UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Procesamento analítico en liña de datos medioambientais en arquitecturas Big Data

Autor:
Diego Ferrón Lea

Directores:
José Ramón Ríos Viqueira
Anselmo Tomás Fernández Pena

Grao en Enxeñaría Informática

Xullo 2016

Traballo de Fin de Grao presentado na Escola Técnica Superior de Enxeñaría da Universidade de Santiago de Compostela para a obtención do Grao en Enxeñaría Informática

D. José Ramón Ríos Viqueira, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela, e **D. Anselmo Tomás Fernández Pena**, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela,

INFORMAN:

Que a presente memoria, titulada *Procesamento analítico en liña de datos medioambientais en arquitecturas Big Data*, presentada por **D. Diego Ferrón Lea** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxeñaría Informática, realizouse baixo a nosa dirección no Departamento de Electrónica e Computación da Universidade de Santiago de Compostela.

E para que así conste a efectos oportunos, expiden o presente informe en Santiago de Compostela a 1 de Xullo de 2016:

O director,	O codirector,	O alumno,
José R. Ríos Viqueira	A. Tomás Fernández Pena	Diego Ferrón Lea

Contidos

Índice de figuras

Índice de gráficos

Índice de táboas

Índice de fórmulas

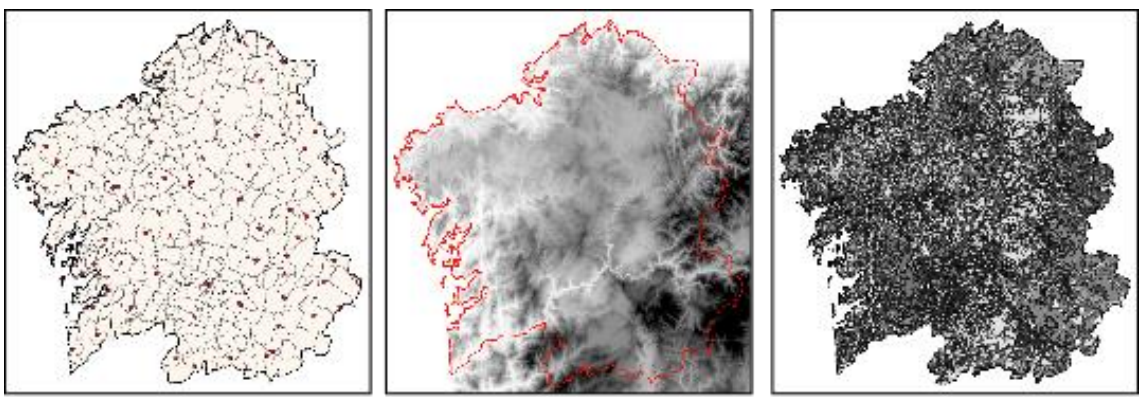
INTRODUCIÓN

A evolución das solucións Big Data nos últimos anos vén de ser exponencial, motivando un novo enfoque respecto do tratamento de grandes cantidades de información que non poden ser procesadas ou analizadas con ferramentas tradicionais. Ditas solucións constitúen o fundamento dos modernos sistemas de apoio á toma de decisións (DSS) en diversos dominios de aplicación, como por exemplo a análise do mercado levada a cabo por unha aeroliña [25] ou a monitorización e xestión do sistema de saúde dun territorio [8].

Entre eses dominios podemos destacar a análise de datos medioambientais, xerados principalmente por sensores ou por software de modelado de datos, como poden ser as medicións físicas realizadas por estacións meteorolóxicas ou as características medioambientais dun determinado espazo xeográfico. Neste eido podemos identificar dous grandes tipos de datos: dunha banda existen mostraxes de propiedades específicas sobre espazos multidimensionais, e doutra banda existen entidades que representan propiedades específicas de obxectos que poden evolucionar ó longo do tempo.

As tecnoloxías dispoñibles na actualidade para a análise destes dous grandes tipos de datos son os xestores de bases de datos relacionais (RDBMS) e un subgrupo de solucións Big Data (por exemplo, bases de datos NoSQL e ecosistemas tipo Hadoop) para o caso das entidades, e solucións *ad-hoc* sobre arquivos de arrays xunto con *scientific array data managers* (por exemplo, SciDB [18, 22] ou Rasdaman [5, 21]) para o caso das mostraxes. Sen embargo, non existe no mercado ningunha tecnoloxía que posibilite a xestión integrada e uniforme de ambos tipos de datos.

A xustificación de dita necesidade manifféstase en casos de uso que requiran a integración de fontes de datos diversas [27, 26]: un exemplo sería a avaliación do risco de incendios dunha determinada zona xeográfica en base a un conxunto de datos entre os que poden considerarse mostraxes temporais obtidas de observacións meteorolóxicas, mostraxes espaciais da elevación do terreo e entidades con datos sobre especies presentes e modelos de combustible.



(a) Estacións meteorolóxicas. (b) Elevación sobre o nivel do mar. (c) Modelo de combustible.

Figura 1.- Fontes de datos para un risco de incendio.

No eido destas preocupacións xorde esta proposta de traballo de fin de grao para a cal nos propoñemos como obxectivo principal a elaboración unha solución baseada en tecnoloxías *Big Data* para a análise integrada de datos de entidades e mostraxes no ámbito da avaliación do risco de incendio forestal. Máis especificamente, definimos os seguintes obxectivos concretos:

- Realizar un estudo do estado da arte relacionado con tecnoloxías Big Data aplicables neste ámbito.
- Deseñar e implementar estruturas de datos, tanto en memoria como en disco, para os dous tipos de datos previamente mencionados: mostraxes e entidades.
- Deseñar e implementar operadores de análise integrada para ambos tipos de datos.
- Levar a cabo unha análise de rendemento sobre aspectos como os tempos de execución ou o escalamento de diferentes prototipos, identificando operacións ineficientes e propoñendo solucións ós problemas atopados.
- Deseñar e implementar un módulo software que permita exportar os datos procesados en algún dos estándares existentes para poder visualizalos.

Os resultados deste traballo foron enviados como candidatura para presentación nun congreso de nivel internacional [10] e noutro de nivel nacional [11], onde foron avaliados por expertos na materia. En ámbolos casos foron aceptados.

A estrutura da que se compón esta memoria pretende describir o proceso de obtención de ditos obxectivos paso por paso, de xeito que imos a describir os seguintes aspectos:

- En primeiro lugar, cabe contextualizar a nivel técnico cal é o problema a afrontar, no marco do desenvolvemento tecnolóxico existente no momento de realización do traballo, pasando polas diferentes necesidades que se desprenden do caso de uso proposto.
- En segundo lugar, cabe determinar con claridade aqueles aspectos de xestión que son ineludiblemente necesarios para a consecución do proxecto. Isto comprende desde a definición do alcance do mesmo e dunha metodoloxía a empregar, ata a planificación temporal do mesmo así como a estimación económica. Cómpre non esquecer tampouco unha correcta xestión dos riscos asociados a el.
- Seguindo a estrutura habitual dun proxecto de software, definiremos a continuación cales son os requisitos, tanto funcionais como non funcionais, asociados ó software a desenvolver, así como os principais casos de uso que ilustran o uso de dito software por parte dun usuario.
- Na seguinte parte describiremos o proceso de deseño global levado a cabo para especificar a composición do sistema e a interacción entre os seus diferentes elementos, tanto a nivel estático como a nivel dinámico.
- Logo cómpre explicar a evolución das diferentes implementacións que temos levado a cabo xunto coas probas de rendemento que nos permitirán xustificar cales son as vantaxes e inconvenientes de cada unha delas, e tamén os problemas que nos fomos atopando.
- Por último, redactaremos unhas conclusións que engloben as leccións aprendidas e os logros acadados.

GLOSARIO

- **Almacén de datos (*data warehouse*).-** Repositorio de datos integrados a partir de fontes diversas, tipicamente centralizado. Pode almacenar tanto datos históricos como datos actualizados, proporcionar información organizacional.
- **Arquivos de arrays.-** Trátase de arquivos con formatos específicos para Sistemas de Información Xeográfica que conteñen vectores de valores representando propiedades espazo-temporais específicas definidas na metainformación ou coberturas xeográficas. Por exemplo, os formatos GeoTIFF e NetCDF.
- **Big Data.-** Con este termo faise referencia a grandes conxuntos de datos que cumpres a regra das tres "v": volume, variedade e velocidade (no seu tratamento). En xeral enmárcanse en sistemas de tipo distribuído, escalable e tolerantes a fallos.
- **Cubo de datos.-** Estruturas n-dimensionais que permiten a representación de conxuntos de funcións (*mappings*), cuxos dominios están definidos por produtos cartesianos de dimensións e cuxos rangos están definidos por subconxuntos de datos do sistema.
- **DataFrame.-** Colección distribuída de datos que se organizan en columnas, conceptualmente equivalentes ás táboas das bases de datos relacionais ou ós dataframes de R. Forma parte de Spark SQL.
- **Dimensión.-** Subconxunto finito de elementos dun tipo de dato non xeométrico.
- **Driver (Spark).-** Neste contexto este termo fai referencia ó proceso principal que leva a cabo o control da execución dun programa en Spark, sincronizando os executores.
- **DSS (*Decision Support System*).-** Sistema computacional de información que serve de soporte ó proceso de toma de decisións nunha organización.
- **Entidades.-** Forma de representación de datos consistente en obxectos con propiedades específicas que poden (ou non) evolucionar ó longo do tempo.
- **ETL (*Extract, Transform, Load*).-** Proceso que ten lugar en bases de datos e almacéns de datos consistente na extracción dos datos das súas fontes, transformación nun formato definido e carga no seu destino final (normalmente o almacén).
- **Executor (Spark).-** Neste contexto este termo fai referencia a cada proceso distribuído que se executa nun nodo do clúster para procesar e almacenar datos dun programa en Spark. En xeral, cada nodo físico pode albergar varios executores.
- **GeoSpark.-** *Framework* de computación en clúster para o procesamento a gran escala de datos de tipo espacial.
- **GeoTIFF.-** Estándar de metadatos, de dominio público, para a inclusión de información xeograficamente referenciada en arquivos TIFF.
- **GIS (*Geographical Information System*).-** Sistema deseñado para capturar, almacenar, manipular, analizar e xestionar tipos de datos espaciais ou xeográficos.
- **Hadoop.-** *Framework* que permite o procesamento distribuído de grandes conxuntos de datos sobre clústeres de computación facendo uso de modelos de programación simples.
- **LiDAR (*Light Detection And Ranging*).-** Tecnoloxía de exploración que mide a distancia iluminando o obxectivo cunha luz láser, tipicamente usada para crear mapas.

- **Mostraxes (*Samplings*).-** Secuencia ordenada de elementos dun tipo de dato espacial ou temporal, desde un elemento inicial ata un elemento final.
- **NetCDF.-** Aínda que o termo fai referencia a un conxunto de librerías de software desenvolvidas pola UCAR (*University Corporation for Atmospheric Research*), neste documento utilizámolo como sinónimo do formato de arquivo derivado para o almacenamento de arrays de datos científicos.
- **OLAP (*Online Analytical Processing*).-** Esquema de procesamento desenvolvido como resposta a consultas multidimensionais de tipo analítico, frecuentemente utilizadas en intelixencia de negocio.
- **Parquet.-** Formato de almacenamento columnar dispoñible para calqueira proxecto do ecosistema Hadoop.
- **PostGIS.-** *Software open source* que engade soporte para obxectos de tipo xeográfico á base de datos PostgreSQL.
- **RDD.-** Colección de obxectos dunha linguaxe de programación en particular que se particiona sobre o clúster onde se executa. Forma parte da API de Spark.
- **SequenceFile.-** Ficheiro binario conformado por pares clave-valor que se utiliza en Hadoop como formato de entrada/saída para operacións MapReduce.
- **Shapefile.-** Formato de arquivo que serve para almacenar modelos de datos xeorrelacionais que representan entidades xeográficas.
- **Solucións espaciais clásicas.-** Con este termo facemos referencia ás extensións para datos espazo-temporais de bases de datos relacionais, como por exemplo PostGIS.
- **Spatial Hadoop.-** Extensión *open source* para MapReduce que pemite manexar grandes cantidades de datos xeoespaciais en Hadoop.
- **Spark.-** Sistema de computación distribuída de propósito xeral deseñado para o procesamento de grandes cantidades de datos.
- **Spatial Big Data.-** Rama do mundo Big Data adicada especificamente ó manexo de datos xeoespaciais.

PARTICIPANTES

- **Diego Ferrón Lea**, en calidade de autor do proxecto, desempeñando os seguintes roles: analista, deseñador, programador, investigador e xestor.
- **José Ramón Ríos Viqueira**, en calidade de titor do proxecto: participará na toma de decisións e no apoio académico, desempeñando o rol de director/xefe do proxecto.
- **Anselmo Tomás Fernández Pena**, en calidade de cotitor do proxecto: participará na toma de decisións e no apoio académico, desempeñando fundamentalmente un rol de analista.

CONTEXTO DO PROBLEMA

A cantidade e precisión dos datos de tipo medioambiental xerados é cada vez maior, debido aos grandes avances na fabricación de dispositivos de sensorización. Sen embargo, o procesamento de datos medioambientais en entornos científicos realízase mediante implementacións *ad-hoc* sobre formatos de arquivo estandarizados, ás veces utilizando librerías de xeo-procesamento dispoñibles na área dos Sistemas de Información Xeográfica. Neste eido, a nosa proposta xorde como resposta á existencia de dous grandes problemas:

a) A necesidade de extraer, transformar e cargar (ETL) nun sistema de almacenamento uniforme todos eses datos que proveñen de fontes diversas e que seguen modelos diversos. Estas fontes poden comprender desde módulos que engaden soporte de obxectos xeográficos en bases de datos relacionais, como por exemplo o módulo PostGIS [19] para PostgresSQL, ata formatos de arquivo do tipo GeoTiff, NetCDF ou Shapefile, amplamente utilizados neste tipo de sistemas.
b) O posterior procesamento analítico en liña (OLAP) desas grandes cantidades de datos importadas. Como mencionamos na introdución, eses datos inclúen tanto entidades espaciais como por exemplo estacións meteorolóxicas, ata grandes arrays de valores numéricos resultantes de procesos de mostraxe temporais (coma os xerados polas estacións) ou espaciais (coma os datos de elevación do terreo xerados por Lidar).

Posto que esta variedade e cantidade de datos se pode enmarcar no rango do que se coñece como Big Data, cabe analizar brevemente o grao de desenvolvemento deste tipo de sistemas baixo o paraugas de ditas tecnoloxías. Probablemente os dous *frameworks* de computación distribuída máis coñecidos na actualidade son Apache Hadoop e Apache Spark. Ambos xogan un papel fundamental á hora de proporcionar plataformas de código aberto sobre as cales desenvolver aplicacións de procesamento en paralelo de grandes cantidades de datos sobre clústeres de computadores de baixo custo. Proporcionan, ademais, funcionalidades adicionais ligadas ós ecosistemas nos que se desenvolven, como por exemplo o soporte de sistemas de ficheiros distribuídos.

Tal e como se explica na páxina web do proxecto, Apache Hadoop [3, 26] é un *framework* que permite o procesamento distribuído de grandes conxuntos de datos sobre clústeres de computación facendo uso de modelos de programación simples. Está deseñado para escalar horizontalmente desde un só servidor ata miles de máquinas, ofrecendo cada unha delas a súa propia capacidade de computación e almacenamento. A librería está deseñada para detectar e xestionar erros a nivel de aplicación, independizando dito aspecto da alta dispoñibilidade que poida proporcionar o hardware.

O proxecto inclúe unha serie de módulos que se complementan mutuamente:

- *Hadoop Common* incorpora as utilidades comúns que permiten a integración do resto de módulos.

- *Hadoop Distributed File System (HDFS)* é un sistema de arquivos distribuído que proporciona alto rendemento no acceso aos datos da aplicación así como replicación dos mesmos.
- *Hadoop YARN* é un *framework* para a planificación de traballos e a xestións dos recursos dun clúster.
- *Hadoop MapReduce* é un sistema, basado en YARN, para o procesamento paralelo de grandes conxuntos de datos.

Apache Spark [4, 16] é un sistema de computación distribuída de propósito xeral deseñado para o procesamento de grandes cantidades de datos. Proporciona APIs para varias linguaxes de programación (Scala, Java e Python) e un motor optimizado. A estrutura primitiva do sistema é o *Resilient Distributed Dataset* (RDD), que non é máis ca unha colección de obxectos dunha linguaxe de programación en particular que se particiona sobre o clúster onde se executa. O *framework* tamén proporciona un conxunto de operacións que permiten a avaliación de funcións sobre os RDD en paralelo, sobre os nodos dispoñibles do clúster. Cabe citar aquí dúas características importantes dos RDDs:

a) Son tolerantes a erros, isto é, o sistema pode recuperar datos perdidos en calquera momento.
b) Son avaliados de forma tardía (*lazy evaluation*), isto é, as operacións sobre os RDDs só se levan a cabo cando os datos resultantes desas operacións se requiren noutro proceso.

Spark SQL extende Apache Spark engadindo características de procesamento relacional eficiente. En particular, proporciona aos usuarios unha nova estrutura de datos denominada *DataFrame* coa cal as operacións relacionais se poden executar de forma eficiente apoiándose nun optimizador extensible propio. O DataFrame, ó igual ca un RDD, é unha colección distribuída de datos que se organizan en columnas, conceptualmente equivalentes ás táboas das bases de datos relacionais ou ós dataframes de R. Sen embargo, diferénciase dun RDD en que o DataFrame mantén o esquema dos datos e almacénase por columnas, sendo máis compacto e eficiente. Os DataFrames poden construírse directamente desde fontes de datos externas ou xerándoos desde coleccións de obxectos nun RDD, e tamén permiten avaliación tardía das operacións relacionais. Spark SQL proporciona, en definitiva, un *framework* que integra eficientemente o procesamento relacional declarativo co procesamento procedimental expresado na linguaxe correspondente. Adicionalmente, desde as versións máis recentes de Spark (v1.6) incorpórase unha nova estrutura, denominada *Dataset*, que pretende combinar a eficiencia dos DataFrames coa flexibilidade dos RDDs.

As vantaxes destes *frameworks* motivaron a recente aparición de solucións tipo *SpatialHadoop* [1, 9, 24] ou *GeoSpark* [12, 29] que pretenden aplicar ditas tecnoloxías ó campo específico dos datos espazo-temporais e que poden servir de inspiración para o noso sistema. Sen embargo, non existe ningunha implementación eficiente de ETL e OLAP para datos medioambientais de todo tipo, deseñada para integrar entidades espaciais e grandes arrays resultantes de mostraxes espazo-temporais. A solución que propoñemos baséase na utilización dalgún dos *frameworks* Big Data mencionados anteriormente, que dispoña dun sistema de arquivos distribuído no cal almacenar os

datos que se vaian a procesar posteriormente. Neste senso, esbózanse dúas cuestións principais:

1) Definir, deseñar e implementar as estruturas de datos que sexan necesarias para almacenar os datos do problema no sistema de arquivos distribuído importándoos desde as diversas fontes de datos das que dispoñamos, tendo en conta a dualidade entre entidades e mostraxes. Haberá que desenvolver tanto estruturas de almacenamento en disco como estruturas de almacenamento en memoria, para garantir unha eficiencia adecuada na xestión das cantidades de datos deste calibre, tomando man das técnicas que sexan necesarias. Isto conformará un *data warehouse* distribuído sobre o que se realizará o procesamento.
2) Definir, deseñar e implementar os operadores que sexan necesarios tanto para a importación dos datos coma para o seu procesamento. O proceso de importación ou ETL consistirá en extraer, transformar e cargar os datos desde as fontes iniciais ata o sistema de arquivos distribuído, namentres as operacións de procesamento, que dependerán do caso de uso ó que se destine o sistema, beneficiaranse do paralelismo computacional que nos proporciona o *framework* que escollamos.

En base á problemática descrita, un exemplo de caso de uso, xa mencionado na introdución, que require unha solución como a proposta é o cálculo do risco de incendio para un determinado territorio. A análise de factores que inflúen na aparición de incendios forestais benefíciase da dispoñibilidade de diversos conxuntos de datos, entre os que cabe mencionar datos meteorolóxicos xerados por redes de estacións públicas e privadas, datos de elevación xerados por sensores Lidar ou datos de cobertura vexetal do terreo. Cabe aclarar que a intención desta proposta de caso de uso é simplemente a ilustración do procesamento integrado de datos medioambientais de tipo array e de tipo relacional, sen pretender ser unha contribución ó dominio relevante de aplicación.

Para facerse unha idea da cantidade de datos xerados, MeteoGalicia dispón dunha rede dunhas 80 estacións meteorolóxicas que xeran datos cunha frecuencia temporal de 10 minutos, mentres que a resolución espacial máxima dos datos de elevación proporcionados polo Instituto Xeográfico Nacional é de 5 metros, co que se necesitan uns 2.500 millóns de valores de elevación para cubrir o territorio de Galicia.

As estruturas de datos que compoñen o modelo de almacenamento de datos espazo-temporais a implementar son as mesmas que se propoñen en traballos previos dos autores S. Villarroya e J.R. Viqueira [27, 28]:

- **Dimensións**.- Trátase dun subconxunto finito de elementos dun tipo de dato (non xeométrico). Casos especiais de dimensións son as mostraxes 1D (temporais e espaciais) e 2D (espaciais). As mostraxes conteñen unha secuencia ordenada de elementos dun tipo de dato (temporal ou espacial), desde un elemento inicial a un elemento final. No caso das mostraxes 2D é necesario utilizar un ordenamento espacial definido pola curva de recheo do espazo. Un exemplo de mostraxe 1D temporal sería aquel que se xera sobre o tipo de datos *TimeInstant*(86400) a partir das datas inicial e final dunha serie de datos agregados de temperatura xerados cunha frecuencia diaria por unha estación

meteorolóxica. De xeito similar, un exemplo de mostraxe 2D espacial sería aquela que se pode definir sobre o tipo de datos *Point2D* para datos de elevación do terreo desde un punto inicial do mapa ata un punto final.

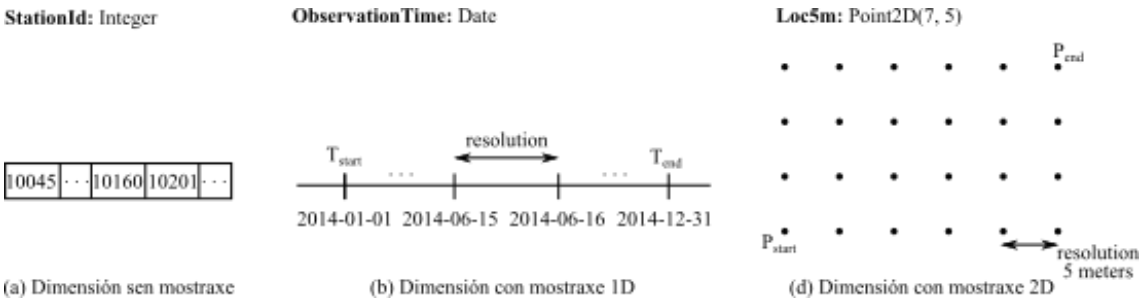


Figura 2.- Exemplos dos diferentes tipos de dimensións.

- **Cubos de datos**.- Trátase de estruturas n-dimensionais que permiten a representación de conxuntos de funcións, cuxos dominios están definidos por produtos cartesianos de dimensións e cuxos rangos están definidos por subconxuntos de datos do sistema. Por exemplo, o resultado da interpolación espacial de datos de temperatura, humidade e vento xerados polas estacións represéntase mediante un cubo de datos con tres funcións: temperatura, humidade e vento, de tipo numérico de precisión fixa, definido sobre mostraxes temporais (data da observación) e espaciais (punto do territorio representado).

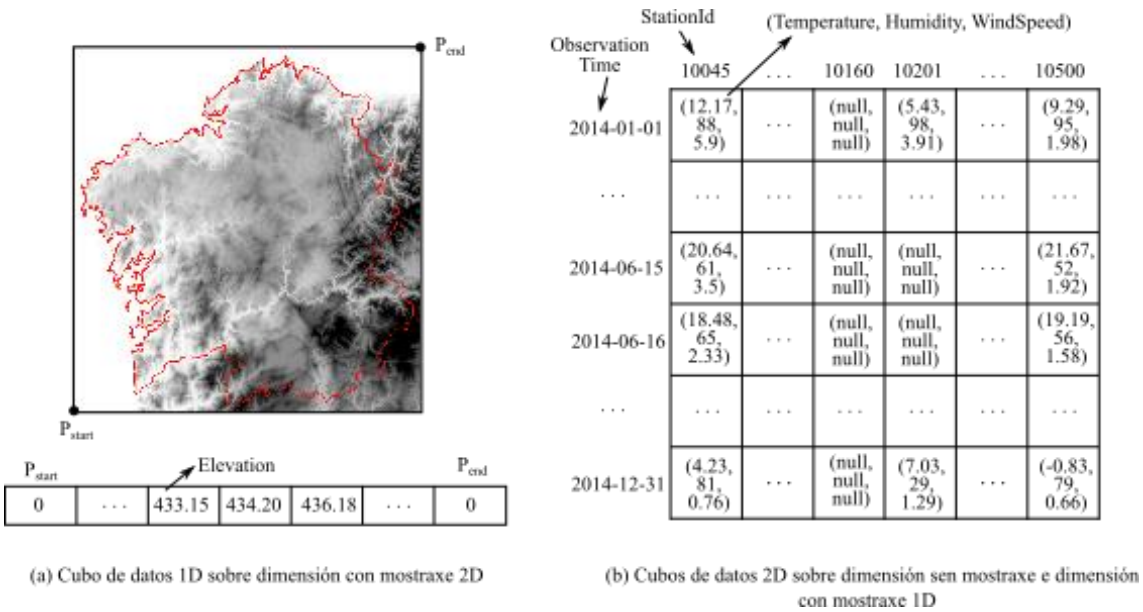


Figura 3.- Exemplos de cubos de datos.

- **Constantes**, definidas por un valor do tipo atómico T.

Desde o punto de vista das bases de datos relacionais, as dimensións e os cubos de datos permiten o modelado de entidades e relacións entre elas. Por exemplo, a dimensión *idEstación* contén o identificador de cada estación meteorolóxica e as propiedades restantes das estacións modélanse mediante o correspondente cubo de datos, que neste caso conterá as funcións *nome* e *localización*. Respecto ás mostraxes de datos temporais e espaciais (1D e 2D), podemos tomar como exemplo os datos de elevación dun arquivo GeoTiff, onde a localización (coa súa correspondente resolución) se considerará unha dimensión e os propios valores de elevación do terreo serán a función que formará parte dun cubo de datos.

Ademais dos tipos de datos convencionais que proporciona a linguaxe de programación a utilizar, será necesario definir tipos de datos específicos do tipo de sistema que imos a construír. En primeiro lugar, é importante o coñecemento da precisión e da escala dos datos numéricos para levar a cabo un almacenamento eficiente dos datos de tipo real, polo que cómpre pedir ó usuario que indique dita información e logo utilizar unha representación de precisión fixa para os datos deste tipo. Por outra parte, para representar datos que modelen xeometrías en espazos euclidianos de dúas dimensións cómpre definir un conxunto de datos primitivos, que no caso de uso do risco de incendios son os seguintes:

- Point2D(P, R): $\{(x * R, y * R) | x, y \in Integer \wedge -10^P < x, y < 10^P\} \cup \{\perp\}$
- LineString(P, R): Secuencia de Point2D(P, R).
- Polygon(P, R): Constituído por un borde, representado mediante unha *LineString*, e un conxunto de buracos representado mediante un vector de *LineString*.
- MultiPolygon(P, R): Colección de xeometrías de tipo *Polygon*.

Adicionalmente, para os datos de tipo temporal pódense engadir as seguintes representacións:

- TimeInstant(R): $\{t * R \mid t \in Integer \wedge -10^{MP} < t < 10^{MP}\} \cup \{\perp\}$
- Date: TimeInstant(86400)

FERRAMENTAS

Hardware

O hardware dispoñible a utilizar para o proxecto é o seguinte:

a) Para as probas en local, un ordenador de sobremesa coas seguintes características:
   - AMD Athlon 64 X2 Dual Core Processor 4800+ @ 2.5 GHz.
   - 4 GB de memoria RAM repartidos en 2 módulos DDR2 @ 800MHz.
   - Disco duro SATA de 240 GB.
   - Sistema operativo Ubuntu Desktop 14.04 LTS.

b) O clúster de computación distribuída que imos a utilizar está montado no CiTIUS e está formado por 5 nodos computacionales HP Proliant DL 370 G6. Cada un conta con:
   - 2 procesadores Intel Xeon E5506 @ 2.13GHz.
   - 12 GB de memoria RAM ECC repartidos en 6 módulos DDR3 @ 800MHz.
   - 2 discos duros de 2 TB montados en dos RAID0.
   - 4 portos Gigabit Ethernet.
   - Instalación do sistema CentOS 6.5 mínima.

Software

Todo o software que se vai a utilizar é software sen custo de licenza.

XESTIÓN DO PROXECTO

Neste apartado descríbense todos os aspectos relacionados coa xestión do proxecto que son necesarios para a súa correcta execución.

Enunciado do alcance do proxecto

Descrición do alcance do produto

O software a desenvolver permitirá a importación de datos tanto de mostraxes como de entidades, sobre fontes diversas, recoñecendo en concreto o tipo de arquivo GeoTiff, e permitirá tamén o procesamento en paralelo de ditos datos importados, en concreto permitirá calcular o risco de incendio a partir dos datos de partida definidos. O usuario disporá dunha interface, que pode ser un arquivo, sobre a cal poderá definir as características das fontes de datos. Por último, deberase implementar un módulo que posibilite a exportación dos valores de risco ó formato estándar ASCII para a súa visualización.

Criterios de aceptación do produto

- O usuario dispón de flexibilidade para definir as súas fontes de datos.
- O software permite importar grandes cantidades de datos (da orde de GB) sobre un sistema de almacenamento distribuído.
- O software permite procesar en paralelo (nun clúster) as operacións definidas.
- O software permite realizar probas de rendemento sobre operacións concretas.
- O software permite exportar os valores de risco a un ficheiro ASCII correctamente estandarizado para a súa visualización.

Entregables do proxecto

- Software en formato executable, na linguaxe de programación que corresponda, e coas dependencias de execución incorporadas de xeito que se poida executar directamente sobre o *framework* de computación distribuída elixido.
- Documento da memoria do proxecto, na cal se especificarán todas as características que definen o software desenvolvido así como as fases de análise, deseño, implementación e probas levadas a cabo, e a documentación asociada á xestión do proxecto.

Exclusións do proxecto

En xeral consideramos como exclusións do proxecto todo aquilo que non estea incluído na especificación de requisitos. En concreto, quedan fóra do seu alcance:

- A implementación de operacións non relacionadas co caso de uso elixido (risco de incendio).

- A inclusión de soporte para outros estándares de arquivos de datos xeográficos (por exemplo, Shapefile).
- A implementación dunha interface de representación e exploración de ditos formatos.

Restricións do proxecto

- O proxecto suporá 401,25 horas de traballo autónomo e 11,25 horas de titorías que se distribuirán nun máximo de 12 semanas.
- O volume de datos do que se dispón para facer as probas non supera a orde de decenas de GB.
- A linguaxe de programación a utilizar deberá ser compatible co *framework* de computación distribuída elixido.
- As restricións respecto do uso de memoria virán marcadas polo entorno do que se dispón para realizar as probas.

Supostos do proxecto

- Asumimos que dispoñemos do presuposto necesario para levar a cabo o proxecto.
- Asumimos que dito presuposto non se vai a modificar.
- Asumimos que dispoñemos dun entorno de computación distribuída adecuado para levar a cabo as probas.

Estrutura de descomposición do traballo

O seguinte diagrama ofrece unha descomposición xerárquica dos principais paquetes de traballo a realizar no proxecto. Posteriormente, no apartado de planificación, agrégase un maior nivel de detalle a ditos grupos de tarefas.

Figura 4.- Estrutura de descomposición de tarefas.

Descrición do paquete de traballo	
Número de paquete	1
Nome do paquete	Xestión do proxecto
Descrición	Este paquete agrupa as tarefas de definición do alcance e da metodoloxía a utilizar, realización da planificación temporal e da estimación de custos, e definición dun proceso de xestión de riscos e outro de control da configuración do proxecto. Tamén se inclúen aquí as reunións.
Duración estimada	45h
Responsable	Diego
Produtos	Enunciado do alcance do proxecto, Planificación temporal, Presuposto inicial, Proceso de xestión de riscos, Proceso de xestión da configuración

Descrición do paquete de traballo	
Número de paquete	2
Nome do paquete	Exploración do estado da arte
Descrición	Este paquete agrupa tres tarefas principais dedicadas a documentarse sobre as principais solucións existentes no noso campo de traballo: solucións espaciais clásicas, Big Data e Spatial Big Data.
Duración estimada	40h
Responsable	Diego
Produtos	-

Descrición do paquete de traballo	
Número de paquete	3
Nome do paquete	Análise de requisitos
Descrición	Este paquete contén como tarefas a especificación dos requisitos tanto funcionais como non funcionais e a especificación dos casos de uso.
Duración estimada	11h
Responsable	Diego
Produtos	Especificación de requisitos, Especificación de casos de uso

Descrición do paquete de traballo	
Número de paquete	4
Nome do paquete	Deseño global
Descrición	As tarefas deste paquete consistirán en facer un deseño a nivel global do sistema a desenvolver tanto desde un punto de vista estático como desde un punto de vista dinámico.
Duración estimada	8h
Responsable	Diego
Produtos	Diagramas de compoñentes, Diagramas de secuencia

Descrición do paquete de traballo	
Número de paquete	5
Nome do paquete	Prototipado inicial
Descrición	Inicialmente cómpre desenvolver un prototipo funcional do software definindo para elo as estruturas de datos e os algoritmos cos que vamos a traballar posteriormente. Tamén cómpre especificar un formato de definición de fontes para o usuario.
Duración estimada	60h
Responsable	Diego
Produtos	Ficheiro de definición de fontes, Prototipo inicial

Descrición do paquete de traballo	
Número de paquete	6
Nome do paquete	Desenvolvemento iterativo
Descrición	O desenvolvemento consistirá nunha serie de iteracións orientadas a producir software funcional sobre o cal se poderán facer probas continuas. Para elo, cada iteración requirirá dun breve proceso previo de análise e deseño.
Duración estimada	172h
Responsable	Diego
Produtos	Versión funcional do software

Descrición do paquete de traballo	
Número de paquete	7
Nome do paquete	Documentación do proxecto
Descrición	Este grupo de tarefas está orientado á realización da memoria do traballo e da preparación da presentación, e de ser o caso, a documentación de uso do programa.
Duración estimada	84h
Responsable	Diego
Produtos	Memoria do proxecto, Documentos de presentación, Documentos de uso

Metodoloxía a utilizar

A metodoloxía de desenvolvemento elixida para realizar o traballo é unha variación de programación extrema especialmente adaptada ás necesidades deste proxecto. En concreto, considéranse os seguintes aspectos:

- O desenvolvemento consistirá na implementación iterativa de pequenas melloras ou adaptacións de propostas conducentes a afrontar os problemas atopados, partindo dun prototipo inicial.
- En cada iteración crearanse versións funcionais do software en forma de prototipos que permitan levar a cabo probas sobre cada módulo por separado e sobre funcionalidades específicas dentro de cada módulo:
    - As probas sobre cada módulo correspóndense coa execución do caso de uso correspondente a ese módulo e a comprobación da saída. Por exemplo, para un módulo de procesamento, haberá que comprobar que o resultado do procesamento é correcto.
    - A proba sobre funcionalidades específicas requirirá a definición de casos de probas sobre os principais métodos do módulo, quedando suxeitos ás posibilidades que ofreza o *framework* elixido.
- En cada iteración levaranse a cabo melloras do código que incrementen a súa eficiencia e sobre as que haberá que levar a cabo probas de rendemento.
- A busca da simplicidade será un dos principios reitores á hora de garantir a funcionalidade do software.
- Estará ausente unha das prácticas máis habituais dentro da programación extrema, como é a programación por pares, debido ás limitacións propias do proxecto, mentres que outras características como o o desenvolvemento orientado a probas (TDD, *Test Driven Development*) se utilizará naquelas iteracións que se considere oportuno para o proxecto.

A xustificación desta elección despréndese do tipo de traballo que se propón realizar, de forma que ó establecerse como un dos obxectivos principais a medida do rendemento de diferentes alternativas, se require unha metodoloxía que axilice o desenvolvemento e se oriente á mellora continua a través da realización de probas e medidas de rendemento para os diferentes módulos a construír. A programación extrema permite, neste senso, incrementar a produtividade á hora de responder ante propostas de cambio continuas, e introduce puntos de control que se constitúen no fío condutor da mellora do software.

