UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Título do Traballo de Fin de Grao
Subtítulo do Traballo de Fin de Grao

Autor:
Nome do autor

Directores:
Nome do director
Nome do codirector

Grao en Enxeñaría Informática

Febreiro 2011

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. (Nome do Director), Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela, e **D. (Nome do Codirector)**, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela,

INFORMAN:

Que a presente memoria, titulada *(Título do traballo)*, presentada por **D. (Nome do autor do traballo)** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxeñaría Informática, realizouse baixo nosa dirección no Departamento de Electrónica e Computación da Universidade de Santiago de Compostela.

E para que así conste aos efectos oportunos, expiden o presente informe en Santiago de Compostela, a (Data):

O director, O codirector, O alumno,

(Nome do director) (Nome do Codirector) (Nome do Alumno)

Agradecementos

Se se quere pór algún agradecemento, este vai aquí.

Resumo

Se se quere pór resumo, este vai aquí.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introdución

Introdución: composta por Obxectivos Xerais, Relación da Documentación que conforma a Memoria, Descrición do Sistema, Información Adicional de Interese (métodos, técnicas ou arquitecturas utilizadas, xustificación da súa elección, etc.).

Capítulo 2

Gestión del proyecto

2.1. Alcance

2.2. Análisis de requisitos

El objetivo principal de este trabajo es el de paralelizar el algoritmo Progressive Hedging del módulo PySP. Para conseguirlo se establecen los siguientes requisitos:

RF-01	Ejecución de trabajos en Spark
Descripción	Ejecutar el algoritmo PH en Spark de forma paralela.
Importancia	Imprescindible

RF-02	Integración con Pyomo
Descripción	La solución implementada debe funcionar como parte del programa Pyomo.
Importancia	Imprescindible

RF-03	Interoperabilidad con funcionalidad previa
Descripción	La nueva implementación no debe impedir el correcto funcionamiento de los módulos de Pyomo existentes.
Importancia	Importante

RF-04	Establecer medición de rendimiento de Spark
Descripción	Cuantificar la diferencia de rendimiento y escalabilidad de la nueva implementación.
Importancia	Importante

2.3. Análisis de riesgos

2.4. Análisis de costes

2.5. Gestión de la configuración

2.6. Planificación temporal

2.6.1. EDT

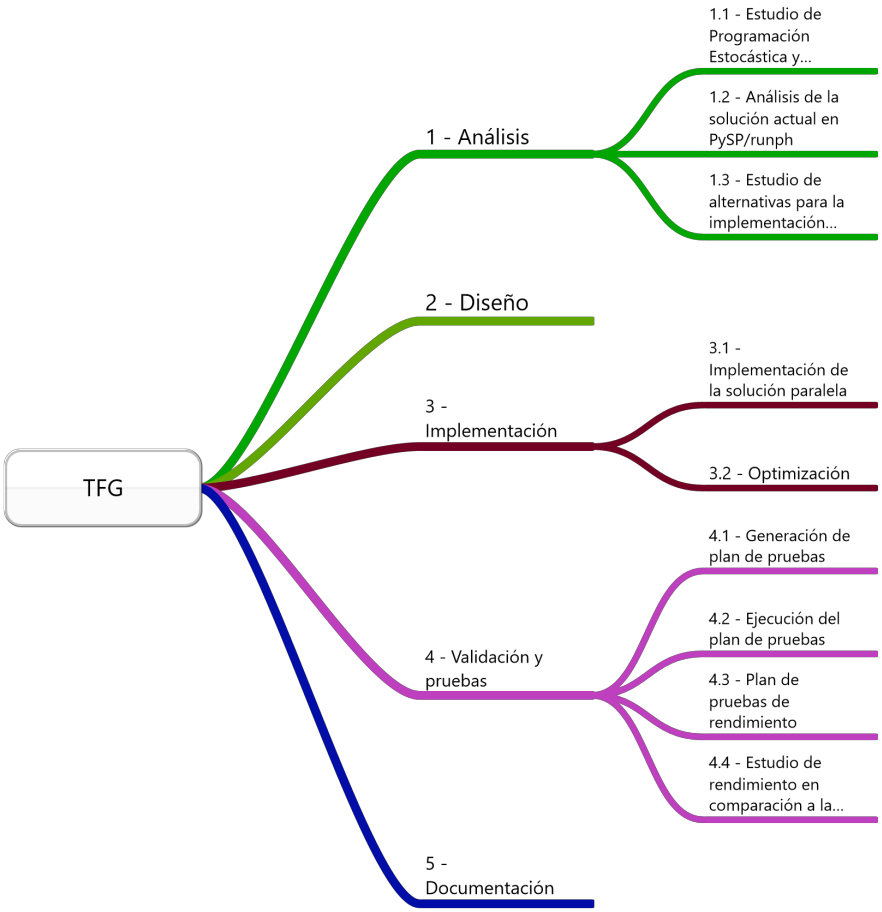


Figura 2.1: EDT inicial

Id	1.1
Tarea	Estudio de Programación Estocástica y Progressive Hedging
Duración	7 días
Descripción	Se investigará el funcionamiento del algoritmo *Progressive Hedging* haciendo uso principalmente de [2] como referencia.
Salida	N/A

Id	1.2
Tarea	Análisis de la solución actual en PySP/runph
Duración	8 días
Descripción	Una vez conocido el funcionamiento teórico del algoritmo, estudiar su implementación sobre el proyecto Pyomo.
Salida	Diagrama de funcionamiento PH [4].

