0.0-sxa1952 [...]
px-0.1-1.0-a.0.0.0-sxa1952 [...]
px-0.1-1.1-a.0.0.0-sxa1952 [...]
px-1.0-1.0-a.0.0.0-sxa1952 [...]
px-1.0-1.1-a.0.0.0-sxa1952 [...]
px-1.1-1.1-a.0.0.0-sxa1952 [...]
zR kpa=px-A.A-A.A-a-0-0-0-,Adjusted stack=[0,0,0,0]
zR kpa=px-A.A-A.A-a-0-0-0-,Adjusted shifts=[]
zR st=0.0,lti_stats=[1054989870,1023800000,0,0]
zR st=0.1,lti_stats=[1054989870,1023800000,0,0]
zR st=1.0,lti_stats=[1054851059,1023661190,0,102]
zR st=1.1,lti_stats=[1054851059,1023661190,0,102]
zR Failed accs=0,dismissed accs=2,in=a31189870.0
```

Si se comparan estos resultados con los de la Sección 5.5.4, puede verse que en este caso las cifras correspondientes con número de FFTs sumadas por ventanas de acumulación corresponde con aproximadamente 100x los del caso del dataset reducido, como era de esperar. Los tamaños de los ficheros de salida serán similares, ya que estos son independientes de la frecuencia de muestreo (la única diferencia es el número de muestras que entran en cada ventana de acumulación.

5.6.3. Rendimiento

Los resultados obtenidos se muestran en la Tabla 5.3. En este caso la tasa de procesado media es 0.60 MB/s. En el siguiente capítulo se comparan estos resultados con los de la línea base de pruebas.

Tabla 5.3: Tasas de procesado por vCore del sistema presentado en este TFM según número de estaciones ejecutando el procesado en AWS EMR.

Número de estaciones	*Tasa por vCore con 8 vCores (2x4)*	*Tasa por vCore con 16 vCores (4x4)*	*Tasa por vCore media*
2	0.61 MB/s	0.59 MB/s	**0.60 MB/s**

Cabe recordar que el número de tareas por nodo ejecutor en este caso está limitada por memoria, y si se dispusiera de máquinas con el doble de RAM pero mismo número de núcleos, el rendimiento debería ser mayor. En efecto, durante la ejecución de un procesado en un clúster con 4 máquinas, se comprobó que el número de tareas activas simultáneas era de 8 (en vez de 16), y con 2 máquinas de 4 (en vez de 8). A pesar de ello y como medida conservadora, se han tenido en cuenta en los cálculos todos los núcleos de las máquinas, aunque no estén todos trabajando.

5.6.4. Discusión

Se intentó también el procesado del dataset completo para 4 estaciones, pero en este caso no fue posible obtener resultados por problemas de memoria [3]. Hay que recordar que la configuración

[3]Fueron problemas similares los que llevaron a incrementar en un nivel la capacidad de las máquinas utilizadas en las pruebas.

del clúster de la línea base disponía de máquinas de 104 GB (Sección 5.2.2), lo que está un orden de magnitud por encima de la memoria por máquina para estas pruebas (16 GB). Este problema de escalabilidad se tuvo en cuenta durante el diseño del sistema legado, que permite un modo de funcionamiento en el que crea una tarea de procesado por cada par de estaciones [1], lo que permite continuar escalando horizontalmente. Aunque este modo está también por tanto incluido en el sistema propuesto (por la parte de código compartida con el sistema legado), sería necesario modificar la parte correspondiente con el cálculo del número de tareas de reducción, por lo que no se llegó a probar. Esta pequeña adaptación para garantizar la escalabilidad de los cálculos con el incremento del número de estaciones se deja como línea futura. Sería interesante también, como línea futura, revisar el uso de memoria del procesado de reducción para intentar mejorar la escalabilidad.

Aprovechando este despliegue, de mayor capacidad que el utilizado para el dataset reducido, se volvió a procesar el dataset reducido a modo de comprobación. Para el clúster de 4 máquinas `m4.xlarge`, con la configuración de 4 GB por ejecutor, el procesado para dos estaciones llevó aproximadamente 240 s, que corresponde con una tasa muy similar a la obtenida para la configuración de 8 máquinas `m4.large` (en ambos casos 16 núcleos virtuales por máquina).

5.7. Resumen

En este capítulo se han descrito las pruebas en la nube del sistema presentado, incluyendo pruebas de escalabilidad. Se ha intentado que los resultados obtenidos fueran comparables a unos resultados de referencia para un correlacionador considerado estándar de facto, de manera que en el siguiente capítulo se realizará esta comparación.

Capítulo 6

Evaluación

En este capítulo se estudian los resultados obtenidos a través de una comparación con los resultados de referencia de la línea base.

6.1. Dataset

Como se vio en las Secciones 5.2.1, 5.5.1 y 5.6.1, las únicas diferencias en cuanto a los datasets utilizados son (el tamaño de los ficheros para la prueba de escalabilidad) y el número de estaciones consideradas en los experimentos. Estas diferencias se resumen en la Tabla 6.1.