Seguimento e control do proxecto

Este apartado está dedicado a especificar un proceso xeral que permita controlar a configuración do proxecto ó longo da súa duración. O control do cronograma, así como o control de custos e o control dos riscos serán explicados posteriormente nos seus correspondentes apartados.

O contexto de desenvolvemento deste proxecto posúe certas particularidades debido principalmente a que a maior parte do traballo recae sobre unha persoa, o alumno, que debe desempeñar diversos roles durante a súa realización. Doutra banda, o seu obxectivo final non deixa de ser a creación dun produto software que se pode considerar de complexidade media e polo tanto requírese organizar e manter a integridade de tódolos documentos que se manexen ó longo do traballo. Será o alumno polo tanto quen, en última instancia, xestione todos os cambios que se van a levar a cabo sobre os mencionados documentos e deberá asegurarse de que se realicen de forma controlada para evitar resultados indesexados.

Sistema de xestión da configuración

Para manter unha liña base ó longo do proxecto cómpre definir algún tipo de sistema que permita realizar versións dos documentos de forma sinxela e ó alcance de tódolos actores implicados no seu tratamento. Para o noso caso, decidimos tomar man de dúas ferramentas amplamente empregadas tanto no mundo profesional como no mundo académico:

- Para controlar a edición de documentos de tipo técnico e descritivo, así como a realización de suxerencias, utilizamos Microsoft Word, que posúe un sistema de control de versións propio e permite a posibilidade de realizar suxerencias sobre o texto. Este foi o caso, por exemplo, da memoria. Os medios de compartición de ditos documentos foron tanto o correo electrónico como unha carpeta compartida na rede interna da Universidade.
- Respecto do propio software, cuxo desenvolvemento se realizou no coñecido IDE Netbeans, decidimos crear unha nova versión do proxecto en cada iteración, mantendo as versións anteriores de forma segura e redundante na nube a través da ferramenta Dropbox.

Control dos cambios

O proceso de control dos cambios vén en parte derivado das ferramentas utilizadas, e pivotou en torno a un mecanismo de toma de decisións conxunto. Dunha banda, os titores puideron realizar as súas suxerencias ao alumno mediante o sistema que provee o propio Word para elo, sendo neste caso responsabilidade do alumno aceptalas ou non. Doutra banda, os cambios relativos ós requisitos ou a outros aspectos do software coma o deseño foron discutidos nas reunións periódicas entre os membros do proxecto, chegando a acordos por consenso. A implementación posterior de ditos cambios correu a cargo do alumno.

Planificación temporal

O desenvolvemento do cronograma do proxecto consiste en determinar as datas de inicio e finalización planificadas para as actividades do proxecto. Trátase dun proceso iterativo sobre unha liña base que se definirá inicialmente mediante unha estimación da duración e dos recursos necesarios para levar a cabo cada tarefa.

Para realizar o cronograma podemos valernos de ferramentas como *Microsoft Project* ou *ProjectLibre*. Haberá que dispoñer previamente dunha lista de actividades e dunha lista de recursos xunto co calendario de dispoñibilidade de cada un deles. En xeral, ditas ferramentas utilizan o *método do camiño crítico* para calcular a duración total do proxecto, recorrendo aquela secuencia de tarefas do proxecto que non admite folgura. Adicionalmente, haberá que comprobar que non existen recursos sobreasignados na planificación.

Como vista principal da planificación adxunto a seguinte captura xerada polo software *Microsoft Project* onde aparece a lista detallada de tarefas do proxecto xunto con información relativa a duración, datas de realización e recursos e ó seu carón o diagrama de Gantt correspondente:

DIAGRAMA DE GANTT

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predec	Nombres de los recu
1		**TFG**	**60 días**	**lun 15/02/16**	**mié 04/05/16**		
2		Reunión inicial	1 hora	lun 15/02/16	lun 15/02/16		ADP;XP;AN
3		**Exploración do estado da arte**	**5,71 días**	**lun 15/02/16**	**lun 22/02/16**		
4		Solucións espaciais clásicas	8 horas	lun 15/02/16	mar 16/02/16	2	ADP
5		Big Data	16 horas	mar 16/02/16	jue 18/02/16	4	ADP
6		Spatial Big Data	16 horas	jue 18/02/16	lun 22/02/16	5	ADP
7		Reunión	1 hora	lun 22/02/16	lun 22/02/16	6	ADP;XP;AN
8		Enunciado do alcance	4 horas	lun 22/02/16	lun 22/02/16	7	ADP
9		Definición do control da configuración	4 horas	lun 22/02/16	mar 23/02/16	8	ADP
10		Realización da EDT	4 horas	mar 23/02/16	mar 23/02/16	9	
11		Reunión	1 hora	mar 23/02/16	mar 23/02/16	10	ADP;XP;AN
12		**Análise de requisitos**	**1,57 días**	**mié 24/02/16**	**jue 25/02/16**		
13		Requisitos funcionais	4 horas	mié 24/02/16	mié 24/02/16	11	ADP
14		Requisitos non funcionais	4 horas	mié 24/02/16	mié 24/02/16	13	ADP
15		Casos de uso	3 horas	jue 25/02/16	jue 25/02/16	14	ADP
16		Planificación temporal	4 horas	jue 25/02/16	jue 25/02/16	15	ADP
17		Estimación de custos	4 horas	jue 25/02/16	vie 26/02/16	16	ADP
18		Análise e planificación de riscos	8 horas	vie 26/02/16	lun 29/02/16	17	ADP
19		Reunión	1 hora	lun 29/02/16	lun 29/02/16	18	ADP;XP;AN
20		Seguemento e control	1 hora	lun 29/02/16	lun 29/02/16	19	ADP
21		**Deseño global**	**1,14 días**	**lun 29/02/16**	**mar 01/03/16**		
22		Compoñentes do sistema	4 horas	lun 29/02/16	mar 01/03/16	20	ADP
23		Interaccións do sistema	4 horas	mar 01/03/16	mar 01/03/16	22	ADP
24		Reunión	1 hora	mar 01/03/16	mar 01/03/16	23	ADP;XP;AN
25		Seguemento e control	1 hora	mar 01/03/16	mar 01/03/16	24	ADP
26		**Iteración 1**	**8,57 días**	**mar 01/03/16**	**vie 11/03/16**		
27		Análise e deseño	8 horas	mar 01/03/16	mié 02/03/16	25	ADP
28		Interface de definición de fontes	8 horas	mié 02/03/16	jue 03/03/16	27	ADP
29		Implementación do módulo ETL	16 horas	jue 03/03/16	lun 07/03/16	28	ADP

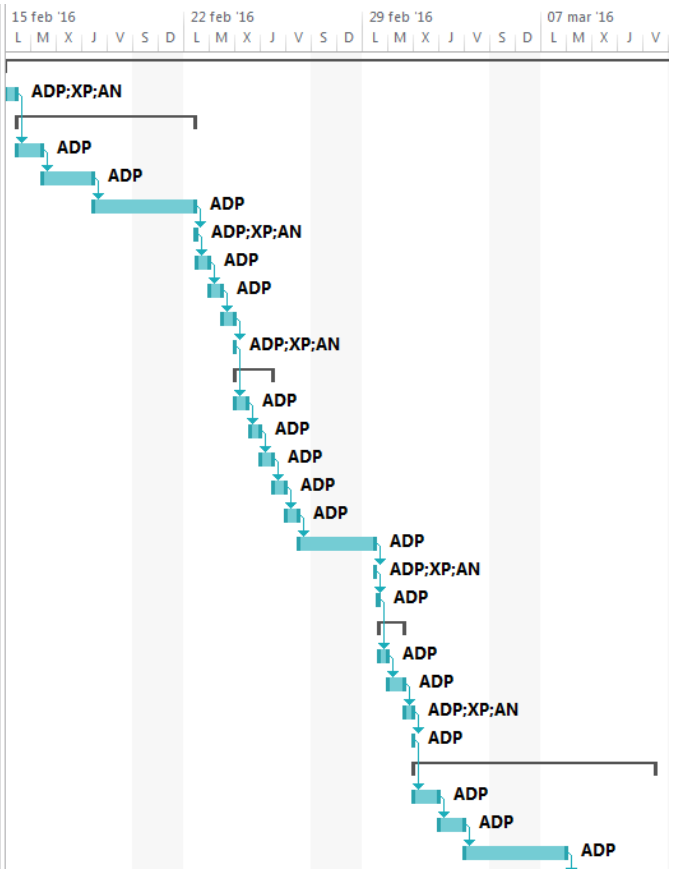


Figura 5.- Diagrama de Gantt (parte 1).

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predec	Nombres de los recu
30		Implementación do módulo OLAP	16 horas	lun 07/03/16	mié 09/03/16	29	ADP
31		Probas unitarias	2 horas	mié 09/03/16	jue 10/03/16	30	ADP
32		Probas de integración	2 horas	jue 10/03/16	jue 10/03/16	31	ADP
33		Probas de rendemento	8 horas	jue 10/03/16	vie 11/03/16	32	ADP
34		Reunión	1 hora	vie 11/03/16	vie 11/03/16	33	ADP;XP;AN
35		Seguemento e control	1 hora	vie 11/03/16	vie 11/03/16	34	ADP
36		**Iteración 2**	**7,43 días**	**vie 11/03/16**	**mar 29/03/16**		
37		Análise e deseño	8 horas	vie 11/03/16	lun 14/03/16	35	ADP
38		Implementación do módulo ETL	16 horas	lun 14/03/16	mié 16/03/16	37	ADP
39		Implementación do módulo OLAP	16 horas	mié 16/03/16	vie 18/03/16	38	ADP
40		Probas unitarias	2 horas	vie 18/03/16	vie 18/03/16	39	ADP
41		Probas de integración	2 horas	vie 18/03/16	lun 28/03/16	40	ADP
42		Probas de rendemento	8 horas	lun 28/03/16	mar 29/03/16	41	ADP
43		Reunión	1 hora	mar 29/03/16	mar 29/03/16	42	ADP;XP;AN
44		Seguemento e control	1 hora	mar 29/03/16	mar 29/03/16	43	ADP
45		**Iteración 3**	**4 días**	**mar 29/03/16**	**vie 01/04/16**		
46		Análise e deseño	8 horas	mar 29/03/16	mié 30/03/16	44	ADP
47		Implementación do módulo de exportación	16 horas	mié 30/03/16	vie 01/04/16	46	ADP
48		Probas unitarias	2 horas	vie 01/04/16	vie 01/04/16	47	ADP
49		Probas de integración	2 horas	vie 01/04/16	vie 01/04/16	48	ADP
50		Reunión	1 hora	vie 01/04/16	vie 01/04/16	49	ADP
51		Seguemento e control	1 hora	lun 04/04/16	lun 04/04/16	50	ADP

Figura 6.- Diagrama de Gantt (parte 2).

52							
53	★	◢ **Iteración 4**	**7,43 días**	**lun 04/04/16**	**mar 12/04/16**		
54	★	Análise e deseño	8 horas	lun 04/04/16	mar 05/04/16	51	ADP
55	★	Implementación do módulo ETL	16 horas	mar 05/04/16	jue 07/04/16	53	ADP
56	★	Implementación do módulo OLAP	16 horas	jue 07/04/16	lun 11/04/16	54	ADP
57	★	Probas unitarias	2 horas	lun 11/04/16	lun 11/04/16	55	ADP
58	★	Probas de integración	2 horas	lun 11/04/16	lun 11/04/16	56	ADP
59	★	Probas de rendemento	8 horas	lun 11/04/16	mar 12/04/16	57	ADP
60	★	Reunión	1 hora	mar 12/04/16	mar 12/04/16	58	ADP;XP;AN
61	★	Seguemento e control	1 hora	mar 12/04/16	mar 12/04/16	59	ADP
62	★	◢ **Iteración 5**	**5,71 días**	**mar 12/04/16**	**mar 19/04/16**		
63	★	Análise e deseño	8 horas	mar 12/04/16	mié 13/04/16	60	ADP
64	★	Implementación	24 horas	mié 13/04/16	lun 18/04/16	62	ADP
65	★	Probas de rendemento	8 horas	lun 18/04/16	mar 19/04/16	63	ADP
66	★	Reunión final	1 hora	mar 19/04/16	mar 19/04/16	64	ADP;XP;AN
67	★	Seguemento e control	1 hora	mié 20/04/16	mié 20/04/16	65	ADP
68	★	Realización da memoria	10 días	mié 20/04/16	lun 02/05/16	66	ADP
	★	Revisión	2 días	lun 02/05/16	mié 04/05/16	67	ADP;XP;AN

Figura 7.- Diagrama de Gantt (parte 3).

O proxecto comeza cunha reunión inicial para a presentación dos participantes e da temática así como do seu contexto. A primeira fase de traballo correspóndese coa exploración do estado da arte, que dura aproximadamente uns seis días. Logo prodúcese outra reunión co obxectivo de definir o alcance do proxecto. A raíz desta reunión elabóranse o enunciado do alcance do proxecto e a estrutura de descomposición de tarefas, e defínese un sistema de control de configuración. A seguinte reunión ten como obxectivo identificar os requisitos do sistema, e a partir do borrador de requisitos obtido nela elabórase a lista de requisitos funcionais e non funcionais así como os casos de uso. Logo concrétanse a planificación temporal, o presuposto e o plan de riscos do proxecto. A seguinte reunión serve para identificar os principais compoñentes do sistema, que posibilitarán a realización dun deseño global do mesmo tanto desde un punto de vista dinámico como desde un punto de vista estático. A continuación, comeza a fase de desenvolvemento (a máis longa do proxecto), que no marco do ciclo de vida elixido se produce iterativamente, comezando co desenvolvemento dun prototipo inicial totalmente funcional e introducindo melloras nas seguintes iteracións. A iteración 3 resérvase para a realización do módulo de exportación e visualización, namentres as restantes se dedican á implementación de melloras específicas. Cada iteración ten as súas fases de análise, deseño, implementación e probas. Por último, resérvanse algo máis dunha decena de días para a realización da memoria e a súa revisión. Periodicamente, despois de cada reunión, dedícase un tempo moderado a facer tarefas de seguimento e control do proxecto.

Control da planificación

Dentro do paquete de traballo de xestión do proxecto, a tarefa de seguimento e control do proxecto debe incluír tamén as comprobacións periódicas necesarias para verificar o cumprimento da planificación inicial especificada. O informe do avance sobre a planificación do traballo será un dos temas a tratar en cada reunión periódica co xefe de proxecto ou, neste caso, titor. Os cambios na planificación deberán estar sometidos á súa aprobación, e realizaranse seguindo o proceso xeral de control de cambios definido.

Estimación de custos

A estimación de custos non é máis ca unha aproximación dos custos dos recursos necesarios para completar as actividades que compoñen o proxecto. Para poder definir un presuposto inicial podemos partir da suma dos custos estimados, en base a prezos de mercado, das actividades individuais ou paquetes de traballo. Este presuposto vai a constituír a liña base de custo do proxecto e o posterior control de custos consistirá en realizar un seguimento sobre as posibles variacións que poidan xurdir sobre esa estimación inicial.

Na seguinte táboa detallamos a estimación inicial de custos do proxecto:

Táboa 1.- Estimación de custos.

Concepto	**Recurso**	**Horas**	**Custo**
Xestión do proxecto	XP	10	393.50
	AN	10	293.00
	ADP	45	863.10
Exploración do estado da arte	ADP	40	767.20
Análise de requisitos	ADP	11	210.98
Deseño global	ADP	8	153.44
Prototipado inicial	ADP	60	1150.80
Desenvolvemento iterativo	ADP	172	3298.96
Documentación do proxecto	XP	14	550.90
	AN	14	410.20
	ADP	84	1611.12
Materiais	PC	420	2.39
Aluguer do clúster	Clúster	240	36.00
Custos indirectos	-	-	2045.74
Fondo de reserva	-	-	500.00
TOTAL			**12287,33 €**

O prezo dos recursos humanos (baseado nunha estimación de prezos de mercado a partir de ofertas de traballo en InfoJobs [15]) é o seguinte:

- XP (Xefe de Proxecto): 39.35 €/hora.
- AN (Analista principal): 29.30 €/hora.
- ADP (Analista-Deseñador-Programador): 19.18 €/hora.

O prezo dos recursos materiais remítese ó PC de sobremesa utilizado e consiste no cálculo da amortización para o número aproximado de horas de uso en base ó prezo dun equipo de similares características en Amazon [2]:

- Valor de mercado do equipo: 559 €
- Valor residual do equipo: 60 €
- Vida útil do equipo: 87600 horas (10 anos)
- Cota de amortización (por hora): $(559 - 60) \,/\, 87600 = 0.0057$ €

O custo do uso do clúster vén determinado polo cálculo da amortización que corresponde ó número de horas de uso rexistradas. Como se menciona no capítulo de ferramentas, dito clúster está composto de cinco nodos HP Proliant DL 370 G6 dos cales se toma de referencia, de novo, o prezo actual de Amazon:

- Valor de mercado do clúster: 2866.5 € * 5 unidades = 14332.5 €
- Valor residual do clúster: 200 € * 5 unidades = 1000 €
- Vida útil do clúster: 87600 horas (10 anos)
- Cota de amortización (por hora): $(14332.5 - 1000) / 87600 = 0.15$ €

Os custos indirectos calcúlanse utilizando unha porcentaxe do 21% sobre o custo do proxecto (excluíndo o fondo de reserva):

- Custos indirectos: $9741.59 \times 0.21 = 2045.74$ €

Control de custos

Convén especificar aqueles aspectos que permitirán levar a cabo un correcto seguimento do cumprimento do presuposto inicial así como o mecanismo mediante o cal poderá modificarse, de ser necesario:

- En primeiro lugar cómpre definir un nivel de precisión para o custo, que será de dous decimais neste proxecto, e as unidades de medida para os diferentes tipos de recursos, que son as seguintes:
    - Número de unidades e prezo por unidade para recursos materiais.
    - Horas/home para indicar a cantidade de traballo dunha persoa e euros/hora para indicar o seu custo económico.
    - Número de meses e prezo por mes para calcular o custo dos servizos que se contraten.
- A exactitude do presuposto mellora usualmente a medida que avanza a realización do proxecto debido á existencia dunha maior cantidade de información. Admitimos unha marxe de erro do presuposto inicial de entre o -10% e o +10%. A desviación permitida abrangue esta marxe, debendo aplicar accións correctivas no caso de rebasalo. Por outra parte, para facer fronte a posibles imprevistos inclúese en dito presuposto un fondo de reserva de ata cincocentos euros.
- En todo caso, a modificación do presuposto inicial requirirá a aprobación dos xefes de proxecto mediante a presentación do correspondente informe xustificativo.

Xestión de riscos

Planificación da xestión de riscos

Como primeiro paso para levar a cabo unha planificación de riscos partimos da definición das principais características que van a definir a cada un dos riscos baseándonos nas especificacións usualmente indicadas nas principais guías de riscos utilizadas no sector [23]:

- **Impacto.-** Representa o efecto que a ocorrencia do risco tería no desenvolvemento do proxecto, en termos de custo, esforzo ou duración total do mesmo.
- **Probabilidade.-** Representa a expectativa da ocorrencia real do risco.
- **Nivel de exposición.-** Produto do impacto pola probabilidade. Tómase como referencia para a xestión dos riscos.

Cómpre definir valores numéricos para cada unha destas características de xeito que se poida cuantificar e clasificar a importancia de cada risco para o proxecto:

Táboa 2.- Nivel numérico de exposición a un risco.

			Probabilidade		
			Alta	**Media**	**Baixa**
			0.65	***0.35***	***0.15***
Impacto	**Alto**	***0.65***	0.42	0.22	0.1
	Medio	***0.35***	0.22	0.12	0.05
	Baixo	***0.15***	0.1	0.05	0.02

Os niveis de impacto que aparecen na táboa teñen o seguinte significado:

- Alto.- Cando os efectos da ocorrencia do risco superen umbrais do 75% sobre o total en termos de custo, tempo ou alcance.
- Medio.- Cando os efectos da ocorrencia do risco se sitúan en umbrais de entre o 15% e o 65% sobre o total en termos de custo, tempo ou alcance.
- Baixo.- Cando os efectos da ocorrencia do risco non superen umbrais do 15% sobre o total en termos de custo, tempo ou alcance.

Os niveis de probabilidade que aparecen na táboa teñen o seguinte significado:

- Alta.- Cando a ocorrencia do risco se espera durante a realización do proxecto.
- Media.- Cando a ocorrencia do risco non se pode predicir durante a realización do proxecto.
- Baixa.- Cando a ocorrencia do risco non se espera durante a realización do proxecto.

Respecto ós códigos de colores da exposición resultante, teñen o seguinte significado:

- Vermello.- Nivel de exposición alto.
- Laranxa.- Nivel de exposición medio.
- Verde.- Nivel de exposición baixo.

Identificación de riscos

O proceso de identificación de riscos asociados ao proxecto consistiu nunha combinación das seguintes metodoloxías:

- **Revisión da documentación**, intentando detectar os riscos a partir dos problemas existentes nos documentos do proxecto.
- **Tormenta de ideas**, realizada nunha das reunións do grupo de traballo.
- **Lista de control**, seleccionando os riscos relevantes para este proxecto concreto dunha lista xenérica[6].

Sendo así, a saída do proceso consistiu na seguinte lista de riscos identificados:

Táboa 3.- Lista de riscos.

Código	Descrición	Indicador
RSC01	Planificación demasiado optimista ou pouco realista.	Grao de cumprimento da planificación.
RSC02	O inicio do proxecto retrásase debido a obxectivos difusos.	Data de finalización da fase inicial do proxecto.
RSC03	As ferramentas para realizar probas non están dispoñibles no momento necesario.	Estado de dispoñibilidade do clúster.
RSC04	Os titores insisten en novos requisitos.	Peticións de cambio nas reunións periódicas.
RSC05	O tempo de desenvolvemento alárgase máis do esperado.	Data de finalización das fases de desenvolvemento.
RSC06	Os requisitos non estaban ben definidos.	Probas de verificación do software en cada iteración.
RSC07	O traballo cun entorno software distribuído causa problemas non previstos.	Data de finalización das probas de rendemento.
RSC08	O produto depende de estándares insuficientemente definidos.	Resultado do proceso de documentación sobre o estado da arte e as librerías existentes.
RSC09	A falta de motivación e de moral reduce a produtividade.	Grao de cumprimento da planificación.
RSC10	Non se pode implementar a funcionalidade desexada ca linguaxe ou bibliotecas utilizadas.	Probas de verificación do software en cada iteración.
RSC11	A burocracia do proxecto produce un proceso máis lento do esperado.	Grao de cumprimento da planificación.

RSC12	O esforzo é maior do estimado.	Horas de traballo adicionais sobre as reflexadas na planificación.

Análise cualitativa de riscos

Este tipo de análise céntrase en asignar unha probabilidade e un impacto a cada risco para poder determinar un valor de exposición, utilizando a táboa de valores numéricos definida no apartado anterior:

Táboa 4.- Análise cualitativa de riscos.

Código	Probabilidade	Impacto	Exposición
RSC01	Alta	Medio	0.22
RSC02	Media	Medio	0.12
RSC03	Alta	Alto	0.42
RSC04	Baixa	Medio	0.05
RSC05	Alta	Medio	0.22
RSC06	Media	Alto	0.22
RSC07	Alta	Alto	0.42
RSC08	Baixa	Medio	0.05
RSC09	Baixa	Medio	0.05
RSC10	Media	Medio	0.12
RSC11	Baixa	Baixo	0.02
RSC12	Baixa	Baixo	0.02

Pódese apreciar mediante os valores de exposición que existen cinco riscos cun nivel de exposición alto, dous riscos cun nivel de exposición medio e cinco riscos cun nivel de exposición baixo.

Planificación da resposta aos riscos

Vistos os niveis de exposición no apartado anterior, cómpre aclarar en primeiro lugar que os riscos cun nivel de exposición baixo se aceptan sen tomar ningunha acción adicional. Respecto ós demais riscos, imos a definir unha estratexia de resposta de entre as que propoñemos na seguinte lista:

- **Evitar.-** A acción a tomar ten como obxectivo intentar que o risco non chegue a suceder.
- **Transferir.-** Derívase a responsabilidade do risco noutra entidade, probablemente externa.
- **Mitigar.-** Defínese unha acción que reduza o dano provocado por un risco no caso de que ocorra.
- **Continxencia.-** Trátase de desenvolver un plan que se levará a cabo no momento no que se detecte que o risco aparece.
- **Aceptar.-** Asumir o risco, cando o resto de alternativas non son viables.

Na seguinte táboa defínese a estratexia elixida para cada risco:

Táboa 5.- Estratexias de resposta aos riscos.	
Código	**Estratexia**
RSC01	Mitigar (10% de folgura en tarefas críticas)
RSC02	Evitar (clarificación dos obxectivos na reunión inicial)
RSC03	Continxencia (realización de probas en local con máquinas virtuais)
RSC04	Aceptar
RSC05	Mitigar (10% de folgura en tarefas críticas)
RSC06	Evitar (clarificación dos requisitos na reunión de especificación dos requisitos)
RSC07	Mitigar (utilización de versións estables e adopción de configuracións base)
RSC08	Aceptar
RSC09	Aceptar
RSC10	Mitigar (utilización dunha linguaxe de propósito xeral e de librerías suficientemente probadas)
RSC11	Aceptar
RSC12	Aceptar

Seguimento e control de riscos

A tarefa de seguimento e control dos riscos está definida na planificación temporal dentro dun paquete de tarefas denominado “Seguimento e control do proxecto”, que se vai a executar periodicamente tras cada iteración. Dito seguimento realizarase mediante os indicadores de risco definidos na táboa 3 e, no seu caso, as accións a tomar basearanse nas indicadas na táboa 5. O control dos riscos reflexarase de ser necesario mediante o correspondente documento de incidencias.

Análise de requisitos

Casos de uso

Os principais casos de uso do sistema a desenvolver son os seguintes:

- O usuario define a configuración das fontes de datos a utilizar. Os dous tipos principais de fontes a considerar son as relacionais e as vectoriais.
- O usuario executa o proceso de extracción, transformación e carga dos datos (ETL), sobre un sistema de almacenamento distribuído.
- O usuario executa a secuencia de operacións de procesamento sobre os datos. En ditas operacións haberá que considerar especificamente as combinacións por clave compartida ou por algunha condición espacial, así como as agregacións de ventás espaciais e/ou temporais.
- O usuario explora e visualiza os datos de entrada e saída. As características dos datos admitirán representacións gráficas e/ou textuais.

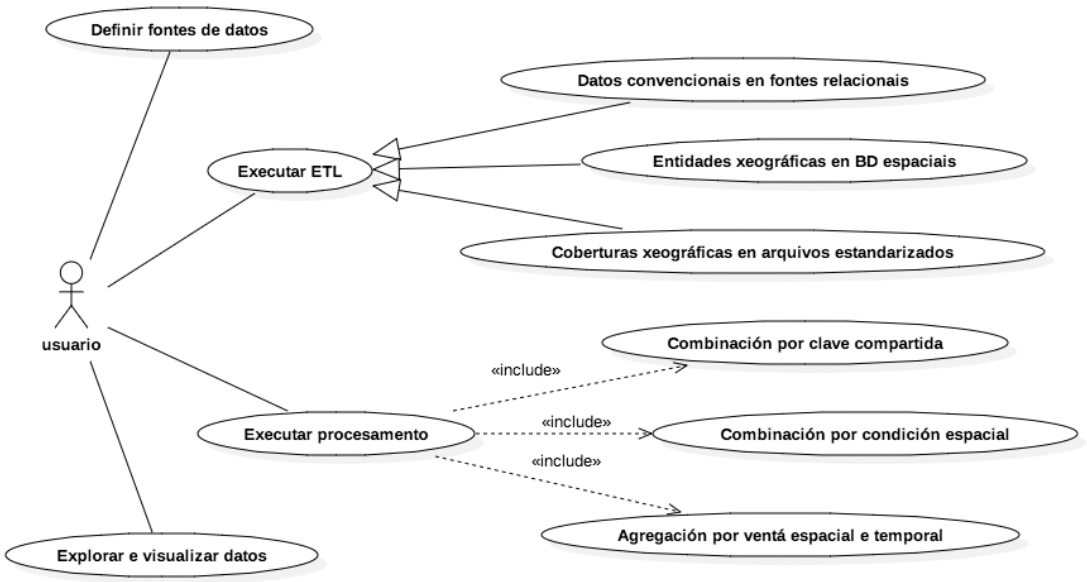


Figura 8.- Diagrama de casos de uso.

A continuación lístanse de xeito máis formal o conxunto de casos de uso que compoñen o sistema, seguindo para elo as instrucións indicadas na guía de Sommerville [23]:

CU01	
Función	Definición das fontes de datos.
Importancia	Alta
Dependencias	-
Descrición	Trátase de definir mediante o formato que se especifique as fontes (arquivos, bases de datos, ...) a partir das cales se van a importar os datos cos que traballa o sistema.
Entradas	Localización do medio de definición de datos.
Fonte	Arquivos e bases de datos.
Saídas	Formato de definición de fontes completado.
Destino	Control principal do programa.
Accións	1. Obter a localización e o formato de definición de fontes de datos. 2. Completar a definición coa información sobre as fontes de datos a utilizar polo sistema.
Excepcións	-
Precondicións	Existen fontes de datos a especificar.
Postcondicións	A información de fontes de datos está dispoñible na localización indicada.

CU02	
Función	Importación de datos convencionais de fontes relacionais.
Importancia	Alta
Dependencias	CU01
Descrición	Trátase de incorporar ó sistema datos como por exemplo os nomes das estacións meteorolóxicas que se almacenan nunha táboa dunha base de datos.
Entradas	Parámetros de conexión coa base de datos.
Fonte	Formato de definición das fontes de datos.
Saídas	Estrutura de datos cos valores importados.
Destino	Control principal do programa.
Accións	1. Comprobar se existen fontes de datos relacionais definidas. 2. Se existen, realizar a importación dos datos de tipo convencional indicados nas fontes.
Excepcións	3. Se non existen, mostrar unha mensaxe informativa.
Precondicións	Existen fontes de datos relacionais definidas.
Postcondicións	O control do programa dispón dos datos en memoria.

CU03	
Función	Importación de datos de entidades xeográficas de bases de datos espaciais, así como a súa evolución temporal.
Importancia	Alta
Dependencias	CU01
Descrición	Trátase de incorporar ó sistema datos como por exemplo a xeometría que indica a localización dunha estación ou a serie temporal de observacións meteorolóxicas desa estación almacenados en táboas de bases de datos espaciais.
Entradas	Parámetros de conexión coa base de datos.
Fonte	Formato de definición das fontes de datos.
Saídas	Estrutura de datos cos valores importados.
Destino	Control principal do programa.
Accións	1. Comprobar se existen fontes de datos relacionais con extensións espaciais definidas. 2. Se existen, realizar a importación dos datos de tipo xeográfico ou temporal indicados nas fontes.
Excepcións	3. Se non existen, mostrar unha mensaxe informativa.
Precondicións	Existen fontes de datos relacionais definidas con extensións espaciais.
Postcondicións	O control do programa dispón dos datos en memoria.

CU04	
Función	Importación de coberturas xeográficas e a súa evolución temporal almacenados en arquivos de tipo *raster* ou de vectores multidimensionais.
Importancia	Alta
Dependencias	CU01
Descrición	Trátase de incorporar ó sistema datos como por exemplo os valores de elevación do terreo que se almacenan nun arquivo cun formato estandarizado tipo NetCDF ou GeoTiff.
Entradas	Información de localización e tipo de arquivo.
Fonte	Formato de definición das fontes de datos.
Saídas	Estrutura de datos cos valores importados.
Destino	Control principal do programa.
Accións	1. Comprobar se existen fontes de datos asociadas a ficheiros definidas. 2. Se existen, realizar a importación, en función do tipo de arquivo, dos datos indicados nas fontes.
Excepcións	3. Se non existen, mostrar unha mensaxe informativa.
Precondicións	Existen fontes de datos asociadas a ficheiros definidas.
Postcondicións	O control do programa dispón dos datos en memoria.

CU05	
Función	Almacenamento dos datos de entrada e saída nun sistema de almacenamento distribuído.
Importancia	Alta
Dependencias	CU02, CU03, CU04
Descrición	Trátase de permitir que os datos importados das fontes de datos se poidan almacenar de forma distribuída para ser procesados en paralelo, así como ofrecer a opción de exportar o resultado ó mesmo sistema de almacenamento.
Entradas	Datos importados das fontes; datos resultado do procesamento.
Fonte	Fontes de datos; procesamento.
Saídas	Datos exportados nun formato compatible co sistema de almacenamento distribuído.
Destino	Sistema de almacenamento distribuído.
Accións	1. Obter os parámetros de acceso ao sistema de almacenamento distribuído e a referencia ao conxunto de datos que se desexa gardar. 2. Realizar a escritura dos datos no formato de almacenamento que se especifique.
Excepcións	3. No caso de erro de comunicacións co sistema de almacenamento distribuído, mostrar mensaxe informativa.
Precondicións	Téñense importado datos das fontes ou resultado do procesamento.
Postcondicións	Os datos están dispoñibles no sistema de almacenamento distribuído especificado.

CU06	
Función	Aplicación de funcións de procesamento sobre fontes de datos de todos os tipos.
Importancia	Alta
Dependencias	CU05
Descrición	Trátase de posibilitar e levar a cabo o procesamento dos datos importados das fontes, implementando operacións típicas como por exemplo a transformación das coordenadas espaciais dos obxectos.
Entradas	Arquivos de datos almacenados para procesar.
Fonte	Sistema de almacenamento distribuído.
Saídas	Conxunto de datos resultado do procesamento.
Destino	Control principal do programa.
Accións	1. Especificar os datos a procesar. 2. Importar os datos desde o sistema de almacenamento distribuído. Se non existen, mostrar mensaxe informativa. 3. Procesar os datos aplicando un *pipeline* de operacións que produce un resultado.
Excepcións	4. Se se produce un error no procesamento, mostrar mensaxe informativa.
Precondicións	Existen os arquivos que conteñen os datos a procesar.
Postcondicións	O control do programa dispón do resultado en memoria.

CU07	
Función	Combinación de varias fontes de datos a través de claves compartidas (*equijoin*).
Importancia	Alta
Dependencias	CU05
Descrición	O sistema debe permitir a unión de dous conxuntos de datos mediante unha clave común; por exemplo: a unión entre obxectos de tipo estación e obxectos de tipo observación a través dun identificador de estación presente en ambos tipos de obxectos.
Entradas	Conxuntos de datos a combinar.
Fonte	Almacenamento distribuído ou resultado de procesamento.
Saídas	Conxunto de datos resultado da combinación.
Destino	Control principal do programa.
Accións	1. Cargar os conxuntos de datos a combinar en memoria. 2. Comprobar que os dous tipos de obxectos conteñen a clave de unión. 3. Realizar a unión dos conxuntos.
Excepcións	4. Se se produce un erro durante a unión, mostrar mensaxe informativa.
Precondicións	Ambos tipos de obxectos conteñen a clave de unión.
Postcondicións	O control do programa dispón do resultado en memoria.

CU08	
Función	Combinación de varias fontes de datos de acordo con algunha condición espacial (*spatial join*).
Importancia	Alta
Dependencias	CU05
Descrición	O sistema debe permitir a unión de dous conxuntos de datos mediante unha condición espacial a especificar; por exemplo: a unión entre obxectos de tipo elevación e obxectos de tipo estación en función dunha determinada distancia xeográfica entre ambos.
Entradas	Conxuntos de datos a combinar.
Fonte	Almacenamento distribuído ou resultado de procesamento.
Saídas	Conxunto de datos resultado da combinación.
Destino	Control principal do programa.
Accións	1. Cargar os conxuntos de datos a combinar en memoria. 2. Comprobar que os dous tipos de obxectos conteñen os atributos xeográficos necesarios. 3. Aplicar a condición ou función de unión sobre ditos atributos e obter o resultado.
Excepcións	4. Se se produce un erro durante a combinación, mostrar mensaxe informativa.
Precondicións	Ambos tipos de obxectos conteñen os atributos xeográficos necesarios para aplicar a condición ou función de unión.
Postcondicións	O control do programa dispón do resultado en memoria.

CU09	
Función	Aplicación de operacións de agregación sobre ventás temporais, espaciais ou espazo-temporais.
Importancia	Alta
Dependencias	CU05
Descrición	O sistema debe permitir o tipo de operacións de agregación mencionado, do cal é un exemplo típico o cálculo da interpolación da temperatura nun determinado punto xeográfico a partir dos datos recollidos por varias estacións cercanas.
Entradas	Conxunto de datos a agregar.
Fonte	Almacenamento distribuído ou resultado de procesamento.
Saídas	Conxunto de datos resultado da agregación.
Destino	Control principal do programa.
Accións	1. Cargar o conxunto de datos en memoria. 2. Especificar a ventá de agregación e comprobar que existen os atributos necesarios para realizala. 3. Aplicar a condición ou función de agregación sobre ditos atributos e obter o resultado.
Excepcións	4. Se se produce un erro durante a agregación, mostrar mensaxe informativa.
Precondicións	Os obxectos dispoñen dos atributos que permiten levar a cabo a agregación.
Postcondicións	O control do programa dispón do resultado en memoria.

