Anexo 4: Arquitectura BigTexts

Ing. Wilson Alzate Calderón[*1] and Ing. Alexandra Pomares Ph.D.[**1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana

5 de diciembre de 2014

Índice

[*]walzate@javeriana.edu.co
[**]pomares@javeriana.edu.co

Índice de figuras

Índice de cuadros

1. Introducción

Teniendo como partida la pregunta de investigación ¿Cómo generar un framework que ofrezca funcionalidades pre-construidas para ejecutar tareas de pre-procesamiento en minería de texto basado en tecnologías de Big Data, en donde se logre adicionar un nivel de abstracción a la complejidad que implica tener que enlazar las diferentes tecnologías disponibles y así poder mejorar los tiempos de procesamiento en sistemas de minería de texto?, se propone BigTexts, un modelo de aplicación de Big Data que soporta la fase de pre-procesamiento de manera paralela, haciéndola más rápida y eficiente como se muestra en la figura 1.

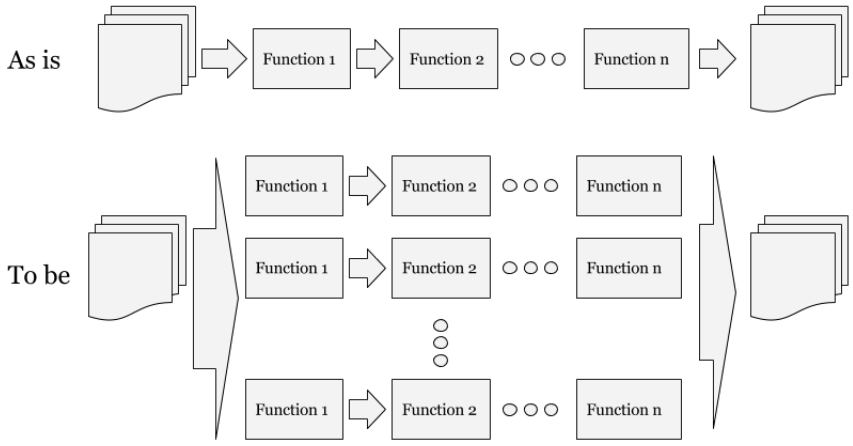

Figura 1: Situación actual vs situación deseada

2. Alcance

Este informe presenta la arquitectura propuesta para desarrollar el proyecto BigTexts: Framework de pre-procesamiento de datos en minería de texto basado en tecnologías de Big Data. Para definirla se utiliza el modelo de Kruchten 4+1 [5], que consiste en un conjunto de 5 vistas para explicar la arquitectura de software.

3. Atributos de Calidad

Los siguientes son los atributos y subatributos de calidad identificados según la clasificación de atributos de calidad de la norma ISO 9126 [7], que se extrajeron y propusieron a partir de los requerimientos.

Prioridad	Atributo	Sub-atributo	Concerns
5	Eficiency	Time behavior	Dado un conjunto n de textos, el tiempo de pre-procesamiento será igual o menor al tiempo de pre-procesamiento secuencial, dividido en el número de máquinas disponibles en el clúster. Donde las máquinas disponibles en el clúster son al menos tan robustas como la máquina inicial
4	Functionality	Interoperability	El Framework debe ser usado por otros sistemas en forma de API o librería
3	Usability	Understandability	El framework debe usar los patrones de usabilidad Cancel, Alerts, Status indication, Tooltips, Help y Undo
2	Reliability	Recoverability	El framework debe ser capaz de volver a lanzar procesos fallidos
1	Maintainability	Changeability	El sistema debe tener la capacidad de fácilmente agregar un nuevo algoritmo de pre-procesamiento

Cuadro 1: Atributos de calidad

4. Estilo Arquitectónico

Para seleccionar los estilos arquitectónicos relevantes para los atributos de calidad identificados se utilizó la siguiente tabla donde se califica cada atributo de calidad (con su respectiva prioridad) en comparación con los estilos arquitectónicos tanto estructurales como para sistemas interactivos. Las calificaciones se basaron en el artículo *A method of selecting appropriate software architecture styles:Quality Attributes and Analytic Hierarchy Process* [8],de la siguiente manera:

- 2: El estilo arquitectónico tiene impacto MUY positivo sobre un atributo de calidad.
- 1: El estilo arquitectónico tiene impacto positivo sobre un atributo de calidad.
- 0: Representa que no tiene impacto, es neutral o no especificado.
- -1: El estilo arquitectónico tiene impacto negativo sobre un atributo de calidad.
- -2: El estilo arquitectónico tiene impacto MUY negativo sobre un atributo de calidad.