Id	1.3
Tarea	Estudio de alternativas para la implementación paralela
Duración	3 días
Descripción	Se barajarán tecnologías Big Data (Spark) o modelos tradicionales (MPI).
Salida	TODO

Id	2
Tarea	Diseño
Duración	20 días
Descripción	Generar un diseño de la implementación a realizar con la tecnología escogida. Será importante la integración con la implementación actual.
Salida	Diseño de la implementación.

Id	3.1
Tarea	Implementación de la solución paralela
Duración	15 días
Descripción	Escribir los nuevos módulos de código e integrarlos en el proyecto.
Salida	Nuevos ficheros de código y modificaciones a archivos existentes.

Id	3.2
Tarea	Optimización
Duración	5 días
Descripción	Una vez la integración es correcta y la implementación funciona, se realizarán optimizaciones de rendimiento intentando aprovechar las carácterísticas de la tecnología escogida para la nueva implementación paralela.
Salida	Modificaciones a la implementación anterior.

Id	4.1
Tarea	Generación de plan de pruebas
Duración	4 días
Descripción	Idear un plan de pruebas para el nuevo módulo.
Salida	Documento de pruebas [**?**].

Id	4.2
Tarea	Ejecución del plan de pruebas
Duración	1 días
Descripción	Implementar y ejecutar las pruebas establecidas para establecer confianza sobre el correcto funcionamiento de la implementación.
Salida	Informe de ejecución de pruebas.

Id	4.3
Tarea	Plan de pruebas de rendimiento
Duración	3 días
Descripción	Idear un plan de pruebas con problemas que permitan estudiar el rendimiento del programa.
Salida	Documento de pruebas de rendimiento [**?**].

Id	4.4
Tarea	Estudio de rendimiento
Duración	2 días
Descripción	Ejecutar el plan de pruebas anterior y compararlo con las versiones originales tanto secuencial como con Pyro.
Salida	Informe de rendimiento [**?**].

Id	5
Tarea	Documentación
Duración	10 días
Descripción	Generar los documentos asociados al desarrollo del proyecto.
Salida	Memoria del proyecto y documentos asociados.

Tareas no planificadas

Tras las modificaciones realizadas sobre el cronograma y explicadas en Subsección 2.6.3, se generan nuevas tareas para el proyecto:

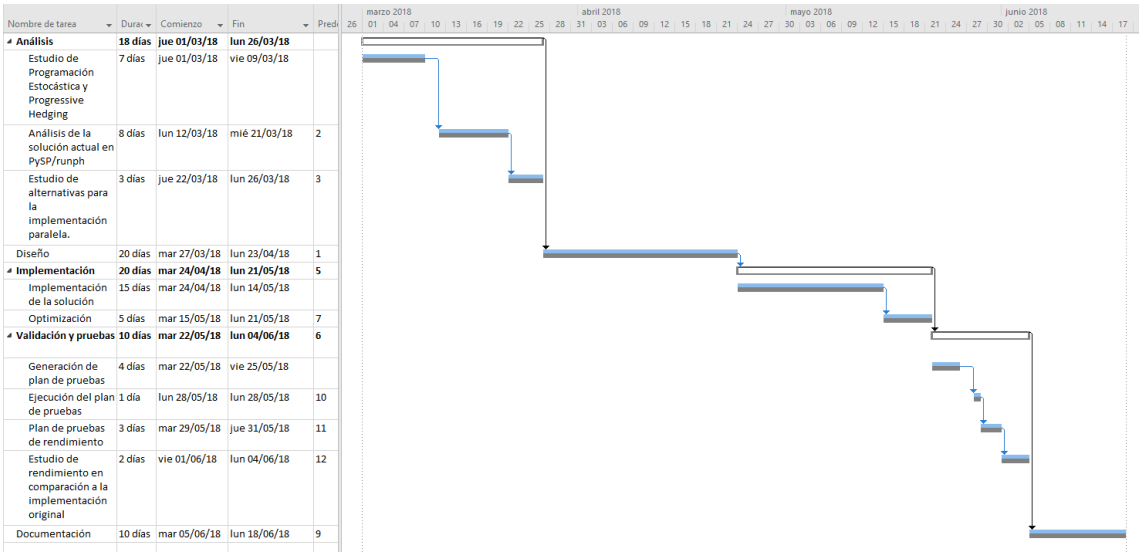
Id	3.*
Tarea	Prototipo aislado
Duración	10 días
Descripción	Crear una aplicación en Python que interactúe con Spark de forma similar a como lo hará la implementación en Pyomo.
Salida	Proyecto de pruebas en Python [5].

