Escuela de Ciencia de la Computación
Big Data

Efficient and Distributed Temporal Pattern Mining

Informe Final de Proyecto

Ariana Villegas
ariana.villegas@utec.edu.pe

Roosevelt Ubaldo
roosevelt.ubaldo@utec.edu.pe

Stephano Wurttele
stephano.wurttele@utec.edu.pe

Victor Peña
victor.pena@utec.edu.pe

Lima, Perú
Febrero 2022

Índice general

Capítulo 1

Introducción

Como parte del curso de Big Data, hemos analizado distintas aplicaciones y lenguajes que nos permiten trabajar información de varias maneras de acuerdo a la estructuración de la información o la forma en la que se quiere utilizar. Para el proyecto descrito en este informe, haremos foco a los métodos de manejo de cantidades masivas de información utilizando computación distribuida orientado a información de patrones temporales. Por ello, usamos la estructura presentada en el documento *Efficient and Distributed Temporal Pattern Mining* [1], que implementa la estructura de datos *Distributed Hierarchical Pattern Graph TPM (DHPG-TPM)*. Esta, a la fecha de publicación del documento, es la primera solución distribuida para *Temporal Pattern Mining*. Considerando el estado del arte para trabajar con esta tipo de información, se utiliza el engine *Apache Spark*.

De acuerdo al planteamiento del documento referenciado, se realizará una recaudación de información comparable para realizar el estudio y construir una base de datos temporal, proceso que se explicará a detalle en la descripción del caso. Luego, se usa un bitmap para recopilar información de los eventos temporales, de lo que conseguiremos los datos de frecuencia y soporte. Con ello, orrganizaremos las secuencias temporales y podremos organizarlas en un DHPG-TPM.

Este informe cumplirá con explicar conceptualmente lo planteado en el documento, los pasos seguidos para reproducirlo tanto en detalles de implementación como estructuras de datos, y finalmente concluir con los logros de la reproducción de acuerdo a los objetivos planteados por el profesor.

Capítulo 2

Descripción del caso del estudio

En las etapas iniciales se consideró usar una de las bases de datos de NIST o del metropolitano. El proceso propuesto para esto, a rasgos generales, fue categorizar cada columna de datos para usar estas categorías como eventos. Después, sé planteo agrupar las categorías que se encuentren secuencialmente para generar los intervalos en los cuales los eventos suceden. Finalmente, al unir el evento e intervalo. Finalmente, convertiríamos nuestra base de datos secuencial en una de eventos temporales.

Este proceso no fue completado por las diversas dificultades que significaba implementar y el tiempo disponible para realizar esto. Una de las dificultades fue la categorización de cada evento, ya que era necesario hacer primero un análisis estadístico de cada variable. Este análisis sería el que determinaría los denominados eventos, como $x > 1$ sería el evento A y así sucesivamente. Esta categorización es complicada por lo irregulares que son los datos y los diferentes rangos de los mismos.

Por lo antes mencionado, el equipo decidió generar los eventos de manera sintética y de esta manera comenzar las pruebas requeridas para las otras partes del proyecto. Esta generación de datos se realiza de manera randomizada y dependiendo de la cantidad n de eventos y una cantidad m de intervalos por evento que se desee.

Capítulo 3

Implementación

3.1. Arquitectura de desarrollo

El programa desarrollado puede recibir data entrada desde un archivo externo o generar su propia data de manera interna. La data recibida es posteriormente preprocesada y minada en la aplicación. Finalmente, los resultados de este procedimiento son mostrados directamente en pantalla.

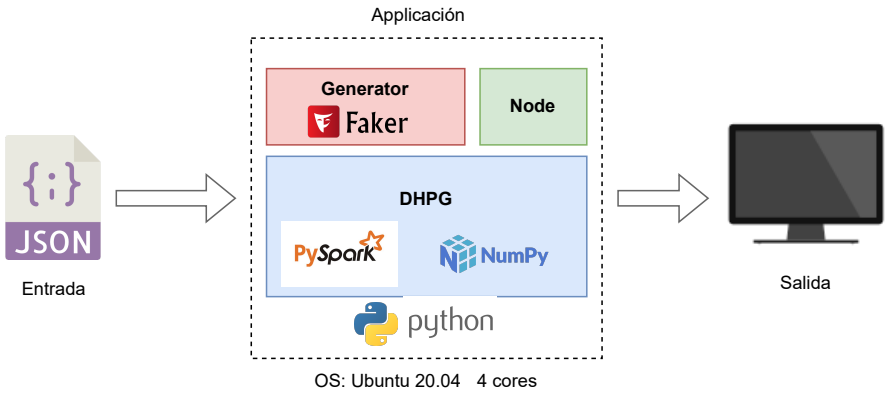

Figura 3.1: Diagrama de Arquitectura.