Tabla 6.1: Diferencias en los entornos de pruebas de la línea base y del proyecto presentado en este TFM.

Característica	*Línea base*	*Este TFM*
Número de estaciones	2-20	2-4
Tamaño de fichero	20 GB	20 GB

En ambos casos los datasets fueron generados "en laboratorio", a partir de muestras de ruido. Para el dataset reducido, esta reducción en los tamaños de fichero se consiguió dividiendo la frecuencia de muestreo original por 100, de manera que esta reducción no afecta al nivel de paralelización alcanzable con el sistema presentado. Cabe destacar de nuevo que esta reducción de tamaño se aplicó para poder conseguir tiempos de ejecución menores sin afectar a los resultados (tasas de procesado). Además las comparaciones de rendimiento se realizarán para el mismo número de estaciones (dos), y para el dataset sin reducir.

6.2. Configuración

Antes de comparar los resultados obtenidos con los de referencia es necesario comparar los entornos de prueba, para asegurarse de que los resultados son comparables, y de no ser el caso, ajustarlos como corresponda.

En la Tabla 6.2 se resumen las principales diferencias entre los entornos de pruebas. Estos datos se han recopilado en las Secciones 5.2.2 y 5.5.2. Se añaden además datos de medidas de rendimiento extraídos de las referencias [66] y [67], en concreto el número de operaciones en coma

flotante para DFFT para un único núcleo. Se eligió este benchmark por considerarse representativo del procesado, ya que como se vio en la Sección 2.1.3 el cálculo de la Transformada de Fourier es parte del mismo. Para conseguir una representación más compacta se representan las máquinas `n1-highmem-16`, `n1-highmem-32`, `n1-highmem-64` y `n1-highmem-96` como `n1-highmem-16/96`.

Tabla 6.2: Diferencias en los entornos de pruebas de la línea base y del sistema presentado en este TFM.

Característica	*Línea base*	*Este TFM*
Proveedor	GCP	AWS
Tipo de instancias	`n1-highmem-16/96`	`m4.large` / `m4.xlarge`
Procesador	Intel Xeon 2.5 GHz	Intel Xeon 2.4 GHz
Número de instancias	1	1-8 / 1-4
vCores/máquina	16-96	2 / 4
Total vCores	16-96	2-16 / 8-16
Benchmark DFFT (single core)	2.57 Gflops	2.48 Gflops
Memoria	104 GB	8 GB / 16 GB
Almacenamiento	128 GB SSD	32 GB EBS / 100 GB EBS
Proveedor	GCP	AWS

A la vista de estos resultados, y aunque las capacidades de las máquinas utilizadas en la línea base son de mayor capacidad (número de núcleos y almacenamiento), se considera que los resultados de rendimiento pueden compararse debido a la utilización de tasas relativas, y que los tipos de núcleo de procesador (y benchmarks) son similares. En cuanto a escalabilidad esto no es así, debido a que los resultados de escalabilidad presentados para la línea base corresponden a escalabilidad vertical, mientras que en los resultados presentados en este TFM corresponden a escalabilidad horizontal (ya que se consideran preferibles, tanto por coste como por escalabilidad, valga la redundancia).

Nótese que aunque la capacidad de las máquinas de la línea base son superiores en cuanto a memoria, almacenamiento, y número de núcleos (un orden de magnitud en todos los casos), el uso de tasas relativas por núcleo virtual elimina de la comparación el número de núcleos de las máquinas. En cuanto a memoria, habría que realizar más pruebas para asegurarse que en efecto esta reducción no afecta negativamente a los resultados presentados. Dado que en el caso de que sí influyera los resultados presentados serían mejores, se considera que los resultados presentados corresponden con el peor caso.

6.3. Rendimiento

Si comparamos los resultados mostrados en la Tabla 5.1 con los de la Tabla 5.3, observamos que el rendimiento de DiFX es aproximadamente 2x el que consigue el sistema presentado en este TFM (Tabla 6.3)[1] considerando la mayor tasa medida para DiFX, pero si tomamos las tasas medias (última columna de las tablas) los resultados son similares.

Cabe destacar de nuevo que las tasas obtenidas son válidas para configuraciones similares a las del dataset descrito en la Sección 5.2.1.

[1] En base a los resultados presentados en la Sección 5.6.3, estos resultados posiblemente serían mejores con máquinas con mayor relación memoria/procesador, pero se mantiene la tasa dada como peor caso.

Tabla 6.3: Diferencias en las tasas de procesado medias por núcleo de la línea base y del sistema presentado en este TFM para la configuración descrita en la Sección 5.2.1.