Id	3.*
Tarea	Prototipo de integración inicial
Duración	5 días
Descripción	Comenzar la implementación sobre Pyomo comprobando que la integración del nuevo módulo con el código existente es correcta y el flujo de ejecución es correcto con respecto al funcionamiento anterior.
Salida	Código añadido a Pyomo.

Id	3.*
Tarea	Prototipo funcional
Duración	10 días
Descripción	Modificar el prototipo anterior añadiendo las funcionalidades esperadas del programa.
Salida	Código añadido a Pyomo.

2.6.2. Cronograma inicial

Para la realización del trabajo se plantea un ciclo de vida en cascada. Este ciclo de vida nos permitirá realizar un seguimiento del progreso del proyecto en función del tiempo disponible.



Nombre de tarea	Dura	Comienzo	Fin	Pred
Análisis	**18 días**	**jue 01/03/18**	**lun 26/03/18**	
Estudio de Programación Estocástica y Progressive Hedging	7 días	jue 01/03/18	vie 09/03/18	
Análisis de la solución actual en PySP/runph	8 días	lun 12/03/18	mié 21/03/18	2
Estudio de alternativas para la implementación paralela.	3 días	jue 22/03/18	lun 26/03/18	3
Diseño	20 días	mar 27/03/18	lun 23/04/18	1
Implementación	**20 días**	**mar 24/04/18**	**lun 21/05/18**	**5**
Implementación de la solución	15 días	mar 24/04/18	lun 14/05/18	
Optimización	5 días	mar 15/05/18	lun 21/05/18	7
Validación y pruebas	**10 días**	**mar 22/05/18**	**lun 04/06/18**	**6**
Generación de plan de pruebas	4 días	mar 22/05/18	vie 25/05/18	
Ejecución del plan de pruebas	1 día	lun 28/05/18	lun 28/05/18	10
Plan de pruebas de rendimiento	3 días	mar 29/05/18	jue 31/05/18	11
Estudio de rendimiento en comparación a la implementación original	2 días	vie 01/06/18	lun 04/06/18	12
Documentación	10 días	mar 05/06/18	lun 18/06/18	9

Figura 2.2: Línea base

2.6.3. Modificaciones al cronograma inicial

Se ha realizado una estimación temporal inicial poco precisa por no utilizar ningún tipo de método probado ni datos concretos.

Esta es la razón principal para los retrasos que se explican a continuación.

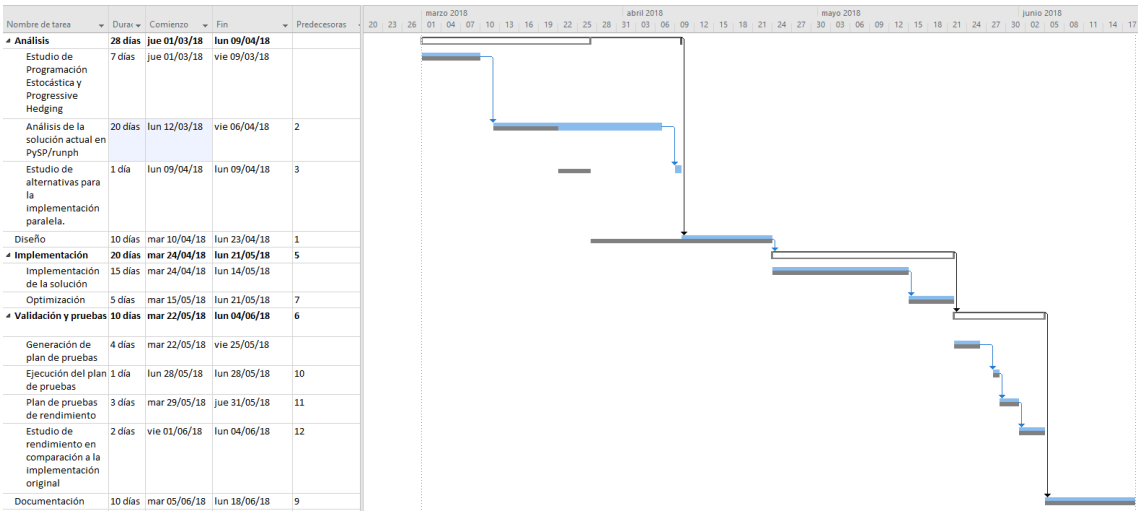
Retraso en análisis

El primer retraso se produce en la fase de análisis de la implementación actual. En esta fase se debe estudiar el funcionamiento del proyecto Pyomo y, en concreto, el módulo de resolución de problemas mediante Progressive Hedging.

A pesar de conocer el funcionamiento teórico del algoritmo mediante [**?**], Pyomo es un proyecto complejo, con multitud de funcionalidades para resolver otros tipos de problemas, soporte para plugins, etc. Todo esto hace que la complejidad del código aumente y sea necesario estudiar múltiples capas de abstracción para entender correctamente el funcionamiento del programa.

Otra complicación añadida es el personal desconocimiento del lenguaje Python previo a la realización de este trabajo.