CU10	
Función	Exploración e visualización dos datos de entrada e saída.
Importancia	Media
Dependencias	CU06, CU07, CU08, CU09
Descrición	O sistema debe permitir analizar as características que teñen os datos que se van a utilizar no procesamento así como o resultado exportado de dito procesamento, e a súa correcta visualización.
Entradas	Conxuntos de datos a utilizar; resultado do procesamento.
Fonte	Fontes de datos ou resultado de procesamento.
Saídas	Imaxe ou texto lexible coas características a representar.
Destino	Pantalla de visualización.
Accións	1. Seleccionar o ficheiro ou fonte de datos a visualizar. 2. Especificar as características a visualizar. 3. Xerar a imaxe ou o texto coa información.
Excepcións	4. Se se produce un erro durante a exportación ou a xeración da visualización, mostrar mensaxe informativa.
Precondicións	Os datos a representar deben ter un formato estandarizado.
Postcondicións	Os datos poden ser visualizados en pantalla.

Requisitos funcionais

Ó existir unha correspondencia un a un entre os casos de uso especificados e os requisitos funcionais do sistema, remitímonos ó expresado no apartado anterior como listado das principais funcionalidades a implementar.

Requisitos non funcionais

- Requisitos operacionais:
  - O sistema debe poder ser executado nunha arquitectura de procesamento distribuído.
  - O sistema debe de utilizar formatos estandarizados para a entrada e saída dos datos.
- Requisitos de rendemento:
  - O sistema debe de garantir a súa escalabilidade horizontal.
- Requisitos de usabilidade
  - O sistema debe de ser flexible na incorporación de novos formatos de datos.
- Requisitos de desenvolvemento:
  - O desenvolvemento do sistema debe de estar finalizado antes de finais de xuño de 2016.

Validación e verificación de requisitos

A validación de requisitos vaise a realizar a través de revisións conxuntas levadas a cabo durante as reunións periódicas do proxecto, axudándonos para elo dos diferentes prototipos que imos desenvolver en cada iteración.

Por outra parte, para levar a cabo a verificación dos requisitos axudarémonos das probas tanto de funcionalidade como de rendemento que imos a executar sobre cada módulo a desenvolver.

Deseño global

A continuación preséntase unha visión integrada e a grandes trazos dos principais compoñentes do sistema. Esta representación ten como obxectivo proporcionar unha idea xeral da arquitectura a desenvolver e como interaccionan entre si os módulos software que a compoñen.

Arquitectura do sistema

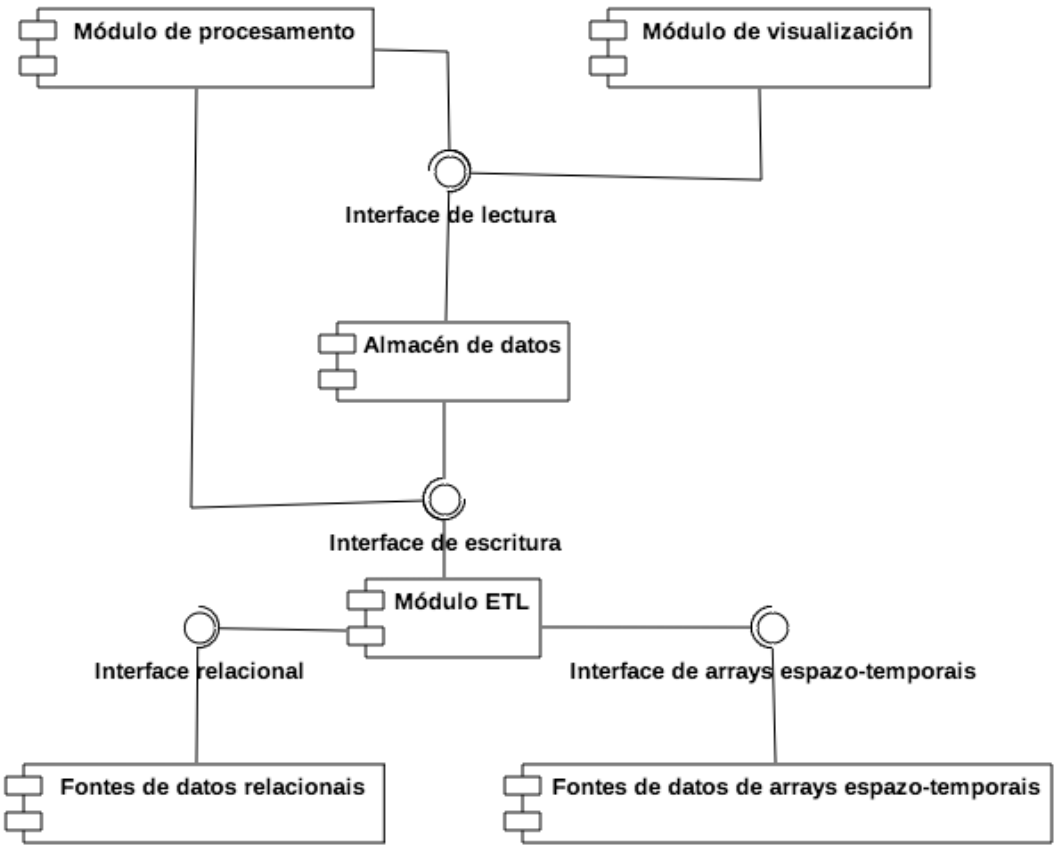


Figura 9.- Arquitectura de compoñentes do sistema.

O sistema compose de tres módulos principais:

1. Módulo ETL para a importación de datos, que accede ós diferentes tipos de fontes a través das correspondentes interfaces.
2. Módulo de procesamento, que le os datos a procesar desde o sistema de almacenamento distribuído para posteriormente volver a escribir o resultado en dito sistema.
3. Módulo de visualización, que accede ós datos do resultado do procesamento escritos no almacenamento distribuído para mostralos por pantalla.

As infraestruturas coas que interaccionan ditos módulos son o sistema de almacenamento distribuído, que permitirá unha xestión integrada de datos procedentes de fontes diversas, e as propias fontes de datos, que poden ser tanto relacionais (contendo datos de tipo entidade) como arquivos de arrays espazo-temporais (contendo coberturas xeográficas) .

Interaccións do sistema

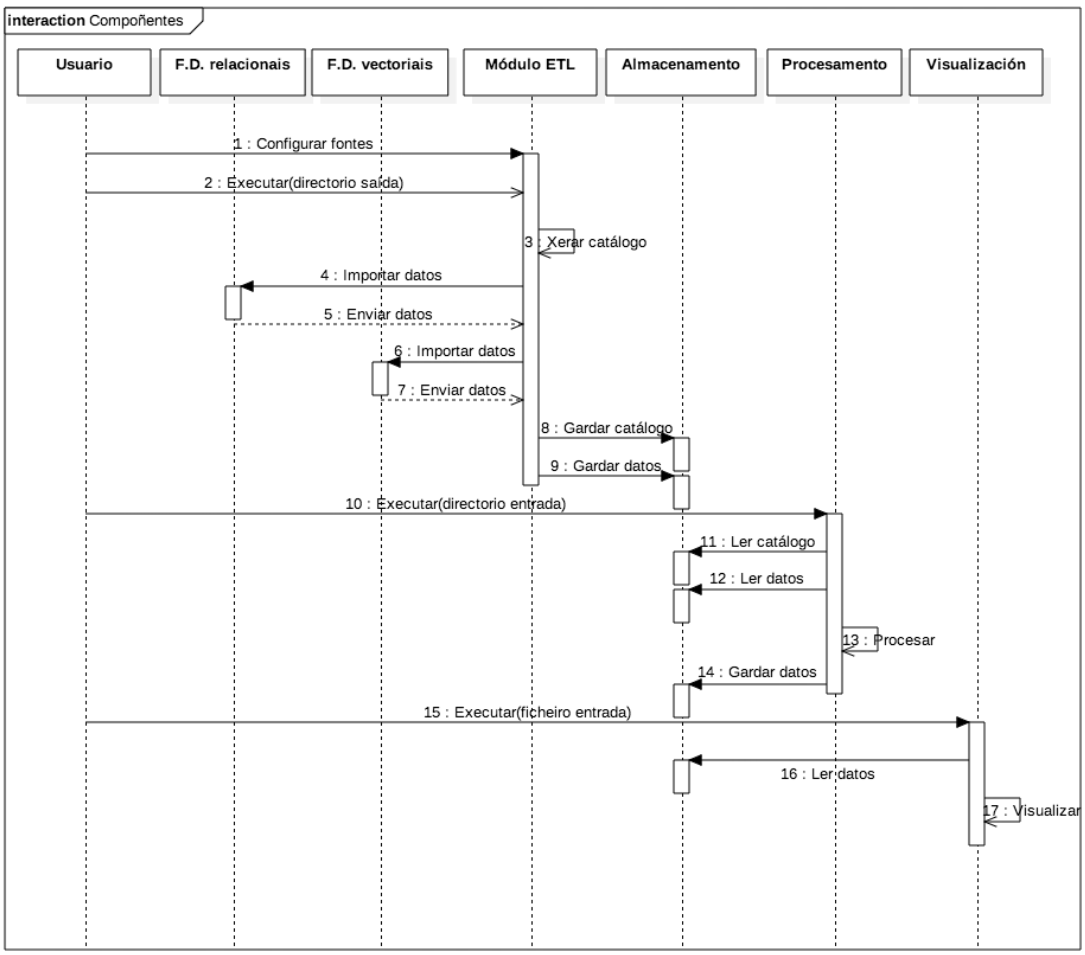


Figura 10.- Interacción entre os compoñentes do sistema.

Como se observa na figura anterior, o usuario inicia a interacción co sistema configurando as fontes de datos para posteriormente executar o módulo de extracción, transformación e carga que realizará a importación dos datos indicados cara o sistema de almacenamento distribuído. Eses datos serán utilizados polo módulo de procesamento para obter o resultado desexado que en última instancia poderá ser visualizado mediante o módulo de visualización. Para a coordinación entre o módulo ETL e o de procesamento utilizaremos un catálogo de datos que tamén se gardará no sistema de almacenamento distribuído.

Tecnoloxías escollidas

En base ós requirimentos especificados para o sistema, esbózanse dúas cuestións principais no referente ás tecnoloxías a utilizar para a súa implementación que están intrinsecamente relacionadas:

1. Dunha banda, para realizar o procesamento e, de ser o caso, a extracción, transformación e carga dos datos en paralelo, o sistema debe executarse sobre unha plataforma de computación distribuída. Segundo o explicado no contexto do problema, as principais tecnoloxías utilizadas na actualidade para levar a cabo proxectos deste tipo son *frameworks* do tipo Hadoop ou Spark. O equipo do proxecto decidiu levar a cabo unha análise das principais fortalezas e debilidades de ditas tecnoloxías e tomar a decisión de cal utilizar:

Táboa 6.- Comparativa de frameworks.

Framework	Fortalezas	Debilidades
Hadoop	• Maior nivel de desenvolvemento • Almacenamento distribuído incorporado (HDFS) • Tolerancia a fallos • Compatibilidade con YARN e Mesos • Open Source • Hardware de baixo custo • Ecosistema máis amplo	• Utilización do esquema de procesamento MapReduce • Escritura en disco tras cada operación intermedia • Destinado a ser executado en clústeres
Spark	• Maior velocidade de procesamento debido á redución de escrituras en disco • Compatibilidade con HDFS e formatos de arquivo de Hadoop • Tolerancia a fallos • Maior flexibilidade na definición de operacións sobre os datos • Módulo Spark SQL • Compatibilidade con YARN e Mesos • Open Source • Hardware de baixo custo • Curva de aprendizaxe máis rápida	• Estado experimental dalgunhas características • Alto consumo de memoria • Destinado a ser executado en clústeres

Despois de levar a cabo pequenos experimentos de rendemento, tiñamos comprobado que o rendemento de Spark na fase de procesamento de datos era superior ó rendemento de Hadoop (nalgúns casos ata dez veces máis rápido), debido ó diferente modo que cada *framework* ten de levar a cabo dito procesamento. En concreto, o modelo procesamento levado a cabo por Spark, ó estar máis orientado cara o encadeamento de operacións sobre conxuntos de datos distribuídos en memoria (RDDs) parecía adecuarse máis á serie de transformacións típicas sobre os tipos de datos xeográficos que se van a utilizar no noso sistema. Esta intuición veuse reforzada por dous engadidos:

- A compatibilidade de Spark con HDFS permitiría dispoñer dun sistema de almacenamento distribuído sobre o cal almacenar os datos obxecto de procesamento e o resultado do mesmo.
- O recente desenvolvemento da extensión para Spark coñecida como *Spark SQL* permitiría o procesamento de datos de tipo estruturado, aínda estando distribuídos, utilizando como linguaxe SQL. Este módulo resultaría especialmente útil á hora de realizar as operacións de combinación e de agregación mencionadas nos requirimentos.

Polo tanto a decisión do equipo foi a de adoptar Spark como *framework* de execución do sistema en conxunto.

2. Debido ó considerado no punto anterior, a cuestión sobre o sistema de almacenamento distribuído a utilizar orientouse cara o uso de HDFS por ter garantida a compatibilidade con Spark (aínda que existen alternativas como Apache HBase [13] ou Apache Cassandra [7]). HDFS é o sistema de ficheiros distribuído e baseado en Java que forma parte de Hadoop, provendo almacenamento escalable e tolerante a fallos. Hai que ter en conta que a maior parte de clústeres nos que se utilizan estes *frameworks* adoptan como configuración predeterminada o uso de HDFS baixo un xestor tipo YARN ou Mesos. Asumindo esta vía de despregue do sistema xurdiu o abanico de posibilidades referentes ó tipo de arquivo para almacenar os nosos datos importados en HDFS. Neste punto foi onde se puxo de manifesto a compatibilidade de Spark cos formatos de arquivo creados no ecosistema de Hadoop. En concreto, valoráronse os seguintes formatos comúns [16]:

Táboa 7.- Comparativa de formatos de arquivos.

Formato	**Estruturado**	**Descrición**
Texto	Non	Arquivos de texto plano. Almacénase un rexistro por liña.
JSON	Semi	Arquivos de texto semi-estruturados.
CSV	Si	Arquivos de texto de uso común con separadores.
SequenceFile	Si	Formato específico de Hadoop que segue un esquema clave-valor e utiliza compresión.
Arquivos de obxectos	Si	Almacenamento serializado de obxectos, dependente do código das clases.

Avro	Si	Formato de almacenamento serializado e con esquema que soporta tipos de datos complexos.
Parquet	Si	Formato de almacenamento por columnas que almacena campos anidados de forma eficiente.

En concreto, o formato SequenceFile foi o utilizado en traballos anteriores debido á utilidade dalgunhas das súas características:

- É o formato utilizado por defecto por parte de Hadoop para o paso da fase de Map á fase de Reduce.
- O acceso ós datos de forma non secuencial é máis eficiente grazas ó formato clave-valor.
- Pódese comprimir e pódese particionar para distribuílo como máis conveña.
- O almacenamento é máis eficiente ó tratarse de ficheiros binarios.

Decidimos implementar o primeiro prototipo do sistema en formato SequenceFile para aproveitarnos desa experiencia previa e reducir o risco, nas fases iniciais, de adoptar un formato co que non tiñamos traballado anteriormente.

Unha terceira cuestión derivada das dúas anteriores foi a de que linguaxe de programación utilizar. Spark é compatible coas linguaxes Scala, Java e Python. Cada unha delas constitúe un enfoque de programación diferente. Pensando na formación recibida ó longo da titulación, tamén cun enfoque de redución de riscos do proxecto en mente, decidimos adoptar Java como linguaxe de desenvolvemento pola nosa experiencia con dita linguaxe, asumindo as súas vantaxes e limitacións.

Iteración 1

A primeira iteración do proxecto consistiu en realizar un prototipo funcional do sistema para comprobar o seu correcto funcionamento sobre o *framework* de computación distribuída elixido e identificar cuestións de deseño concretas que se poderían mellorar en seguintes iteracións. Imos a dividir o desenvolvemento en dous módulos principais: un módulo para o ETL e outro para o procesamento. Como paso previo, cómpre definir cal vai a ser o formato mediante o cal o usuario defina as fontes de datos.

Dado que o desenvolvemento do software se vai a dirixir cara a implementación do cálculo do risco de incendio, que implica tanto un datos de partida concretos como unha secuencia de operacións concretas, temos decidido, para esta primeira iteración, gardar no sistema de almacenamento distribuído os identificadores necesarios en cada conxunto de datos para poder realizar as combinacións entre eles. Esta estratexia ten como desvantaxe o custo adicional tanto en termos de espazo en disco como en termos de tempo de escritura e lectura que supón o almacenamento de ditos identificadores, pero reduce e simplifica o *pipeline* de operacións necesarias para calcular o risco.

Formato de definición das fontes de datos

O rango de posibilidades para a definición de datos de entrada a un programa é amplo. Neste caso, requirimos un formato que estea estandarizado e que permita flexibilidade á hora de engadir novos tipos de fontes, ó mesmo tempo que sexa sinxelo de utilizar para o usuario e non presente problemas de compatibilidade á hora de extraer a información desde o programa. Considerando as características citadas, decidimos utilizar un ficheiro XML como formato de definición de datos. Dito ficheiro contén a seguinte estrutura de etiquetas:

```
<sources>
 <databases>
  <database name='name'>
   <access url="__" user="__" password="__" driver="__"/>
   <dimensions>
    <dimension name="__" type="__" sample="__" table="__" column="__"/>
    <dimension name="__" type="__" sample="__" resolution="__" start="__" end="__"/>
   </dimensions>
   <mappingsets>
    <mappingset name="__" table="__">
     <dimension ref="__" column="__"/>
     <mapping name="__" type="__" column="__"/>
     <mapping name="__" type="__" precision="__" resolution="__" column="__"/>
     <mapping name="__" type="__" precision="__" scale="__" column="__"/>
    </mappingset>
   </mappingsets>
  </database>
 </databases>
 <files>
  <file path="__" filetype="__">
```

```
    <dimensions>
      <dimension name="__" type="__" sample="__" resolution="__" start="__" end="__"/>
    </dimensions>
    <mappingsets>
      <mappingset>
        <dimension ref="__"/>
        <mapping type="__" precision="__" scale="__"/>
      </mappingset>
    </mappingsets>
  </files>
</sources>
<constants>
  <constant name="__" value="__"/>
</constants>
```

No primeiro nivel definimos as etiquetas "sources" e "constants", que permiten indicar fontes de datos e constantes, respectivamente. As constantes simplemente reciben un nome e un valor. Dentro das fontes, estas poden ser bases de datos ou ficheiros. Unha base de datos, etiqueta "database", recibe un nome, e ten uns parámetros de acceso, etiqueta "access", que son a url, o nome de usuario, o contrasinal e o driver a utilizar para facer a conexión. Os ficheiros, etiqueta "file", pola súa parte, veñen definidos por un path, que nos indica onde están, e un tipo para saber de que estándar se trata. Os datos que conteñen as fontes poden ser dimensións ou mapas de valores, como xa temos explicado no contexto. As dimensións teñen o seu propio nivel de definición, onde podemos indicar o seu nome, o seu tipo e se é de tipo mostraxe ou non. En caso afirmativo debemos indicar a resolución (tamaño real de cada punto espacial ou temporal), así como os valores inicial e final. Nas dimensións presentes en bases de datos cómpre indicar a táboa e a columna para acceder ós datos. Pero tamén poden formar parte de cubos de datos (*mappingsets*), e hai que indicalo explícitamente dentro da etiqueta correspondente, utilizando como referencia o nome da dimensión. Os cubos de datos teñen tamén un nome e, no caso de estar nunha base de datos, correspóndense cunha táboa, que hai que indicar. Os mapas de valores (etiqueta *mappings*) teñen tamén un nome e un tipo que, no caso de ser xeométrico ou de precisión fixa, require unha precisión (para indicar o número de cifras necesario para a representación) e unha resolución ou unha escala (para indicar o número de decimais almacenados), respectivamente. Isto é debido a que estes valores se van a almacenar como valores enteiros no sistema, para aumentar a súa eficiencia.

Análise do módulo ETL

Na fase previa de elección de tecnoloxías de implementación decidimos comezar utilizando como formato de almacenamento en HDFS os arquivos de tipo SequenceFile. Isto conleva certas restriccións á hora de deseñar o sistema, xa que os obxectos que se almacenan en ditos arquivos deben implementar a interface *Writable* da API de Hadoop. Dita interface proporciona un protocolo de serialización simple e eficiente para os obxectos que a implementan que será utilizado á hora de escribilos nos arquivos mencionados. Os datos importados desde as fontes definidas polo usuario deberán ser, en consecuencia, encapsulados en contedores que implementen a interface, e en

formato clave-valor (propio dos SequenceFile, como temos explicado). Posto que non se precisa o acceso aleatorio ós datos almacenados, senón que se van a cargar en memoria en lotes para o seu procesamento, a clave utilizada non ten importancia máis aló de ser unha esixencia do formato.

No contexto explicábamos que os RDDs son as coleccióńs de datos distribuídas coas que traballa Spark. Cada RDD pode conter obxectos da linguaxe concreta na que se implementa. Existen dous tipos de operación principais que se poden aplicar sobre os RDDs:

1. Transformacións.- Son operacións que devolven un novo RDD, como por exemplo a aplicación dun filtro sobre os datos.
2. Accións.- Son operacións que devolven un valor final ó programa principal (denominado *driver* en Spark) ou o escriben nun sistema de almacenamento externo, como por exemplo a conta do número de aparicións dun atributo no conxunto de datos.

A aplicación de avaliación tardía (*lazy evaluation*) implica que as transformacións sobre un RDD non se aplican ata que o programa detecta unha acción, e polo tanto redúcense o número de pasadas sobre os datos e o tempo de desenvolvemento do software, xa que é o propio *framework* quen se encarga de xerar unha árbore de execución de operacións eficiente.

Mediante a aplicación dunha transformación sobre un RDD que conteña os nosos datos de partida, podemos construír os contedores de datos tipo Writable que necesitamos para almacenalos en HDFS.

Outro aspecto a ter en conta no deseño son as estruturas de datos que imos a utilizar para almacenar os datos de tipo xeográfico. Temos que contemplar os principais obxectos xeométricos que se poden presentar nas fontes a utilizar, e que son os definidos no contexto: puntos 2D, cadeas de puntos, polígonos e multipolígonos. Posto que van a formar parte dos obxectos a almacenar en HDFS, ditas estruturas deben implementar tamén a interface Writable. Ademais, haberá que contemplar unha interface que permita parsear as diferentes representacións das xeometrías en cada fonte de datos aos tipos xeométricos primitivos do sistema.

Entre os requirimentos do sistema incluíase a posibilidade de importar datos tanto de fontes relacionais, xa sexan entidades convencionais ou entidades xeográficas presentes en bases de datos espaciais, como de fontes GIS estandarizadas que conteñan vectores de valores ou coberturas xeográficas. Isto tamén require do deseño dunha interface que posibilite o acceso aos diferentes tipos de fontes.

Por último, cómpre definir un catálogo do sistema no que se recollan os metadatos e características dos diferentes conxuntos de datos existentes tanto nas fontes como no almacenamento distribuído. Posto que se trata de grupos diverxentes, o deseño dun catálogo diferente para cada unha das dúas partes constitúe unha mellora ó desacoplar e reducir as dependencias entre cada unha.

Resumidamente, o proceso de lectura de datos comeza coa obtención de información do arquivo de configuración para cada dimensión, cubo de datos e constante para gardalas no catálogo de fontes. No caso das dimensións haberá que indicar tanto a súa orixe como se é unha mostraxe ou non. No caso dos cubos de datos haberá que indicar a súa orixe. Se a fonte dos datos é unha base de datos, haberá que indicar os parámetros de conexión, mentres que se se trata dun arquivo deberase indicar o seu tipo. Os datos procedentes das bases de datos poderanse importar en paralelo a través da propia API de Spark SQL, mentres que os datos de arquivos estandarizados haberá que parsealos (valéndonos de librerías externas) e paralelizalos. Como paso final haberá que parsear os datos a obxectos Writables e escribilos en HDFS con formato SequenceFile.

Descrición das fontes de datos

Para o caso de uso sobre o que imos desenvolver o sistema, é dicir, o cálculo do risco de incendios (neste caso para o territorio galego), imos a utilizar as seguintes fontes de datos:

- SXBD espacial:
  - Base de datos PostGIS coas seguintes táboas:
    - *Estacion(id, nombre, loc)*
    - *Observacion(estacion, fecha, temperatura, humedad, viento)*
    - *Municipio(codigo, nombre, poblacion, geo)*
    - *Especies(id, especies, geo)*
    - *ModCombustible(id, modelo, geo)*
  - Arquivo de configuración:
    - Definición de “Dimensions” e “Extensional MappingSets”
    - *Dimensions*
      - *IdEstacion*
      - *FechaObservacion (Sampling)*
      - *CodMun*
      - *idEspecie*
      - *idModelo*
    - *Extensional MappingSets*
      - *Estacion*
        - Dominio: *IdEstacion*
        - Mappings: *Nombre*, *Loc*
      - *Observacion*
        - Dominio: *IdEstacion, FechaObservacion*
        - Mappings: *Temperatura, Humedad, Viento*
      - *Especies*
        - Dominio: *IdEspecie*
        - Mappings: *Especies, Geo*
      - *ModCombustible*
        - Dominio: *IdModelo*
        - Mappings: *Modelo, Geo*
  - Arquivo GeoTiff de elevación:
    - Resolución a 200 metros para o territorio galego

        - Arquivo de configuración:
            - *Dimensions*
                - *Loc200m*
            - *Extensional Mappings*
                - Topo
                    - Dominio: *Loc200m*
                    - Mappings: *elevacion*
    - Constantes (a engadir no arquivo XML):
        - Distancia IDW.
        - Pesos para media ponderada.
        - Máximos e mínimos para normalización.

Deseño do módulo ETL

A continuación mostramos tanto a visión estática (diagrama de clases) como a visión dinámica (diagrama de secuencia) do módulo ETL desta primeira iteración.

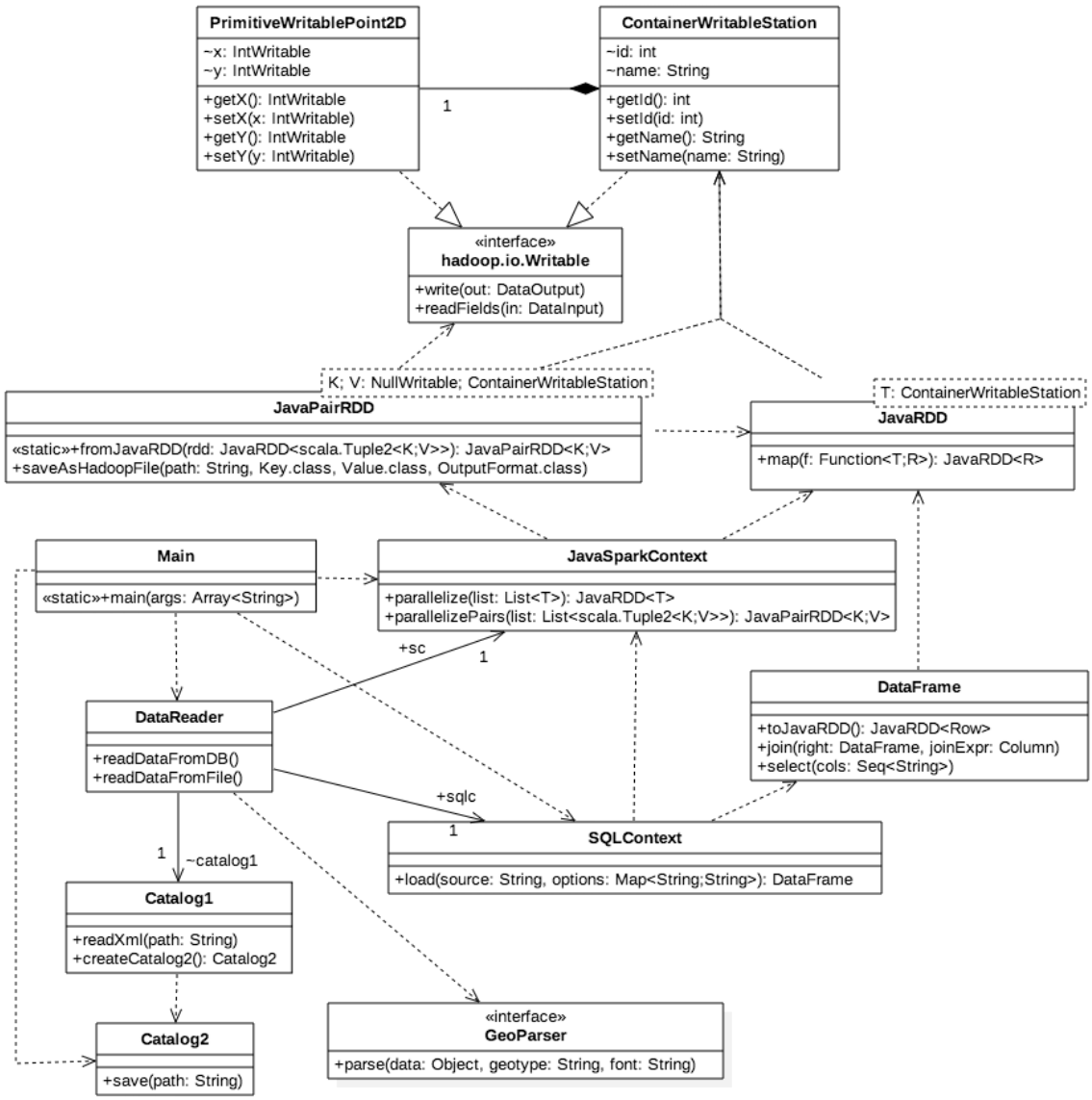


Figura 11.- Diagrama de clases do módulo ETL.

As principais clases que compoñen este módulo son as que se mostran no diagrama. En primeiro lugar, comezando pola parte superior, temos un exemplo de tipo primitivo do sistema (*PrimitiveWritablePoint2D*) e de clase contedor (*ContainerWritableStation*), neste caso dunha estación meteorolóxica, na cal a súa localización é un atributo almacenado como un punto de dúas dimensións. As clases de tipos xeométricos primitivos son xenéricas para calquera fonte de datos compatible coa interface de importación, mentres que as clases contedor son clases específicas do problema a resolver neste proxecto, que é o cálculo do risco de incendios. A utilización destas últimas non é necesaria pero constitúe unha axuda ó desenvolvemento. Ambas clases herdan da interface Writable de Hadoop para que os seus obxectos poidan ser almacenados en SequenceFiles.

Posto que eses contedores van a formar parte de RDDs (*JavaRDD* en Java, e *JavaPairRDD* para os RDDs de tipo clave-valor), existe unha dependencia entre os métodos destas últimas e ditas clases. No caso do JavaPairRDD, a interface Writable é utilizada polo método *saveAsHadoopFile()* para almacenar ditos obxectos en HDFS. Nótese que se utiliza como clave un obxecto do tipo *NullWritable*, que é unha clase especial da API Writable proporcionada para indicar que non existen valores válidos nas claves das tuplas de datos e polo tanto non é necesario almacenar esa parte cando se escriben en disco. Ambas clases de RDDs son parametrizables. O JavaSparkContext é o obxecto Java que contén os metadatos de execución do programa en Spark (o programa execútase como un traballo no *framework*), e debe ser instanciado polo programa principal ó seu comezo. Tamén permite paralelizar (a través da familia de métodos *parallelize()*) coleccións de obxectos desde o programa principal. O SQLContext, que tamén debe ser instanciado no método *main()*, contén neste caso os metadatos relacionados coa parte relacional do programa, recibindo o seu construtor o JavaSparkContext como parámetro. O SQLContext permite crear DataFrames a partir de fontes relacionais. A partir dos DataFrames que se xeran lendo datos dunha fonte relacional pódese crear un RDD usando o método *toJavaRDD()*.

A clase DataReader é a que se vai a utilizar como interface de lectura entre as fontes de datos e o programa principal. Para elo contén dous métodos, *readDataFromDB()* e *readDataFromFile()*, que permiten a lectura tanto desde bases de datos relacionais, usando o SQLContext, como desde arquivos GIS estandarizados, como NetCDF ou GeoTiff. Esta clase recibe a axuda da interface GeoParser, sobre a cal se poden definir parsers específicos para cada tipo de xeometría que nos poidamos atopar nas fontes de datos.

Por último, temos unha clase Catalog1 sobre a cal se define o catálogo referente ás fontes de datos, que contén información sobre as orixes de ditos datos e as súas características extraída do ficheiro de definición XML co método *readXML()*, e unha clase Catalog2 que contén a información sobre os datos a almacenar no sistema de almacenamento distribuído para o seu posterior procesamento. O segundo catálogo créase a partir da información que contén o primeiro e o destino dos datos en HDFS, mediante o método *createCatalog2()*.

Diagrama de secuencia

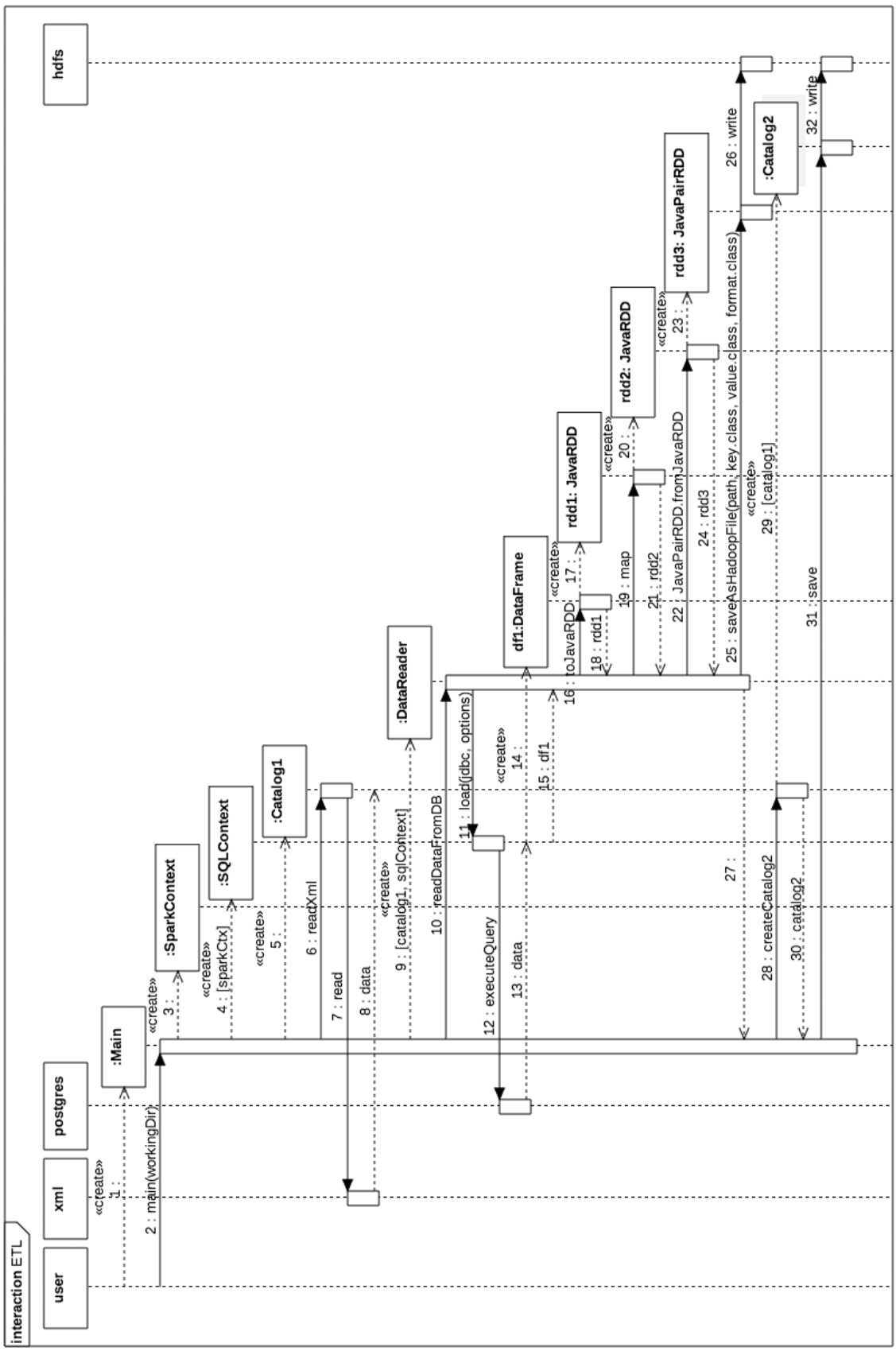


Figura 12.- Diagrama de secuencia para o módulo ETL.