Resultado	*Línea base*	*Este TFM*
Tasa de procesado máxima por núcleo virtual	1.28 MB/s	0.61 MB/s
Tasa de procesado media por núcleo virtual	0.88 MB/s	0.60 MB/s

En cuanto a escalabilidad, en este caso no es posible comparar los resultados. Para el sistema de referencia se observa que los tiempos de ejecución continúan reduciéndose conforme se incrementa el número de núcleos virtuales más allá de 96, aunque el valor de speedup en el rango estudiado es de aproximadamente 1:2 (0.5). En el caso del sistema presentado, este valor sólo se estudia hasta 16 núcleos virtuales, y el valor de speedup en el rango estudiado está cerca del 1:1 (1), excepto hacia el final que es 6:8 (0.75). Si calculamos el límite teórico para el número de núcleos hasta donde el sistema presentado escala, es fácil ver que éste es aproximadamente 40 (coincide con el número de particiones), dado que a partir de este valor no es posible seguir reduciendo tiempos de ejecución. Por otra parte, cabe mencionar también que para los resultados de referencia se considera escalabilidad vertical (aumentando la capacidad de las máquinas), mientras que en este trabajo se considera escalabilidad horizontal (agregando más máquinas al clúster).

Así, aunque el valor de Speedup es mejor para el sistema presentado en este TFM (0.75 frente a 0.5), el número de CPUs virtuales para los cuales los tiempos siguen mejorando es más bajo (40 frente a más de 96). En la siguiente sección se profundizará en los motivos de estas diferencias.

Es importante mencionar que no se ha profundizado en presentar una comparación con el sistema legado. Esto es intencionado pues el objetivo del sistema presentado en este TFM no es superar el rendimiento del sistema legado, sino recortar distancias con el sistema de referencia. A modo de comparativa preliminar y por tener al menos una referencia, tras ejecutar el procesado del dataset de prueba (incluido en el proyecto legado) en el propio sistema legado se observó una mejora de un orden de magnitud, aunque estas pruebas se realizaron con un dataset pequeño con lo que deberían ser repetidas para obtener una estimación más precisa.

6.4. Precisión

Dado que el sistema presentado comparte la parte de la base de código correspondiente con los cálculos de correlación con el sistema legado (Sección 3.2), y de hecho se han preparado tests unitarios (para asegurarse que los resultados de ambos son casi exactamente iguales al procesar los mismos datos, Secciones 3.4 y 4.2) y se han realizado pruebas básicas de validación comparando los resultados del procesado sin paralelización (modo “tubería”) y con paralelización con Spark (Sección 5.5.4), podemos afirmar que la precisión de ambos sistemas es la misma.

En cuanto a la comparación de precisiones entre DiFX y el sistema legado (CorrelX), en las referencias [1, 45] se presenta una comparativa de resultados tras procesar los resultados de ambos sistemas para dos datasets (uno de ellos con datos de telescopios del EHT [20] y otro del sistema VGOS [22]) con la herramienta de postprocesado Fourfit [64]. Los resultados obtenidos son muy parecidos, con algunas diferencias de precisión, lo que hace que CorrelX, y por tanto también el sistema propuesto, deban ser considerados todavía prototipos hasta que se puedan depurar estas diferencias de precisión. Por homogeneidad con el resto de secciones, se resumen estos resultados en la Tabla 6.4.

Aunque se han solucionado varios bugs, algunos de ellos que afectaban a la precisión del

Tabla 6.4: Validación del sistema

Resultado	*DiFX vs. CorrelX*	*CorrelX vs. este TFM*
Precisión	Resultados parecidos [1, 45]	Mismos resulados (Secciones 3.4, 4.2 y 5.5.4)

sistema (como se describió en la Sección 4.4), si bien es previsible que esta precisión aumente en algunos casos, queda como línea futura realizar un trabajo de depuración para pasar de un estado de prototipo a un software listo para producción.

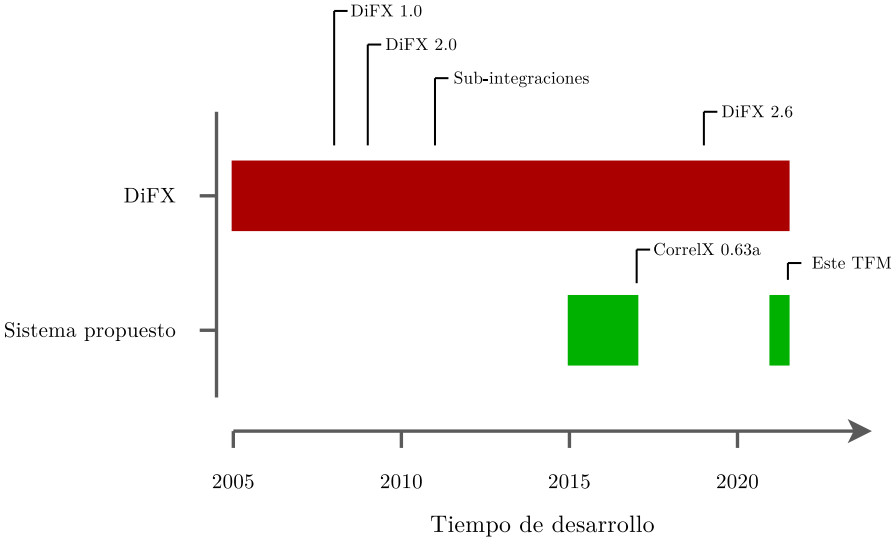
6.5. Discusión

Uno de los resultados más importantes de este TFM es la consecución de un factor rendimiento de 1:2, ya que recorta distancias colocando a ambas tasas en el mismo orden de magnitud.