		Eficiency (Time behavior)	Functionality (Interoperability)	Usability (Understandability)	Reliability (Recoverability)	Maintainability (Changeability)	Total
		5	4	3	2	1	
Estructurales	**Cliente / Servidor**	2	0	0	0	-1	**9**
	Pipes and filters	0	0	-2	0	1	**-5**
	Layers	-2	0	0	0	2	**-8**
Sistemas interactivos	**Model-View-Controller**	0	0	1	0	1	**4**
	Presentation - Abstraction - Control	-1	0	1	0	1	**-1**
Comunicación	**Publish - Subscribe**	-1	2	0	2	0	7
	Forwarder - receiver	-1	0	0	0	-1	**-6**

Cuadro 2: Tabla de escogencia de estilos arquitectónicos

Se realiza la suma ponderada por cada estilo arquitectónico obteniendo unos totales para cada uno de ellos[1], llegando a un estilo arquitectónico estructural, uno de sistemas interactivos y uno de comunicación. Dichos estilos se definen a continuación:

Cliente / Servidor: Se escoge este estilo arquitectónico debido a que existe la posibilidad de contar con varios clientes y además se tiene la posibilidad de distribuir el procesamiento en un clúster de servidores. Se promueve mediante este patrón la modificabilidad y el reuso, ya que se crean servicios comunes, los cuales al existir una necesidad de modificación, disminuye el impacto, ya que dicho cambio solamente se tiene que hacer en un solo sitio, o en un número pequeño de componentes. Se mejora la escalabilidad y la disponibilidad al centralizar el control de esos recursos y servicios, mientras que se pueden distribuir dichos recursos en múltiples servidores físicos.

Model-View-Controller: es el patrón que determina el comportamiento y la interacción con el usuario, es decir, describe cómo funcionará el componente cliente del patrón dominante. Es muy útil dado a que la interfaz de un sistema de software es normalmente la porción más modificada e interactiva de la aplicación. También se plantea por la posibilidad existente de tener la necesidad de contar con diferentes perspectivas de los datos (Vistas) del estado de los datos (Modelo).

[1]Para el estilo arquitectónico Cliente / Servidor en el atributo de calidad *Eficiency (Time behavior)* se establece un valor de 1, dado que se pretende usar componentes relacionados a Big Data como Hadoop que permitan tener mejores tiempos de respuesta en clústers de servidores.

Publish - Subscribe: Es el que permitirá el llamado asíncrono entre el cliente y el servidor, garantizando la extensibilidad y confiabilidad del sistema, ya que a través de su esquema de colas permitirá que los nodos se comuniquen asincrónicamente. Es la manera de crear mecanismos de integración que soportan la habilidad de transmitir mensajes entre productores y consumidores de manera tal que no conocen la identidad del otro o probablemente su existencia. Es decir, se logra bajo acoplamiento entre los clientes y el servidor. Por consiguiente el número de componentes, interfaces y conexiones es: añadido, eliminado y modificado, junto con una caracterización de la complejidad de los cambios, cancelaciones, modificaciones.

5. Descripción de la arquitectura

Para describir la arquitectura se utilizará el modelo de Kruchten [5]; este modelo consiste en 5 vistas las cuales se explican a continuación:

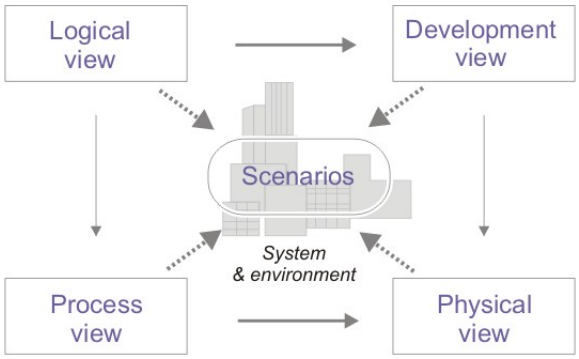

Figura 2: Modelo 4+1 de Kruchten

5.1. Vista de Escenarios

Esta vista permite la descripción de la arquitectura usando un conjunto de casos de uso o escenarios, siendo un punto de partida para el diseño. Se identifican las interacciones entre los usuarios y los requerimientos del sistema[5].