Neste diagrama podemos observar a secuencia típica de execución do módulo ETL para o caso da lectura de datos en fontes relacionais. A lectura de ficheiros segue un esquema similar, pero facendo uso dos seus correspondentes métodos. En primeiro lugar, é o usuario quen executa o programa pasándolle como parámetro o directorio de traballo. O proceso principal representado polo método *main()* é o encargado de instanciar o SparkContext e logo o SQLContext, que recibe como parámetro o anterior. A continuación crea un obxecto de tipo Catalog1 e fai unha chamada ó seu método *readXml()* que se encargará de parsear a información que contén o ficheiro de definición de datos e almacenala nese obxecto. Posteriormente, desde o control principal créase o DataReader, que é o obxecto encargado de ler os datos das fontes de datos, pasándolle como parámetros o catálogo de fontes e o SQLContext, De seguido, a chamada ó seu método *readDataFromDB()* cede o control ó proceso de importación que comeza lendo os datos de fontes relacionais a partir da información que contén o catálogo. Para elo utiliza o SQLContext, cuxo método *load()* permite executar consultas contra bases de datos relacionais indicándolle unha serie de parámetros, e que devolve o resultado de dita consulta en forma de DataFrame. Para poder encapsular os datos importados nos contedores Writable definidos é necesario convertir o DataFrame nun RDD mediante unha chamada ó método *toJavaRDD()*. O método *map()* do RDD é o que nos vai a permitir aplicar unha función de transformación para extraer os datos do DataFrame e crear os novos obxectos, que a continuación deberán ser transformados á súa vez en tuplas (con clave nula), formando parte dun JavaPairRDD. Esta clase é a que contén o método *saveAsHadoopFile* que permite especificar como formato de saída o SequenceFile desexado. No caso de que os obxectos conteñan estruturas de tipo espazo-temporal cómpre parsear ditas estruturas, que xeralmente se extraerán da fonte de datos como un vector de bytes. Pode ser necesario o uso de librerías externas con soporte para datos de tipo xeográfico. Por último, cando o control regresa ó proceso principal, hai que crear o catálogo para as novas estruturas de datos, utilizando para elo o método *createCatalog2()* do catálogo de fontes, e gardalo no almacenamento distribuído co método do novo obxecto Catalog2 *save()*.

Probas unitarias sobre o módulo ETL

Debido a que os módulos software a desenvolver están destinados a executarse nun *framework* de computación distribuída, a realización de probas está supeditada ás ferramentas que poida proporcionar dito *framework* para monitorizar a súa execución, especialmente no referido ás probas de rendemento. Afortunadamente para nós, Apache Spark proporciónanos tanto unha interface web a través da cal podemos monitorizar a execución dos traballos como abundante información en modo texto no caso de realizar a execución por terminal.

Probas de funcionalidade

Realizouse un seguimento da execución do programa para comprobar que non existisen anomalías de funcionamento que imposibilitaran a súa correcta finalización. O resultado do programa son arquivos SequenceFile en HDFS cuxo contido se pode visualizar a través da ferramenta *Spark-Shell* que proporciona o propio *framework*. Esta é a forma de

comprobar que o resultado final da execución do proceso de ETL é correcto. Respecto a funcións específicas, deseñamos e executamos os seguintes casos de proba:

CP_IT1_ETL_01	
Método	Catalog1.readXML()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que o catálogo de fontes se crea correctamente a partir da información do XML.
Entradas requiridas	Ficheiro XML definido polo usuario.
Saídas esperadas	Catálogo de fontes ben formado.

CP_IT1_ETL_02	
Método	Catalog1.readXML()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que se lanza unha excepción cando o ficheiro XML está mal definido.
Entradas requiridas	Ficheiro XML definido polo usuario.
Saídas esperadas	Excepción indicando o tipo de erro.

CP_IT1_ETL_03	
Método	DataReader.readDataFromDB()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a lectura de datos en fontes relacionais se leva a cabo correctamente.
Entradas requiridas	Catálogo de fontes de datos.
Saídas esperadas	RDDs cos datos especificados no catálogo.

CP_IT1_ETL_04	
Método	DataReader.readDataFromDB()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a lectura de datos en fontes relacionais non se pode levar a cabo se os datos de acceso son incorrectos.
Entradas requiridas	Datos de acceso incorrectos no catálogo de fontes.
Saídas esperadas	Excepción indicando o tipo de erro.

CP_IT1_ETL_05	
Método	DataReader.readDataFromFile()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a lectura de datos en arquivos estandarizados se leva a cabo correctamente.
Entradas requiridas	Catálogo de fontes de datos.
Saídas esperadas	RDDs cos datos especificados no catálogo.

CP_IT1_ETL_06

Método	DataReader.readDataFromFile()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a lectura de datos en arquivos estandarizados non se pode levar a cabo se os datos de acceso son incorrectos.
Entradas requiridas	Datos de ficheiro incorrectos no catálogo de fontes.
Saídas esperadas	Excepción indicando o tipo de erro.

Probas de rendemento

Para avaliar o rendemento intentamos identificar problemas concretos en cada fase da execución que puidesen revelar a existencia de ineficiencias na estratexia de programación ou no tipo de formatos utilizados para almacenar os datos. Para elo servímonos da interface web de Spark, onde podemos comprobar diversos parámetros de execución de cada etapa do traballo e polo tanto identificar problemas potenciais asociados a esa fase. En concreto, observamos que a fase de transformación de DataFrames a RDDs e o posterior parseo a obxectos de tipo Writable, de forma xerárquica desde os tipos de datos básicos de cada obxecto, implicaba un custo adicional considerable en tempo de execución así como en consumo de memoria pola necesidade de crear os novos obxectos. Este problema repetíase na fase de creación das tuplas clave-valor necesarias para poder gardar os datos en formato SequenceFile.

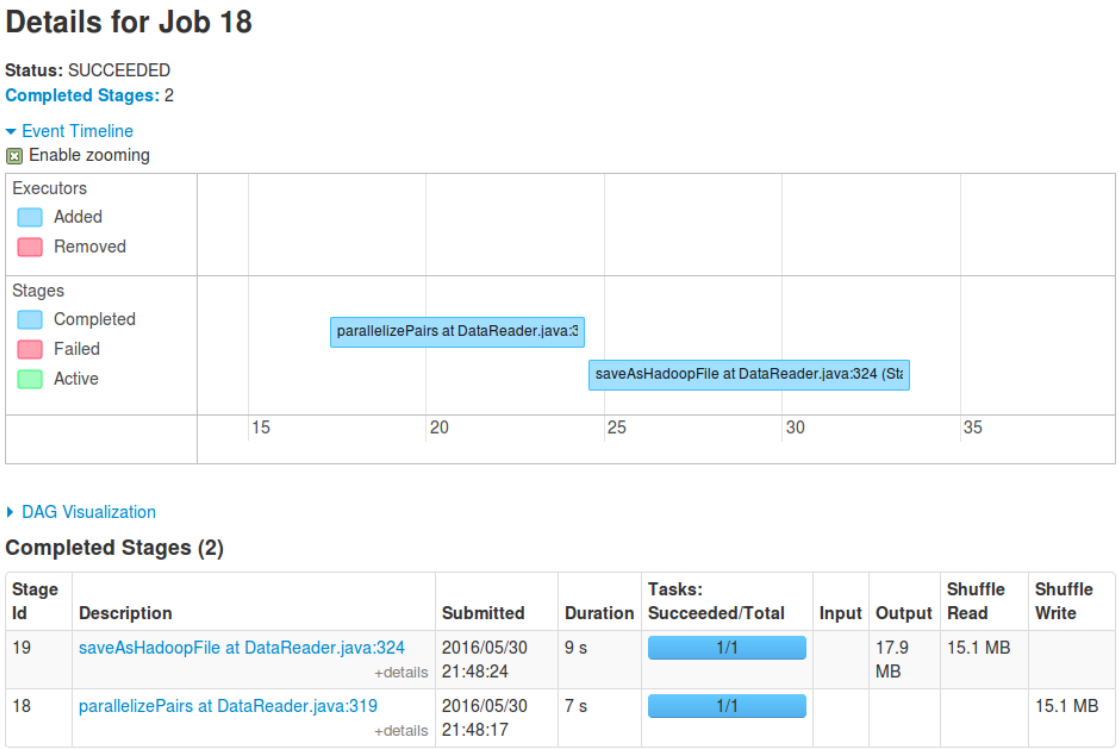


Figura 13.- Captura da Spark Web UI con información referente á escritura dun SequenceFile.

Mediante a monitorización das tarefas de creación dos obxectos Writable e escritura en disco (como se mostra na figura anterior), puidemos comprobar que o tempo dedicado a ditas tarefas predominaba sobre a lectura dos datos a partir das fontes e incorría en ineficiencias probablemente prescindibles.

Análise do módulo de procesamento

Os arquivos SequenceFile exportados co módulo ETL conteñen os datos sobre os que imos a realizar o procesamento, e están almacenados en HDFS de forma distribuída e replicada, polo cal a implementación da lectura dos datos vai a ser dependente de ditos formatos. Respecto á parte de tratamento deses datos, para o caso de uso do cálculo de risco de incendios o procesamento a levar a cabo implica a realización de operacións típicas de transformación, unión ou agregación de conxuntos de datos. Os propios métodos tanto da clase JavaRDD como da clase DataFrame implementan algunhas destas operacións e a maiores dan a posibilidade ó programador de definir as súas funcións específicas para aquelas operacións que non estean contempladas. Logo, para importar os datos en memoria haberá que utilizar a interface correspondente que permita acceder ó formato SequenceFile, e despois facer as transformacións que sexan necesarias para aplicar os operadores que permitan calcular o risco de incendio. Finalmente, os valores de risco que se obteñen como resultado deben ser almacenados no sistema de almacenamento distribuído, podendo utilizar para elo o formato de escritura por defecto que proporcione a API.

Explicación do cálculo do risco de incendios

A secuencia de operacións para calcular o risco de incendios sobre os datos de partida propostos implica unha serie de lecturas, transformacións, combinacións, agregacións e cálculos que resumimos brevemente a continuación:

1. Lectura dos datos de estacións.
2. Lectura dos datos das observacións.
3. Unión entre estacións e observacións utilizando o identificador de estación.
4. Lectura de datos de elevación.
5. Cálculo da pendente en cada píxel a partir da elevación. Imos a seguir o enfoque proposto por Horn [14]: denotando *elevation*(p) a elevación na localización p e denotando *pnw, pn, pne, pw, pe, psw, ps* e *pse* as localizacións veciñas correspondentes ó p noroeste, norte, nordeste, oeste, este, suroeste, sur e surdeste, entón a pendente no punto p denotada por *slope*(p) obtense coa seguinte fórmula:

$$Xcomp(\mathrm{p}) = 1 * elevation(\mathrm{pnw}) + 2 * elevation(\mathrm{pw}) + 1 * elevation(\mathrm{sw}) - 1 * elevation(\mathrm{pne}) - 2 * elevation(\mathrm{pe}) - 1 * elevation(\mathrm{se})$$

$$Ycomp(\mathrm{p}) = 1 * elevation(\mathrm{psw}) + 2 * elevation(\mathrm{ps}) + 1 * elevation(\mathrm{se}) - 1 * elevation(\mathrm{pnw}) - 2 * elevation(\mathrm{pn}) - 1 * elevation(\mathrm{ne})$$

$$slope(\mathrm{p}) = atan1(\sqrt{Xcomp(p)^2 + Ycomp(p)^2})$$

Fórmula 1.- Cálculo da pendente.

6. Unión por distancia dos datos de pendente co conxunto de datos obtido en (3).
7. Agrupación de (6) por localización, data e pendente, e cálculo da interpolación IDW para a temperatura, a humidade e o vento. O método IDW (*Inverse Distance Weighted*) é un método de interpolación espacial que que asigna pesos ós datos do entorno dun punto en función inversa da distancia que os separa. Por exemplo, se para un instante específico denotamos *temperature*(p) como a interpolación da temperatura en cada localización ou píxel, o cálculo realízase a partir das temperaturas observadas en cada estación s_i, denotada *temperature*(s_i), coa seguinte fórmula:

$$temperature(\mathrm{p}) = \begin{cases} \dfrac{\sum_i \frac{temperature(s_i)}{distance(p,s_i)^2}}{\sum_i \frac{1}{distance(p,s_i)^2}}, & if\ 0 < distance(p,s_i) < d \\ temperature(s_i), & if\ distance(p,s_i) = 0 \end{cases}$$

Fórmula 2.- Interpolación dos valores de observación.

8. Lectura de datos dos modelos de combustible.
9. Unión, por intersección xeométrica, entre os modelos e (7).
10. Agrupación por localización, pendente, data, e valores interpolados das observacións, seleccionando o mínimo modelo de combustible.
11. Normalización e agregación dos parámetros. En cada localización p, os valores de temperatura, humidade, velocidade do vento e modelo de combustible deben ser normalizados a valores no intervalo [0, 1].
12. Por último, o risco de incendio en cada localización p calcúlase como unha media aritmética ponderada dos compoñentes normalizados. Para a ponderación, hai que ter en conta que o risco increméntase coa pendente, a temperatura, a velocidade do vento e o modelo de combustible, mentres que diminúe coa humidade.

Deseño do módulo de procesamento

A continuación mostramos tanto a visión estática (diagrama de clases) como a visión dinámica (diagrama de secuencia) do módulo de procesamento desta primeira iteración.

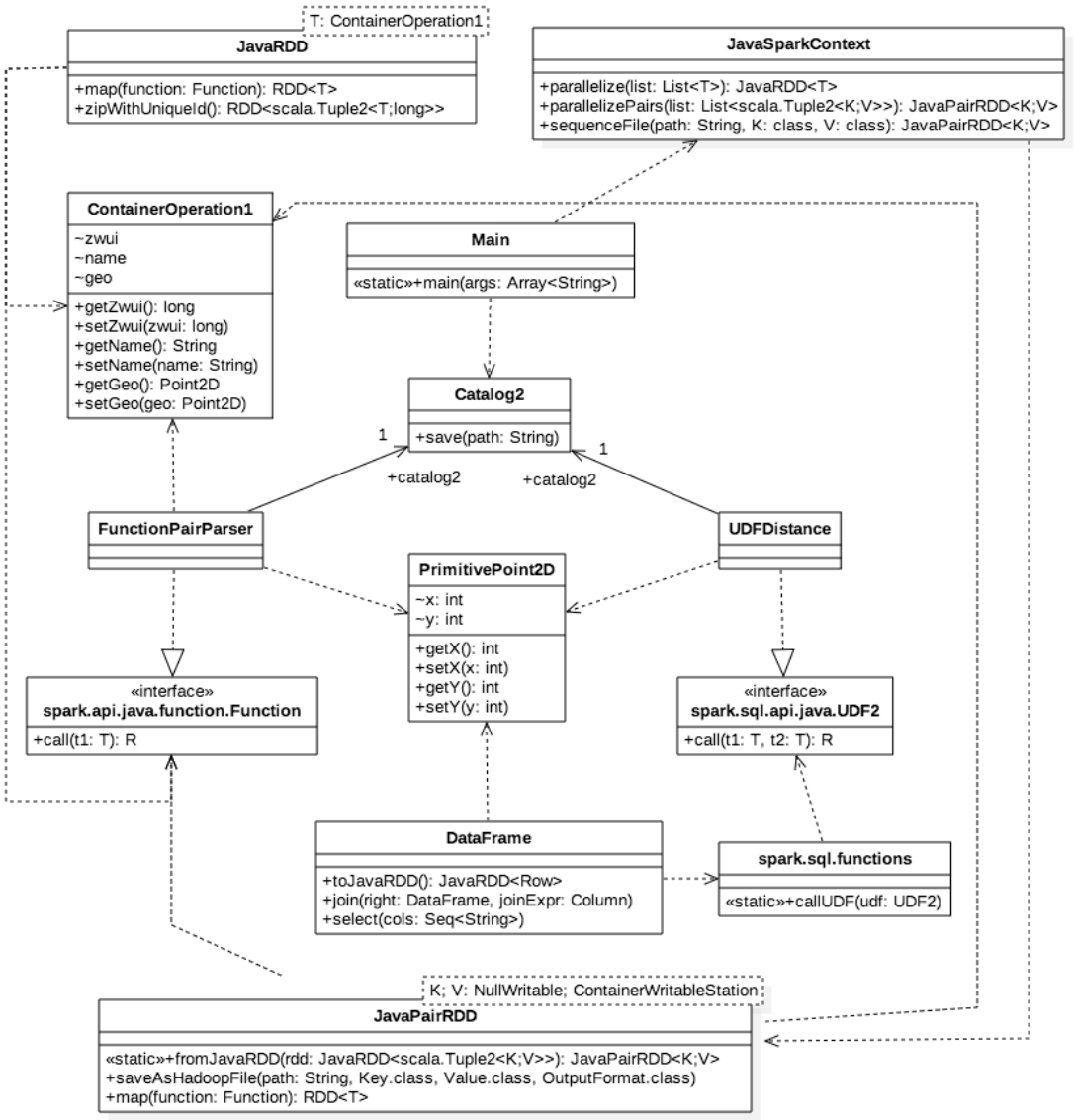


Figura 14.- Diagrama de clases para o módulo OLAP.

A clase principal do programa constitúese coma sempre no punto central ó redor do cal vai a xirar a creación dos obxectos necesarios. Imos a reutilizar a clase Catalog2 para poder instanciar o catálogo do sistema que temos almacenado en HDFS. Este catálogo pode ser utilizado dentro das funcións propias definidas polo programador. Un exemplo de función para aplicar sobre un RDD é a FunctionPairParser, que se encargaría de parsear as tuplas clave-valor importadas desde un SequenceFile a un JavaRDD de contedores, enviando unha instancia de dita función como parámetro do método *map()* aplicado sobre o JavaPairRDD inicial (que se pode xerar en memoria lendo o SequenceFile co método *sequenceFile()* do JavaSparkContext). Neste caso os contedores son obxectos destinados a almacenar datos en memoria e polo tanto non é necesario que implementen a interface Writable. As funcións definidas polo usuario deben implementar o método *call()* da interface Function da API de Spark. No caso de ser funcións a executar sobre DataFrames, existe outra interface diferente na API

denominada UDF. En función dos parámetros que recibe cada función, existen interfaces específicas: UDF1, UDF2, UDF3, ..., e Function, Function2, Function3, ... O exemplo de UDF que se mostra é unha función que calcular a distancia entre dous puntos que forman parte dos atributos de dous DataFrames diferentes. As funcións definidas van a traballar en moitos casos con datos primitivos do sistema, coma o Point2D (cuxa clase xa non necesita implementar Writable tampouco). Por último, utilizaremos métodos específicos da clase JavaRDD para operacións específicas, como por exemplo a xeración dun índice autoincremental usando *zipWithUniqueId()*.

Diagrama de secuencia

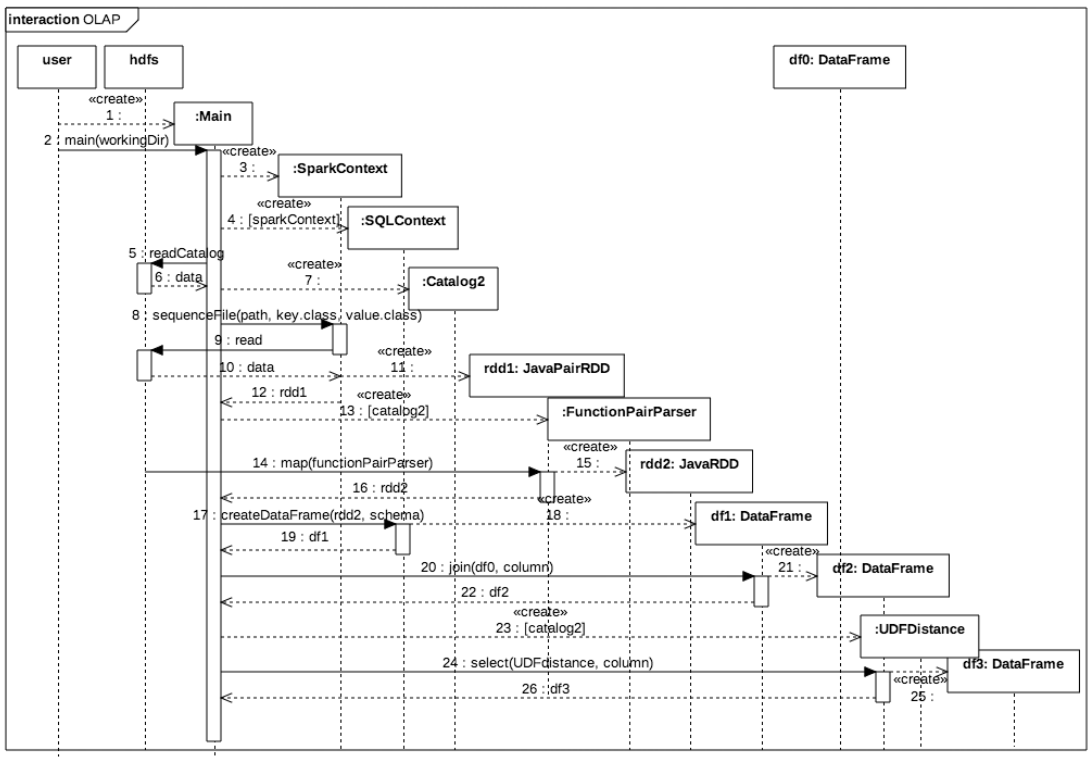


Figura 15.- Diagrama de secuencia para o módulo OLAP.

No diagrama de secuencia podemos observar a orde das accións a executar para a lectura dun determinado conxunto de datos e o seu posterior procesamento. Unha vez creados o contextos e lido o catálogo desde o método *main()*, realízase unha chamada ó metodo *sequenceFile()* do JavaSparkContext indicándolle como parámetros a ruta do arquivo a ler en HDFS e as clases que actúan como clave e como valor. Este método devolve un JavaPairRDD cos obxectos Writable importados. Para realizar o parseo debemos que aplicar a función que temos definido como FunctionPairParser, a través do método *map()* do PairRDD, que vai a devolver un novo RDD de contedores co nome e a xeometría (en datos primitivos do sistema) das estacións. No seguinte paso podemos aplicar operacións concretas que sexan necesarias en cada conxunto de datos. Despois de facer as transformacións sobre os RDDs necesitamos convertelos en DataFrames para poder aplicar operadores relacionais específicos como a unión por atributos (método

join()) ou funcións propias. Para crear un DataFrame a partir dun JavaRDD utilizamos a mesma estratexia ca no módulo de ETL, facendo unha chamada ó método *createDataFrame()* do SQLContext que recibe como argumentos o RDD de partida e o esquema de datos a aplicar.

Probas unitarias sobre o módulo de procesamento

Probas de funcionalidade

O obxectivo destas probas foi en primeiro lugar a comprobación do correcto funcionamento do módulo, valéndonos da ferramenta *Spark-Shell* para visualizar o contido do arquivo SequenceFile resultado cos índices de risco. Respecto ás funcionalidades concretas, definimos e executamos os seguintes casos de proba:

CP_IT1_OLAP_01	
Método	FunctionPairParser.call()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que os parseo dos pares clave-valor dos datos de SequenceFile se leva a cabo correctamente.
Entradas requiridas	JavaPairRDD de pares clave-valor a partir dun SequenceFile.
Saídas esperadas	JavaRDD de contedores de datos.

CP_IT1_OLAP_02	
Método	FunctionPairParser.call()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que se evita o parseo de arquivos SequenceFile mal formados.
Entradas requiridas	JavaPairRDD de pares clave-valor a partir dun SequenceFile con datos mal formados.
Saídas esperadas	Excepción indicando o tipo de erro.

CP_IT1_OLAP_03	
Método	UDFDistance.call()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que o filtrado de datos nun DataFrame aplicando unha función de distancia se leva a cabo correctamente.
Entradas requiridas	Point2D de orixe e DataFrame cunha columna tipo Point2D.
Saídas esperadas	DataFrame coas filas que cumpren a condición de distancia avaliada.

CP_IT1_OLAP_04	
Método	UDFDistance.call()
Técnica	Conxectura de erros

Descrición	Trátase de comprobar que non se pode aplicar a función de distancia a columnas de tipo diferente a Point2D.
Entradas requiridas	Point2D de orixe e DataFrame cunha columna dun tipo diferente a Point2D.
Saídas esperadas	Excepción indicando o tipo de erro.

Os defectos atopados fóronse corrixindo en base ó resultado das probas, e no caso de erros específicos, sobre a marcha.

Probas de rendemento

Respecto ó rendemento fixemos un intento de identificar posibles puntos débiles na implementación que puidesen ter efectos indesexables nos tempos de execución e polo tanto debían ser mellorados. Para elo fixemos uso, de novo, da interface web de monitorización provida por Spark que permite realizar o seguimento de cada etapa específica de execución.

Neste punto cabe explicar a forma en que Spark executa os seus traballos. Isto significa adentrarse brevemente no deseño interno do *framework*. Cando executamos un traballo, Spark utiliza a representación lóxica de operacións sobre os RDDs derivada do código programado para crear un plan de execución físico que introduce melloras de eficiencia agrupando múltiples operacións en *tasks* (tarefas). Este plan de execución ten a forma dun grafo acíclico dirixido (DAG), que é o elemento que permite ó *framework* tomar a decisión de cando se deben executar as transformacións definidas sobre os datos. Trátase, noutras palabras, dun mapa da ascendencia de cada RDD que permite planificar a orde de execución máis adecuada, e para elo introdúcese aínda outro nivel de abstracción agrupando as tarefas en *stages* (etapas). Unha etapa física executa tarefas que levan a cabo operacións similares pero en particións específicas de datos. En xeral, o *pipeline* de operacións que ocorren nunha determinada tarefa terá tres pasos principais: obtención dos datos de entrada, realización das operacións de transformación e envío dos datos á saída que corresponda (pode ser outro RDD ou un *shuffle*, que é unha operación de intercambio de datos entre nodos do clúster).

Polo tanto, mediante a Spark Web UI podemos obter información de progreso e de múltiples métricas que concernen a cada etapa e a cada tarefa dentro dela. Esta capacidade foi a que nos permitiu volver a observar que a etapa de lectura e parseo dos obxectos Writable almacenados en SequenceFile supoñía un custo adicional considerable tanto en tempo de execución como en consumo de memoria (por instanciación de novos obxectos), ó igual ca no caso do módulo ETL para a escritura. Para elo levamos a cabo a implementación dun subconxunto do *pipeline* de operacións necesarias para calcular o risco de incendio, en concreto, as referentes á importación, transformación, unión e agregación dos datos de estacións, observacións e elevación. Doutra banda, a posibilidade de definir funcións propias tanto sobre DataFrames como sobre RDDs puxo en cuestión a necesidade de realizar transformacións continuas entre ambos tipos de conxuntos, motivando a busca dunha solución máis optimizada que se levaría a cabo na seguinte iteración (en cuxo apartado de probas se mostra unha comparativa de rendemento respecto a esta versión).

Iteración 2

Na segunda iteración do proxecto establecemos como obxectivo principal a implementación dunha versión mellorada do prototipo inicial. Facendo fronte ás ineficiencias detectadas nel, o equipo propúxose a utilización dalgún outro formato de almacenamento para os datos que supuxese unha mellora substancial na eficiencia da lectura, da escritura e do espazo de almacenamento, e que á súa vez permitise reducir o número de operacións de parseo necesarias para importar e exportar os datos tanto no módulo ETL como no módulo de procesamento.

Análise das melloras

As características que nun principio fixeron atractivo o uso do formato SequenceFile para a implementación do prototipo inicial víronse eclipsadas trala constatación do seu rendemento na primeira iteración. Púxose de manifesto a necesidade dun cambio de formato, polo que o equipo decidiu levar a cabo unha análise de posibles alternativas partindo das posibilidades descritas na táboa 7 (véxase "Tecnoloxías escollidas"). Isto supuxo a aparición dun risco derivado da necesidade dun cambio nos requirimentos que motivou un breve proceso de reflexión e avaliación das posibles implicacións. Finalmente, xurdiu como proposta a utilización do formato Parquet. Pese a ser un formato co que non tiñamos experiencia previa, o estudo detallado das súas características revelou a súa potencialidade de aplicación no noso proxecto.

Apache Parquet é, como se indica na páxina do seu proxecto [17], un "*formato de almacenamento columnar dispoñible para calqueira proxecto do ecosistema Hadoop, independentemente da elección do framework de procesamento, modelo de datos ou linguaxe de programación"*. Entre as súas características destacan, ademais do almacenamento columnar, a posibilidade de almacenar estruturas anidadas complexas e a utilización de métodos de codificación e compresión avanzados. Ademais, a súa integración con Spark vai máis aló dos aspectos de compatibilidade xa que é o formato utilizado por defecto para almacenar os DataFrames en disco. A adecuación de Parquet para o proxecto fundaméntase, por unha parte, na posibilidade de almacenar de forma eficiente os elementos complexos que forman parte dos datos cos que traballa o noso sistema, como por exemplo os tipos xeométricos polígonos ou multipolígonos, e por outra parte no aumento da eficiencia que se deriva do uso dun formato columnar de almacenamento, xa que o tipo de operacións que levamos a cabo implica frecuentemente o acceso aos datos por algunha columna específica, e cómpre evitar o sobrecusto da carga en memoria de columnas innecesarias.

Posto que a escritura de DataFrames en disco con formato Parquet non require de parseos adicionais, desaparece a necesidade de implementar contedores tipo Writable para encapsular os nosos datos. Aínda máis, a lectura de datos desde fontes relacionais a través do SQLContext xera directamente DataFrames, polo cal resulta moito máis sinxelo facer transformacións sobre os datos antes de almacenalos en HDFS. Os

DataFrames conteñen obxectos de tipo *Row* (filas) que encapsulan vectores de tipos básicos (enteiros, cadeas, ...). Dado que a implementación a baixo nivel consiste nun RDD ó que se lle engade un esquema de metadatos cos tipos que definen cada columna, podemos levar a cabo consultas sobre os datos con sentencias SQL.

Deseño das melloras

Unha vez tomada a decisión de utilizar Parquet como formato de almacenamento en disco e DataFrames como estruturas de datos en memoria levamos a cabo tanto o deseño do novo módulo ETL reutilizando o prototipo inicial pero modificando a parte específica da transformación e escritura en disco dos datos, como o deseño do módulo de procesamento cuxa característica principal será o uso exclusivo de DataFrames para a computación do risco.

Diagramas de clases

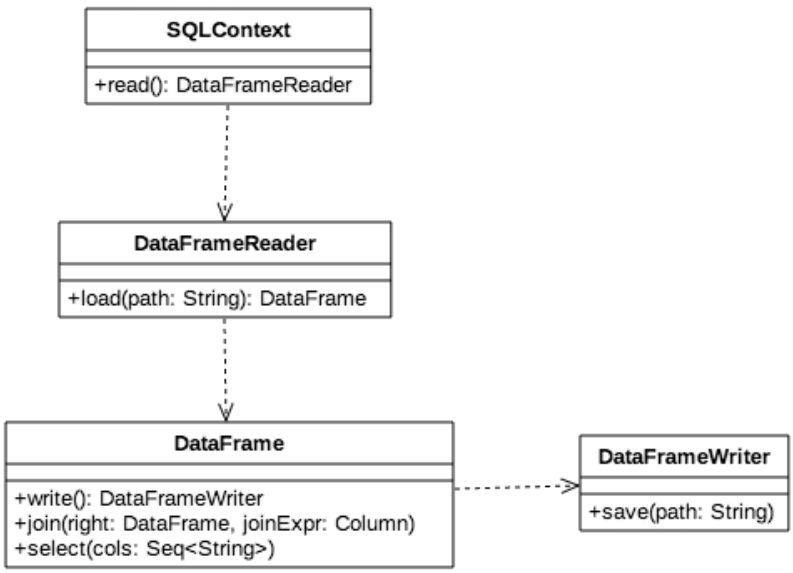


Figura 16.- Diagrama coas novas clases do módulo ETL.

O diagrama de clases é similar ó mostrado na figura 11 (véxase "Iteración 1"), salvo que desaparecen as clases relativas ós RDDs e á utilización de contedores Writable, xa que o uso desta interface era un requirimento específico do formato SequenceFile. En concreto, aparecen en contexto dúas novas clases da API de Spark, DataFrameWriter e DataFrameReader, que nos van a permitir tanto a escritura en disco dos DataFrames como a súa posterior lectura para procesalos. O DataFrameWriter é o parámetro de retorno do método *write()* da clase DataFrame que imos a utilizar para gardalo en disco, mentres que o DataFrameReader é o parámetro de retorno do método *read()* da clase

SQLContext, que permite ler un arquivo de disco en formato Parquet ou similares e vai a ser utilizado para cargar en memoria os datos a procesar desde HDFS.

Outra cousa que desaparece é a necesidade de definir os datos xeométricos primitivos como implementacións da clase Writable, pola mesma razón que os contedores. Logo o paquete de datos xeométricos que imos a utilizar no sistema vai a ser o mesmo tanto no módulo de ETL como no módulo de procesamento. Para o noso caso de uso, a xerarquía de clases deste tipo de datos queda como se reflexa no seguinte diagrama:

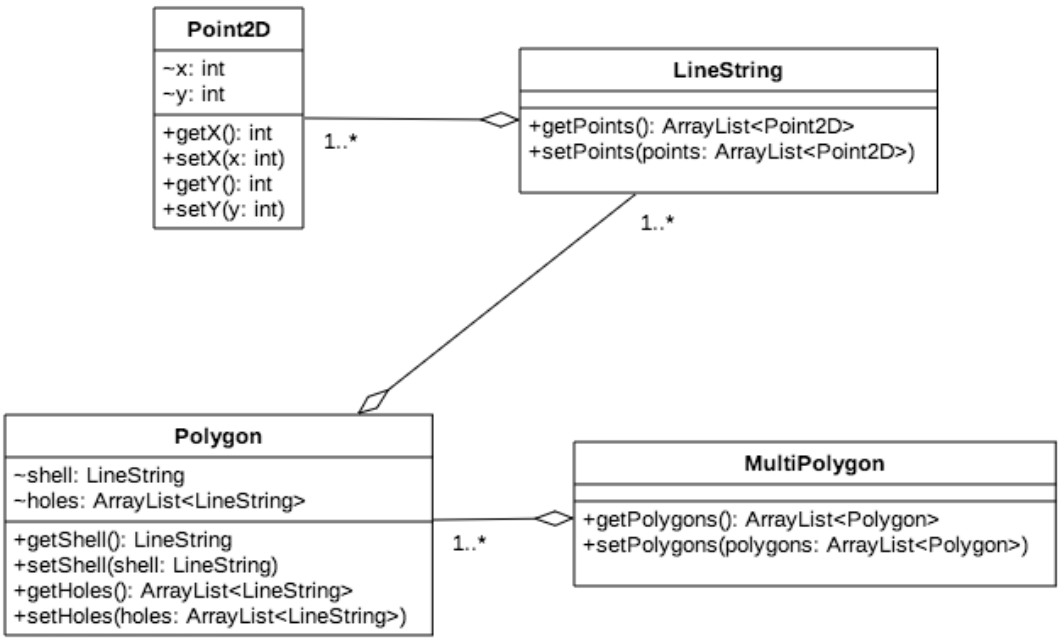


Figura 17.- Diagrama coas clases de tipos xeométricos primitivos.