Además hay que tener en cuenta varios aspectos que se detallan a continuación que abren la puerta a futuras mejoras de rendimiento que puedan reducir este factor todavía más.

Madurez de los proyectos Hay que tener en cuenta que se están comparando dos proyectos con tiempos de desarrollo muy distintos. Por un lado, el proyecto de referencia tiene más de 14 años de historia a sus espaldas y es utilizado actualmente, mientras que el proyecto presentado en esta tesis tiene poco más de dos años contando los cuatro meses de trabajo correspondientes con el trabajo presentado en este TFM (compaginado con trabajo y con el resto de asignaturas del máster del cuadrimestre). En la Figura 6.1 se comparan los tiempos de desarrollo del sistema de referencia y del sistema propuesto.

Figura 6.1: Comparativa de tiempos de desarrollo.



Ausencia de micro-optimizaciones El procesado actual realizado por el sistema presentado no incorpora mejoras que incrementen el nivel de paralelización como lo hace DiFX. Éste último incorpora mecanismos como las "sub-integraciones" [65], que permite dividir las ventanas de

acumulación en partes más pequeñas para permitir su cálculo en paralelo, con un paso final en el que se agrupan los resultados de estas sub-ventanas en los intervalos de acumulación configurados.

Este problema se pone de manifiesto con la configuración del dataset, que únicamente tiene un canal, resultando como se vio en un total de 40 particiones en la etapa de cálculo de la correlación. En la etapa de procesado de ficheros de entrada los bloques se eligieron para que no superar este límite. La implementación de este tipo de optimizaciones está fuera del alcance de este TFM, no obstante se propone su incorporación como mejora futura para incrementar la escalabilidad del sistema.

Teniendo en cuenta estas limitaciones, los resultados obtenidos se consideran muy prometedores, y se espera que este trabajo sea un empujón en el desarollo del proyecto en la comunidad.

Capítulo 7

Conclusiones

7.1. Conclusiones

En este TFM se ha presentado la ejecución de un proyecto para implementar un sistema de correlación para interferometría de base muy larga a partir de un sistema legado basado en Apache Hadop streaming de Python (desarrollado por el autor de este TFM entre 2015 y 2017), replanteando la arquitectura del sistema y cambiando su motor de paralelización a Apache Spark.

Tras presentar un estudio de viabilidad y describir la planificación, se han introducido varios conceptos para comprender el ámbito del problema y se han comparado las principales soluciones existentes al mismo. Posteriormente se ha descrito en detalle la implementación, que va mucho más allá de cambiar el motor de procesado de Hadoop a Spark, con mejoras en arquitectura, usabilidad, funcionalidad, fiabilidad, etc. Esta implementación se ha probado en local pero también en un servicio en la nube (AWS EMR), donde se han realizado pruebas de escalabilidad.

La metodología ha tenido en cuenta resultados de referencia presentados en un artículo de 2019 para el sistema considerado estándar de facto, de manera que ha sido posible comparar los resultados obtenidos en condiciones similares. Los resultados más importantes son que se ha conseguido incrementar el rendimiento del sistema legado hasta acercarse a un factor 1:2 (la mitad)[1] aproximadamente del rendimiento que da el sistema estándar, recortando distancias en un proyecto con poco más de 2 años de desarrollo de una prueba de concepto frente a más de 10 años de ejecución en entornos de producción.

En la Tabla 7.1 se resumen los objetivos presentados en la Sección 1.2. Se consideran todos estos objetivos cumplidos, y para cada uno de ellos se enlaza a la sección de este documento donde se recogen los resultados referentes al mismo. A continuación se dan detalles adicionales sobre cada uno de ellos.

- OBJ-DES-01: relativo al estudio de las soluciones existentes y su comparación con la solución presentada. Los resultados relativos a este objetivo pueden consultarse en la Sección 2.3.
- OBJ-DES-02: relativo al diseño del sistema presentado. Este diseño se presentó en detalle en el Capítulo 3, describiendo desde la arquitectura hasta la organización del código.
- OBJ-DES-03: relativo a la actualización de tecnologías, tiene en cuenta principalmente los cambios requeridos para la actualización de versiones de Python descrita en la Sección 4.1, y la adaptación para correr en Spark descrita en la Sección 4.7.

[1] Posiblemente la diferencia sería incluso menor con máquinas con mayor relación memoria/procesador.

- OBJ-DES-04: relativo a la creación de un entorno controlado para poder desarrollar el sistema evitando regresiones, corresponde con el plan de pruebas descrito en la Sección 3.4, y la implementación y ejecución de los tests descritas en las Secciones 4.2 y 4.9.