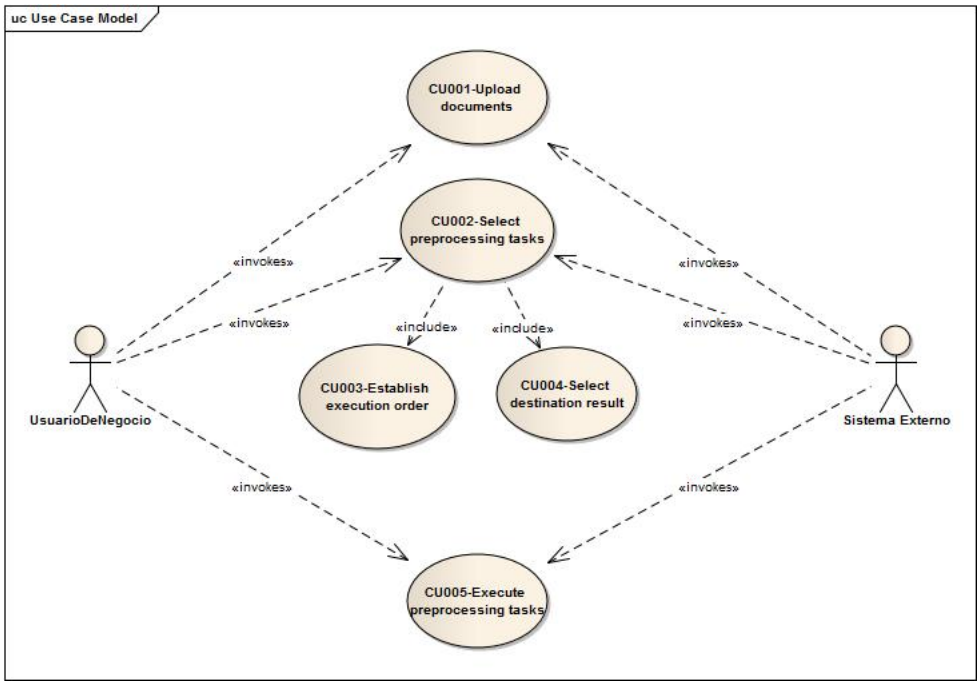

Figura 3: Vista de Escenarios

5.1.1. Elementos de la vista

1. CU001-Upload documents: Describe el proceso de realizar el cargue de documentos para su posterior procesamiento

2. CU002-Select preprocessing tasks: Describe la selección de las tareas de pre-procesamiento que se desean realizar en el sistema. Las tareas que se podrán seleccionar son Part Of Speech, Splitter, Tokenizer, Entity Recognition, búsqueda basado en expresiones regulares(NE Transducer) e identificación de coreferencias

3. CU003-Establish execution order: Se establece el orden en que se deben ejecutar las tareas seleccionadas y configuradas en el CU002.

4. CU004-Select destination result: El sistema elige la forma en que quiere que le sea entregado el resultado, teniendo como opciones: una ubicación en un directorio FTP, en el sistema de archivos local, o en un directorio HDFS.

5. CU005-Execute preprocessing tasks: El sistema ejecuta la lista de tareas de pre-procesamiento sobre los documentos cargados.

Para mayor información acerca de los casos de uso revisar el archivo DocumentoDeCasosDeUso.docx.

5.2. Vista Lógica

Esta vista permite la identificación de los elementos estructurales del sistema y el estilo arquitectónico seleccionado[5].

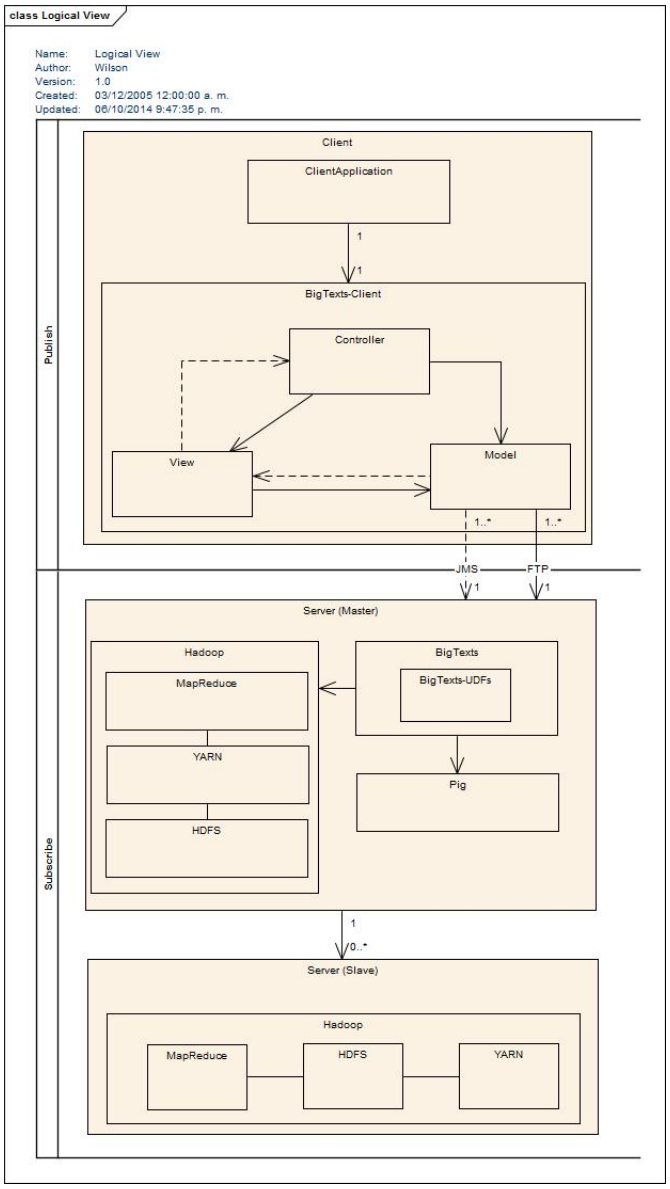

Figura 4: Vista Lógica

5.2.1. Elementos de la vista

1. Client: Es la máquina cliente en donde se encuentra tanto el cliente de BigTexts, como la aplicación externa que lo utiliza.

   a) ClientApplication: Es el sistema externo que usa el API de BigTexts.
   b) BigTexts-Client: Es la aplicación de escritorio que es usada tanto por un usuario de negocio(por medio de una interfaz gráfica), como por un sistema externo(como librería que expone un API). Dentro de este cliente se encuentra implementado el patrón MVC. Es el componente que invoca los servicios del componente servidor. Se utiliza el patrón MVC para mantener separada la funcionalidad de interfaz de la funcionalidad de aplicación.
      1) Model: Contiene la funcionalidad principal y los datos.
      2) View: Muestra la información al usuario.
      3) Controller: Maneja las entradas del usuario.