Diagramas de secuencia

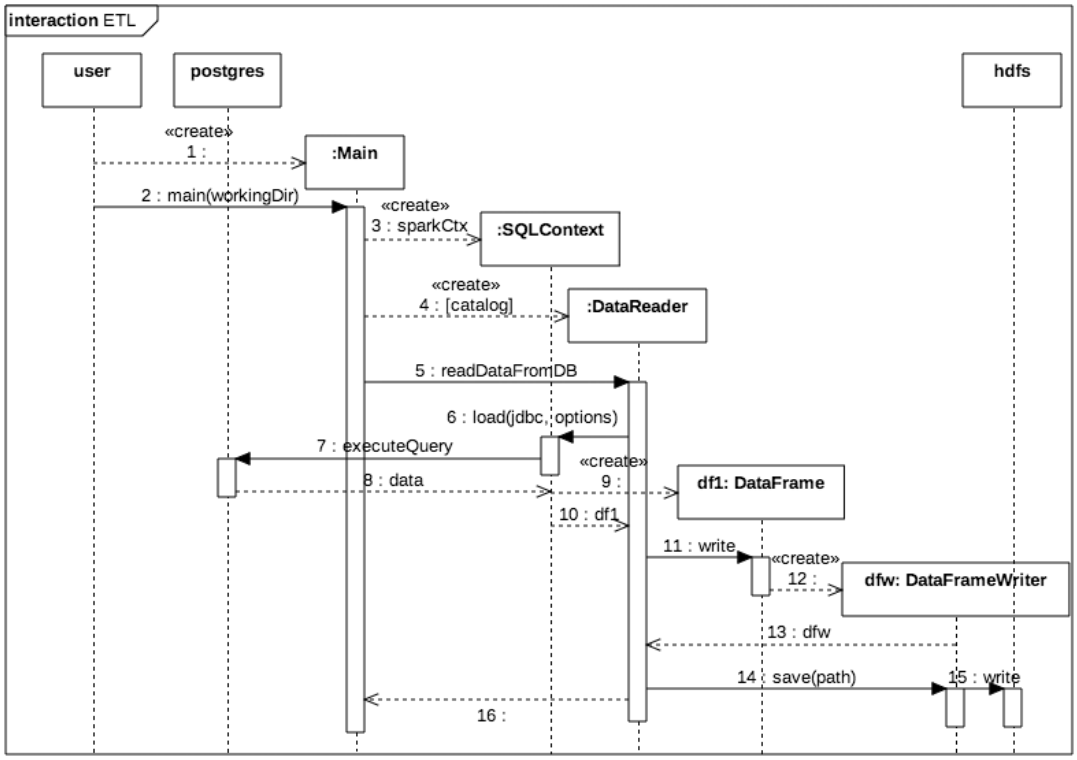


Figura 18.- Diagrama de secuencia para a importación de datos procedentes dunha base de datos relacional.

No diagrama de secuencia podemos observar claramente o aforro en termos de número de accións necesarias para exportar os datos a HDFS. No exemplo da figura 18, que representa a importación dun conxunto de datos similar ó mostrado na figura 12 (véxase "Iteración 1"), tras ler os datos de Postgres co SQLContext a través do DataReader obtemos un DataFrame que pode ou non ser sometido a transformacións, pero que non necesita ser parseado para ser almacenado en disco con formato Parquet.

Respecto ó modulo de procesamento, o *pipeline* de operacións simplifícase con respecto ó mostrado na figura 15 (véxase "Iteración 1"). Agora, como temos mencionado, o método *main()* realiza unha chamada ó metodo *read()* do SQLContext para importar os datos almacenados en formato Parquet en HDFS. Internamente, dito método crea un obxecto de tipo DataFrameReader para levar a cabo a lectura. Posteriormente podemos levar a cabo operacións tanto de tipo relacional, como por exemplo combinacións ou agregacións, como outro tipo de algoritmos definidos polo programador en funcións propias, directamente sobre os DataFrames desde a entrada dos datos ata a obtención do resultado do cálculo do risco, sen necesidade de transformacións adicionais en RDDs. Isto mellora a eficiencia do código e a rapidez coa que se executan as transformacións e os cálculos. Podémolo apreciar na seguinte secuencia:

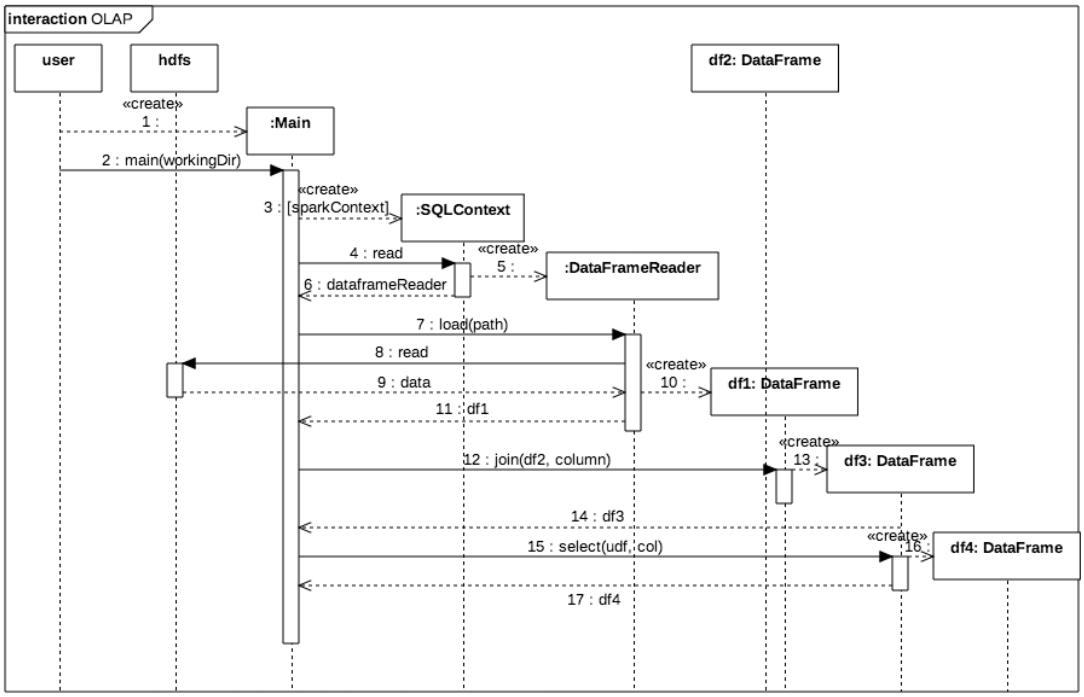


Figura 19.- Diagrama de secuencia para a unión entre dous conxuntos de datos obtidos de HDFS.

Comparativa entre formatos de almacenamento

Para comprobar se a decisión de cambiar o formato de arquivo no que almacenamos os datos en HDFS foi acertada, decidimos levar a cabo unhas breves probas co obxectivo de apreciar as diferencias entre ambos. En primeiro lugar, fixemos medicións acerca do tamaño que ocupa en disco cada tipo de arquivo para un mesmo conxunto de datos:

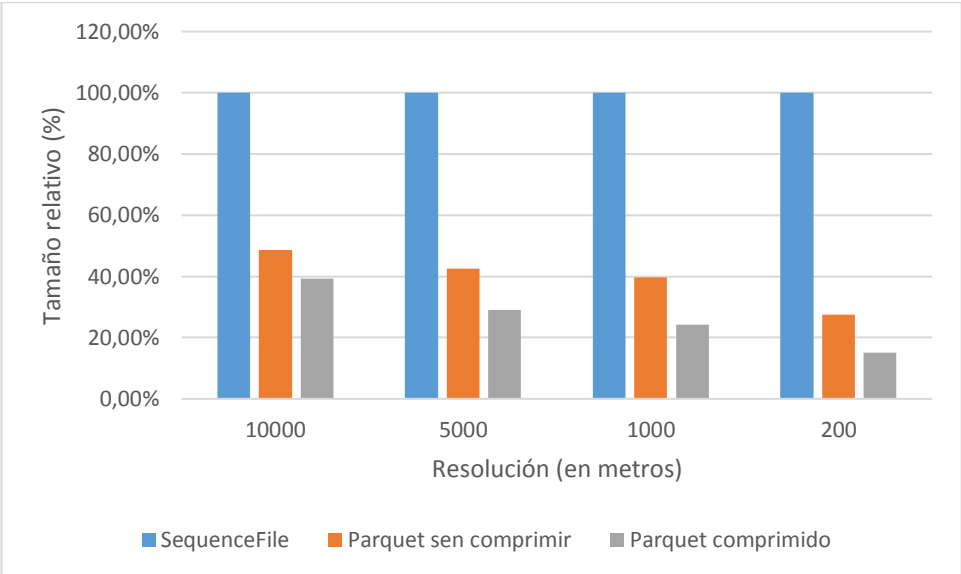


Gráfico 1.- Tamaños do arquivo de datos de elevación.

Estes datos xeráronse sobre un arquivo GeoTiff con datos de elevación no territorio galego para diferentes resolucións: 200, 1000, 5000 e 10000 metros. Para poder graficar correctamente as medidas transformamos os valores absolutos en porcentaxes relativas sobre o tamaño do arquivo SequenceFile para cada resolución. O que podemos apreciar no gráfico anterior é que para calquera resolución o formato Parquet, grazas á maior eficiencia do almacenamento columnar, ocupa en disco menos do 50% do que ocupa un SequenceFile coa codificación por defecto. Aplicando compresión en Parquet, obtéñense aínda mellores resultados. Doutra banda, a medida que medra a resolución e polo tanto o tamaño dos datos de partida, as diferencias entre ambos formatos fanse máis pronunciadas.

En segundo lugar, queríamos comprobar se existía unha mellora nos tempos de execución debido á menor carga de operacións derivada da ausencia da fase de parseo de obxectos Writable a DataFrames. Para comprobalo, executamos dúas versións modificadas do módulo de procesamento, unha que importa os datos de SequenceFile en RDDs e outra que importa os datos de Parquet en DataFrames, para levar a cabo a combinación entre os valores da pendente do terreo en cada píxel cos valores interpolados das observacións. As probas executáronse, neste caso, sobre Spark en modo *standalone* (é dicir, nunha máquina local e con dous núcleos dispoñibles para simular o paralelismo).

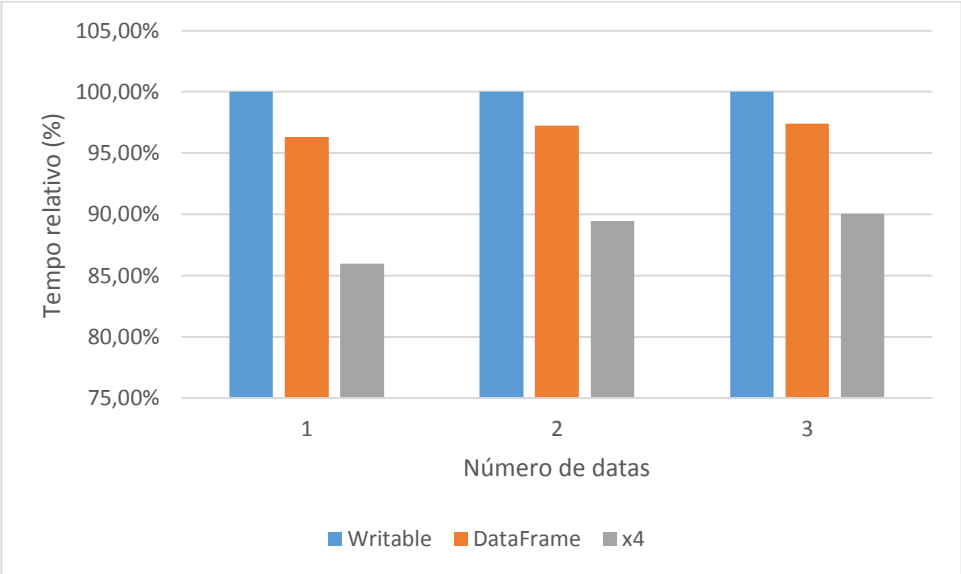


Gráfico 2.- Descensos dos tempos de lectura para cada formato.

Os resultados mostran que na versión con Parquet e DataFrames o tempo de lectura descende entre un 3 e un 4% con respecto á versión de SequenceFile e Writables. Agora ben, se extrapolamos estes valores a un *pipeline* coma o noso no que se se levan a cabo varias lecturas de arquivos (no gráfico anterior poñemos un exemplo con catro lecturas), a redución do tempo de execución acumúlase, chegando neste caso a reducirse entre un 10 e un 15%.

Probas de rendemento sobre a execución completa

Unha vez que tivemos totalmente implementada a versión completa do módulo de procesamento utilizando Parquet e DataFrames, puidemos levar a cabo medidas sobre o tempo de execución de todo o *pipeline* de operacións necesarias para calcular o risco de incendios. Neste caso, tiñamos dous obxectivos principais:

- Comprobar se para o noso caso de uso, coas súas operacións concretas, se producía escalamento ó executar o módulo no clúster.
- Detectar posibles pescozos de botella en puntos concretos do *pipeline* de operacións.

Respecto do primeiro punto, utilizamos datos de elevación con resolucións de 10 e 5 kilómetros para calcular o risco e tomar as medidas. Os executores (nodos en Spark) utilizados dispoñían cada un deles de 2GB de memoria RAM e un núcleo de procesamento. As seguintes gráficas mostran os resultados obtidos:

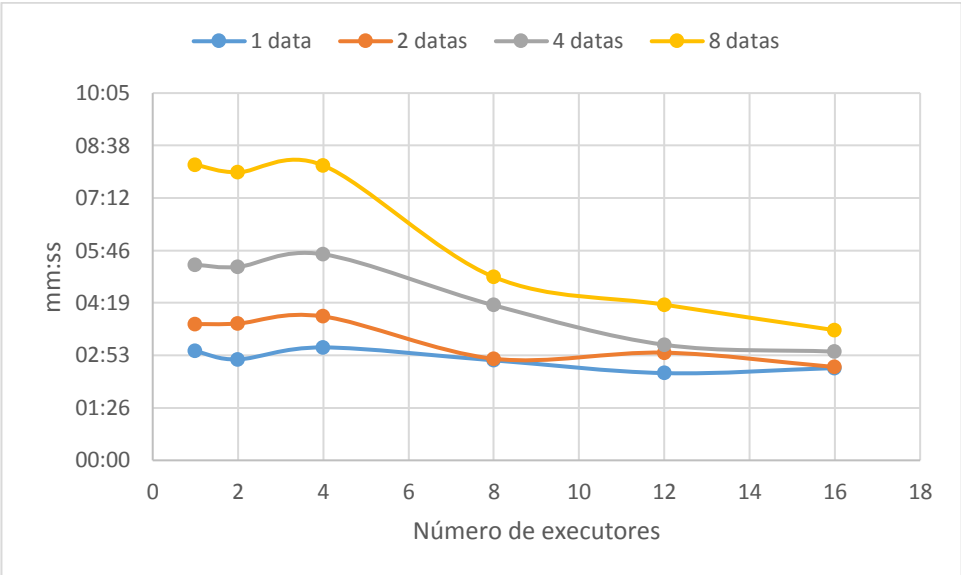


Gráfico 3.- Tempo de execución para 10km de resolución.

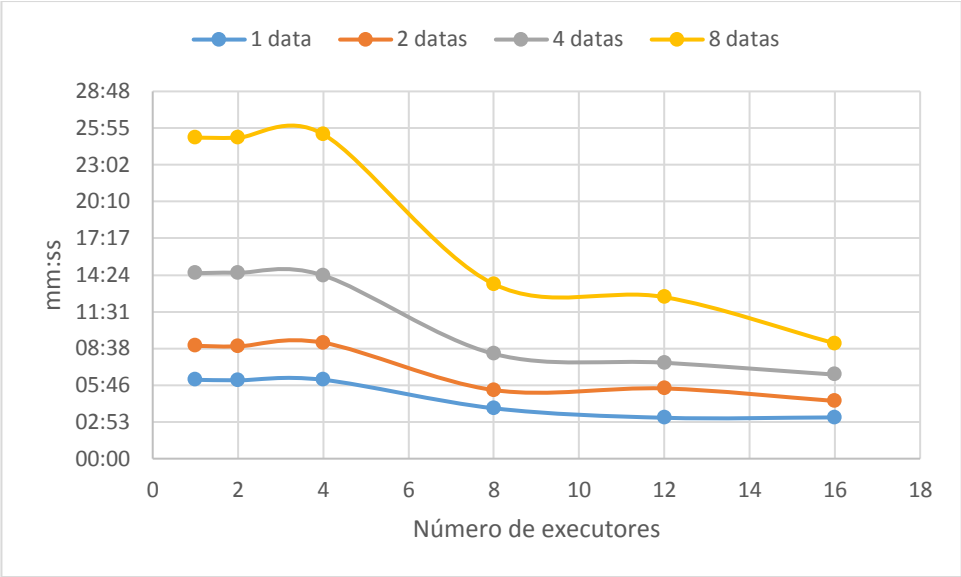


Gráfico 4.- Tempo de execución para 5km de resolución.

En ambos casos podemos observar que existe escalamento entre catro e oito executores (aínda que no caso dunha e dúas datas a diferenza é menor). Nos casos de utilización de catro ou menos executores, os resultados son similares ó uso dun só executor, é dicir, non se obteñen beneficios do paralelismo de operacións. A utilización de doce executores mellora lixeiramente os tempos de execución obtidos con oito, sendo máis pronunciado o descenso no caso do uso de dezaseis executores. A razón de que sexa a partir de oito executores cando comeza o escalamento pode estar relacionada co número de nodos físicos do clúster utilizado, que é de 5. Ó utilizar oito executores, existen nodos que conteñen máis dun executor. Spark detecta automaticamente a configuración descrita e utiliza algún dos nodos que contén máis dun executor para paralelizar tarefas que con menos executores non paralelizaba para evitar transferencias de datos presuntamente innecesarias.

Unha análise máis pormenorizada da execución, mediante a interface web de Spark, levounos a comprobar que, debido á forma na que o *framework* leva a cabo as unións de conxuntos de datos (esta operación é a máis común no noso *pipeline* de operacións) existen algunhas operacións que resultan máis custosas de paralelizar. En concreto, lévase a cabo:

- Unha unión por identificador entre os datos de estación e os datos de observacións.
- Unha unión por veciñanza entre os datos de elevación.
- Unha unión por distancia entre os datos de pendente e os datos de estación-observación.
- Unha unión por intersección entre os datos de combustible e o grupo anterior (coas observacións interpoladas).

A primeira unión é do tipo *equijoin* mentras que as tres seguintes son de tipo *spatial join*. Revisando o grafo de execución xerado por Spark, podemos coñecer cales son os algoritmos que se están aplicando para levar a cabo estas unións:

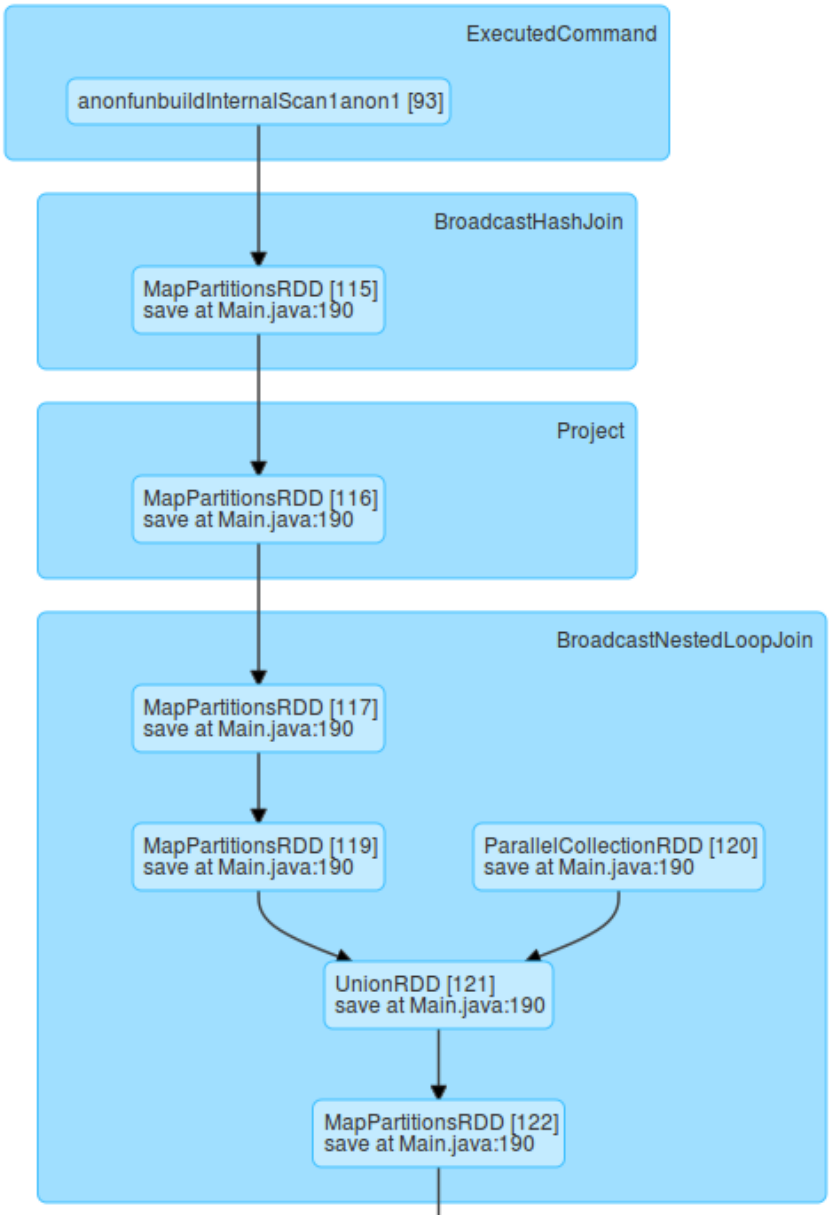


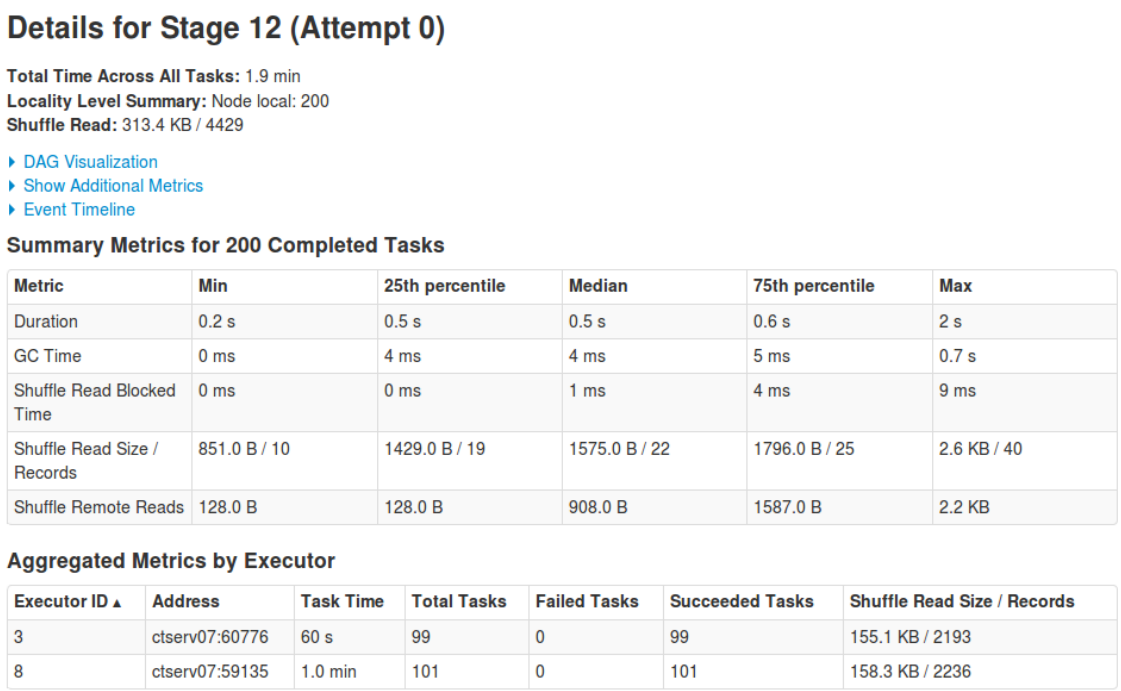
Figura 20.- Grafo de execución para a unión entre os datos de observacións e de pendente.

En efecto, a figura anterior, extraída da interface web, correspóndese coa parte do *pipeline* na cal se unen as estacións coas observacións, para o cal se aplica un algoritmo de tipo *hash join*, e o resultado se une á súa vez cos datos de pendente, utilizando para elo unha función de distancia, o cal require a utilización dun algoritmo de tipo *nested loop join*. A diferencia entre ambos algoritmos é a seguinte:

- O *hash join* é un algoritmo que tipicamente se implementa nas bases de datos relacionais, e a súa formulación clásica para a unión de dúas relacións sería:
    - Aplicar unha función *hash* sobre a relación de menor tamaño, utilizando o atributo de unión.
    - Escanear a relación maior e buscar as filas da relación menor coincidentes mediante a táboa *hash*.

- O *nested loop join* leva a cabo a unión de dous conxuntos utilizando dous bucles aniñados. Para dúas relacións R (maior) e S (menor), estando os datos distribuídos entre varios nodos, habería que:
    - Copiar todos os datos de S en cada nodo.
    - Para cada tupla de R, recorrer o conxunto de tuplas de S e aplicar a condición de unión.

Intuitivamente se pode apreciar que o segundo dos algoritmos é o máis custoso, tanto en termos de tempo de execución como en termos de consumo de memoria (e de tráfico de datos). Posto que Spark utiliza ese algoritmo cando levamos a cabo as unións por condicións espaciais, e nas súas versión máis recentes (a maio de 2016) non se implementa ningún tipo de indexación ou distribución espacial, este tipo de unións constitúe un hándicap de rendemento no noso sistema. Como exemplo, adxuntamos a seguinte captura da Spark Web UI:



Details for Stage 12 (Attempt 0)

Total Time Across All Tasks: 1.9 min
Locality Level Summary: Node local: 200
Shuffle Read: 313.4 KB / 4429

▸ DAG Visualization
▸ Show Additional Metrics
▸ Event Timeline

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0.2 s	0.5 s	0.5 s	0.6 s	2 s
GC Time	0 ms	4 ms	4 ms	5 ms	0.7 s
Shuffle Read Blocked Time	0 ms	0 ms	1 ms	4 ms	9 ms
Shuffle Read Size / Records	851.0 B / 10	1429.0 B / 19	1575.0 B / 22	1796.0 B / 25	2.6 KB / 40
Shuffle Remote Reads	128.0 B	128.0 B	908.0 B	1587.0 B	2.2 KB

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Shuffle Read Size / Records
3	ctserv07:60776	60 s	99	0	99	155.1 KB / 2193
8	ctserv07:59135	1.0 min	101	0	101	158.3 KB / 2236

Figura 21.- Falta de paralelismo nunha unión espacial.

Nesta figura observamos que, pese a executar o programa nun clúster con oito executores, esta etapa correspondente a unha unión espacial, que consta de 200 tarefas, só se esta a executar en dous executores (o número 3 e o número 8) que pertencen ó mesmo nodo (*ctserv07*).

Outra cuestión de rendemento a ter en conta é o aumento do tempo de lectura e de consumo de memoria segundo aumenta o tamaño dos arquivos importados desde o almacenamento distribuído para calcular o risco. En efecto, para a orde de xigabytes de datos de entrada, que é un nivel típico para os casos de uso deste tipo de sistemas, a porcentaxe de execución dedicada a ler os datos de entrada comeza a ser significativa e a predominar sobre as operacións de cálculo levadas a cabo cos datos en memoria. Nesta versión do software estamos almacenando por duplicado as dimensións

(compartidas) que identifican a cada cubo de datos para despois utilizalas na unión dos conxuntos de datos, e non estamos tendo en conta a posibilidade de rexenerar en memoria columnas de valores que seguen intervalos fixos, a partir dos datos do catálogo.

Polo tanto xorden aquí dúas vías de exploración de posibles melloras: dunha banda, a utilización da orde implícita de certas columnas de datos para aforrar almacenamento en disco e tempo de lectura, así como a evitación da duplicación de dimensións, e doutra banda a implementación dalgún tipo de indexación e distribución espacial dos datos que permita mellorar a eficiencia das unións de tipo espacial.

Iteración 3

Unha vez obtivemos unha implementación completa e funcionalmente correcta dos módulos de ETL e procesamento, chegou o momento de construír o módulo de visualización para poder representar e mostrar por pantalla os resultados do cálculo do risco de incendio, así como visualizar os datos de entrada.

Análise do módulo de visualización

Durante o proceso de avaliación de alternativas por parte do equipo, optamos por poñer a énfase en non volver a desenvolver solucións que xa puidesen existir no mercado a disposición dos usuarios e de forma gratuíta. Isto permitíanos reducir, entre outros, os riscos do proxecto asociados á aparición de novos requirimentos, e optimizar o tempo da planificación dedicado a este módulo, para poder centrar os nosos esforzos en buscar solucións aos problemas de rendemento detectados na iteración anterior.

Decidimos desenvolver un módulo software que importase o arquivo cos resultados do procesamento almacenado en HDFS e levase a cabo as operacións de parseo necesarias para exportar tipos de arquivo compatibles cos estándares GIS habituais (GeoTiff, Ascii, NetCDF, ...). Desta forma, poderíamos levar a cabo a visualización en calquera das múltiples solucións gratuítas existentes para este tipo de arquivos. En concreto, o candidato elixido para a visualización foi QGIS [20] (anteriormente chamado *Quantum GIS*), un sistema de información xeográfica para escritorio gratuíto, multiplataforma e *open-source*, que permite visualización, edición e análise de datos. Trátase dun software que se utiliza habitualmente como *front-end* para outros sistemas de información xeográfica, e que ademais de ser compatible con estándares de arquivos habituais, proporciona funcionalidades avanzadas como por exemplo integración con PostGIS. Esta última característica posibilita a visualización de todas aquelas fontes de datos do noso problema que están almacenadas en PostGIS.

Deseño do módulo de exportación

Para o deseño do módulo de exportación dedicimos utilizar o patrón *Factory*. Tratábase de definir unha interface de exportación común para todos os formatos de arquivo de xeito que desde o cliente (neste caso o método *main()*) puidésemos crear obxectos exportadores sen necesidade de expoñer a súa lóxica de creación. Este módulo tamén está destinado a executarse en Spark, xa que a importación dos datos de risco pode levarse a cabo en paralelo.

Diagrama de clases

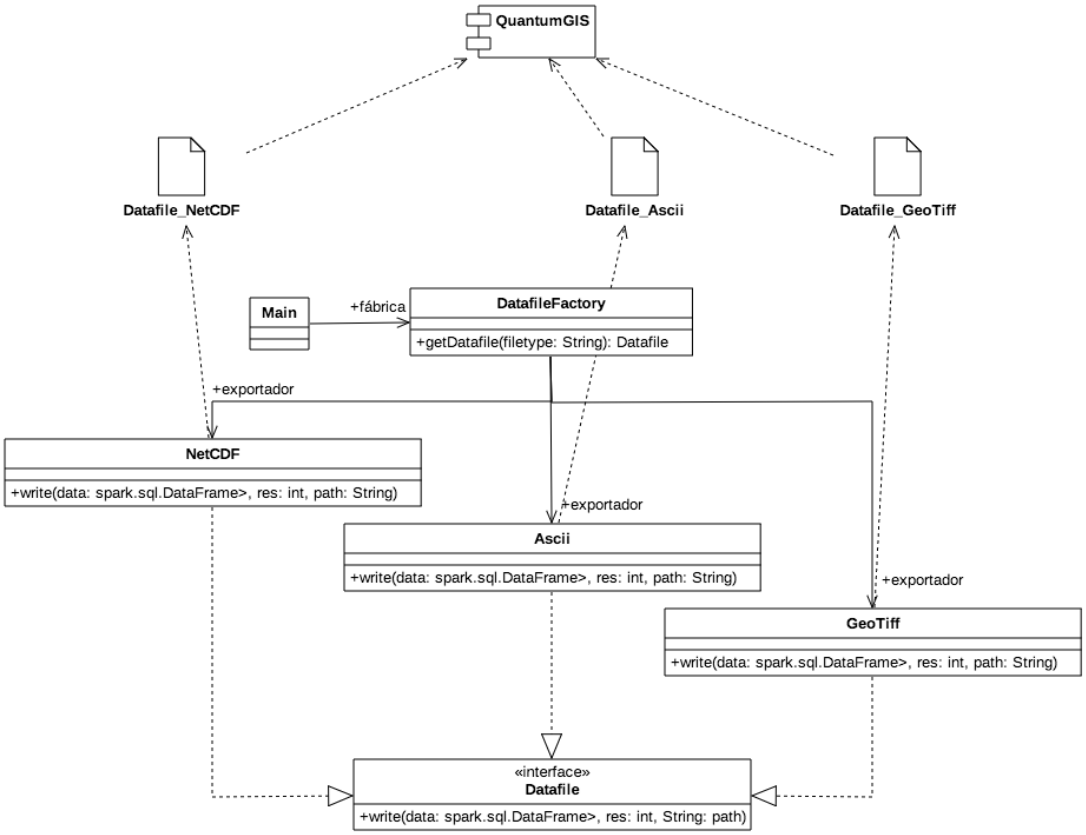


Figura 22.- Diagrama de clases do módulo de exportación.

Neste caso temos unha interface xenérica para o exportador denominada "Datafile" que se instancia a través da fábrica de exportadores, denominada "DataFileFactory", chamando ó seu método *getDatafile()* e indicándolle como argumento o tipo de formato imos a exportar. Posteriormente, cando desexamos escribir os datos, facemos unha chamada ó método *write()* da interface, que é implementado por cada exportador concreto. Independentemente da implementación concreta de cada exportador para cada formato, o método de creación e exportación dos datos é o mesmo. Isto beneficia a reutilización e o mantemento do código.

Diagrama de secuencia

Como exemplo dos pasos a tomar para exportar o risco a un ficheiro Ascii axuntamos o diagrama da figura 23. Despois de instanciar a fábrica de exportadores e obter o exportador para un tipo de ficheiro concreto, utilizamos o método *write()* da interface para gardalo no almacenamento que corresponda.

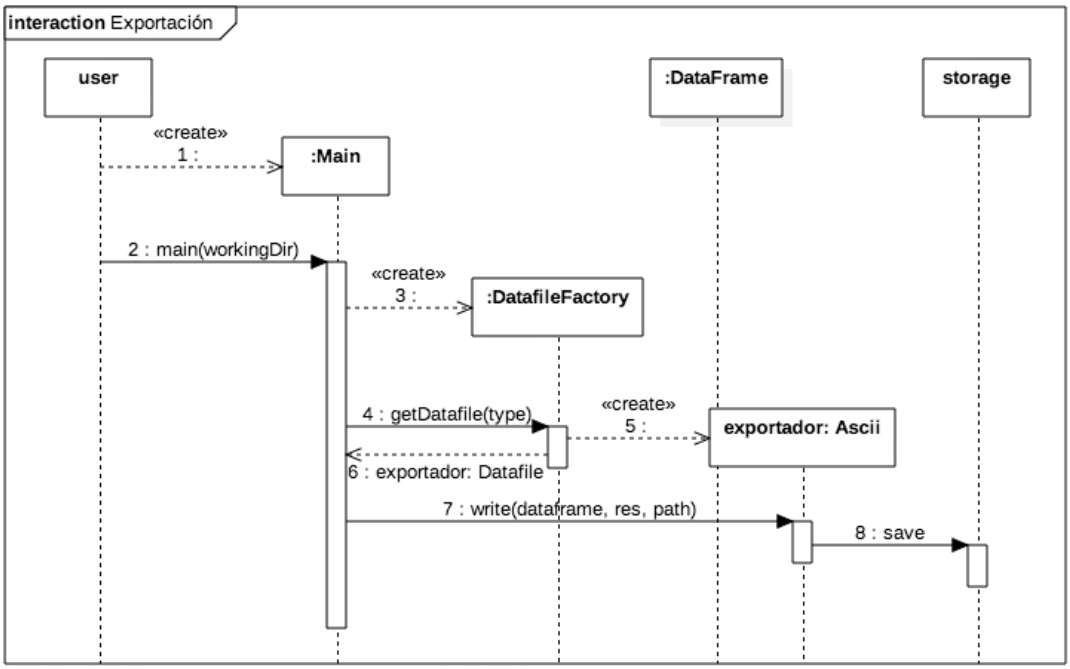


Figura 23.- Secuencia de exportación do risco a un ficheiro Ascii.

Probas unitarias do módulo de exportación

Para poder visualizar os resultados, levamos a cabo o cálculo do risco de incendio para varias resolucións e escribimos o resultado no sistema de almacenamento distribuído, en formato Parquet. Logo, mediante o módulo de exportación, transformamos ese arquivo no formato de saída desexado.

Probas de funcionalidade

Para a implementación da exportación en formato Ascii non foi necesaria a utilización de librerías de terceiros, xa que é un formato en texto plano que non require máis ca unha cabeceira con metainformación e de seguido os vectores de datos separados por espazos en branco e cambios de liña. O deseño e execución de casos de probas para funcións específicas remitiuse ó seguinte:

CP_IT3_EXP_01	
Método	DatafileFactory.getDatafile()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a fábrica de exportadores devolve unha instancia do exportador correcto.
Entradas requiridas	Tipo de exportador.
Saídas esperadas	O exportador é unha instancia do tipo indicado.

CP_IT3_EXP_02	
Método	DatafileFactory.getDatafile()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a fábrica de exportadores non devolve instancia para tipos descoñecidos.
Entradas requiridas	Tipo de ficheiro descoñecido (“unknown”).
Saídas esperadas	Excepción indicando o tipo de erro.

CP_IT3_EXP_03	
Método	Ascii.write()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a implementación do exportador ao estandar Ascii xera un ficheiro correcto.
Entradas requiridas	DataFrame, resolución e ruta.
Saídas esperadas	Ficheiro correcto na ruta indicada.