Tras este primer retraso se intenta ajustar la planificación reduciendo el tiempo de diseño a la mitad:



Nombre de tarea	Dura	Comienzo	Fin	Predecesoras
Análisis	**28 días**	**jue 01/03/18**	**lun 09/04/18**	
Estudio de Programación Estocástica y Progressive Hedging	7 días	jue 01/03/18	vie 09/03/18	
Análisis de la solución actual en PySP/runph	20 días	lun 12/03/18	vie 06/04/18	2
Estudio de alternativas para la implementación paralela.	1 día	lun 09/04/18	lun 09/04/18	3
Diseño	10 días	mar 10/04/18	lun 23/04/18	1
Implementación	**20 días**	**mar 24/04/18**	**lun 21/05/18**	**5**
Implementación de la solución	15 días	mar 24/04/18	lun 14/05/18	
Optimización	5 días	mar 15/05/18	lun 21/05/18	7
Validación y pruebas	**10 días**	**mar 22/05/18**	**lun 04/06/18**	**6**
Generación de plan de pruebas	4 días	mar 22/05/18	vie 25/05/18	
Ejecución del plan de pruebas	1 día	lun 28/05/18	lun 28/05/18	10
Plan de pruebas de rendimiento	3 días	mar 29/05/18	jue 31/05/18	11
Estudio de rendimiento en comparación a la implementación original	2 días	vie 01/06/18	lun 04/06/18	12
Documentación	10 días	mar 05/06/18	lun 18/06/18	9

Figura 2.3: Primer retraso

Teniendo en cuenta que los primeros retrasos fueron principalmente causados por el desconocimiento de la tecnología a usar así como de una mala estimación, es muy probable que en las fases siguientes se produzcan otros retrasos. En este punto se considera entonces abandonar el ciclo de vida en cascada. Su principal atractivo era poder ajustarnos a una planificación temporal que nos permita acabar el proyecto dentro de tiempo, pero esta ventaja no se está cumpliendo en la práctica. Buscando reducir el tiempo de implementación con tecnologías que serán usadas por primera vez, se decide adaptar la planificación a un ciclo de vida por prototipos.

La creación de sucesivos prototipos permite ir acostumbrándose a las tecnologías desconocidas, en este caso Python y Spark, así como ir probando el rendimiento y la integración a medida que se desarrolla.

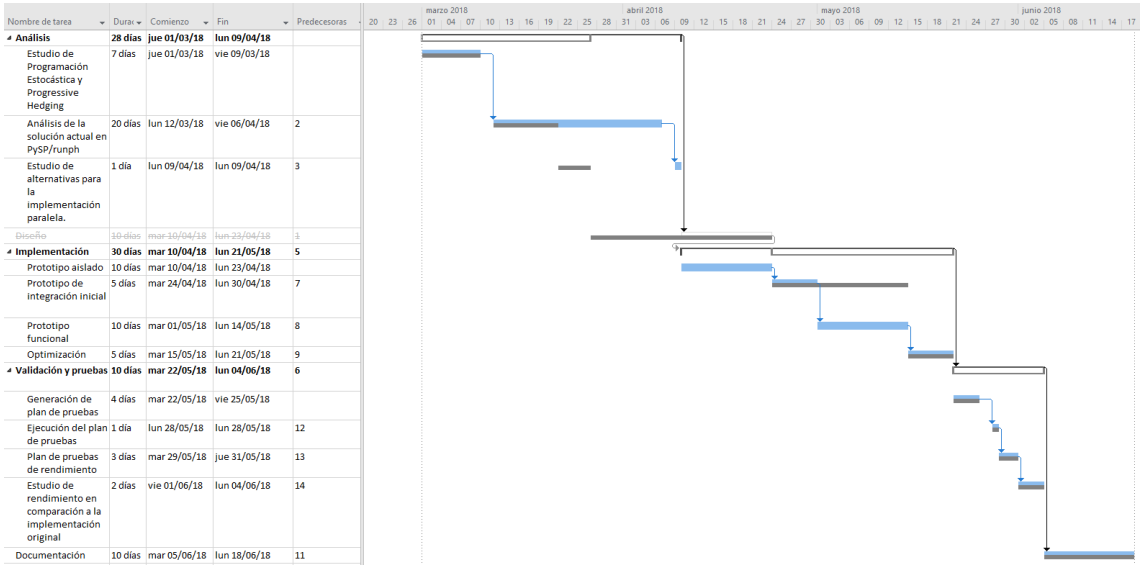
En primer lugar se crea un prototipo aislado para comprobar la implementación de Spark con una arquitectura similar a la que se implementará en Pyomo. Este primer prototipo sirve como aprendizaje de la instalación de Spark y el despliegue de una aplicación en el mismo, así como la implementación en python que interactuará con Spark. Es deseable utilizar una arquitectura de objetos python similar a la que se usará en Pyomo para concretar el uso de Spark y descubrir posibles problemas con la implementación elegida.

Posteriormente se realizarán prototipos sucesivos sobre Pyomo para integrar el nuevo módulo e ir solucionando posibles problemas de rendimiento o funcionamiento que vayan surgiendo.