2. Server (Master): Es el nodo que hace las veces de maestro en el clúster de Hadoop. Es la máquina en la que se instala el servidor de BigTexts.
   a) Hadoop
      1) MapReduce: Componente de Hadoop para el procesamiento en paralelo.
      2) YARN: Componente de Hadoop para la planificación de tareas y gestión de recursos en clúster
      3) HDFS: Componente de Hadoop para la gestión distribuida de archivos.
   b) BigTexts: Es el servidor de BigTexts, es el encargado de la gestión de la ejecución de las tareas de preprocesamiento usando la infraestructura de Hadoop.
      1) BigTexts-UDFs: Son las implementaciones de las tareas de preprocesamiento, las cuales para poder ser usadas por Apache Pig, se tienen que programar con *User Defined Functions*[2]
   c) Pig: Apache Pig, provee un motor para la ejecución de flujos de datos en paralelo sobre Hadoop, hace uso directo de HDFS Y MapReduce [4].

3. Server(Slave): Es una de las máquinas del clúster que hacen las veces de esclavos, tienen instalado hadoop y sirven para el proceso en paralelo de las tareas.
   a) Hadoop

1) MapReduce: Es un estilo de computación que se puede usar para gestionar muchos cálculos de gran escala en una manera que es tolerante a fallos de hardware [6]. Se compone de dos tareas, la primera es la Map, que toma un conjunto de datos y lo convierte en otro donde los elementos individuales se dividen en tuplas y la tarea Reduce que toma la salida de la tarea Map como entrada y combina aquellos datos en un conjunto más pequeño de tuplas [9].
2) YARN: Es un *framework* para la planificación de tareas y gestión de recursos en clúster
3) HDFS: Es un sistema de ficheros distribuido que almacena los archivos a través de todos los nodos en un cluster Hadoop

5.3. Vista de Desarrollo

En esta vista se presenta un mayor nivel de detalle en el diseño arquitectónico, se muestran componentes del sistema y sus interacciones a un nivel de granularidad mayor[5].