- OBJ-DES-05: relativo a las mejoras introducidas en este sistema con respecto al sistema legado. Las mejoras funcionales se describieron en la Sección 4.5, los bugs solucionados en la Sección 4.4, el empaquetado en la Sección 4.6, y la mejora de rendimiento en la Sección 6.3.
- OBJ-DES-06: relativo a la ejecución de pruebas en un entorno local. Sus resultados se describieron en la Sección 4.8.
- OBJ-DES-07: relativo a la ejecución de pruebas en un entorno en la nube. Sus resultados se describieron en la Sección 5.4.
- OBJ-DES-08: relativo a las pruebas de escalabilidad en la nube. Estos resultados se describieron en las Secciones 5.5 y 5.6.
- OBJ-DES-09: relativo a la validación del sistema. Estos resultados se describieron mediante comparación en la Sección 6.4.
- OBJ-EDU-01: relativo a los conocimientos adquiridos sobre Apache Spark. Estos conocimientos se consideran mejorados, tanto los relativos a la interacción con Spark a través de pyspark (Sección 4.7) como los relativos a la configuración del clúster (Sección 4.8).
- OBJ-EDU-02: relativo a los conocimientos adquiridos sobre el entorno en la nube de AWS. Estos conocimientos se consideran mejorados, tanto los relativos al aprovisionado de las máquinas del clúster EMR (Sección 5.3)como los relativos a la interacción general con este servicio (Sección 5.4).

Tabla 7.1: Objetivos del TFM.

Objetivo	*Resumen*	*Completado*	*Justificación*
OBJ-DES-01	Comparación	✓	Sección 2.3
OBJ-DES-02	Diseño	✓	Capítulo 3
OBJ-DES-03	Actualización de tecnologías	✓	Secciones 4.1 y 4.7
OBJ-DES-04	Entorno controlado	✓	Secciones 3.4, 4.2 y 4.9
OBJ-DES-05	Mejoras	✓	Secciones 4.4, 4.5, 4.6 y 6.3
OBJ-DES-06	Pruebas en local	✓	Sección 4.8
OBJ-DES-07	Pruebas en la nube	✓	Sección 5.4
OBJ-DES-08	Escalabilidad en la nube	✓	Sección 5.5
OBJ-DES-09	Validación	✓	Secciones 6.4 y 5.6
OBJ-EDU-01	Apache Spark	✓	Secciones 4.7 y 4.8
OBJ-EDU-02	Amazon AWS	✓	Secciones 5.3 y 5.4

Los resultados presentados se consideran muy prometedores, y se espera que sirvan de empujón al proyecto para su uso por parte de la comunidad radioastrónomica y de computación del alto rendimiento.

7.2. Líneas Futuras

A lo largo de este TFM se han presentado varias líneas de desarrollo futuro que se resumen a continuación:

Arquitectura para Streaming En la Sección 3.1 se dieron las bases para poder migrar el proyecto a una arquitectura en tiempo real. Esto abriría la puerta a aplicaciones como la correlación simultánea a la recepción de las señales, lo que daría cabida a aplicaciones como la detección temprana de problemas de configuración.

Test unitarios Aunque en este TFM se ha incorporado una infraestructura básica de pruebas, a largo plazo sería recomendable incorporar tests unitarios a nivel local en cada uno de los sub-módulos del proyecto para facilitar el desarrollo (Sección 3.2.2).

Eliminación de Hadoop A futuro podría ser interesante elmininar las dependencias con Hadoop, ya que como se vio en la Sección 3.2.3 actualmente se están manteniendo dos arquitecturas, y centrar el desarrollo en Spark simplificaría la gestión de la base de código. Esto únicamente requeriría migrar la funcionalidad del modo "tubería" (Sección 3.2.3).

Organización del código En la Sección 3.2.3 se propuso la reorganización del código siguiendo una aproximación orientada a objetos. Dado que se dispone de un prototipo funcional, esto simplificaría las interfaces internas del sistema.

Formato de salida Como se indicó en la Sección 4.7, se propone como línea futura el cambio en el formato de salida de los ficheros del actual (tipo texto) a un formato binario para mejorar el rendimiento. Se evitaría la transformación a texto de los resultados, reduciendo así tiempos, y se reduciría considerablemente el tamaño de los archivos de salida, reduciendo así también tiempos de transferencia.

Escalabilidad En la Sección 5.6.4 se describieron los ajustes mínimos necesarios para poder sacar partido al modo de funcionamiento que, aunque más lento, garantiza la escalabilidad horizontal al incrementar el número de estaciones. Posiblemente antes debería dedicarse tiempo también a mejorar el uso de memoria en los nodos ejecutores, para conseguir un uso más eficiente de los recursos.

Optimizaciones En la Sección 6.5 se puso de manifiesto la ausencia de optimizaciones específicas como por ejemplo la computación de sub-ventanas de acumulación para aumentar el grado de paralelización de la etapa de reducción del procesado, lo que aumentaría la escalabilidad del procesado en casos con pocos canales y pocas ventanas de acumulación.