Dado que la implementación partirá de un protipo inicial de baja calidad será importante realizar una fase de optimización y refactorización al final de la implementación para asegurarse un código final de calidad. Definir la calidad del código no es algo trivial y en este caso calificaremos el código como ”de calidad”si cumple:

- Funciona correctamente y es resistente a errores. Para esto nos apoyaremos en un plan de pruebas funcional.
- Funciona eficientemente y otorga un buen rendimiento, en comparación a las implementaciones existentes. En este caso nos apoyaremos en el plan de pruebas de rendimiento.
- Se integra adecuadamente al proyecto actual. Debe seguir una filosofía de diseño análoga al resto del código así como funcionar correctamente de forma paralela a todo lo implementado previamente.

Tras esta modificación en la planificación, se genera una nueva planificación que podemos ver en la figura y se guardará como una nueva línea base.



Nombre de tarea	Durac	Comienzo	Fin	Predecesoras
Análisis	**28 días**	**jue 01/03/18**	**lun 09/04/18**	
Estudio de Programación Estocástica y Progressive Hedging	7 días	jue 01/03/18	vie 09/03/18	
Análisis de la solución actual en PySP/runph	20 días	lun 12/03/18	vie 06/04/18	2
Estudio de alternativas para la implementación paralela.	1 día	lun 09/04/18	lun 09/04/18	3
~~Diseño~~	~~10 días~~	~~mar 10/04/18~~	~~lun 23/04/18~~	~~1~~
Implementación	**30 días**	**mar 10/04/18**	**lun 21/05/18**	**5**
Prototipo aislado	10 días	mar 10/04/18	lun 23/04/18	
Prototipo de integración inicial	5 días	mar 24/04/18	lun 30/04/18	7
Prototipo funcional	10 días	mar 01/05/18	lun 14/05/18	8
Optimización	5 días	mar 15/05/18	lun 21/05/18	9
Validación y pruebas	**10 días**	**mar 22/05/18**	**lun 04/06/18**	**6**
Generación de plan de pruebas	4 días	mar 22/05/18	vie 25/05/18	
Ejecución del plan de pruebas	1 día	lun 28/05/18	lun 28/05/18	12
Plan de pruebas de rendimiento	3 días	mar 29/05/18	jue 31/05/18	13
Estudio de rendimiento en comparación a la implementación original	2 días	vie 01/06/18	lun 04/06/18	14
Documentación	10 días	mar 05/06/18	lun 18/06/18	11

Figura 2.4: Línea base prototipos

En este punto hemos eliminado la fase de Diseño para poder aumentar el tiempo asignado a Análisis e Implementación. En caso de sufrir más retrasos en la fase de implementación podremos reducir el tiempo asignado a pruebas si el retraso no es grave. En caso de ser un retraso mayor, no cumpliremos la fecha de finalización establecida.

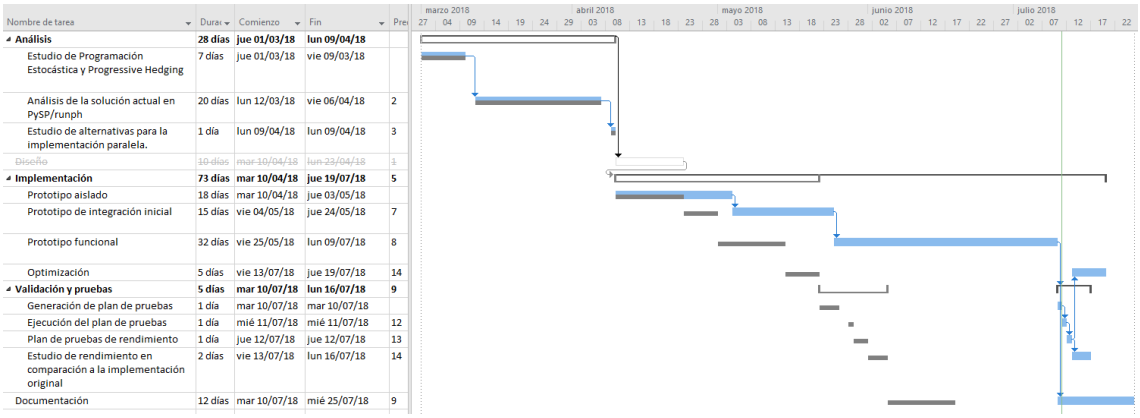
Retraso en implementación

Durante la implementación del prototipo funcional el desarrollo llega a un punto muerto. Las funciones implementadas no devuelven el resultado correcto y se debe hacer una búsqueda de los errores que lo causan. Por falta de experiencia y desconocimiento de las tecnologías, esta fase de implementación se alarga hasta

el día 07/07/2018.

Este retraso sumado a un retraso de 10 días en la creación del prototipo aislado nos fuerza a retrasar la fecha de finalización del proyecto al día 25/07/2018.