CP_IT3_EXP_04	
Método	Ascii.write()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que o exportador de Ascii non pode xerar un ficheiro sobre un conxunto de datos corrompido.
Entradas requiridas	DataFrame con datos non válidos.
Saídas esperadas	Excepción indicando o tipo de erro.

Visualización

A captura que adxuntamos a continuación correspóndese co risco de incendio para o territorio galego a unha resolución de 200 metros, para o día 1 de agosto de 2014. Unha maior intensidade da cor vermella na imaxe representa un maior risco de incendio nesa zona, tomando en consideración todos os factores que forman parte da nosa fórmula:

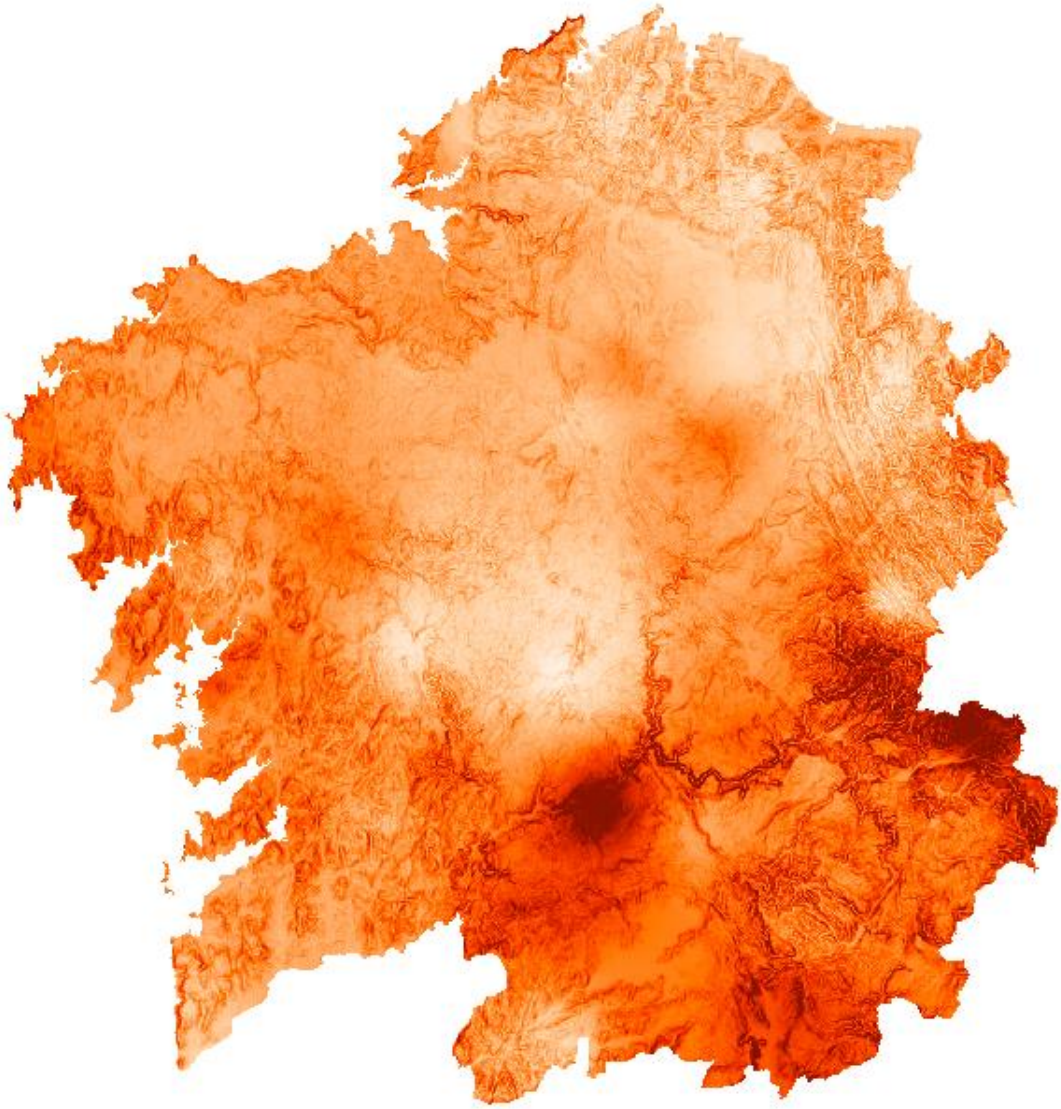


Figura 24.- Risco de incendio (01-08-2014).

Sobre esta imaxe, renderizada no software QGIS a partir do arquivo exportado, podemos consultar individualizadamente os valores de risco para cada píxel, ou características do arquivo como por exemplo o número de píxeles horizontais e verticais. O mesmo serve para a visualización dos datos de entrada.

Iteración 4

Nas probas realizadas na iteración 2 comprobábamos como a diminución do tempo de execución obtida grazas ó aumento do número de nodos de computación non foi a esperada, poñendo de manifesto a necesidade de melloras do software na busca dun maior rendemento. Apuntabamos que estas melloras se enfocaban cara dúas liñas principais: a diminución do almacenamento en disco, e polo tanto do tempo de lectura de datos, mediante o uso da orde implícita de certas columnas de datos, e a implementación dalgún tipo de distribución dos datos que permitise obter beneficios de localidade á hora de levar a cabo as unións de tipo espacial. Nesta iteración imos a afrontar a primeira desas melloras.

Análise do módulo ETL

O deseño da iteración 2, tanto do módulo de ETL como do módulo de procesamento, baseábase no almacenamento de toda a información necesaria para levar a cabo as combinacións entre os diferentes conxuntos de datos que nos ían a servir para calcular o risco de incendios. Esta forma de proceder é similar á que utilizan as solucións de almacenamento clásicas, como por exemplo as bases de datos relacionais, onde se almacenan tuplas de valores contendo cada unha delas o seu identificador correspondente. Retomando a distinción entre dimensións e cubos de datos explicada no contexto, a nosa estratexia requiría que se utilizase unha columna de datos para almacenar cada dimensión e cada cubo de datos, de xeito que os valores de mostraxes se almacenaban todos en disco, e naqueles casos nos que unha dimensións formaba parte dun novo cubo de datos, producíase un duplicado dos seus datos (por exemplo, no caso dos datos de estacións e observacións, que comparten a dimensión *stationid*).

Sen embargo, o paradigma de almacenamento de datos por columnas posibilita a introdución de melloras sobre esa forma de almacenamento, xa que se pode conservar a orde implícita dunha serie de datos e a partir dela rexenerar o identificador que lle corresponde a cada un, utilizando para elo os valores inicial e final almacenados no catálogo. Grazas á inclusión no ficheiro de definición de datos da posibilidade de definir dimensións de tipo mostraxe, podemos agora aplicar esta técnica de mellora da eficiencia no almacenamento e comprobar o seu impacto no rendemento do software.

Deseño do módulo ETL

Aínda que a estrutura de clases a implementar para o módulo ETL é similar na súa maior parte á indicada nas iteracións 1 e 2 (figuras 8 e 12, respectivamente), existen partes concretas que deben sufrir modificacións para poder adaptarnos ós novos requirimentos. En concreto, posto que desexamos almacenar as dimensións e os cubos de datos por separado, cómpre definir na clase que fai de interface á lectura (*DataReader*) novos métodos que dean conta das dimensións e dos cubos de datos por

separado. Isto implica que no catálogo de fontes (*Catalog1*) deben estar ben diferenciadas ambas estruturas, e polo tanto motiva a creación dunha xerarquía de clases que permita xestionar correctamente a información relativa tanto ás dimensións e ós cubos de datos que proceden de fontes relacionais como de aquelas que proceden de ficheiros.

Diagramas de clases

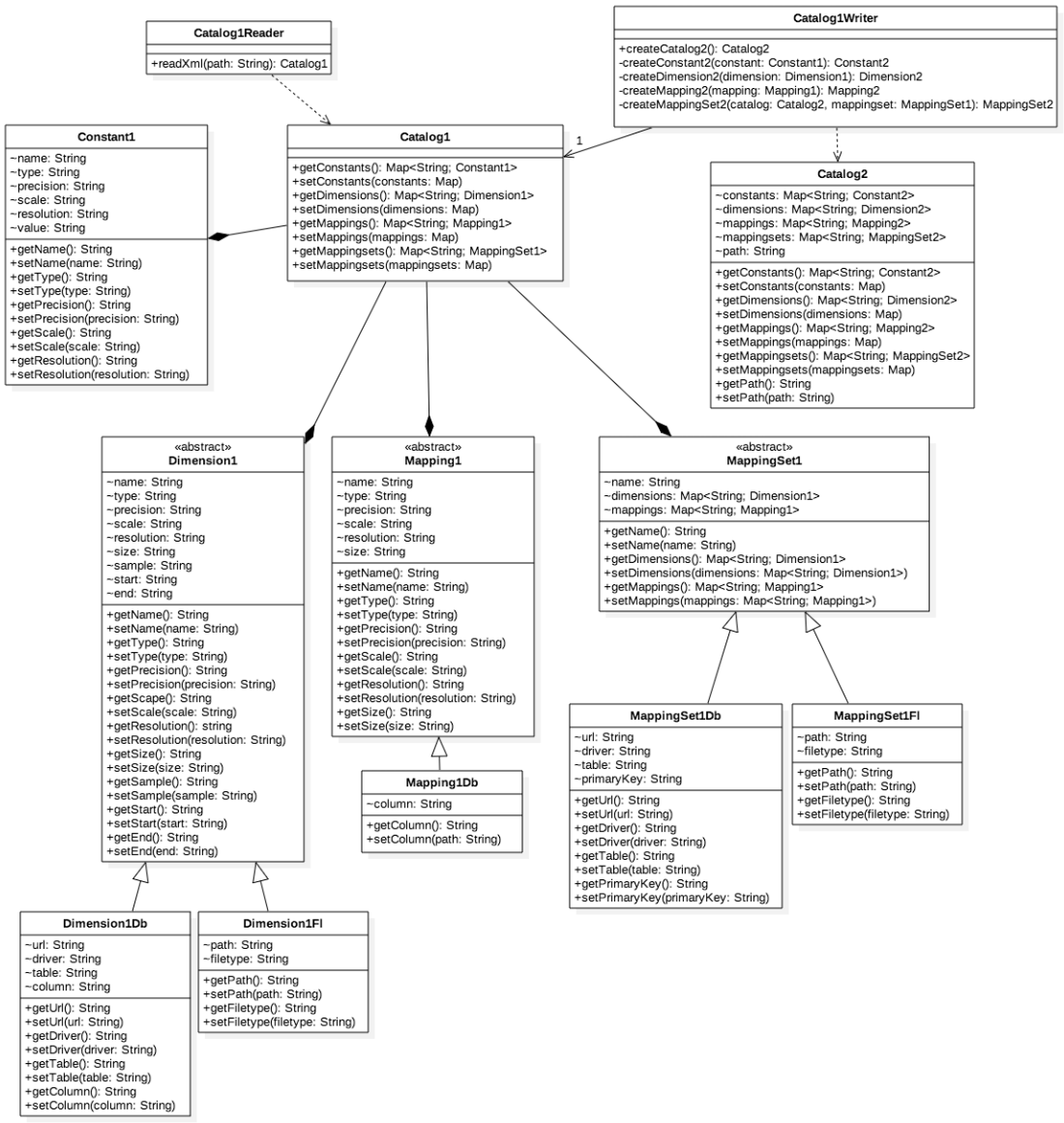


Figura 25.- Xerarquía de clases no catálogo.

Nesta figura podemos apreciar como a clase Catalog1 agora pasa a ser o contedor de varios conxuntos de obxectos que representan respectivamente as dimensións (Dimension1), os mapeos (Mapping1), os cubos de datos (MappingSet1) e as constantes (Constant1) presentes nas fontes de datos. Ademais, posto cada unha das tres primeiras

clases citadas é abstracta, definimos unha implementación diferente segundo a fonte dos datos sexa relacional ou ficheiro. Isto permitiranos almacenar os atributos necesarios para localizar os datos na base de datos ou no ficheiro, respectivamente. Doutra banda, imos a desacoplar a lectura e a escritura dos datos do propio catálogo creando dúas clases novas denominadas Catalog1Reader e Catalog1Writer que servirán respectivamente para ler o ficheiro XML e para transformar o catálogo de fontes ao catálogo de sistema. A relación do catálogo co resto de clases importantes do módulo móstrase no seguinte diagrama:

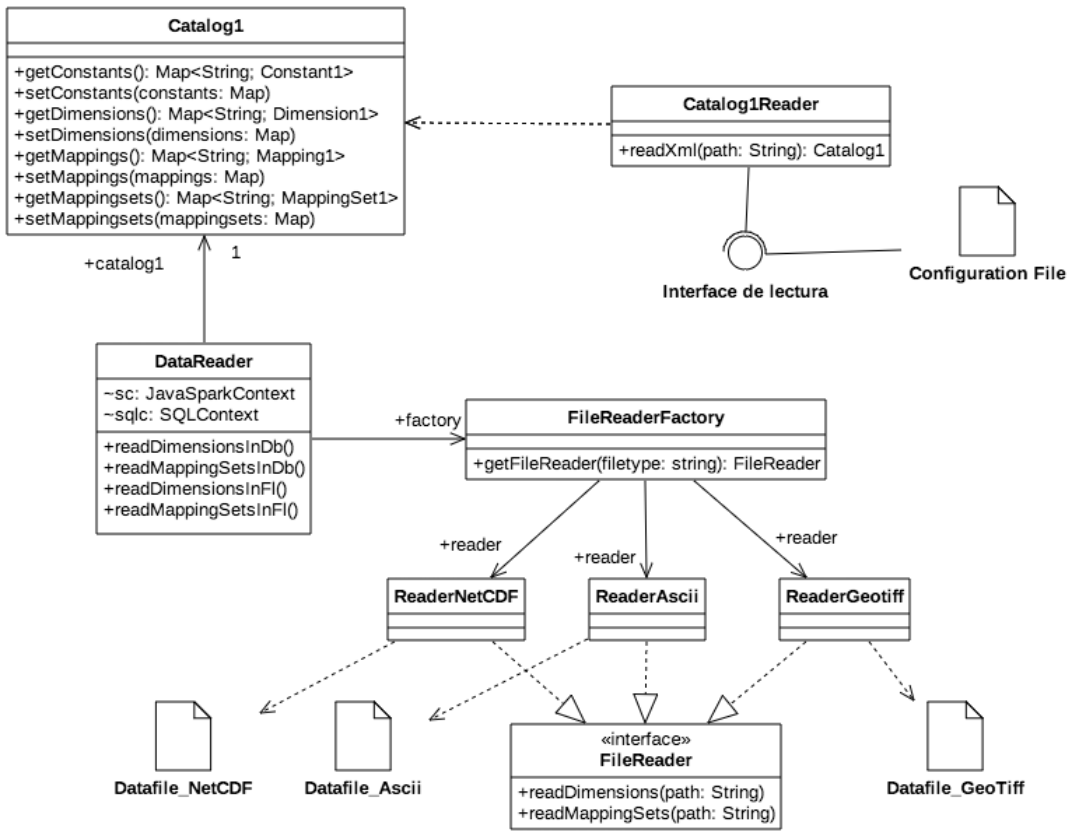


Figura 26.- Diagrama de clases para o módulo ETL.

Seguindo a estratexia que adoptamos na iteración 3 para implementar o módulo de exportación, optamos por utilizar de novo o patrón Factory, neste caso para o deseño da parte de lectura de ficheiros, onde nos podemos atopar con diferentes estándares. A fábrica de lectores de ficheiros devolve ó cliente (neste caso o DataReader) un obxecto da interface FileReader que oculta os detalles de implementación de cada tipo en particular. O cliente simplemente obtén o lector que necesita facendo unha chamada ó método *getFileReader()* indicando como argumento o tipo de ficheiro a ler (que é un dos parámetros do catálogo). O DataReader, que era a nosa clase que cumplía o papel de interface de lectura, dispón agora de catro métodos diferenciados e especializados para cada tipo de datos: lectura de dimensións en fontes relacionais, lectura de cubos de datos en fontes relacionais, lectura de dimensións en ficheiros e lectura de cubos de datos en ficheiros. No caso de ser necesario, poderíase aplicar tamén un Factory á parte

de lectura en fontes relacionais. O DataReader segue facendo uso do catálogo de fontes, que agora se crea a través da clase Catalog1Reader invocando o seu método *readXml()*. Este último é o método que lee o ficheiro de configuración definido polo usuario, instanciando todos os obxectos definidos na xerarquía da figura 26.

Diagrama de secuencia

No diagrama de secuencia podemos ver a orde na que ocorren as operacións citadas. En concreto, neste exemplo lense as dimensións dun ficheiro GeoTiff mediante o DataReader, que obtén a información que necesita do catálogo (é un dos parámetros do seu construtor). Para crear o lector específico utilízase a fábrica de lectores.

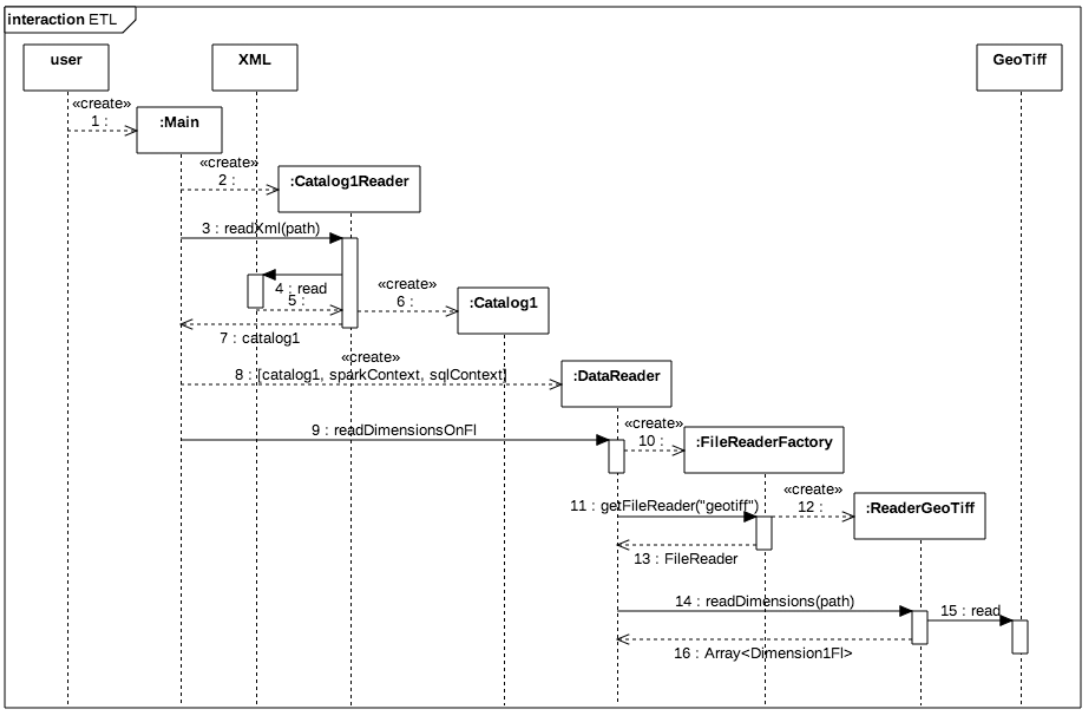


Figura 27.- Secuencia de operacións para a lectura das dimensións dun ficheiro de tipo GeoTiff.

Probas unitarias do módulo ETL

Unha cuestión importante a comprobar durante a fase de probas era o almacenamento na orde correcta dos datos, posto que dita orde vai a ser a base que imos a utilizar no módulo de procesamento para poder combinar os conxuntos de datos de estacións, observacións e elevación. Relacionado co anterior, tivemos que asegurarnos de que no catálogo se escribían correctamente os valores inicial e final de todas as dimensións de tipo mostraxe.

O deseño e execución de casos de probas para funcións específicas remitiuse ó seguinte:

CP_IT4_ETL_01	
Método	FileReaderFactory.getFileReader()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a fábrica de lectores de arquivos devolve unha instancia operativa do tipo de arquivo indicado.
Entradas requiridas	Tipo de arquivo (“ascii”).
Saídas esperadas	Obxecto de tipo FileReader sendo instancia de ReaderAscii.

CP_IT4_ETL_02	
Método	FileReaderFactory.getFileReader()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a fábrica de lectores de arquivos non devolve ningunha instancia cando o tipo de arquivo non se recoñece.
Entradas requiridas	Tipo de arquivo inexistente (“inexistente”).
Saídas esperadas	Excepción indicando o tipo de erro.

CP_IT4_ETL_03	
Método	ReaderAscii.readDimensions()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a lectura de dimensións (de tipo non mostraxe) en arquivos Ascii se leva a cabo correctamente.
Entradas requiridas	Catálogo de fontes con dimensións correctamente definidas para un arquivo Ascii.
Saídas esperadas	DataFrame(s) cos valores correspondentes a cada dimensión.

CP_IT4_ETL_04	
Método	ReaderAscii.readDimensions()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que non se leva a cabo a lectura cando as dimensións están mal definidas.
Entradas requiridas	Catálogo de fontes con dimensións incorrectamente definidas para un arquivo Ascii.
Saídas esperadas	Excepción indicando o tipo de erro.

CP_IT4_ETL_05	
Método	ReaderAscii.readMappingSets()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que a lectura de cubos de datos en arquivos Ascii se leva a cabo correctamente.
Entradas requiridas	Catálogo de fontes con cubos de datos correctamente definidos para un arquivo Ascii.

Saídas esperadas	DataFrame(s) cos valores correspondentes a cada cubo de datos.

CP_IT4_ETL_06	
Método	ReaderAscii.readMappingSets()
Técnica	Conxectura de erros
Descrición	Trátase de comprobar que non se leva a cabo a lectura cando os cubos de datos están mal definidos.
Entradas requiridas	Catálogo de fontes con cubos de datos incorrectamente definidos para un arquivo Ascii.
Saídas esperadas	Excepción indicando o tipo de erro.

Probas de rendemento

As probas de rendemento levadas a cabo sobre este módulo non arroxaron resultados significativamente diferentes dos obtidos na implementación da iteración 2. Isto era o esperado, dado que levamos a cabo unha reestruturación do catálogo para evitar duplicidades e unha extensión da funcionalidade de importación de ficheiros con obxectivos dirixidos á mellora da compatibilidade e das posibilidades de reutilización do código, e non ó rendemento, a diferencia do módulo de procesamento.

Outra cuestión era a de comprobar o tamaño dos arquivos xerados en función da utilización ou non de compresión, para diferentes tamaños. Este aspecto ímolo a analizar nas probas de rendemento que levaremos a cabo trala implementación do módulo de procesamento.

Análise do módulo de procesamento

As modificacións máis importantes desta nova versión do software atinxen a este módulo debido á utilización de columnas implícitas, que requiren o cálculo en memoria da serie de valores que as compoñen, naqueles casos nos que ditos valores sexan necesarios para levar a cabo unha operación.

Como exemplo, imos a describir a unión entre os datos de estación e os datos de observación. Dado un conxunto de datos de estacións, onde cada fila representa a unha estación e as columnas que a compoñen representan información dela, podemos asociar unha orde implícita ás filas onde cada fila se representaría mediante un número enteiro incremental. Agora ben, dado o conxunto de datos de observacións, onde cada fila representa unha observación asociada a unha estación concreta, nunha data concreta, tamén podemos asociar unha orde implícita ás filas, ó igual que no caso das estacións. O punto clave é que a orde que se lle asigna ás estacións sexa equivalente á orde que se lle asigna ós datos de observacións, que terán que comezar por aquelas observacións da estación que ocupe o primeiro lugar no primeiro conxunto. Dentro dese primeiro grupo de observacións que corresponden á primeira estación, haberá que

aplicar unha orde tamén para cada data, por exemplo de menor a maior data. Logo se sabemos cantas observacións se teñen realizado en cada estación (debe ser o mesmo número para todas as estacións), podemos recalcular o número de estación que lle corresponde, e logo a data concreta a partir da data inicial, que debe aparecer no catálogo do sistema (posto que se trata dunha dimensión de tipo mostraxe).

Neste caso, posto que decidimos non almacenar os identificadores alfanuméricos de cada conxunto de datos (por exemplo, o *stationid*), debemos xerar en memoria as columnas implícitas que representen un identificador enteiro incremental baseado na orde dos datos. Posteriormente poderemos levar a cabo a combinación dos conxuntos de datos utilizando eses índices. Para elo imos a facer uso dun método da API de Spark denominado *zipWithUniqueId()*, que asigna a cada elemento dun RDD un enteiro longo que o identifica univocamente en correspondencia co lugar que ocupa dentro dunha determinada partición. O *pipeline* de operacións para calcular o risco de incendios aumenta lixeiramente a súa complexidade, quedando como segue:

1. Lectura dos datos de estacións.
2. Xeración do ZWUI (ZipWithUniqueID) para cada estación.
3. Lectura dos datos das observacións.
4. Xeración do ZWUI para cada observación.
5. Xeración do número de fila orixinal (N) de cada observación, a partir do seu ZWUI, coa seguinte fórmula:

$$N = \frac{ZWUI - PN}{P} + PN * ceil\left(\frac{tam}{P}\right)$$

Fórmula 3.- Xeración do número de fila de cada observación.

   onde *PN* representa o número da partición, *P* o número de particións e *tam* o número de observacións.
6. Xeración do número de estación (NE) e do número de data (ND) para cada observación, coas seguintes fórmulas:

$$NE = \frac{N \ mod \ (tamE * tamD)}{tamD}$$

Fórmula 4.- Xeración do número de estación para cada observación.

$$ND = N \ mod \ tamD$$

Fórmula 5.- Xeración do número de data para cada observación.

   onde *tamE* representa o número de estacións e *tamD* o número de datas.
7. Xeración do ZWUI de estación (zwuiE) que lle corresponde a cada observación, coa seguinte fórmula:

$$zwuiE = \left(NE \ mod \ \left(ceil\left(\frac{tamE}{P}\right)\right)\right) * P + PN$$

Fórmula 6.- Xeración da referencia á estación que lle corresponde a cada observación.

8. Unión entre estacións e observacións utilizando o ZWUI de estación.
9. Lectura de datos de elevación.
10. Xeración de ZWUI para os datos de elevación.
11. Xeración do número de fila orixinal (N) de cada dato de elevación, coa mesma fórmula utilizada en (5).
12. Xeración do Point2D (x, y) que lle corresponde a cada elevación a partir do seu N, e das coordenadas iniciais, que deben estar almacenadas no catálogo do sistema. Para elo utilizamos as seguintes fórmulas:

$$x = (N \ mod \ tamX) * res + startX$$

$$y = (floor(N \ / \ tamX)) * res + startY$$

Fórmula 7.- Xeración do Point2D que lle corresponde a cada valor de elevación.

onde *tamX* representa o tamaño do eixo de abscisas, *res* a resolución en metros e *startX*, *startY* as coordenadas iniciais.

13. Cálculo da pendente en cada píxel a partir da elevación.
14. Unión por distancia dos datos de pendente co conxunto de datos obtido en (8).
15. Agrupación de (14) por localización, data e pendente, e cálculo da interpolación IDW para a temperatura, a humidade e o vento.
16. Lectura de datos dos modelos de combustible.
17. Unión, por intersección xeométrica, entre os modelos e (15).
18. Agrupación por localización, pendente, data, e valores interpolados das observacións, seleccionando o mínimo modelo de combustible.
19. Normalización e agregación dos parámetros. En cada localización p, os valores de temperatura, humidade, velocidade do vento e modelo de combustible deben ser normalizados a valores no intervalo [0, 1].
20. Por último, o risco de incendio en cada localización p calcúlase como unha media aritmética ponderada dos compoñentes normalizados. Para a ponderación, hai que ter en conta que o risco increméntase coa pendente, a temperatura, a velocidade do vento e o modelo de combustible, mentres que diminúe coa humidade.

O deseño de clases é similar ó mostrado no diagrama da figura 11 (véxase “Iteración 1”), excluíndo os contedores de datos e os PairRDDs, xa que usamos DataFrames. A interacción entre elas é, sen embargo, máis complexa desta volta posto que temos que implementar a nova secuencia de operacións para xerar os identificadores de obxectos en memoria:

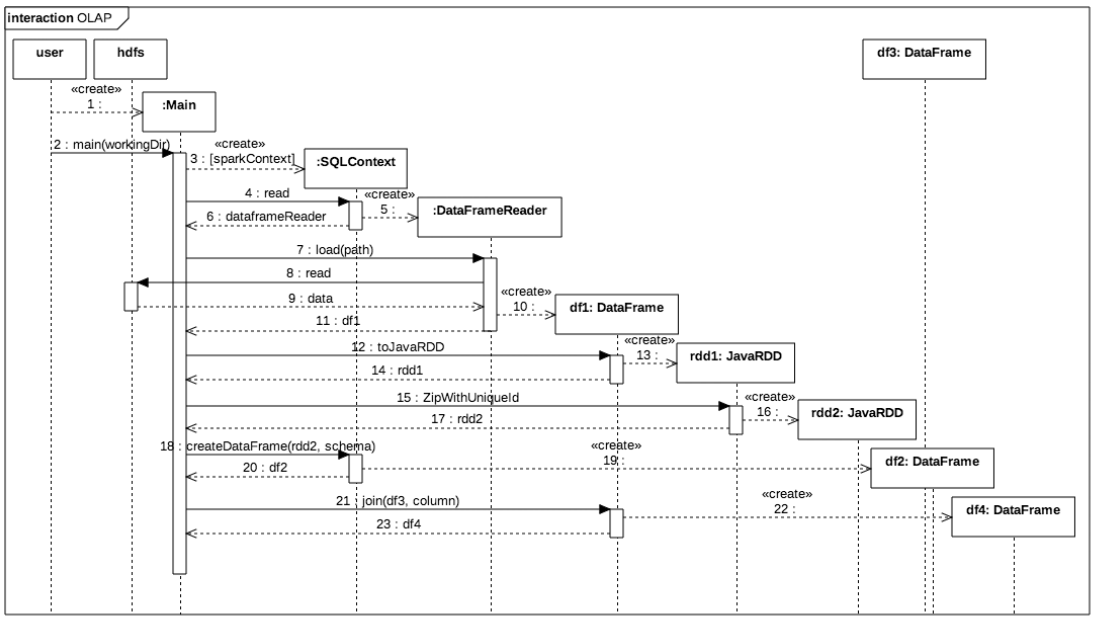


Figura 28.- Secuencia de lectura dos datos de estacións e xeración dos índices en memoria.

Esta secuencia mostra os pasos necesarios para a creación da columna de identificadores dun conxunto de datos, como pode ser o de estacións, e a súa posterior unión con outro conxunto de datos. En primeiro lugar, o programa principal le os datos desde HDFS a través do SQLContext (que utiliza un obxecto DataFrameReader), e debe transformar ese DataFrame a un JavaRDD para poder aplicar o método *zipWithUniqueId()*, que será o que xere os índices únicos para cada fila (obxecto tipo *Row*) do RDD. Esta transformación a RDDs é un *overhead* de procesamento necesario para poder aplicar o método de xeración dos índices, e constitúe un cambio respecto á versión da iteración 2. Ó aplicar o método, xérase outro RDD cos índices para cada obxecto que debe ser transformado de volta nun DataFrame para poder realizar a unión. Nos casos nos que se necesitan realizar máis transformacións, como por exemplo para obter o número de estación e o número de data no conxunto de observacións, hai que aplicar as correspondentes funcións de transformación (baseadas nas fórmulas indicadas no apartado anterior) implementando a interface *Function* da API de Spark. Unha vez obtemos o novo DataFrame resultado de aplicar o *schema* correspondente, podemos realizar a unión con outro DataFrame utilizando a columna de identificadores.

Outra mellora relevante que implementamos nesta iteración é o deseño dun algoritmo propio para o cálculo da intersección entre un punto e un multipolígono, de forma de

eliminamos a dependecia de librerías xeométricas externas neste módulo. O algoritmo é o seguinte:

1. Comprobar se o punto pertence ó rectángulo mínimo que contén o multipolígono. Se non pertence, entón non hai intersección.
2. Se pertence, entón para cada buraco do multipolígono, comprobar se o punto intersecciona. Se intersecciona cun buraco, entón non intersecciona co multipolígono.
3. Se non intersecciona con ningún buraco, entón comprobar se está dentro do borde do multipolígono. Se está dentro, entón intersecciona co multipolígono.

Comparativa de rendemento entre diferentes alternativas

Unha vez chegados a este punto, temos implementadas dúas alternativas para o almacenamento das dimensións nos cubos de datos, en función do uso ou non de columnas implícitas:

- **Alternativa 1.-** Tanto as dimensións como as funcións se almacenan en columnas dun DataFrame, admitindo duplicidades no caso de dimensións compartidas por varios cubos de datos.
- **Alternativa 2.-** Só as funcións se persisten nos DataFrames, ordenadas de forma ascendente polos valores das dimensións, de xeito que se poden rexenerar en memoria.

Xorde, ademais, unha terceira alternativa non analizada ata agora:

- **Alternativa 3.-** Ademais das funcións, almacénase un identificador autoincremental seguindo a orde das dimensións. Esta alternativa resulta unha solución de compromiso entre as dúas anteriores, xa que evita o almacenamento de dimensións de tipos complexos, pero admite o almacenamento dun identificador autoincremental para eliminar a necesidade de utilizar métodos do tipo *zipWithUniqueId()*.

O noso interese diríxese, xa que logo, á comparación dos tempos de execución para cada unha das alternativas propostas. Para elo temos seleccionado unha parte do *pipeline* de operacións, neste caso a unión entre os datos de observacións interpolados e os datos de elevación, que foron tratados previamente para asignarlles o píxel ou punto que lle corresponde no espazo de representación bidimensional. Para a alternativa 1 imos a utilizar como columna de unión unha cadea de texto que representa o punto no espazo (xeometría) de cada dato, de forma que non se necesitan operacións adicionais máis aló da propia unión de cada fila. Para a alternativa 2, só imos almacenamos os valores de observación e de elevación, e polo tanto temos que rexenerar en memoria as columnas implícitas baseándonos na orde dos datos e posteriormente calcular os puntos 2D que lle corresponden para facer a unión. Por último, na alternativa 3 imos a utilizar a columna que contén os identificadores autoincrementais para rexenerar os puntos xeométricos que lles corresponden aos datos. No caso da alternativa 2 imos a utilizar un algoritmo propio que rexenera o índice posicional de cada fila controlando o particionamento dos datos:

1. En escritura:
    a. Almacenar os datos en disco en función do seu número de partición en memoria, en carpetas independentes cuxo nome contén o número de partición. Para elo existe un método en Spark denominado *partitionBy()* que recibe a columna de particionamento como argumento.
    b. Almacenar no catálogo do sistema o índice inicial de cada partición.
2. En lectura:
    a. Ler os datos creando o mesmo número de particións e co mesmo tamaño ca en escritura.
    b. Recalcular o índice orixinal de cada fila usando o número de partición, que se rexenera automaticamente no proceso de lectura a partir do nome da carpeta, e a posición do dato dentro da partición. Isto consíguese obtendo do catálogo o índice inicial da partición e sumándolle a posición do dato dentro da partición.

Como datos de partida xeramos tres arquivos sen comprimir e tres arquivos comprimidos correspondentes ás tres alternativas de almacenamento mencionadas. Os tamaños dos arquivos compáranse no seguinte gráfico:

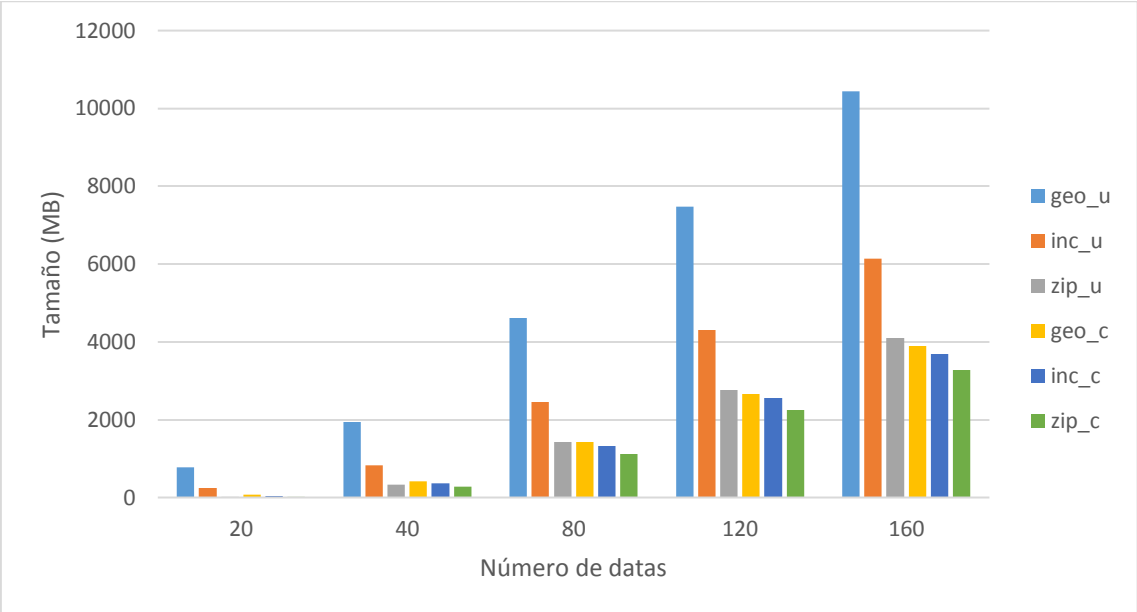


Gráfico 5.- Comparativa do tamaño dos arquivos de observación.