Bibliografía

[1] Vázquez, A. J.; Pankratius, V.; Elosegui, P. *CorrelX User and Developer Guide 1.0*. MIT Haystack Observatory, 2017.

[2] UNED. *Guía de la asignatura del Trabajo de Fin de Máster en Ingeniería y Ciencia de Datos*. Diponible en http://portal.uned.es/portal/page?_pageid=93,69878406&_dad=portal&_schema=PORTAL&idAsignatura=31110075. Fecha del último acceso: 21 de febrero de 2021.

[3] Solus Project. *Ubuntu Budgie*. Disponible en https://ubuntubudgie.org/. Fecha del último acceso: 21 de febrero de 2021.

[4] Git. *git*. Disponible en https://git-scm.com/. Fecha del último acceso: 21 de febrero de 2021.

[5] Brachet, P. *TexMaker*. Disponible en https://www.xm1math.net/texmaker/. Fecha del último acceso: 21 de febrero de 2021.

[6] Python Software Foundation. *Python*. Disponible en https://www.python.org/downloads/. Fecha del último acceso: 21 de febrero de 2021.

[7] Apache. *Spark*. Disponible en https://spark.apache.org/. Fecha del último acceso: 21 de febrero de 2021.

[8] GNU. *GNU Image Manipulation Program (GIMP)*. Disponible en https://www.gimp.org/. Fecha del último acceso: 21 de febrero de 2021.

[9] Dia. *Diagram Editor*. Disponible en http://dia-installer.de/. Fecha del último acceso: 21 de febrero de 2021.

[10] Atlassian. *Jira: software de desarrollo de proyectos e incidencias*. Disponible en https://www.atlassian.com/es/software/jira. Fecha del último acceso: 21 de febrero de 2021.

[11] GitHub. *GitHub*. Disponible en https://github.com/. Fecha del último acceso: 21 de febrero de 2021.

[12] Amazon Web Services (AWS). *Elastic MapReduce (EMR)*. Disponible en https://aws.amazon.com/es/emr. Fecha del último acceso: 21 de febrero de 2021.

[13] Amazon Web Services (AWS). *AWS Pricing Calculator*. Disponible en https://calculator.aws/#/createCalculator/EMR. Fecha del último acceso: 31 de marzo de 2021.

[14] Jobted.es. *Sueldo del Ingeniero de Telecomunicaciones en España*. Disponible en https://www.jobted.es/salario/ingeniero-telecomunicaciones. Fecha del último acceso: 31 de marzo de 2021.

[15] Soffel, M.; Ruder, H.; Schneider, M.; Campbell, J.; Schuh, H. *Relativistic Effects in Geodetic VLBI Mesurements*. Relativity in Celestial Mechanics and Astrometry, 277-282. International Astronomical Union, 1986.

[16] NASA Space Geodesy Project. *Calc/Solve Software*. Disponible en https://space-geodesy.nasa.gov/techniques/tools/calc_solve/calc_solve.html. Fecha del último acceso: 21 de febrero de 2021.

[17] Moran, J.M. y Dhawan, V. *An Introduction to Calibration Techniques for VLBI, in Very Long Baseline Interferometry and the VLBA*. Zensus, J.A., Diamond, P.J., and Napier, P.J., Eds., Astron. Soc. Pacific Conf. Ser., 82, 161–188, 1995.

[18] Andrew Zisserman (rofessor of Computer Vision Engineering at Oxford). *Lecture 2: 2D Fourier transforms and applications*. Disponible en https://www.robots.ox.ac.uk/~az/lectures/ia/lect2.pdf. Fecha del último acceso: 23 de febrero de 2021.

[19] Clery, D. *For the first time, you can see what a black hole looks like*. Disponible en https://www.sciencemag.org/news/2019/04/black-hole. Fecha del último acceso: 21 de febrero de 2021.

[20] Event Horizon Telescope. *Event Horizon Telescope*. Disponible en https://eventhorizontelescope.org/. Fecha del último acceso: 21 de febrero de 2021.

[21] Event Horizon Telescope. *Event Horizon Telescope Array*. Disponible en https://eventhorizontelescope.org/array. Fecha del último acceso: 21 de febrero de 2021.

[22] Zhang, Z.; Wang, G.; Xu, D.; Song, S. *Prospective study of high geodetic resolution to future VGOS reference point determination*. Geodesy and Geodynamics; Vol. 10, I. 2. 2019.

[23] MIT News. *Celebrating 50 years of transatlantic geodetic radio science*. Disponible en https://news.mit.edu/2018/celebrating-50-years-transatlantic-geodetic-radio-science-0405. Fecha del último acceso: 21 de febrero de 2021.

[24] NASA Earth Sciences. *VLBI Global Observing System (VGOS)*. Disponible en https://earth.gsfc.nasa.gov/geo/instruments/vlbi-global-observing-system-vgos. Fecha del último acceso: 21 de febrero de 2021.