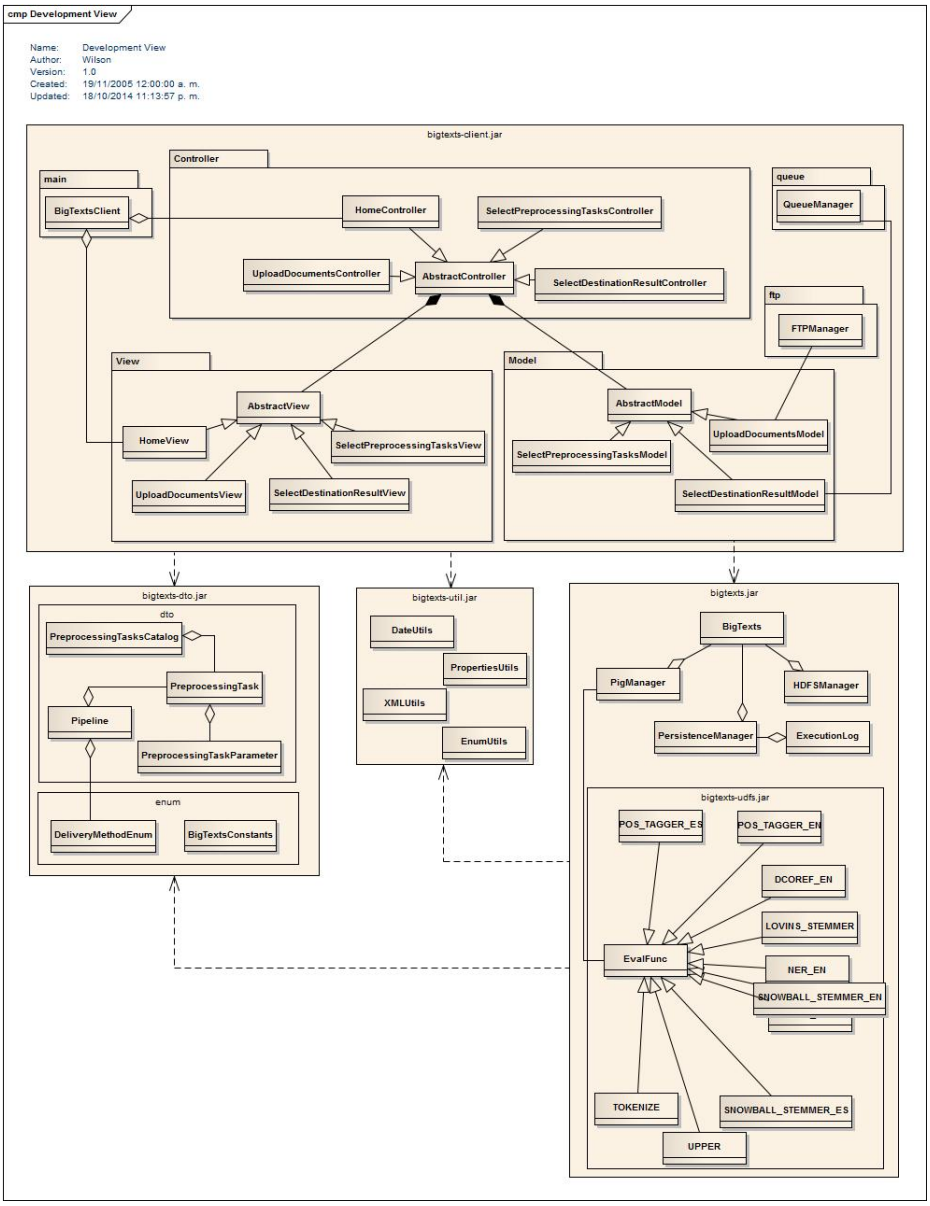

Figura 5: Vista de desarrollo

Se complementa la vista con un diagrama de secuencia del caso principal (CU005-ExecutePreprocessingTasks)

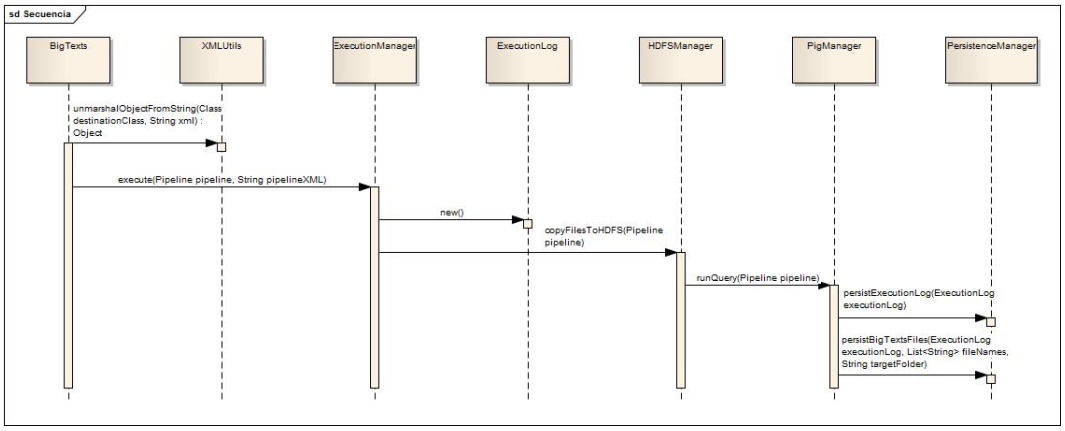

Figura 6: Diagrama de secuencia del CU005-ExecutePreprocessingTasks

5.3.1. Elementos de la vista

1. bigtexts-client.jar: Empaquetado con la aplicación cliente, la cual puede usarse por medio de una interfaz gráfica o como librería.
   a) main:Paquete en el que se almacena la clase principal del cliente
      1) BigTextsClient: Es la clase que inicializa la interfaz gráfica de BigTexts
   b) Controller:Paquete con los componentes que controlan la comunicación entre las vistas y los modelos
      1) AbstractController: Clase abstracta que debe ser extendida por todos los controladores (patrón MVC) del sistema, ofrece utilidades comunes a todos los controladores
      2) HomeController: Controlador asociado a la pantalla Home del sistema
      3) SelectDestinationResultController: Controlador asociado a la pantalla de selección de destino del resultado en el sistema (CU005-Execute preprocessing tasks)
      4) SelectPreprocessingTasksController: Controlador asociado a la pantalla de selección y parametrización de tareas de preprocesamiento del resultado en el sistema (CU002-Select preprocessing tasks, CU003-Establish execution order y CU004-Select destination result)
      5) UploadDocumentsController: Controlador asociado a la pantalla de carga de archivos (CU001-Upload documents)

c) View: Paquete que contiene la forma en que se muestra la información.

   1) AbstractView: Clase abstracta que debe ser extendida por todas las vistas (patrón MVC) del sistema, ofrece utilidades comunes a todos las vistas
   2) HomeView: Vista asociada a la pantalla Home del sistema
   3) SelectDestinationResultView: Vista asociada a la pantalla de selección de destino del resultado en el sistema (CU005-Execute preprocessing tasks)
   4) SelectPreprocessingTasksView: Vista asociada a la pantalla de selección y parametrización de tareas de preprocesamiento (CU002-Select preprocessing tasks, CU003-Establish execution order y CU004-Select destination result)
   5) UploadDocumentsView: Vista asociada a la pantalla de carga de archivos (CU001-Upload documents)

d) Model: Paquete que contiene la funcionalidad principal y los datos.