Tal y como se muestra en la figura 3.1, la aplicación se compone de tres partes principales: el *generator*, *Node* y *DHPG*. En *generator* se realiza la creación sintética del *Temporal Sequence Database (TSD) o* D_{SEQ}. Para ello se provee una lista de nombres de eventos, valores mínimos y máximos de ocurrencia de cada evento, la cantidad de secuencias a generar y si la data generada será escrita en disco o no. Los nombres de los eventos son generados de manera sintética

haciendo uso de la librería *Faker*. Cada secuencia se encuentra conformada por ocurrencias aleatorias de cada valor de la lista de nombres de evento de acuerdo al rango previamente establecido. Los valores de los intervalos de tiempo también son generados de manera aleatoria. En *Node* se define la estructura de datos a usar para almacenar los resultados de cada nivel de minado de frecuencias. Por último, en *DHPG* se realiza la implementación de la estructura del mismo nombre. En esta parte, se realiza la configuración del *Spark Context* haciendo uso de la librería *PySpark* y se realizan las operaciones de pre-procesamiento de D_{SEQ} a D_{EV} y minado *single* y *2-Freq* de manera distribuida. La representación del bitmap se realiza utilizado *np.array()* de la librería *Numpy*.

3.1.1. Diagrama de clases

Tal y como se observa en la figura 3.3, el proyecto se emplearon tres clases principales: *Temporal Sequence Database (TSD)*, *Distributed Hierarchical Pattern Graph (DHPG)* y *Node*. También se emplearon funciones utilitarias para construir el *bitmap*, obtener los valores de *support* y *confidence*, así como construir los nodos del *DHPG*.

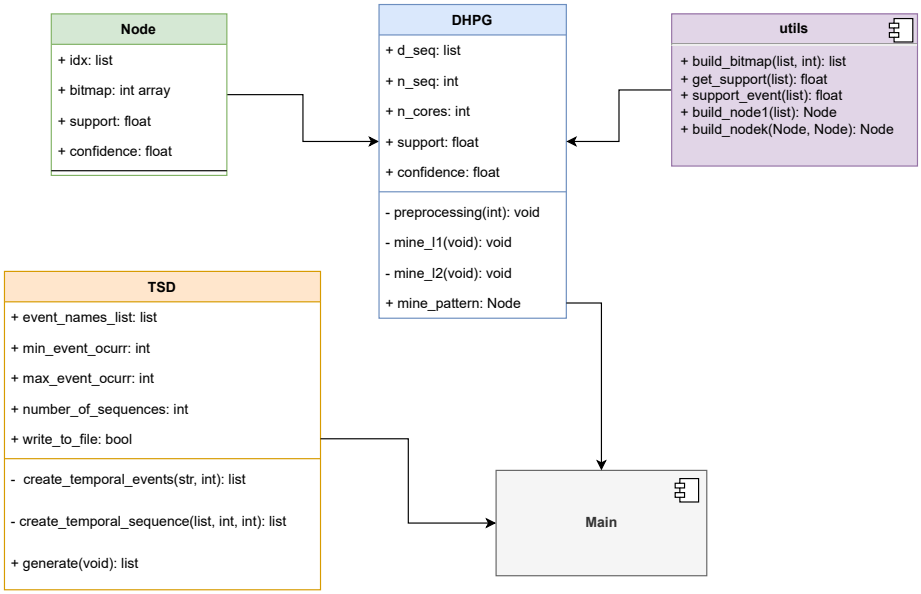

Figura 3.2: Diagrama de clases.

- En TSD se realiza la creación de la data sintética. Para ello se parte desde la generación de eventos temporales, secuencias temporales y múltiples secuencias.

- En DHPG, se realiza el preprocesamiento de D_{SEQ} a D_{EV} y los minados de nivel 1 y 2.
- En Node se almacenan los resultados del minado de cada nivel.

Las clases declaradas son empleadas en la función *Main* que es donde se genera la data sintética y se realiza el minado.