A explicación da lenda é a seguinte:

- *geo_u* é o arquivo que almacena a columna cos puntos xeométricos e os valores sen comprimir.
- *inc_u* é o arquivo que almacena a columna co identificador incremental e os valores sen comprimir.
- *zip_u* é o arquivo que só almacena os valores das observacións, sen comprimir.
- *geo_c* é o arquivo que almacena a columna cos puntos xeométricos e os valores comprimidos.
- *inc_c* é o arquivo que almacena a columna co identificador incremental e os valores comprimidos.

- *zip_c* é o arquivo que só almacena os valores das observacións comprimidos.

Podemos observar que, para as versións de arquivos sen comprimir, o almacenamento das xeometrías implica un aumento considerable do consumo de espazo en disco. En menor medida, este aumento tamén se produce na versión cos identificadores. Respecto ás versións comprimidas, as diferencias son moito menores entre elas, producindo un descenso considerable do consumo de disco para os arquivos coas xeometrías e cos identificadores.

En primeiro lugar, imos a comparar o rendemento de cada versión por separado en función da utilización ou non de compresión no almacenamento.

a) Para a versión que almacena as xeometrías, cun menor número de datas de partida obtemos mellores resultados en tempos de execución prescindindo da compresión, mentres que cun número alto de datas de partida obtemos mellores resultados utilizando compresión. Isto significa que o custo asociado á operación de descompresión é menor que o custo asociado á lectura de máis datos de disco a partir de certos tamaños de arquivo:

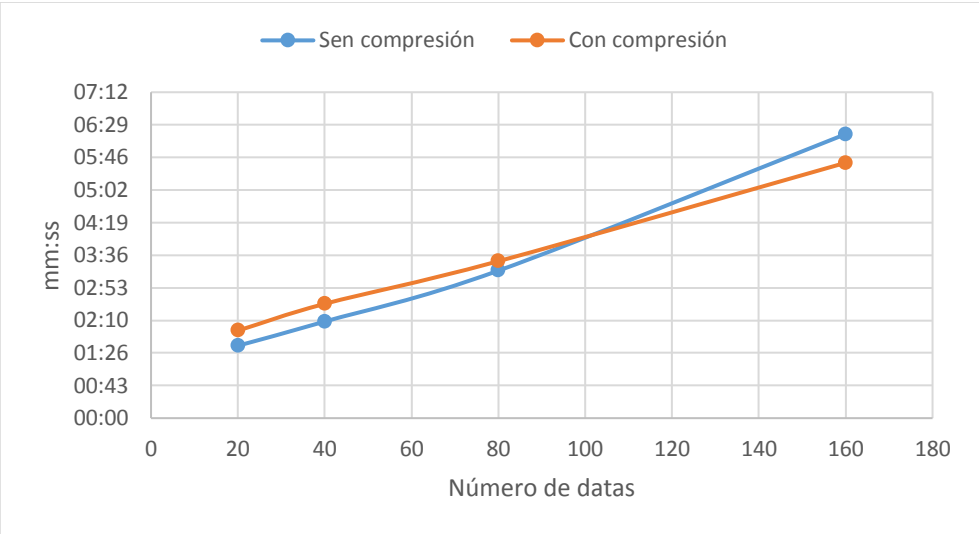


Gráfico 6.- Tempo de execución para a versión cas xeometrías.

b) Para a versión que almacena os identificadores autoincrementais o resultado é similar ó caso anterior, xa que obtemos mellores tempos de execución con poucos datos se prescindimos da compresión, e mellores tempos para moitos datos cando utilizamos a compresión:

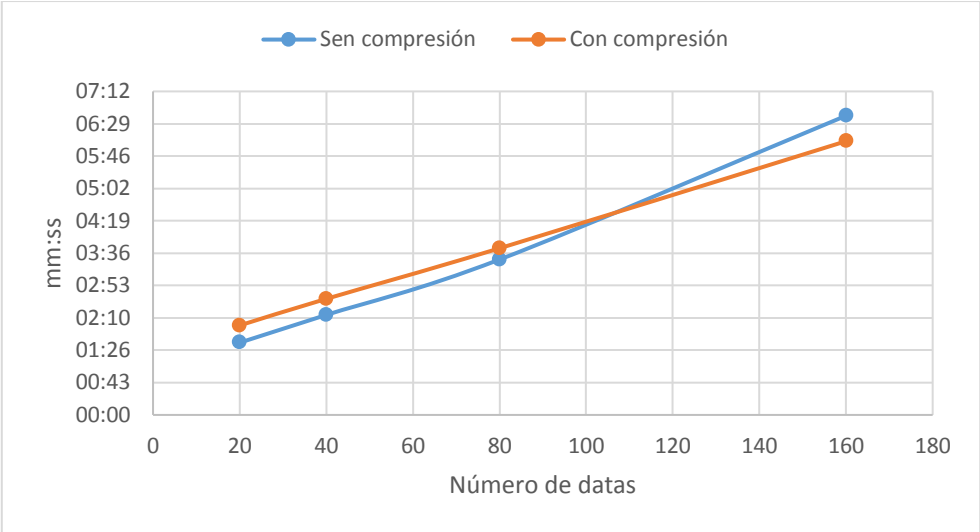


Gráfico 7.- Tempo de execución para a versión co identificador autoincremental.

c) Para a versión que só almacena os valores das observacións, os tempos de execución son similares para poucos datos (compénsanse os efectos da descompresión e o maior tamaño de lectura) pero de novo para moitos datos é máis eficiente a utilización de compresión:

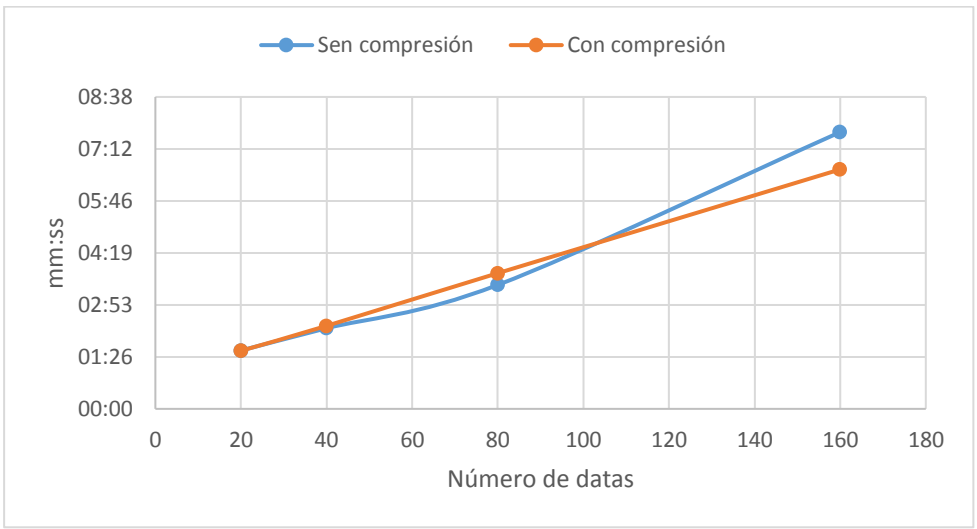


Gráfico 8.- Tempo de execución para a versión que só almacena os valores das observacións.

Agora imos a comparar as diferentes versións de almacenamento entre si, utilizando diferentes tamaños de arquivo en función do número de datas (días) de observacións que conteñen. En primeiro lugar comezamos coas versións que utilizan os arquivos sen comprimir:

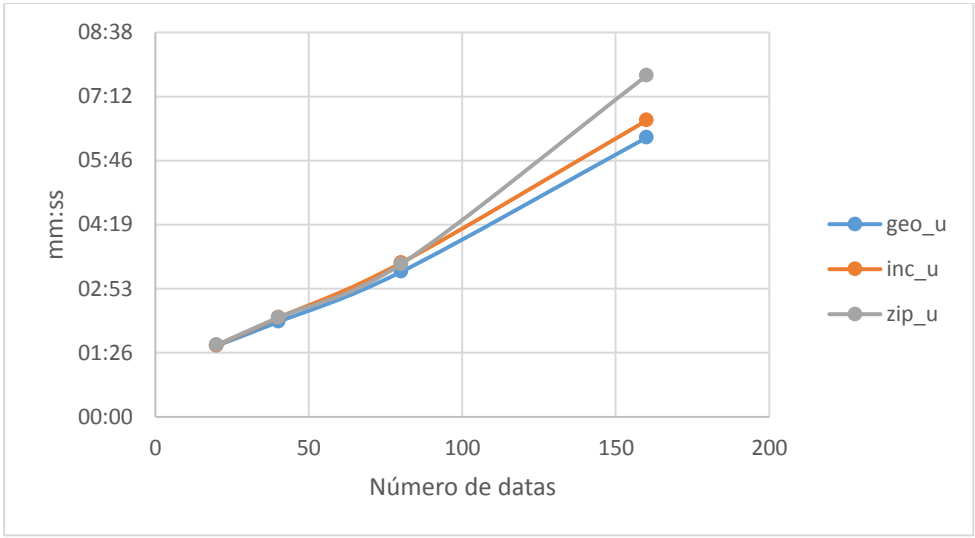


Gráfico 9.- Tempo de execución para as versións que utilizan datos sen comprimir.

No gráfico anterior podemos observar que os tempos de execución para as diferentes versións son similares con poucos datos, mentres que con moitos datos a versión máis eficiente é a que almacena as xeometrías, seguida da versión que almacena os autoincrementais, e a versión menos eficiente é a que só almacena os valores das observacións e ten que recalcular os identificadores e as xeometrías en memoria.

Respecto ás versións que utilizan os arquivos comprimidos, os resultados son os seguintes:

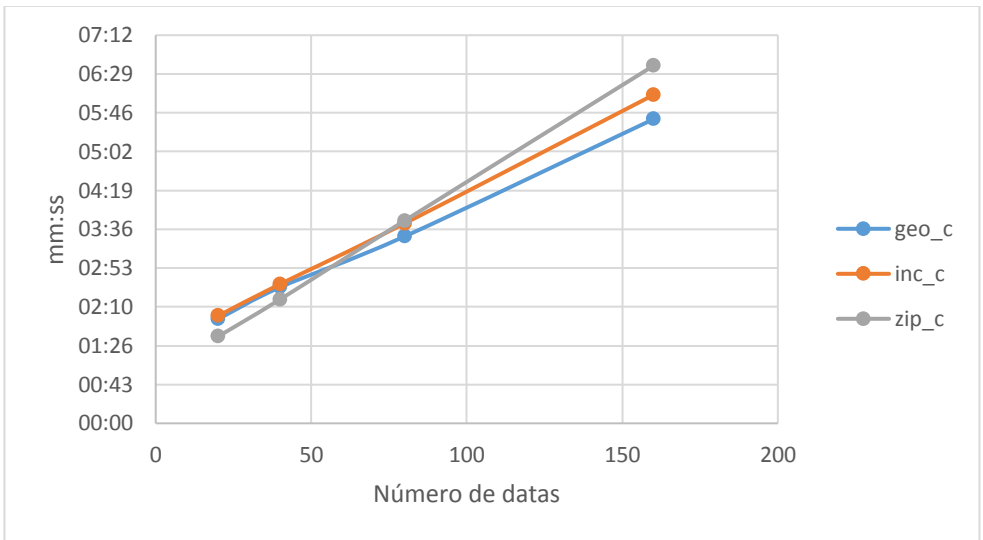


Gráfico 10.- Tempo de execución para as versións que utilizan datos comprimidos.

Neste caso, para poucos datos de entrada, a versión que só almacena os valores das observacións compórtase mellor, debido a que o *overhead* asociado ó procesamento das xeometrías en memoria é menor ca o tempo de lectura asociado a un maior tamaño de arquivo. Sen embargo, para grandes cantidades de datas a tendencia invértese e de novo a versión máis eficiente é a que almacena as xeometrías, seguida da versión con autoincrementais.

Outra cuestión que nos interesa analizar é o escalamento das diferentes alternativas, é dicir, como descende o tempo de execución variando o número de executores. Para as versións que utilizan os arquivos sen compresión obtemos o seguinte gráfico:

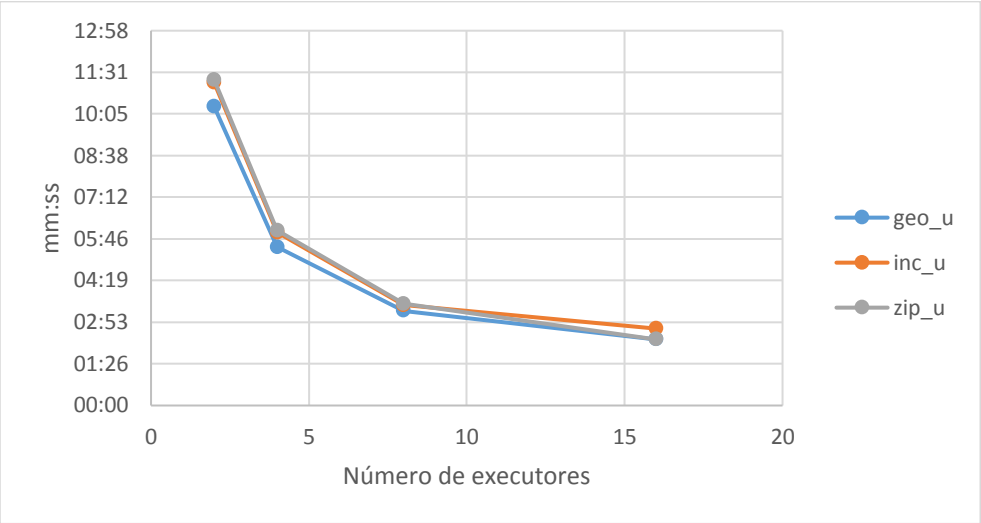


Gráfico 11.- Tempo de execución segundo número de executores (datos sen compresión).

A versión que máis escala (case linealmente) é a que só almacena os valores de observacións e recalcula as xeometrías en memoria. Sen embargo, en termos xerais, a versión máis eficiente é a que almacena as xeometrías. Para as versións que utilizan os arquivos comprimidos como entrada os tempos son os seguintes:

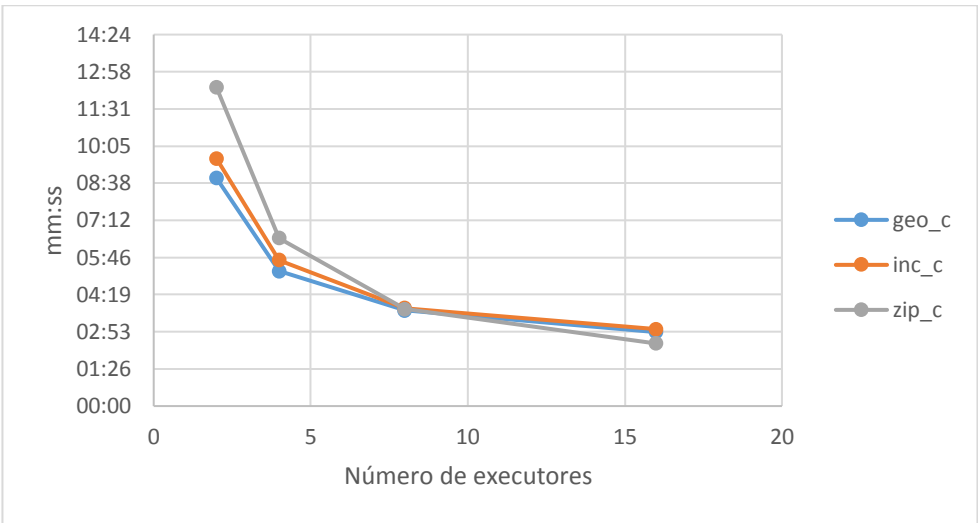


Gráfico 12.- Tempo de execución segundo número de executores (datos con compresión).

De novo a versión que máis escala é a que recalcula as xeometrías en memoria a partir da orde implícita dos elementos. Neste caso semella que a utilización dun número alto de executores beneficia o procesamento adicional requerido, mentres que con poucos executores resulta máis eficiente almacenar máis datos.

En resumo, podemos considerar dunha banda que a utilización da compresión para almacenar os arquivos permite aforrar bastante espazo de almacenamento cun custo adicional de tempo de procesamento que é asumible para poucos datos e inexistente para moitos datos, e en base a elo o cálculo de columnas implícitas en memoria constitúe un *overhead* de procesamento non xustificado pola escasa diferencia de tamaño entre almacenar ou non as xeometrías de forma comprimida, salvo que dispoñamos dun clúster cun alto número de recursos.

Iteración 5

Na iteración 4 afrontabamos unha das posibles melloras de rendemento apuntadas tralas probas realizadas na iteración 2. Nesta iteración 5, que vai ser a última que levemos a cabo para este proxecto, imos afrontar a outra posible mellora de rendemento que tiñamos apuntado: a implementación dalgún tipo de distribución espacial dos datos para obter beneficios relacionados coa súa localidade á hora de levar a cabo unións por condicións espaciais.

Análise

Como mencionamos no contexto, o intento de implementar indexación e particionamento espaciais sobre *frameworks* de computación distribuída xa deu como resultado proxectos da importancia de *SpatialHadoop* ou *GeoSpark*, que proporcionan, entre outras funcionalidades, a creación de índices espaciais para acelerar o procesamento en cada partición ou o uso de operadores espaciais específicos para tipos de datos xeométricos.

Para o noso proxecto, imos a realizar unha proba de concepto que consiste en incorporar ó noso software a distribución espazo-temporal dos datos que forman parte da operación de interpolación de observacións para cada píxel, de forma que poidamos obter a localidade espacial e temporal desexada á hora de calcular dita interpolación. Isto é debido a que, para calcular o valor dunha observación, por exemplo a temperatura, nun determinado píxel, se toman en consideración aquelas observacións de estacións que están dentro dunha distancia D (distancia de interpolación) definida previamente, e para unha data concreta. Esa distancia formaba parte da fórmula da interpolación que tiñamos explicado, baseada no método IDW:

$$temperature(\mathrm{p}) = \begin{cases} \dfrac{\sum_i \dfrac{temperature(s_i)}{distance(p,s_i)^2}}{\sum_i \dfrac{1}{distance(p,s_i)^2}}, & if\ 0 < distance(p,s_i) < d \\ temperature(s_i), & if\ distance(p,s_i) = 0 \end{cases}$$

A distribución espazo-temporal que propoñemos consiste na aplicación dunha malla 3D abstracta sobre o territorio analizado, neste caso Galicia, que permite dividir o territorio en diferentes cadrados ou *chunks* co obxectivo de particionar os datos de acordo co *chunk* no que se enmarquen. A dimensión de profundidade da malla corresponderíase coas datas das observacións (dimensión temporal). O particionamento espacial, visto de forma gráfica, quedaría así:

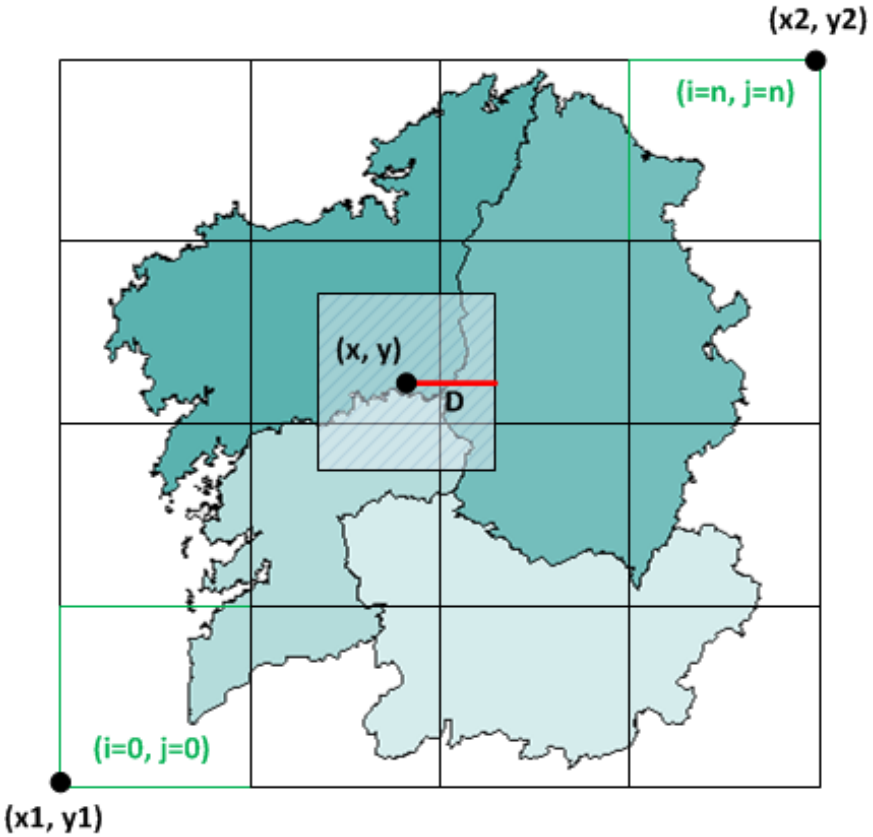


Figura 29.- Representación do particionamento espacial dividindo o territorio galego mediante unha cuadrícula.

O punto (x, y) representa un píxel ó que se lle asigna un valor de temperatura interpolado a partir das observacións de estacións que están a menos dunha distancia D. A maioría desas observacións recaen sobre o *chunk*[i][j], que será procesado nun determinado nodo do clúster. Para acelerar aínda máis o cálculo, imos a almacenar as observacións de forma repetida en cada *chunk* sobre o que recaen (unha mesma estación pode estar dentro da distancia de interpolación de varios *chunks*, como máximo catro, na figura de exemplo). Isto permitirá que cada nodo dispoña de todos os datos necesarios para levar a cabo a interpolación, reducindo o tráfico no clúster. Ademais, engadimos a dimensión temporal mediante outro índice (k), de forma que cada *chunk* espacial se divide en varios *chunks* temporais e se referencia mediante *chunk*[i][j][k].

Os parámetros de particionamento deben ser definidos polo usuario. A opción que imos a adoptar para a nosa implementación é a de utilizar un ficheiro de propiedades que se almacenará no directorio de traballo do programa. Dito ficheiro ten a seguinte estrutura:

```
// sistema_coordenadas [cadea de texto]
// orixe eixo x [real]
// orixe eixo y [real]
```

// fin eixo x [real]
// fin eixo y [real]
// num bloques x [enteiro] (función sobre o número de máquinas)
// num bloques y [enteiro] (función sobre o número de máquinas)
// resolución [real]
// ancho bloque [enteiro]
// altura bloque [enteiro]
// ancho malla [enteiro]
// altura malla [enteiro]
// distancia interpolación [real]

A data da observación podémola extraer directamente do catálogo ou dos datos almacenados en Parquet, en función de se utilizamos a versión con columnas implícitas ou a versión sen elas. O parámetro de referencia para levar a cabo o particionamento espacial é, neste caso, o número de máquinas ou nodos, que vai a determinar o número de *chunks*[i][j] no que dividimos o espazo bidimensional. Para determinar cal é o *chunk* que lle corresponde a cada estación, e polo tanto a cada medida, calculamos o *envelop* ou rectángulo que a contén dentro da distancia D de interpolación, e extraemos os seus puntos inicial e final, en coordenadas de *chunk*, aplicando as seguintes fórmulas:

$$i_1 = floor\,(\frac{x - D - x_0}{res * bwidth})$$

$$j_1 = floor\,(\frac{y - D - y_0}{res * bheight})$$

$$i_2 = floor\,(\frac{x + D - x_0}{res * bwidth})$$

$$j_2 = floor\,(\frac{y + D - y_0}{res * bheight})$$

Fórmula 8.- Cálculo das coordenadas de chunk para o particionado espacial.

onde *x, y* representan as coordenadas da estación, *D* a distancia de interpolación, x_0, y_0 a orixe de coordenadas, *res* a resolución e *bwidth, bheight* a anchura e a altura de cada bloque, respectivamente. A anchura e a altura (en píxeles) de cada bloque obtéñense aplicando as seguintes fórmulas:

$$bwidth = ceil\left(\frac{ceil\,(\frac{(x_1 - x_0)}{res})}{B_x}\right)$$

Fórmula 9.- Cálculo do ancho de cada chunk.

$$bheight = ceil\left(\frac{ceil\,(\frac{(y_1 - y_0)}{res})}{B_y}\right)$$

Fórmula 10.- Cálculo do alto de cada chunk.

onde x_1, y_1 representan as coordenadas finais do bloque e B_x, B_y o número de bloques ou divisións no eixo de abscisas e no de ordenadas, respectivamente.

Para este caso a secuencia de operacións a aplicar é relativamente sinxela posto que só implica cinco pasos importantes:

1. Lectura dos datos de estacións e de observacións desde o almacenamento distribuído.
2. Unión entre os dous conxuntos de datos, utilizando para elo ou ben o identificador almacenado en disco ou ben o identificador xerado en memoria.
3. Mapeo das observacións, xunto co id e a xeometría da estación, a pares clave-valor onde a clave ten a forma: *id_data* e o valor é un contedor cos datos mencionados.
4. Mapeo dos pares clave-contedor a cada *chunk* definido sobre o territorio, utilizando para elo a fórmula 8, definida no apartado anterior. Agora a clave pasa a ser *i_j_data*, onde i e j representan os índices de *chunk* e *data* a dimensión temporal.
5. Reparticionamento das tuplas obtidas en (4) a partir da súa clave.
6. Cálculo da interpolación para cada píxel a partir das observacións que están a menos dunha distancia D.

Deseño

En base ás operacións mencionadas, temos que deseñar dunha banda as clases das funcións de mapeo e de interpolación e doutra banda a clase contedor para os datos das tuplas. A maiores imos a utilizar un *wrapper* para almacenar as propiedades que o usuario indica no ficheiro de configuración.

Diagrama de clases

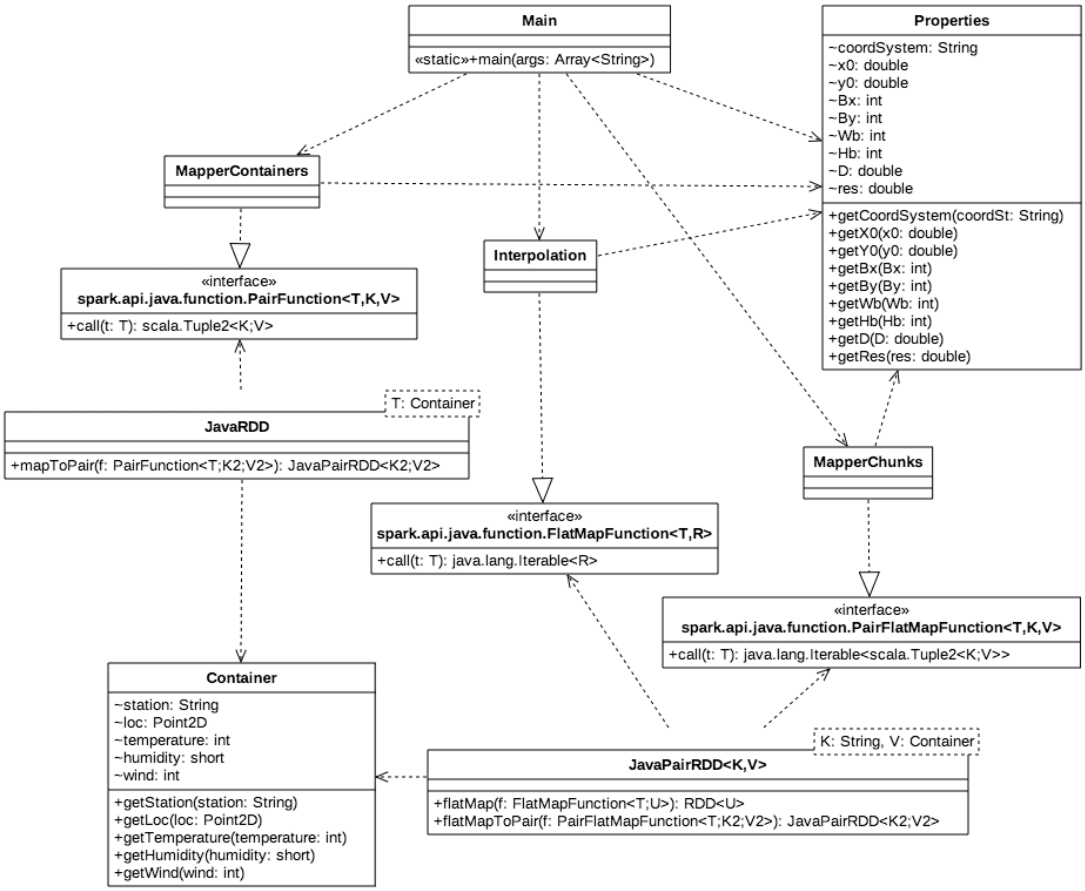


Figura 30.- Diagrama de clases para a implementación con particionamento espacial.

No diagrama de clases podemos apreciar as interfaces que implementan as funcións a codificar: no caso da función mediante a cal se crean os contedores de datos implementa a interface PairFunction, cuxo método *call()* devolve tuplas clave-valor, e no caso da función que asigna cada contedor ós *chunks* que lle corresponden, implementa a interface PairFlatMapFunctio*n*, cuxo método *call()* ten como tipo de retorno un obxecto java.Lang.Iterable. O que nos permite esta última función é xerar, para cada tupla de observacións, un array de tuplas de observacións que teñen como claves aqueles *chunks* nos que se van a almacenar. Os valores dos JavaPairRDDs que imos a construír son contedores que engloban o id da estación, a súa localización, e as tres observacións dispoñibles (temperatura, humidade e vento). A función de interpolación implementa a interface FlatMapFunction, cuxo método *call()* tamén devolve un tipo java.Lang.Iterable que no noso caso será un array (por *chunk*) de obxectos *Row* cos datos interpolados e as coordenadas para cada píxel e data. O método da clase principal do programa, ademais de crear un obxecto tipo Properties a partir do ficheiro de configuración definido polo usuario, é o responsable, como de costume, de facer as chamadas ós métodos dos RDDs enviándolles as correspondentes instancias das funcións. Aínda que non o indicamos explicitamente no diagrama, os datos lense como DataFrames desde os arquivos Parquet facendo uso do SQLContext, ó igual ca no resto de versións do módulo de procesamento.

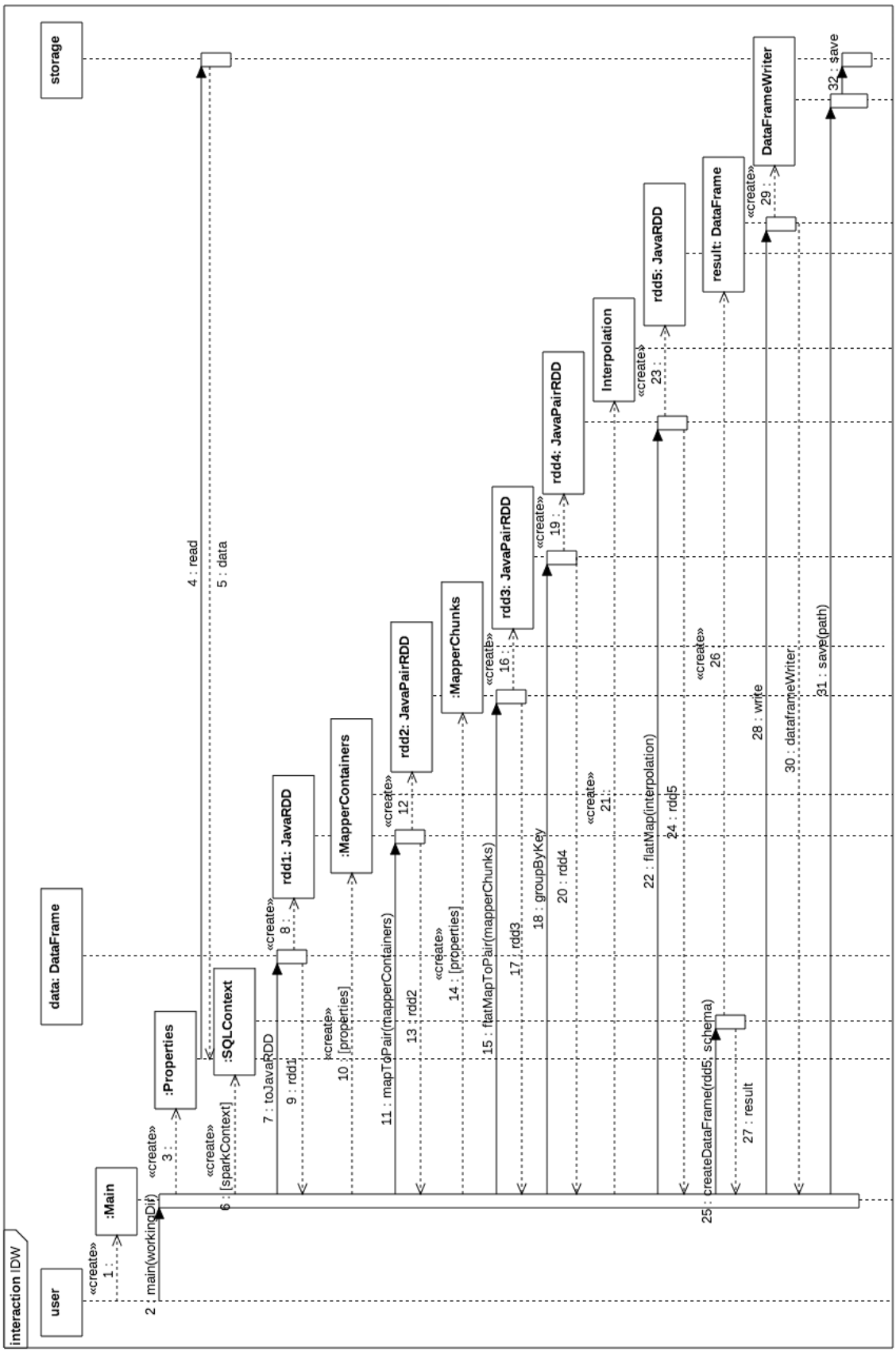


Figura 31.- Diagrama de secuencia completo para a realización da IDW en paralelo.

O inicio da execución do programa é similar á dos outros módulos, sendo o usuario o encargado de executar en Spark o programa principal, cuxo método *main()* será o encargado de crear o SparkContext e o SQLContext, e neste caso tamén o *wrapper* para o ficheiro de propiedades, que pode estar almacenado tanto localmente (no *driver* do programa) como en HDFS. Aínda que non o mostramos neste diagrama por ser un proceso similar ó de diagramas anteriores, o SQLContext é o encargado de ler os ficheiros Parquet de estacións e observacións e crear os DataFrames correspondentes, que deben ser unidos por igualdade utilizando como condición o identificador de estación. O resultado sería o DataFrame de partida que indicamos no diagrama como "data". Ese DataFrame temos que transformalo a un JavaRDD para aplicarlle as funcións que temos definido. Todas elas reciben no momento da súa instanciación unha copia do obxecto de propiedades, para utilizalas dentro das súas operacións. A primeira delas, MapperContainers, envíase como argumento do método *mapToPair()* e leva a cabo a transformación do RDD de obxectos tipo Row nun JavaPairRDD de pares clave-valor, onde a clave é unha cadea da forma *idEstación_data* e o valor un container cos datos de id e localización da estación e os valores das observacións. Ó resultado desta operación aplícase a seguinte función de transformación, MapperChunks, que se envía como argumento do método *flatMapToPair()* e xera como resultado un array de novas tuplas cuxa clave ten a forma *i_j_data* (sendo i e j as coordenadas de *chunk*) e como valor de novo o contedor de datos. Estes arrays parciais incorpóranse a un novo JavaPairRDD (o número de elementos deste RDD é superior o do anterior, xa que cada contedor de datos pode aparecer en máis dun *chunk*). O JavaPairRDD agrúpase a continuación mediante a nova clave definida chamando ó seu método *groupByKey()* que pode recibir como argumento un particionador definido polo usuario ou o número de particións, en cuxo caso utiliza o particionador *hash* implementado por defecto na API. Este punto é importante posto que é a forma mediante a que imos a obter a localidade espacial desexada para levar a cabo o último paso, a interpolación. Para este último paso definimos a función Interpolation, que lle enviamos como argumento ó método *flatMap()*, de forma que imos a calcular uns valores de temperatura, humidade e vento interpolados para cada píxel e data dentro de cada *chunk*, e almacenalos como obxectos tipo Row nun JavaRDD. Neste momento xa temos un conxunto de obxectos cos valores interpolados de cada píxel e data, que podemos transformar nun DataFrame e almacenar para ser exportados co módulo de exportación.