   1) AbstractModel: Clase abstracta que debe ser extendida por todos los modelos (patrón MVC) del sistema, ofrece utilidades comunes a todos los modelos
   2) SelectDestinationResultModel: Modelo de la pantalla de ejecución
   3) SelectPreprocessingTasksModel: Modelo de la pantalla de selección de tareas de pre-procesamiento
   4) UploadDocumentsModel: Modelo de la pantalla de carga de archivos

e) queue: Paquete en el cual se encuentran las utilidades con la gestión de colas de mensajes

   1) QueueManager: Clase con utilidades para la gestión de colas de mensajes

f) ftp: Paquete en el cual se encuentran las utilidades relacionadas con la transferencia de archivos mediante FTP

   1) FTPManager: Clase con utilidades para la gestión de archivos en FTP

2. bigtexts-dto.jar: Es el componente con los objetos que se transferirán entre el cliente y el servidor

   a) Pipeline: Clase que representa todo lo que BigTexts va a ejecutar, las tareas de preprocesamiento, los archivos, y la forma de entrega
   b) PreprocessingTask: Clase que representa las tareas de preprocesamiento, usada tanto para el catálogo como para establecer las tareas de preprocesamiento que se van a ejecutar
   c) PreprocessingTaskParameter: Clase que representa los parámetros asociados a las tareas de preprocesamiento que se usan tanto para el catálogo como para establecer las tareas de preprocesamiento que se van a ejecutar
   d) PreprocessingTasksCatalog: Clase que representa la lista de tareas de preprocesamiento disponibles en el sistema

3. bigtexts-util.jar: Es el componente con clases utilitarias del sistema. Estas clases se usan tanto en el cliente como en el servidor
   a) DateUtils: Clase con utilidades para el manejo de fechas
   b) EnumUtils: Clase con utilidades para la gestión de enumeraciones
   c) PropertiesUtils: Clase con métodos utilitarios para la gestión de archivos de propiedades
   d) XMLUtils: Clase con métodos utilitarios para la serialización y desserialización de objetos a XML usando JAX-B

4. bigtexts.jar: Es el componente servidor del sistema
   a) BigTexts: Clase principal del servidor de BigTexts, recibe el mensaje de la cola y ejecuta el pipeline
   b) PigManager: Clase con métodos de gestión con Pig
   c) HDFSManager: Gestor de trabajos en Hadoop Distributed File System
   d) PersistenceManager: Gestor de conexión con la base de datos
   e) ExecutionLog: Entidad que almacena los datos de las ejecuciones
   f) bigtexts-udfs.jar: Componente con las implementaciones de las tareas de pre-procesamiento
      1) DCOREF_EN: UDF de identificación de correferencias en inglés
      2) LOVINS_STEMMER: UDF Pig con la implementación de lematización de Lovins

3) NER_EN: UDF Pig que implementa el Named Entity Recognition en inglés
4) NER_ES: UDF Pig que implementa el Named Entity Recognition en español
5) POS_TAGGER_EN: UDF que implementa Part Of Speech en inglés
6) POS_TAGGER_ES: UDF que implementa Part Of Speech en español
7) REGEXNER: UDF de Named Entity Recognition basado en expresiones regulares
8) SNOWBALL_STEMMER_EN: UDF Pig con la implementación de lematización usando snowball para inglés
9) SNOWBALL_STEMMER_ES: UDF Pig con la implementación de lematización usando snowball para español
10) SSPLIT: UDF de Partición de textos por frases
11) TOKENIZE: UDF de tokenización
12) UPPER: UDF de Pig que recibe una cadena de caracteres y lo pone en mayúsculas

El sistema está diseñado para que la adición de nuevos UDFs sea lo más sencilla posible. Para ello simplemente es necesaria la creación de una clase que implemente la función y agregarla en el archivo XML del catálogo de tareas de preprocesamiento.

5.4. Vista de Procesos

En esta vista se presentan los componentes de ejecución del sistema, describiendo los aspectos de concurrencia y sincronización[5].