3.1.2. Librerías Externas

Se emplearon tres librerias externar: *PySpark*, *Numpy* y *Faker*.

- *PySpark*: librería que nos permite ejecutar Apache Spark desde Python.
- *Numpy*: librería que nos facilita definir el bitmap y realizar operaciones con él,
- *Faker*: libreria que nos ayuda a generar nombres de eventos.

3.2. Estructuras de datos

La principal estructura de datos empleada es *DHPG*, la cual se encuentra compuesta por niveles, y cada nivel por una determinada cantidad de nodos. El nivel más alto se encuentra vacío, por sugerencias de la implementación original. En el primer nivel se almacena en nodos aquellos eventos que poseen un valor de *support* mayor al del *support_threshold*. En cada nodo, se almacena el nombre del evento o eventos, el bitmap asociado y sus respectivos valores de *support* y *confidence*. En el segundo nivel, se almacenan los resultados de *2-Frequent* events, para ellos se almacen los nombres de cada evento, su bitmap correspondiente, y sus respectivos valores de *support* y *confidence*. A continuación, se muestra un ejemplo de la estructura empleada.

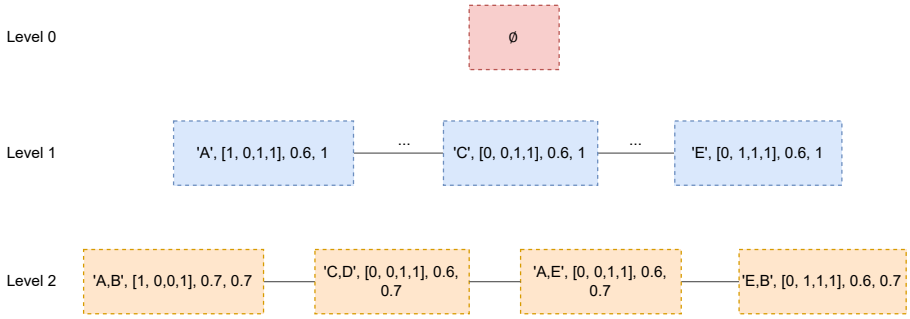

Figura 3.3: Implementación de la estructura *DHPG*.

Capítulo 4

Conclusiones

Este trabajo, como proyecto experimental, nos ha permitido descubrir y confirmar empíricamente algunos conceptos teóricos que teníamos entendidos. Estos relacionados a las estructuras de datos requeridas para trabajar con distintos tipos de datos; y la escalabilidad y eficiencia del paralelismo y procesamiento distribuido.

Por un lado, es muy importante notar como la estructura de datos utilizada en este proyecto ha sido específicamente modificada para poder tratar correctamente la información de tipo de **patrones temporales**, considerando cada una de sus particularidades. Esto nos permite concluir que las estructuras de datos se usan de la mejor forma cuando son modificadas para el caso particular. En este trabajo, la modificación permitió acoplar la estructura de datos con cada paso del proceso de los eventos, lo que permitió a los autores poder definir cada nivel de acuerdo a lo que ellos consideraron, haciendo así posible el minado de *k-event patterns*. Cabe mencionar, esta complejidad va escalando conforme a los niveles, por la cantidad de requisitos y cálculos que se hacen como se explica en el documento, aunque a nivel de implementación nosotros no abarcamos esa extensión y por lo tanto no sentimos empíricamente ese crecimiento.

Aún así, dada la cantidad de datos manejados y la importancia de la optimización, el *engine* utilizado **Apache Spark** (o su equivalente en Python, la librería *PySpark*) demostró ser de gran utilidad. Sus propiedades como sistema para trabajar con información distribuida nos ayudó en gran medida para separar las tareas tanto en el pre-procesamiento y en el minado; pues es cierto que estas tareas son separables y generalmente puedan ocupar gran cantidad de memoria por el volumen de información y la eventual complejidad de los datos. Finalmente entonces, comprobamos empíricamente la utilidad de sistemas de trabajo distribuido, y nos abre la oportunidad de probar trabajos de escala similar con otros como podría serlo por ejemplo Dask.

Bibliografía

[1] N. Ho, V. L. Ho, T. Bach Pedersen, and M. Vu, "Efficient and distributed temporal pattern mining," in *2021 IEEE International Conference on Big Data (Big Data)*, pp. 335–343, 2021.