Con estos nuevos cambios es necesaria una nueva planificación temporal:



Nombre de tarea	Duraci	Comienzo	Fin	Pred
Análisis	**28 días**	**jue 01/03/18**	**lun 09/04/18**	
Estudio de Programación Estocástica y Progressive Hedging	7 días	jue 01/03/18	vie 09/03/18	
Análisis de la solución actual en PySP/runph	20 días	lun 12/03/18	vie 06/04/18	2
Estudio de alternativas para la implementación paralela.	1 día	lun 09/04/18	lun 09/04/18	3
~~Diseño~~	~~10 días~~	~~mar 10/04/18~~	~~lun 23/04/18~~	~~1~~
Implementación	**73 días**	**mar 10/04/18**	**jue 19/07/18**	**5**
Prototipo aislado	18 días	mar 10/04/18	jue 03/05/18	
Prototipo de integración inicial	15 días	vie 04/05/18	jue 24/05/18	7
Prototipo funcional	32 días	vie 25/05/18	lun 09/07/18	8
Optimización	5 días	vie 13/07/18	jue 19/07/18	14
Validación y pruebas	**5 días**	**mar 10/07/18**	**lun 16/07/18**	**9**
Generación de plan de pruebas	1 día	mar 10/07/18	mar 10/07/18	
Ejecución del plan de pruebas	1 día	mié 11/07/18	mié 11/07/18	12
Plan de pruebas de rendimiento	1 día	jue 12/07/18	jue 12/07/18	13
Estudio de rendimiento en comparación a la implementación original	2 días	vie 13/07/18	lun 16/07/18	14
Documentación	12 días	mar 10/07/18	mié 25/07/18	9

Figura 2.5: Línea base final

Capítulo 3

Planificación e presupostos

Planificación e presupostos: debe incluír a estimación do costo (presuposto) e dos recursos necesarios para efectuar a implantación do Traballo, xunto coa planificación temporal do mesmo e a división en fases e tarefas. Recoméndase diferenciar os costos relativos a persoal dos relativos a outros gastos como instalacións e equipos.

Capítulo 4

Plan de pruebas

4.1. Pruebas funcionales

4.1.1. Introducción

El actual plan de pruebas tiene como objetivo asegurar que los requisitos especificados son cumplidos satisfactoriamente por el módulo de código implementado. Adicionalmente se buscará que el código implementado funcione correctamente cuando dispone de entradas incorrecta.

Este plan de pruebas está basado en el estándar IEEE 829 [3].

4.1.2. Restricciones

La principal restricción de este plan de pruebas es el tiempo. Sólo se dispone de 1 día para la creación del plan y otro día para su ejecución y la recolección de resultados.

La complejidad de las entradas que recibe el módulo limita el alcance de las pruebas de caja negra que podremos diseñar en el tiempo disponible.

4.1.3. Objeto de pruebas

El código a probar será la implementación del gestor de Spark, concretamente la clase *SolverManager_PHSpark*. Su ejecución se realiza mediante el comando *runph* con la opción *–solver-manager=phspark*.

Se realizarán pruebas de caja negra sobre las diferentes opciones de configuración del solver manager. Adicionalmente se ejecutarán los ejemplos existentes en el proyecto y se comparará la solución generada mediante spark y mediante el algoritmo secuencial.

4.1.4. Características a probar y exclusiones

Se probarán las diferentes opciones de configuración que modifican la ejecución de phspark. Con esto verificamos el requisito RF-01 y RF-02.

Adicionalmente se ejecutan los ejemplos existentes para PySP mediante phspark y la implementación secuencial, comparando los resultados. De esta forma se verifica un funcionamiento correcto en distintos escenarios así como el requisito RF-03.

Queda excluído de este plan de pruebas la generación de pruebas mediante estrategias de caja blanca. No podremos asegurar una cobertura óptima de código pero no es posible realizarlas en el tiempo disponible dada la complejidad de las entradas.

4.1.5. Estrategia

Se utilizará el módulo "unittest" de Python para generar una clase que ejecute los casos de prueba diseñados.

Será necesario generar un archivo de salida correcto para usar como punto de comparación con las ejecuciones de pruebas. Será necesario filtrar las diferencais en la salida que causa la ejecución con Spark o, alternativamente, comprobar manualmente el resultado de cada pareja de archivos.

4.1.6. Criterios de aceptación

Se considerará que una prueba ha sido pasada correctamente si, tras introducir los datos de entrada escogidos, la aplicación devuelve una salida que concuerda exactamente con la que se estableció como esperada.

Será posible concluir que la aplicación aprobada pasó las pruebas si el 100 % de los casos válidos pasaron los criterios de aceptación. En caso contrario será necesaria la corrección de la implementación.

4.1.7. Diseño de pruebas

Prueba P-01	Conexión a Spark
Objetivo	Valida la creación de una conexión a Spark.
Requisitos validados	RF-01, RF-02
Técnicas aplicadas	▪ **Por caja negra:** • host ◦ 1. Clase válida: Cadena que represente una IP correcta y disponible. ◦ 2. Clase válida: Null. ◦ 3. Clase no válida: Cadena que represente una IP errónea. ◦ 4. Clase no válida: Cadena que represente una IP correcta pero no disponible. • port ◦ 5. Clase válida: Entero que represente un puerto correcto y disponible. ◦ 6. Clase válida: Null. ◦ 7. Clase no válida: Entero que represente un puerto erróneo. ◦ 8. Clase no válida: Entero que represente un puerto correcto pero ocupado.
Casos de prueba	CP-01, CP-02, CP-03, CP-04, CP-05, CP-06
Salida esperada	Conexión correcta y ejecución que genere la salida esperada en el caso de las pruebas para clases válidas. Para las pruebas con entradas no válidas se debe mostrar un error especificando el problema.