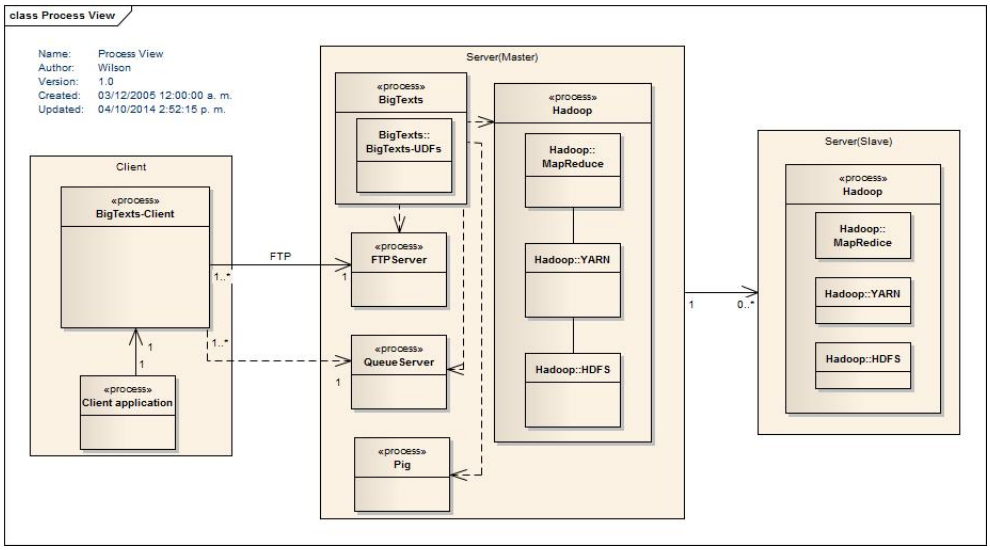

Figura 7: Vista de procesos

5.4.1. Elementos de la vista

1. Client: es la máquina cliente en la que se tiene tanto la aplicación externa como el cliente de BigTexts.

   a) BigTexts-Client: Aplicación cliente, la cual puede usarse por medio de una interfaz gráfica o como librería.

   b) Client application: La aplicación externa que usa el cliente de BigTexts para el preproccesamiento de texto.

2. Server(Master): Máquina principal del clúster Hadoop

   a) BigTexts: Aplicación servidor de BigTexts, es la encargada de ejecutar el pipeline usando la infraestructura de Hadoop.

   b) FTPServer: Servidor FTP, mediante el cual, el cliente le envía los documentos al servidor.

   c) QueueServer: Servidor de colas, mediante el cual, el cliente le envía el mensaje asíncrono de ejecución de un pipeline.

   d) Pig: Es la instalación de Apache Pig, se usa para la ejecución de tareas en el clúster Hadoop.

   e) Hadoop: Instalación principal del *framework* para procesamiento distribuido de conjuntos grandes de datos.

1) MapReduce: Componente de Hadoop para procesamiento en paralelo.
2) YARN: Componente de Hadoop para la planificación de tareas y gestión de recursos en clúster.
3) HDFS: Componente de Hadoop para gestión distribuida de archivos.

3. Server(Slave): Una de las máquinas esclavo del clúster Hadoop

   a) Hadoop: Instalación esclava(secundaria) del *framework* para procesamiento distribuido de conjuntos grandes de datos.
      1) MapReduce: Componente de Hadoop para procesamiento en paralelo.
      2) YARN: Componente de Hadoop para la planificación de tareas y gestión de recursos en clúster.
      3) HDFS: Componente de Hadoop para gestión distribuida de archivos.

5.5. Vista Física

Esta vista describe la correspondencia entre los componentes de Software y los de Hardware reflejando sus aspectos de distribución[5].

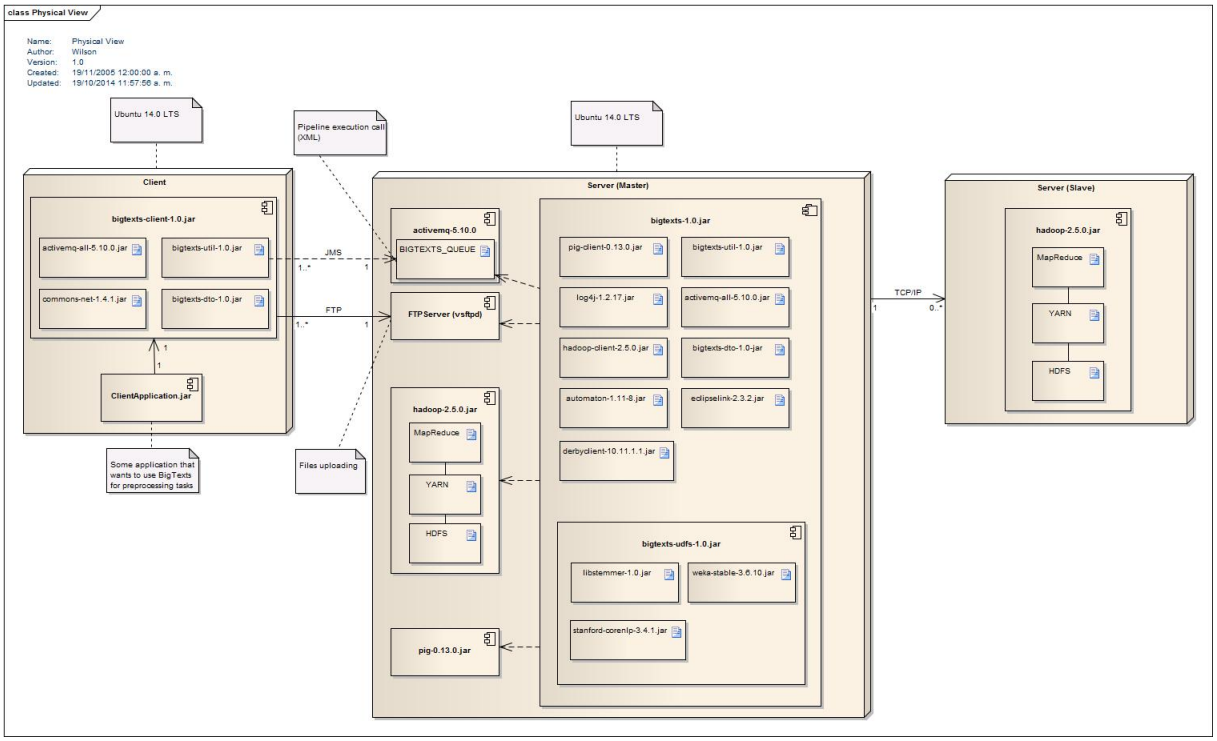

Figura 8: Vista física

5.5.1. Elementos de la vista

1. Client: Máquina cliente en donde se encuentra instalada tanto la instalación de la aplicación externa, como el cliente de BigTexts.

    a) bigtexts-client.jar: Empaquetado con la aplicación cliente, la cual puede usarse por medio de una interfaz gráfica o como librería.