[25] Deller, A. T.; Tingay, S. J.; Bailes, M.; West, C. *DiFX: A software correlator for very long baseline interferometry using multi-processor computing environments*. arXiv:astro-ph/0702141v1, 2007.

[26] MIT Haystack. *Haystack VLBI Correlator*. Disponible en https://www.haystack.mit.edu/about/haystack-telescopes-and-facilities/haystack-vlbi-correlator/. Fecha del último acceso: 21 de febrero de 2021.

[27] Beowulf.org. *Frequently Asked Questions*. Disponible en https://beowulf.org/overview/faq.html. Fecha del último acceso: 21 de febrero de 2021.

[28] Max Planck Institute for Radio Astronomy. *DiFX Correlator*. Disponible en https://www.mpifr-bonn.mpg.de/771785/DiFX-CORRELATOR. Fecha del último acceso: 21 de febrero de 2021.

[29] ATNF VLBI Wiki. *DiFX Users*. Disponible en https://www.atnf.csiro.au/vlbi/dokuwiki/doku.php/difx/users. Fecha del último acceso: 21 de febrero de 2021.

[30] Keimpema, A.; Kettenis, M. M.; Pogrebenko, S. V.; Campbell, R. M.; Cimó, G.; Duev, D. A.; Eldering, B.; Kruithof, N.; van Langevelde, H. J.; Marchal, D.; Molera, G.; Ozdemir, H.; Paragi, Z.; Pidopryhora, Y.; Szomorou, A.; Yang, J. *The SFXC software correlator for Very Long Baseline Interferometry: Algorithms and Implementation*. arXiv:astro-ph.IM, 2015.

[31] Surkis, I; Ken, V; Kurdubova, Y; Melnikov, A; Mishin, V; Mishina, N; Shantyr, V; Zhuravov, D; Zimovsky, V. *IAA Correlator Center Biennial Report 2015+2016*. IAA Correlator Center, 2016.

[32] Clark, M. A.; La Plante, P. C.; L. J. Greenhill; MacMahon, D.; Barsdell, B. *A GPU based FX correlator for radio astronomy*. Disponible en https://github.com/GPU-correlators/xGPU. Fecha del último acceso: 21 de febrero de 2021.

[33] Open MPI: Open Source High Performance Computing. *A High Performance Message Passing Library*. Disponible en https://www.open-mpi.org/. Fecha del último acceso: 21 de febrero de 2021.

[34] OpenMP: OpenMP. *A High Performance Message Passing Library*. Disponible en https://www.openmp.org/. Fecha del último acceso: 24 de febrero de 2021.

[35] MIT Haystack. *CorrelX: A Cloud-Based Software Correlator for Very Long Baseline Interferometry (VLBI)*. Disponible en https://github.com/MITHaystack/CorrelX. Fecha del último acceso: 21 de febrero de 2021.

[36] Gill, A.; Blackburn, L.; Roshanineshat, A.; Chan, C.; Doeleman, S.; Johnson, M.; Raymond, W.; Weintroub, J. *Prospects for Wideband VLBI Correlation in the Cloud*. Publications of the Astronomical Society of the Pacific, 131:124501 (13pp), 2019.

[37] Google Cloud. *Servicios de Cloud Computing*. Disponible en https://cloud.google.com/. Fecha del último acceso: 22 de febrero de 2021.

[38] Apache. *Hadoop*. Disponible en https://hadoop.apache.org/. Fecha del último acceso: 21 de febrero de 2021.

[39] Databricks. *Apache Spark and Hadoop: Working Together*. Disponible en https://databricks.com/blog/2014/01/21/spark-and-hadoop.html. Fecha del último acceso: 22 de febrero de 2021.

[40] Apache Spark. *Spark wins Daytona Gray Sort 100TB Benchmark*. Disponible en https://spark.apache.org/news/spark-wins-daytona-gray-sort-100tb-benchmark.html. Fecha del último acceso: 22 de febrero de 2021.

[41] Nan, S; Su, Z. *Spark vs. Hadoop MapReduce*. Disponible en https://www.cs.rochester.edu/courses/261/spring2017/termpaper/07/Paper.pdf. Fecha del último acceso: 21 de febrero de 2021.

[42] Google Trends *Apache Spark ,Apache Flink, Apache Hadoop, OpenMP, OpenMPI*. Disponible en https://trends.google.es/trends/explore?date=all&q=Apache%20Spark,Apache%20Flink,Apache%20Hadoop,OpenMP,OpenMPI. Fecha del último acceso: 24 de febrero de 2021.

[43] Google Trends *Python, Scala, C++*. Disponible en https://trends.google.es/trends/explore?date=all&q=%2Fm%2F05z1_,%2Fm%2F091hdj,%2Fm%2F0jgqg. Fecha del último acceso: 24 de febrero de 2021.