Visualización

A continuación axúntase unha captura do resultado exportado da interpolación da temperatura para unha resolución de 200 metros, a partir das observacións tomadas o día 3 de agosto de 2014.

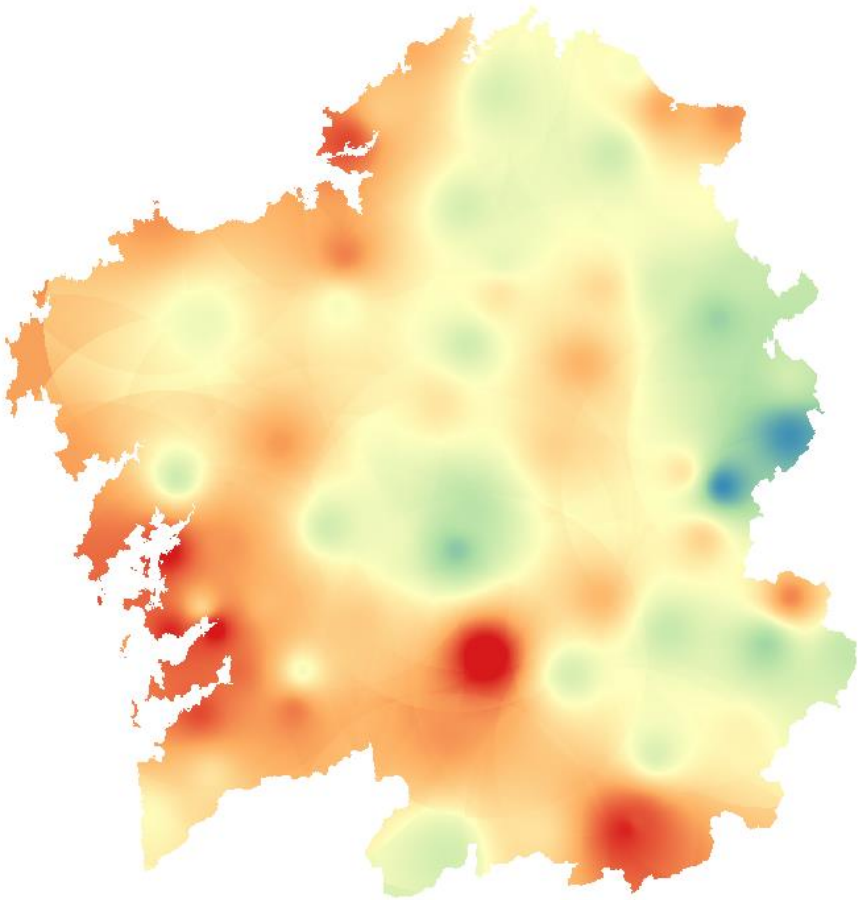


Figura 32.- Temperatura interpolada (03-08-2014).

As zonas vermellas representan os puntos onde a temperatura é máis alta, mentres as zonas azuis representan os puntos onde a temperatura é máis baixa.

Comparativa de rendemento

O aspecto máis relevante desde o punto de vista das probas deste módulo era comparar a súa resposta ó escalamento no clúster con respecto á implementación da interpolación sen aplicar distribución espacial que tiñamos realizado na iteración 2.

Para comprobar en primeiro lugar se a versión con distribución espacial que temos desenvolvido conseguía escalar adecuadamente utilizamos unha alta resolución (200 metros) dun número considerable de datas (100 días):

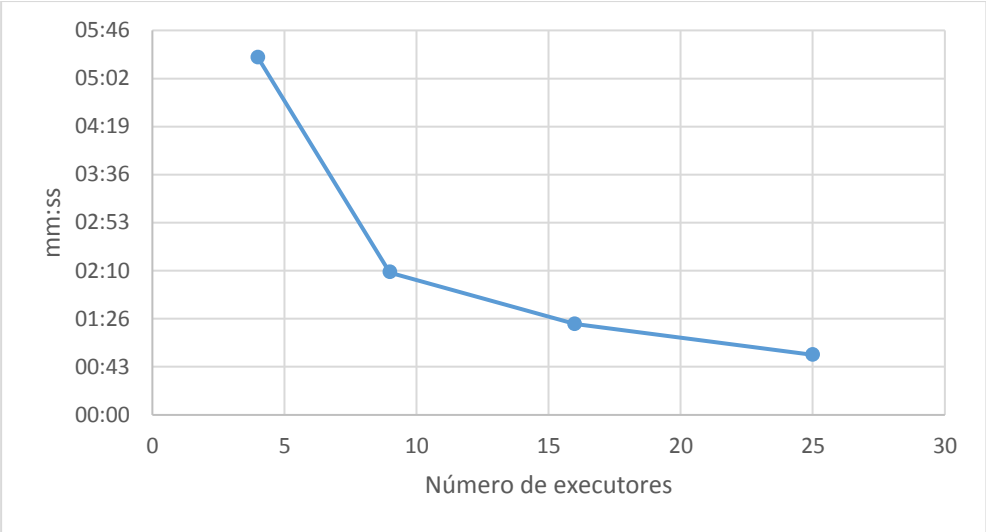


Gráfico 13.- Tempo de execución (resolución de 200 metros, 100 datas) para a versión con particionamento espacial en función do número de executores.

No gráfico anterior podemos observar a diminución do tempo de execución en función do número de executores. Debido a que o particionamento espacial se leva a cabo neste caso en dúas dimensións x e y, decidimos adoptar un número de bloques similar para ambas de xeito que o número de executores necesarios para paralelizar cada bloque fose o cadrado do número de bloques por dimensión:

$$num_{executors} = num_{blocks}(x) \times num_{blocks}(y)$$

Fórmula 11.- Número de executores necesarios para paralelizar os chunks.

Partindo dun tamaño de dimensión igual a 2, obtivemos 2*2=4 executores, 3*3=9 executores, 4*4=16 executores e 5*5=25 executores. O descenso do tempo de execución ó aumentar o número de executores é practicamente lineal salvo cando o tempo de execución descende do minuto, onde os efectos da transmisión de datos entre nodos comezan a predominar.

Respecto á comparación entre os tempos de execución das dúas versións (con particionamento espacial e sen el), tivemos que utilizar unha resolución máis baixa para poder facer as probas adecuadamente (xa que os tempos de execución da versión sen particionamento se disparan con altas resolucións):

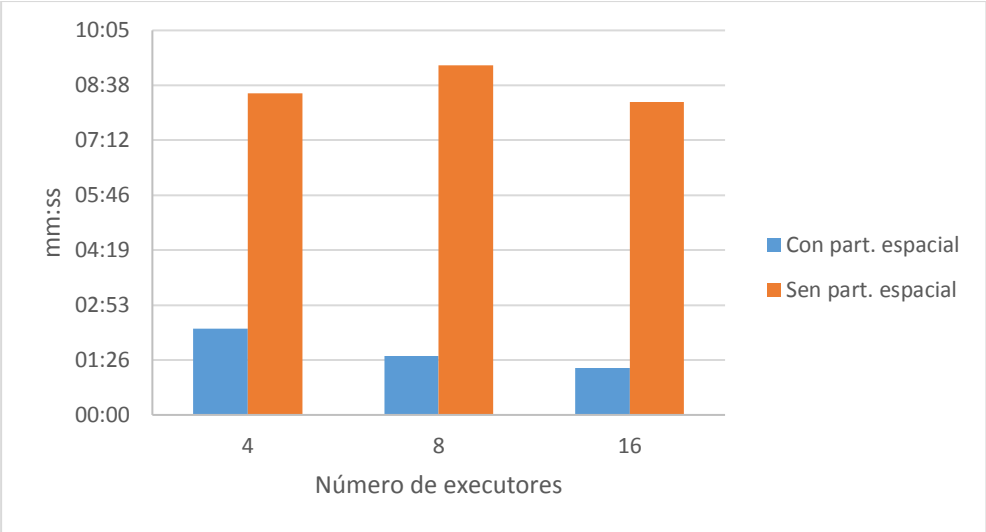


Gráfico 14.- Comparativa de tempo de execución entre versións (con e sen particionamento espacial).

En concreto, para unha resolución de 1000 metros e datos de 80 días os tempos de execución para a versión sen particionamento chegan a ser entre seis e sete veces superiores ós tempos de execución da versión con particionamento. Ademais, podemos apreciar que a versión sen particionamento non escala ó aumentar o número de executores, senón que nalgúns casos empeora.

Polo tanto o uso do particionamento espacial demostra ser unha estratexia que pode acelerar drasticamente os tempos de execución do noso *pipeline* de operacións para calcular o risco de incendios, tendo en conta que levamos a cabo varias unións de tipo espacial entre os nosos conxuntos de datos.

Conclusións

Afrontabamos neste traballo o reto de elaborar un prototipo de solución software que permitise o procesamento de grandes cantidades de datos de tipo medioambiental de xeito eficiente dando solución a dous problemas fundamentais:

1) A integración de fontes diversas de datos, xa fosen de natureza relacional ou grandes arrays de valores numéricos almacenados en arquivos estandarizados.
2) O procesamento eficiente desas inxentes cantidades de datos mediante a utilización dalgún *framework* de computación distribuída.

Para elo valémonos do caso de uso do cálculo dun risco de incendio para o conxunto do territorio galego. A implementación que levamos a cabo permitiunos explorar unha vía de solución ós dous problemas citados:

1) A utilización dun deseño de clases flexible e desacoplado favorece a integración de novas fontes de datos e formatos de arquivo. Grazas a Spark, podemos levar a cabo o proceso de extracción, transformación e carga dos datos de forma paralelizada, mentres que o formato de almacenamento Parquet sobre HDFS posibilita a creación dun almacén de datos de tipo columnar, distribuído e replicado, que garante a eficiencia na súa lectura.
2) O procesamento sobre Spark posibilita a realización das operacións requiridas para o cálculo do risco de forma paralelizada. Dunha banda temos comprobado que a utilización de columnas implícitas non proporciona un aumento de rendemento significativo, salvo do caso dun clúster con abundancia de recursos. A existencia de compresión no formato Parquet posibilita ademais a minimización do impacto de almacenar columnas de datos adicionais. Doutra banda, para asegurar a paralelización de certas operacións complexas, como as unións de tipo espacial, tivemos que analizar a baixo nivel o comportamento de Spark para elaborar algoritmos que nos permitisen implementar algún tipo de distribución e indexación espacial sobre os nosos datos.

Como resumo específico dos resultados de rendemento podemos concluír, xa que logo:

- O uso de Parquet como formato de almacenamento columnar ofrece resultados superiores a outros formatos, xa que se adapta eficientemente ás estruturas de datos utilizadas.
- O feito de non almacenar columnas implícitas, como por exemplo xeometrías ou series temporais, non ofrece un aforro significativo de espazo de almacenamento e pode significar un aumento non desprezable dos tempos de execución.
- A distribución espacial dos datos é fundamental á hora de conseguir o paralelismo desexado nas operacións de procesamento, especificamente con Spark.

Por outra parte, o desenvolvemento dun módulo de exportación, tamén flexible e desacoplado, permitiunos satisfacer a necesidade de visualizar os datos, xerando arquivos estandarizados que se poden importar en calquera das ferramentas *open source* que existen para a exploración de datos de tipo medioambiental.

Como se menciona na introdución, os resultados deste traballo foron enviados como candidatura para presentación nun congreso de nivel internacional [10] e noutro de nivel nacional [11], onde foron avaliados por expertos na materia. En ámbolos casos foron aceptados, e ademais a presentación a nivel internacional xa foi realizada, a data de depósito oficial desta memoria.

A realización deste traballo suscita algunhas posibles vías de continuación futura:

a) Por unha parte cómpre xeneralizar o algoritmo de particionamento espacial que temos desenvolvido a calquera tipo de operación espacial que se poida presentar noutros casos de uso do sistema.
b) Por outra parte cómpre aumentar o rango de fontes relacionais e estándares de arquivos soportados tanto no módulo de importación como no módulo de exportación.
c) Unha cuestión adicional sería a de engadir ó sistema capacidades de almacenamento e procesamento multirresolución a partir das características dos datos definidos polo usuario para aumentar a súa eficiencia.

Bibliografía

1. Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., e Saltz, J.: *Hadoop GIS: A high performance spatial data warehousing system over MapReduce*. Proc. VLDB Endow., 6(11):1009-1020, agosto 2013.
2. Amazon: https://www.amazon.es/. Accedida: maio de 2016.
3. Apache Hadoop: https://hadoop.apache.org/. Accedida: febreiro de 2016.
4. Apache Spark: http://spark.apache.org/. Accedida: febreiro de 2016.
5. Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., and Widmann, N.: *The multidimensional database system rasdaman*. En Proceedings of the 1998 ACM SIGMOD international conference on Management of data, SIGMOD '98, páxinas 575-577, New York, NY, USA, 1998. ACM.
6. Boehm, B.W.: *Software Risk Management*. En Ghezzi, C.; McDermid, J. A. *Proceedings of 2nd European Software Engineering Conference*. ESEC'89. LNCS. pp. 1–19. doi:10.1007/3-540-51635-2_29. ISBN 3-540-51635-2. ISSN 0302-9743.
7. Cassandra: http://cassandra.apache.org/. Accedida: maio de 2016.
8. Diario Expansión: http://www.expansion.com/economia-digital/innovacion/2015/12/04/5661c421e2704ee52c8b45e1.html, accedida: febreiro de 2016.
9. Eldawy, A. e Mokbel, M.F.: *Pigeon: A spatial MapReduce language*. En Data Engineering (ICDE), 2014 IEEE 30th International Conference, páxinas 1242-1245, marzo 2014.
10. Ferrón, D., Villarroya, S., Viqueira, J.R.R. e Pena, A.T.F.: *Towards large scale environmental data processing with Apache Spark*. En 20th Pacific Asia Conference on Information Systems (PACIS 2016), Chiayi, Taiwan, 27 Xuño – 1 Xullo, 2016. Ranking CORE A. Enlace: http://www.pacis2016.org/Abstract/ALL/673.pdf.
11. Ferrón, D., Villarroya, S., Viqueira, J.R.R. e Pena, A.T.F.: *Procesamiento paralelo de datos medioambientales con Apache Spark*. En XXI Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2016), V Congreso Español de Informática (CEDI 2016), Salamanca, España, 13-16 de Setembro, 2016. Aceptado.
12. GeoSpark: http://geospark.datasyslab.org/. Accedida: febreiro de 2016.
13. HBase: https://hbase.apache.org/. Accedida: maio de 2016.
14. Horn, B.K.P. (1982): *Hill shading and the reflectance map.* Geo-processing, 2(1), 65-146.
15. Infojobs: https://www.infojobs.net/. Accedida: maio de 2016.
16. Karau, H., Konwinski, A., Wendell, P. e Zaharia, M. (2015): *Learning Spark: Lightning-Fast Big Data Analysis* (1st ed.). O'Reilly Media. ISBN: 978-1-4493-5862-4.
17. Parquet: https://parquet.apache.org/. Accedida: maio de 2016.
18. Paul G. Brown. *Overview of scidb: large scale array storage, processing and analysis*. En Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, SIGMOD '10, páxs. 963-968, New York, NY, USA, 2010. ACM.
19. Postgis: http://postgis.net/. Accedida: febreiro de 2016.
20. QGIS: http://www.qgis.org/en/site/. Accedida: maio de 2016.
21. Rasdaman: http://www.rasdaman.com/. Accedida: febreiro de 2016.
22. SciDB: http://www.paradigm4.com/technology/. Accedida: febreiro de 2016.

23. Sommerville, I. (2011): *Software Engineering*, ed. Pearson, 9th edition, páxs. 369-375 (especificación de riscos) e páxs. 94-99 (especificación de requisitos).
24. SpatialHadoop: http://spatialhadoop.cs.umn.edu/. Accedida: febreiro de 2016.
25. Tnooz: http://www.tnooz.com/article/big-data-airlines. Accedida: febreiro de 2016.
26. Tom White (June 16, 2015). *Hadoop: The Definitive Guide* (4th ed.). O'Reilly Media.
27. Villarroya, S., Viqueira, J.R.R., Regueiro, M.A., Taboada, J.A., e Cotos, J.M.: *Soda: A framework for spatial observation data analysis*. Distributed and Parallel Databases, páxinas 1-35, 2014.
28. Villarroya, S., Viqueira, J.R.R., Regueiro, M.A. e Cotos, J.M.: *Spatio-temporal integrated analysis with mapal*. En Beniamino Murgante, Sanjay Misra, Ana María A.C. Rocha, Carmelo Torre, Jorge Gustavo Rocha, María Irene Falcao, David Taniar, Bernady O. Apduhan, e Osvaldo Gervasi, (editores), Computational Science and Its Applications - ICCSA 2014, volume 8579 de Lecture Notes in Computer Science, páxs. 283-297. Springer International Publishing, 2014.
29. Yu, J., Wu, J. and Sarwat, M.: *GeoSpark: A Cluster Computing Framework for Processing Large-Scale Spatial Data*. En Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '15 (2015).

Anexo I - Manual de usuario

Para executar os diferentes módulos do sistema cómpre ter instalado:

a) O *Java Runtime Environment* na súa versión 1.7 ou superior (recoméndase a 1.8), que pode descargarse da seguinte páxina:

   http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

b) No caso da execución en local, a versión *standalone* de Spark (1.6 ou superior) con Hadoop incluído, cuxas instrucións de descarga e instalación se indican na seguinte páxina:

   http://spark.apache.org/docs/latest/

   No caso de querer executalo nun clúster, este debe ter instalado HDFS e unha versión de Spark igual ou superior á 1.6, xunto coa versión de JRE 1.7 ou superior.

Os módulos software consisten en paquetes *.jar* coas correspondentes clases Java compiladas que se executan como aplicacións en Spark mediante a liña de comandos. Todos os módulos reciben como argumento a ruta absoluta ó directorio de traballo, que debe conter no caso do módulo ETL un ficheiro *sources.xml* no que o usuario definiu previamente, axudándose dun editor de texto, as súas fontes de datos (segundo o esquema indicado no apartado "Iteración 1" da memoria):

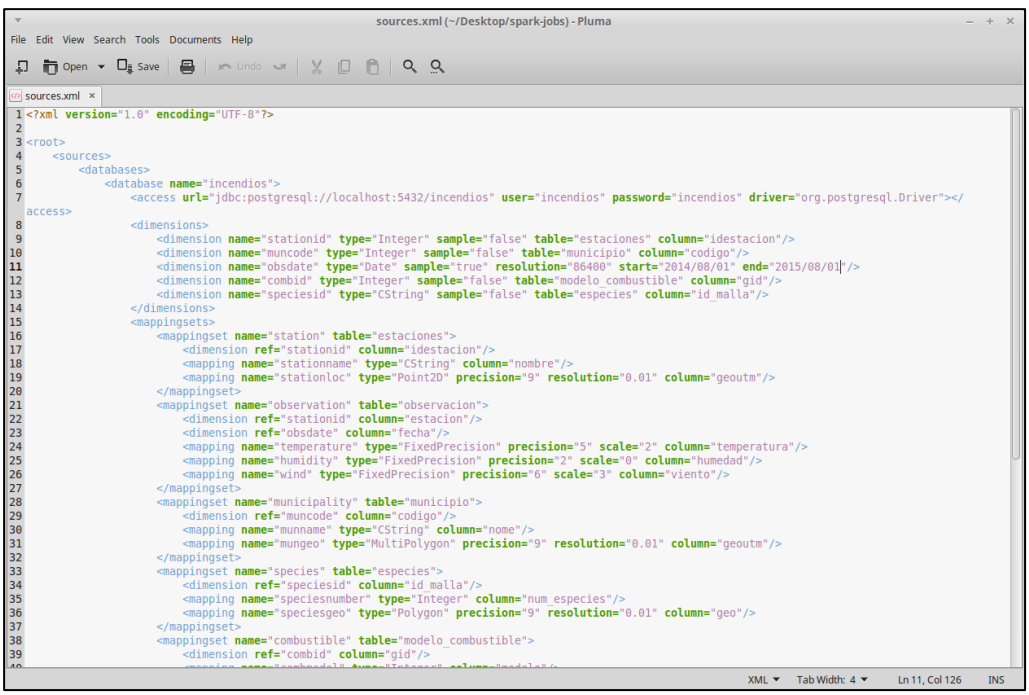


Figura 33.- Exemplo de ficheiro de definición de fontes de datos.

No caso do módulo de procesamento o directorio de traballo debe conter os ficheiros Parquet e o catálogo do sistema exportados previamente co módulo ETL (recoméndase utilizar o mesmo directorio para todos os módulos para evitar a copia manual dos arquivos). Por último, para o módulo de exportación o directorio de traballo debe conter o ficheiro Parquet co índice de risco calculado e exportando mediante o módulo OLAP. O comando de execución de aplicacións en Spark é o seguinte:

```
$SPARK_HOME/bin/spark-submit \
        --driver-cores $SPARK_DRIVER_CORES \
        --driver-memory $SPARK_DRIVER_MEMORY \
        --executor-cores $SPARK_EXECUTOR_CORES \
        --executor-memory $SPARK_EXECUTOR_MEMORY \
        --num-executors $NUM_EXECUTORS\
        --class $MAIN_CLASS $APP.jar $WORKING_DIR
```

onde *SPARK_HOME* é o directorio raíz de Spark, *SPARK_DRIVER_CORES* e *SPARK_DRIVER_MEMORY* o número de núcleos e a cantidade de memoria que se asignan ó driver, *SPARK_EXECUTOR_CORES* e *SPARK_EXECUTOR_MEMORY* o número de núcleos e a cantidade de memoria que se asignan a cada executor, *NUM_EXECUTORS* o número total de executores a utilizar, *MAIN_CLASS* a clase principal do programa, *APP.jar* o arquivo .jar do módulo e *WORKING_DIR* o directorio de traballo. Os módulos realizados para as probas da iteración reciben un segundo argumento (número enteiro) que lles indica o grado de reparticionamento dos datos a aplicar para obter paralelismo (un valor xeralmente adecuado oscila entre 1000 e 2000).

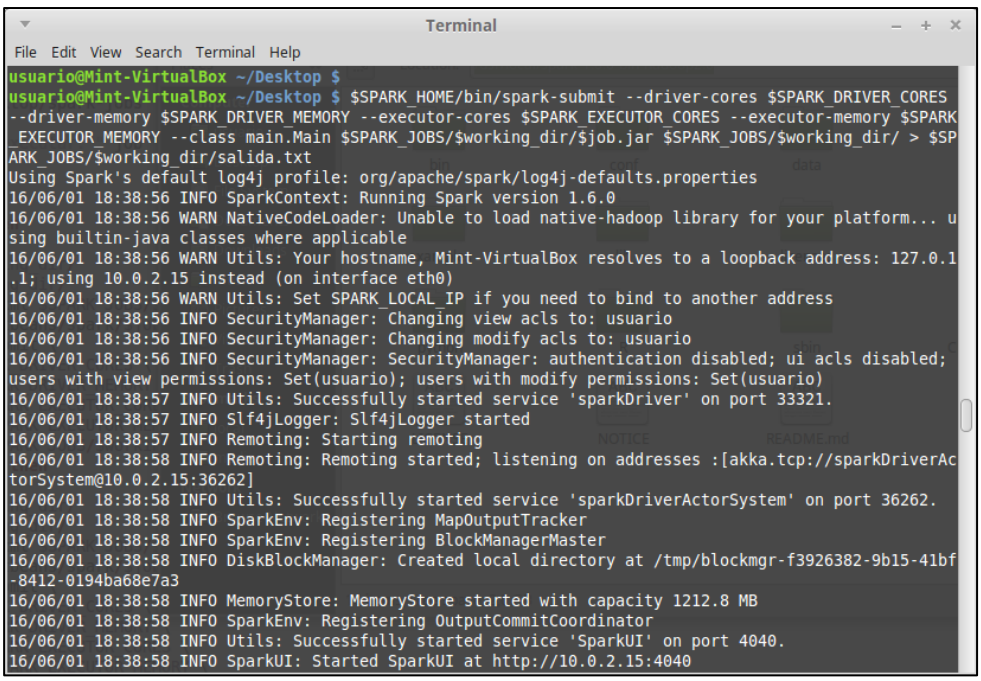


Figura 34.- Execución do módulo ETL mediante terminal.

Aínda que por defecto se mostra información de estado polo terminal, a monitorización en detalle das aplicacións executadas pódese levar a cabo a través da interface web de Spark, cuxa URL por defecto en modo *standalone* é: http://localhost:4040/jobs/.

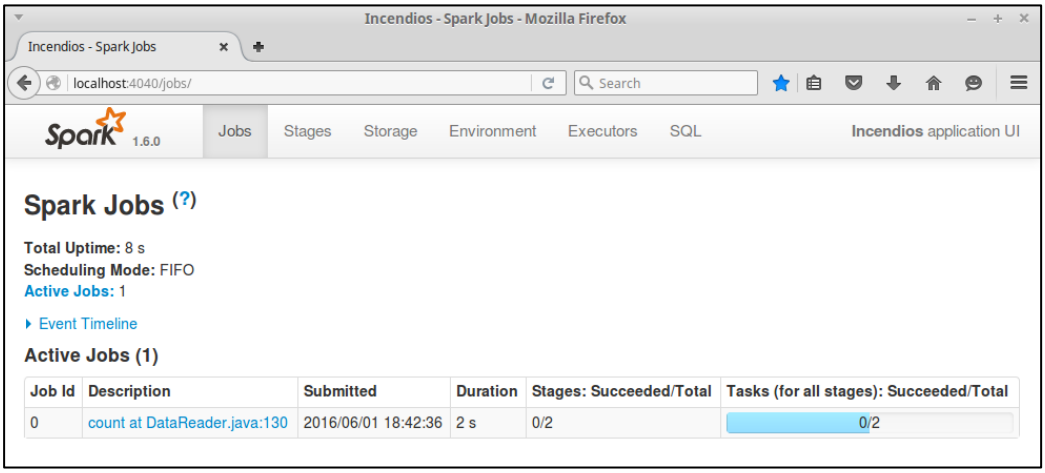


Figura 35.- Monitorización mediante a Spark Web UI.

O arquivo por defecto que xera o módulo de exportación é de tipo Esri ASCII (texto plano) e pódese abrir cun editor de texto convencional. Para poder visualizar o seu contido graficamente en QGIS hai que ter instalado o programa, que se pode descargar aquí:

http://qgis.org/es/site/forusers/download.html

Unha vez instalado, a forma máis directa de abrir un arquivo é arrastralo á barra esquerda lateral da interface de QGIS, titulada “Capas”, como no seguinte exemplo (é posible que pida ó usuario que indique o sistema de coordenadas por defecto):

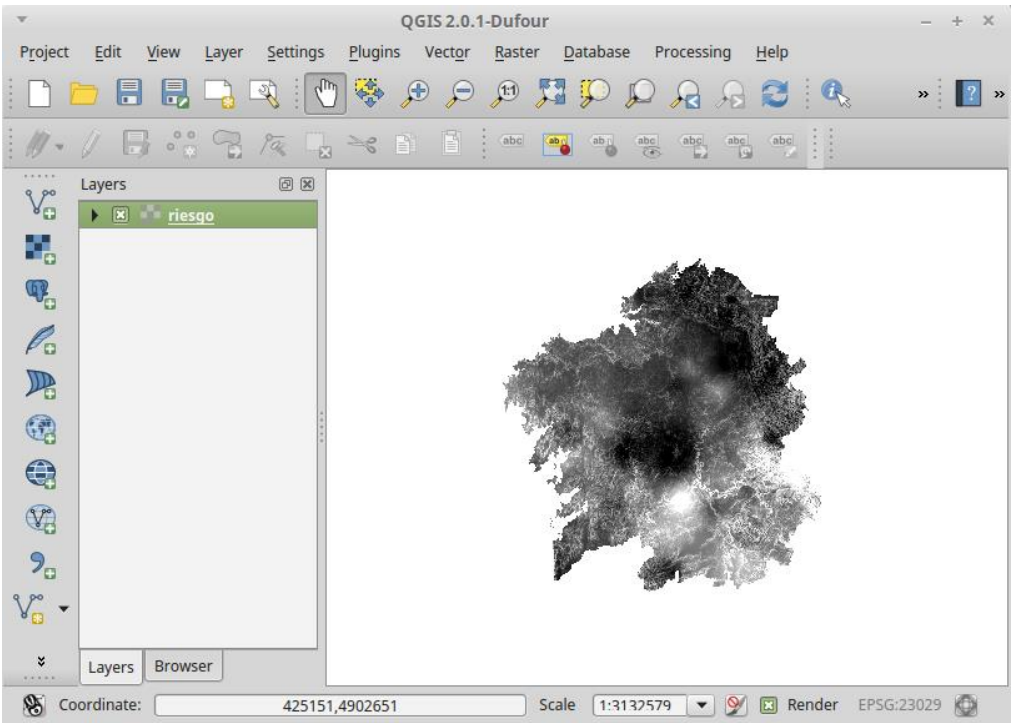


Figura 36.- Interface de usuario en QGIS.

Aquí podemos explorar os puntos e propiedades da imaxe e facer modificacións das capas.

Anexo II - Manual técnico

Para facer probas ou continuar o desenvolvemento do proxecto, todo o indicado no manual de usuario respecto da execución e monitorización dos módulos se mantén. A maiores, os arquivos de desenvolvemento do proxecto están xerados en Netbeans, e polo tanto se se desexa continuar o desenvolvemento con este IDE, cómpre instalar unha versión igual ou superior á 8.0, que se pode descargar da seguinte páxina:

https://netbeans.org/downloads/

O tipo de proxectos creados é *Maven -> Java Application*, xa que facemos uso de Maven para resolver as dependencias de librerías externas tipo Hadoop, Spark, JDBC ou Geotools (librería para os tipos e arquivos xeoespaciais). Estas dependencias están indicadas no arquivo *pom.xml* que se pode atopar na raíz de cada carpeta de proxecto, tendo o seguinte aspecto:

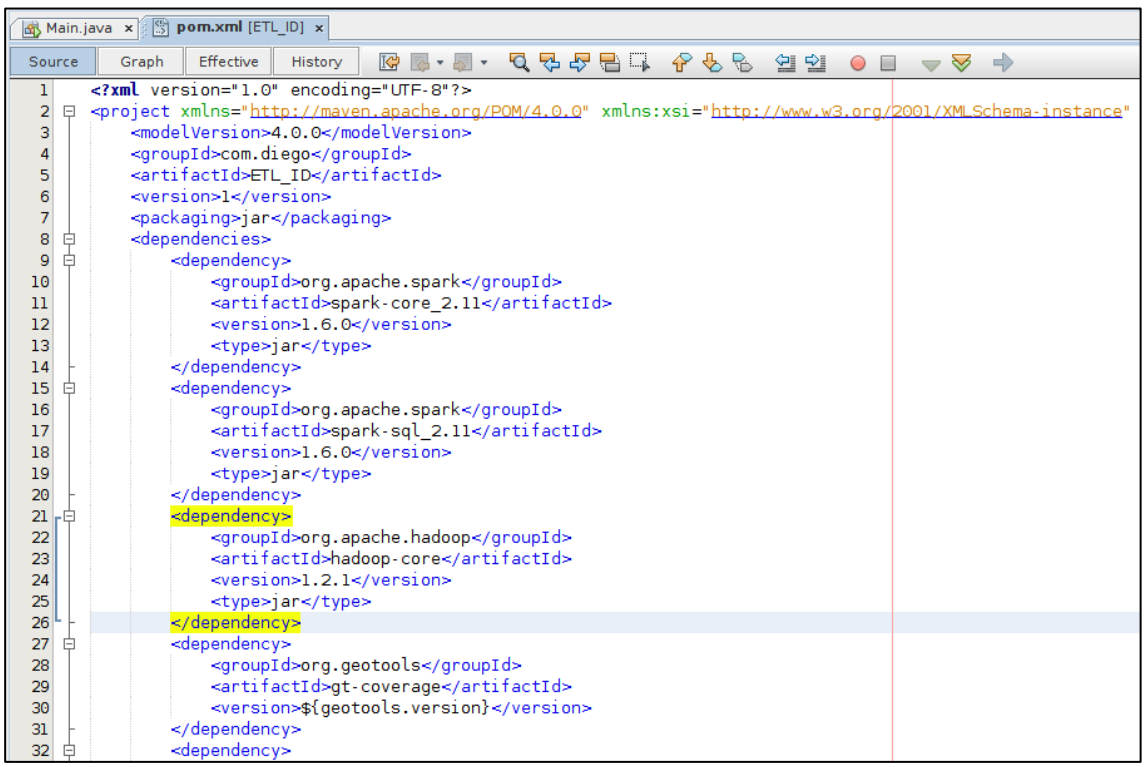


Figura 37.- Exemplo de arquivo pom.xml.

Unha vez importado e configurado o proxecto, simplemente utilizamos a interface do IDE para explorar e modificar o código como desexemos. Cando o teñamos listo, debemos compilar o noso programa para xerar un paquete .jar que contén todas as clases da aplicación compiladas e se pode executar mediante a liña de comandos en Spark. Para crear un .jar con todas as dependencias necesarias e evitar ter que cargar as librerías en tempo de execución en Spark, utilizamos o *shade plugin* de Maven, cuxa configuración se indica (e se pode comprobar) no *pom.xml*. A seguinte captura mostra ese fragmento do *pom.xml* así como a finalización do proceso de compilación:

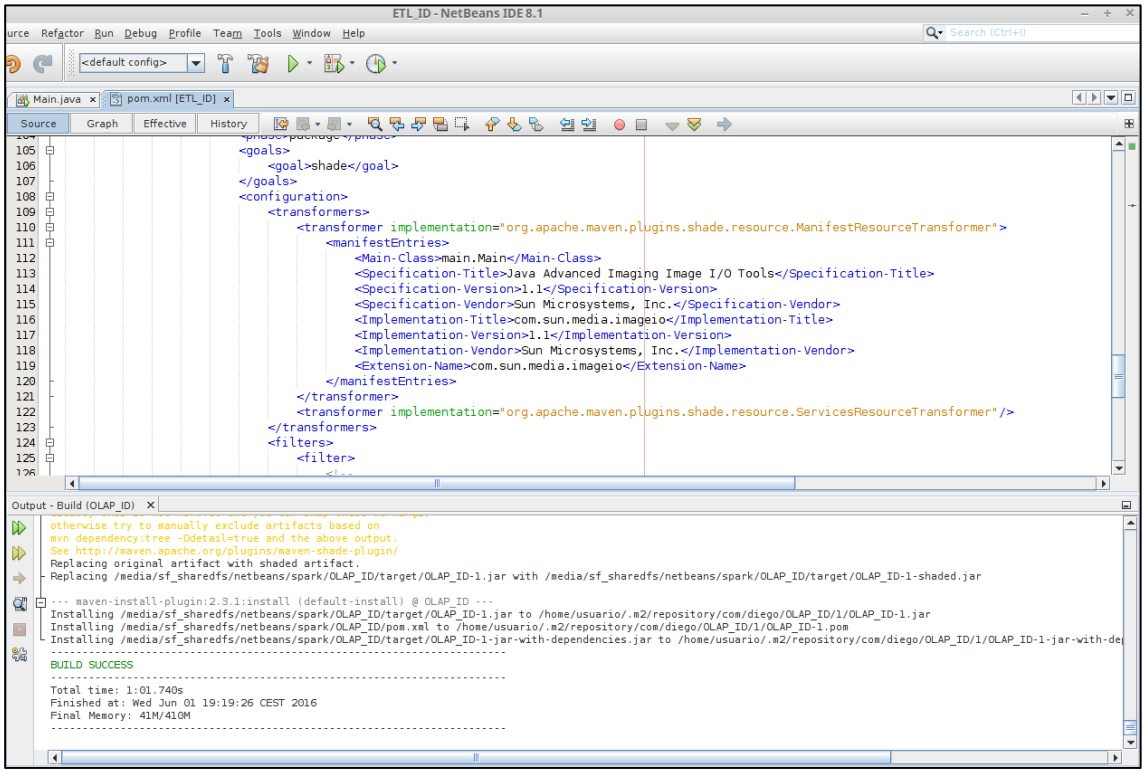


Figura 38.- Compilación do programa mediante o plugin "Shade" de Maven.

O arquivo .jar xerado pódese atopar no subdirectorio target/ da carpeta do proxecto.