        1) activemq-all-5.10.0.jar: Librería que permite el envío de mensajes al servidor de colas.
        2) bigtexts-util-1.0.jar: Empaquetado con las clases de utilidades de BigTexts.
        3) commons-net-1.4.1.jar: Librería para el envío de los documentos al servidor FTP.
        4) bigtexts-dto-1.0.jar: Empaquetado con las clases que representan los objetos que comparten tanto el cliente como el servidor de BigTexts.

b) ClientApplication.jar: Representa la aplicación externa que usa el cliente de BigTexts como librería.

2. Server(Master): Máquina en la cual se instala la instancia principal de Hadoop, así como también el servidor de BigTexts.

   a) activemq-5.10.0: Servidor de colas

      1) BIGTEXTS_QUEUE: Cola de mensajes mediante la cual se comunican los llamados desde el cliente hacia el servidor.

   b) FTPServer (vsftpd): Es el servidor FTP que se instala en Ubuntu.

   c) bigtexts-1.0.jar: Aplicación servidor, es la encargada de ejecutar las tareas de pre-procesamiento en la infraestructura BigData.

      1) pig-client-0.13.0.jar: Es la librería que permite la conexión con la instalación de Pig.
      2) bigtexts-util-1.0.jar: Empaquetado con las clases de utilidades de BigTexts.
      3) log4j-1.2.17.jar: Gestor de la bitácora de la aplicación.
      4) hadoop-client-2.5.0.jar: Librería que permite la interación con HDFS(Hadoop Distributed File System) y con Hadoop en general.
      5) bigtexts-dto-1.0-jar: Empaquetado con las clases de comunicación entre el cliente y el servidor de BigTexts.
      6) automaton-1.11-8.jar: Librería de autómatas de estado finito, usada por el cliente de Pig.
      7) eclipselink-2.3.2.jar: Implementación de JPA(Java Persistence API) de Eclipse
      8) derbyclient-10.11.1.1.jar: cliente del motor de base de datos liviano Apache Derby.
      9) bigtexts-udfs-1.0.jar: Empaquetado con las implementaciones de las tareas de pre-procesamiento.
         a′ libstemmer-1.0.jar: Librería para la implementación de Snowball Stemmer.
         b′ stanford-corenlp-3.4.1.jar: Librería de Natural Language Processing usada para las implementaciones de las tareas de pre-procesamiento.
         c′ weka-stable-3.6.10.jar: Librería para la implementación de Lovins Stemmer.

d) hadoop-2.5.0.jar: Instalación principal(maestra) del framework para procesamiento distribuido de conjuntos grandes de datos a través de clusters de computadores.

1) MapReduce: Componente de Hadoop para procesamiento en paralelo.
2) YARN: Componente de Hadoop para la planificación de tareas y gestión de recursos en clúster.
3) HDFS: Componente de Hadoop para gestión distribuida de archivos.

e) pig-0.13.0.jar: Instalación de la librería para el acceso a Hadoop.

3. Server (Slave): Una de las máquinas secundarias(esclavas) en el clúster Hadoop.

4. hadoop-2.5.0.jar: Instalación principal(maestra) del framework para procesamiento distribuido de conjuntos grandes de datos a través de clusters de computadores.

a) MapReduce: Componente de Hadoop para procesamiento en paralelo.

b) YARN: Componente de Hadoop para la planificación de tareas y gestión de recursos en clúster.

c) HDFS: Componente de Hadoop para gestión distribuida de archivos.

Referencias

[1] Apache Software Foundation. Hadoop. http://hadoop.apache.org/.

[2] Apache Software Foundation. User defined functions. http://pig.apache.org/docs/r0.13.0/udf.html.

[3] DoFactory. Facade design pattern. http://www.dofactory.com/Patterns/PatternFacade.aspx.

[4] Alan Gates. *Programming Pig*. O'Reilly Media, 1 edition edition, September 2011.

[5] P.B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6):42–50, November 1995.

[6] Anand Rajaraman and Jeffrey D Ullman. *Mining of massive datasets*. Cambridge University Press, New York, N.Y.; Cambridge, 2012.

[7] SQA.net. ISO9126 - software quality characteristics. http://www.sqa.net/iso9126.html.

[8] Qiushi Wang and Zhao Yang. A method of selecting appropriate software architecture styles: Quality attributes and analytic hierarchy process. August 2012.

[9] Paul Zikopoulos and Chris Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 1 edition edition, October 2011.