Prueba P-02	Ejecución correcta de ejemplos
Objetivo	Valida que los resultados devueltos son correctos.
Requisitos validados	RF-01, RF-02, RF-03
Técnicas aplicadas	Se ejecutarán todos los ejemplos disponibles en *"/pyomo/examples/pysp"* usando la versión secuencial para generar el resultado esperado y la versión Spark para verificar su funcionamiento.
Casos de prueba	CP-01, CP-02, CP-03
Salida esperada	Los resultados de todos los ejemplos son iguales para la versión secuencial y paralela.

4.1.8. Casos de prueba

Caso de prueba	CP-01
Descripción	Prueba la entrada correcta
Clases que valida	P-01(1,5)
Necesidades del entorno	Instancia de Spark funcionando en "spark://localhost:7077"
Entrada	host = "localhost" port = 7077
Salida esperada	Igual a la ejecución con -solver-manager=serial

Caso de prueba	CP-02
Descripción	Prueba la url por defecto
Clases que valida	P-01(2,6)
Necesidades del entorno	Instancia de Spark funcionando en "spark://localhost:7077"
Entrada	host = None port = None
Salida esperada	Igual a la ejecución con -solver-manager=serial

Caso de prueba	CP-03
Descripción	Prueba la gestión de error al intentar conectarse a una IP incorrecta
Clases que valida	P-01(3)
Necesidades del entorno	N/A
Entrada	host = “111.111” port = None
Salida esperada	Mensaje de error indicando que no se ha podido conectar a Spark en la URL especificada.

Caso de prueba	CP-04
Descripción	Prueba la gestión de error al intentar conectarse a una IP donde no se está ejecutando Spark
Clases que valida	P-01(4)
Necesidades del entorno	La URL “spark://localhost:7077” no debe tener ningún servicio escuchando.
Entrada	host = “localhost” port = 7077
Salida esperada	Mensaje de error indicando que no se ha podido conectar a Spark en la URL especificada.

Caso de prueba	CP-05
Descripción	Prueba la gestión de error al especificar un puerto inválido
Clases que valida	P-01(7)
Necesidades del entorno	N/A
Entrada	host = None port = -1
Salida esperada	Mensaje de error indicando que no se ha podido conectar a Spark en la URL especificada.

Caso de prueba	CP-06
Descripción	Prueba la gestión de error al especificar un puerto donde no se está ejecutando Spark
Clases que valida	P-01(8)
Necesidades del entorno	La URL "spark://localhost:8080" no debe ser una instancia de Spark
Entrada	host = "localhost" port = 8080
Salida esperada	Mensaje de error indicando que no se ha podido conectar a Spark en la URL especificada.

4.1.9. Procedimiento de pruebas

Los casos de prueba especificados son idempotentes y no es necesario especificar un orden concreto para su ejecución. Será necesario para cada uno establecer el entorno especificado en cada caso concreto. Adicionalmente, para verificar los resultados será necesario ejecutar la misma entrada de forma secuencial para comparar.

Para la prueba P-02 no se especifican casos de prueba concretos porque todos se ejecutan de la misma forma cambiando el directorio de entrada. Para esta prueba se ejecutarán todos los ejemplos disponibles para PySP de forma secuencial y, posteriormente, usando Spark. Si Spark devuelve siempre los mismos resultados (dentro de un margen de error) se considerará la prueba como superada.

4.1.10. Ejecución de las pruebas

4.2. Pruebas de rendimiento

4.2.1. Introducción

4.2.2. Restricciones

4.2.3. Objeto de pruebas

4.2.4. Características a probar y exclusiones

4.2.5. Estrategia

4.2.6. Criterios de aceptación

4.2.7. Diseño de pruebas

4.2.8. Casos de prueba

4.2.9. Procedimiento de pruebas

4.2.10. Ejecución de las pruebas

Capítulo 5

Especificación de requisitos

Especificación de requisitos: debe indicarse, polo miúdo, a especificación do Sistema, xunto coa información que este debe almacenar e as interfaces con outros Sistemas, sexan hardware ou software, e outros requisitos (rendemento, seguridade, etc).

Capítulo 6

Deseño

Deseño: cómo se realiza o Sistema, a división deste en diferentes compoñentes e a comunicación entre eles. Así mesmo, determinarase o equipamento hardware e software necesario, xustificando a súa elección no caso de que non fora un requisito previo. Debe achegarse a un nivel suficiente de detalle que permita comprender a totalidade da estrutura do produto desenvolvido, utilizando no posible representacións gráficas.

Capítulo 7

Exemplos

7.1. Un exemplo de sección

Esta é *letra cursiva*, esta é **letra negrilla**, esta é letra subrallada, e esta é `letra curier`. Letra tiny, scriptsize, small, large, Large, LARGE e moitas más. Exemplo de fórmula: $a = \int_o^\infty f(t)dt$. E agora unha ecuación aparte:

$$S = \sum_{i=0}^{N-1} a_i^2. \tag{7.1}$$

As ecuaciones se poden referenciar: ecuación (7.1).