[44] IVS Annual Report 2009 *VLBI Data Interchange Format (VDIF) Specification, Release 1.0*. 26 de junio de 2009.

[45] Pankratius, V.; Vázquez, A. J.; Elosegui, P. *CorrelX: A Cloud-Based VLBI Correlator* Disponible en https://www.haystack.mit.edu/wp-content/uploads/2020/07/conf_NEROC2016_vasquez_web.pdf. Fecha del último acceso: 29 de mayo de 2021.

[46] Apache. *Spark Streaming Programming Guide*. Disponible en https://spark.apache.org/docs/latest/streaming-programming-guide.html. Fecha del último acceso: 21 de febrero de 2021.

[47] Apache. *Kafka*. Disponible en https://kafka.apache.org/. Fecha del último acceso: 8 de abril de 2021.

[48] Koyama, Y; Kondo, T; Kimura, M; Hirabaru, M; Takeuchi. *Real-time High Volume Data Transfer and Processing for e-VLBI*. Advances in Geoscientes, pp. 81-90, 2007.

[49] Akidau, T.; Chernyak, S.; Lax, Reuven. *Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing*. O'Reilly Media, 2018.

[50] Martin, R.C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[51] Buxton, J.N. y Randell, B. *Software Engineering Techniques*, April 1970, p. 16. Informe conferencia patrocinada por el Comité de Ciencia de la OTAN, Roma, Italia, 27–31 de octubre de 1969.

[52] Git. *gitk*. Disponible en https://git-scm.com/docs/gitk/. Fecha del último acceso: 18 de abril de 2021.

[53] Project Jupyter. *Jupyter Notebooks*. Disponible en https://jupyter.org/. Fecha del último acceso: 18 de abril de 2021.

[54] Amazon Web Services. *Amazon EMR - Big Data Platform*. Disponible en https://aws.amazon.com/emr/. Fecha del último acceso: 11 de mayo de 2021.

[55] Microsoft Docs. *Azure HDInsight documentation*. Disponible en https://docs.microsoft.com/en-us/azure/hdinsight/. Fecha del último acceso: 11 de mayo de 2021.

[56] Google Cloud. *Dataproc*. Disponible en https://cloud.google.com/dataproc. Fecha del último acceso: 11 de mayo de 2021.

[57] DZone. *Who Is Leading Among The Big Three?: AWS vs. Azure vs. Google Cloud Market Comparison*. Disponible en https://dzone.com/articles/who-is-leading-among-the-big-three-aws-vs-azure-vs. Última actualización en 2020. Fecha del último acceso: 11 de mayo de 2021.

[58] Jay Chapel (medium.com). *AWS vs Azure vs Google Cloud Market Share 2020: What the Latest Data Shows*. Disponible en https://jaychapel.medium.com/aws-vs-azure-vs-google-cloud-market-share-2020-what-the-latest-data-shows-dd700fd75c2d. Última actualización en noviembre de 2020. Fecha del último acceso: 11 de mayo de 2021.

[59] Amazon EMR. *Using S3 Select with Spark to Improve Query Performance*. Disponible en https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-s3select.html. Fecha del último acceso: 11 de mayo de 2021.

[60] Microsoft Docs. *Azure Blob Storage: Azure Databricks*. Disponible en https://docs.microsoft.com/es-es/azure/databricks/data/data-sources/azure/azure-storage. Fecha del último acceso: 11 de mayo de 2021.

[61] Google. *Usa el conector de Cloud Storage con Apache Spark*. Disponible en https://cloud.google.com/dataproc/docs/tutorials/gcs-connector-spark-tutorial?hl=es-419. Fecha del último acceso: 11 de mayo de 2021.

[62] Google Cloud. *Tipos de máquinas*. Disponible en https://cloud.google.com/compute/docs/machine-types#n1_high-memory_machine_types. Fecha del último acceso: 16 de mayo de 2021.

[63] Amazon Web Services. *Tipos de instancias de Amazon EC2*. Disponible en https://aws.amazon.com/es/ec2/instance-types/. Fecha del último acceso: 14 de mayo de 2021.

[64] MIT Haystack Observatory. *Haystack Observatory Postprocessing System (HOPS)*. https://www.haystack.mit.edu/haystack-observatory-postprocessing-system-hops/. Fecha del último acceso: 30 de mayo de 2021.

[65] Leonid Petrov. *Accumulation period (integration time) in DiFX.*. Disponible en http://astrogeo.org/petrov/discussion/ap_len.html. Fecha del último acceso: 14 de mayo de 2021.

[66] Geekbench Browser. *GCP: n1-highmem-16, 16vCPU / 102GB Google Google Compute Engine*. Disponible en https://browser.geekbench.com/v3/cpu/8262786. Fecha del último acceso: 16 de mayo de 2021.

[67] Geekbench Browser. *EC2: m4.large,2 vCPUs / 8 GB Xen HVM domU*. Disponible en https://browser.geekbench.com/v3/cpu/8265501. Fecha del último acceso: 16 de mayo de 2021.