7.1.1. Un exemplo de subsección

O texto vai aquí.

7.1.2. Otro exemplo de subsección

O texto vai aquí.

Un exemplo de subsubsección

O texto vai aquí.

Un exemplo de subsubsección

O texto vai aquí.

Un exemplo de subsubsección

O texto vai aquí.

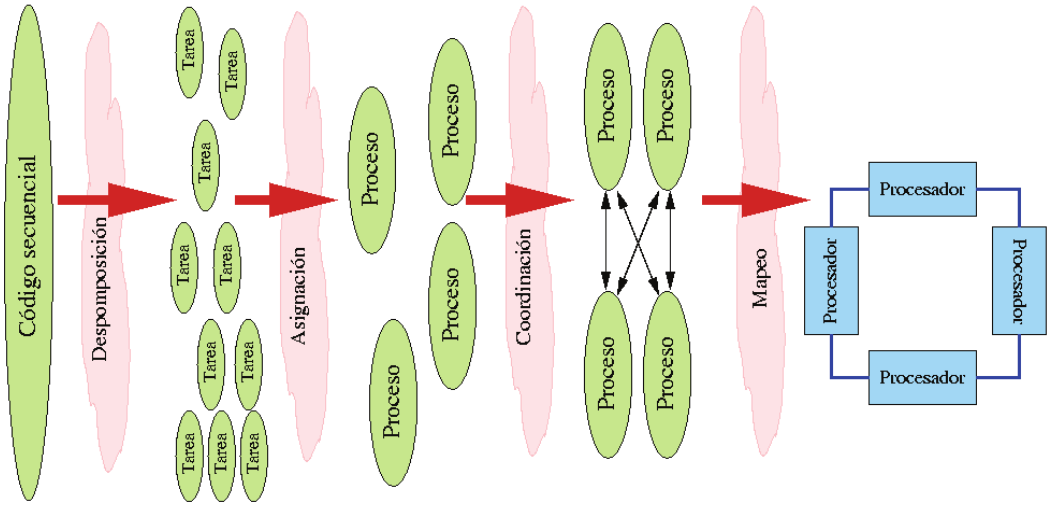


Figura 7.1: Esta é a figura de tal e cal.

Izquierda	Derecha	Centrado
ll	r	cccc
llll	rrr	c

Cuadro 7.1: Esta é a táboa de tal e cal.

7.2. Exemplos de figuras e cadros

A figura número 7.1.
O cadro (taboa) número 7.1.

7.3. Exemplos de referencias á bibliografía

Este é un exemplo de referencia a un documento descargado da web [**?**]. E este é un exemplo de referencia a unha páxina da wikipedia [**?**]. Agora un libro [**?**] e agora unha referencia a un artigo dunha revista [**?**]. Tamén se poden pór varias referencias á vez [**?**, **?**].

7.4. Exemplos de enumeracións

Con puntos:

- Un.
- Dous.

- Tres.

Con números:

1. Catro.
2. Cinco.
3. Seis.

Exemplo de texto verbatim:

```
O texto        verbatim
     se visualiza tal
            como se escribe
```

Exemplo de código C:

```c
#include <math.h>
main()
{   int i, j, a[10];
    for(i=0;i<=10;i++) a[i]=i; // comentario 1
    if(a[1]==0) j=1; /* comentario 2 */
    else j=2;
}
```

Exemplo de código Java:

```java
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

Capítulo 8

Conclusións e posibles ampliacións

Conclusións e posibles ampliacións

Apéndice A

Manuais técnicos

Manuais técnicos: en función do tipo de Traballo e metodoloxía empregada, o contido poderase dividir en varios documentos. En todo caso, neles incluirase toda a información precisa para aquelas persoas que se vaian a encargar do desenvolvemento e/ou modificación do Sistema (por exemplo código fonte, recursos necesarios, operacións necesarias para modificacións e probas, posibles problemas, etc.). O código fonte poderase entregar en soporte informático en formatos PDF ou postscript.

Apéndice B

Manuais de usuario

Manuais de usuario: incluirán toda a información precisa para aquelas persoas que utilicen o Sistema: instalación, utilización, configuración, mensaxes de erro, etc. A documentación do usuario debe ser autocontida, é dicir, para o seu entendemento o usuario final non debe precisar da lectura de outro manual técnico.

Apéndice C

Licenza

Se se quere pór unha licenza (GNU GPL, Creative Commons, etc), o texto da licenza vai aquí.

Bibliografía

[1] Matei Zaharia, Patrick Wendell, Andy Konwinski, Holden Karau, *Learning Spark: Lightning-Fast Big Data Analysis*, 1ª edición, O'Reilly Media, 2015.

[2] John R. Birge, François Louveaux, *Introduction to Stochastic Programming*, 2ª Edición, Springer, 2011.

[3] IEEE Std 829-2008, IEEE Standard for Software and System Test Documentation

[4] PONER AQUÍ LA RUTA

[5] Proyecto de pruebas en: "/DistributedTest"