Instituto Tecnológico Autónomo de México

Comparación de Dask y Spark en procesamiento de grandes volúmenes de datos

Tesis

que para obtener el título de

Licenciado en Matemáticas Aplicadas

Presenta

Iker Antonio Olarra Maldonado

Asesora

Mtra. Liliana Millán

Ciudad de México 2021

«Con fundamento en los artículos 21 y 27 de la Ley Federal del Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada "**Confianza institucional, inclusión social, intensidad religiosa y percepción tecnológica como factores de innovación para el crecimiento económico**", otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la Biblioteca Raúl Bailléres Jr., la autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una contraprestación.»

Fecha

Iker Antonio Olarra Maldonado

A la comunidade de código abierto,
por sus aportaciones al progreso de la tecnología.

Agradecimientos

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Resumen

El volumen de datos y las capacidades de almacenamiento han incrementado de forma considerable en las últimas décadas. Esto ha incrementado la necesidad de desarrollar *software* de procesamiento de datos que utilice cómputo en paralelo y distribuido para poder procesar los conjuntos masivos de datos creados diariamente. Dos de las herramientas de código abierto más populares para el procesamiento de datos son Dask y Spark. El objetivo de esta investigación es comparar su desempeño al realizar operaciones como obtención de estadísticas y procesamiento de datos sobre un conjunto de datos grande (8 GB aproximadamente), correspondiente a vuelos dentro de Estados Unidos. Los diferentes procesos se realizarán por ambos *softwares*, en un orden aleatorio y con un número específico de ejecuciones, en un ambiente local y en en la nube. Los resultados serán comparados para determinar bajo qué circunstancias un software es superior e identificar las características que permiten ese desempeño.

Summary

Data volume and storage capabilities have increased considerably during the past decades. With that in mind, data processing software has leveraged parallelism to keep up with the massive datasets created everyday. Two of the most popular and open-source parallel data processing tools are Dask and Spark. The objective of this research is to compare their performance on a large dataset (8GB approximately), corresponding to flights within the United States, while completing different tasks like computing statistics and performing data processing jobs. The tasks will be executed by both softwares in an isolated Cloud environment in random order and with a fixed number of executions for each task. The results will be compared to determine under which circumstances one software is superior and identify the characteristics that enable such performance.

Índice general

Índice de cuadros

Índice de figuras

Introducción

La Economía tiene distintas definiciones.

Capítulo 1

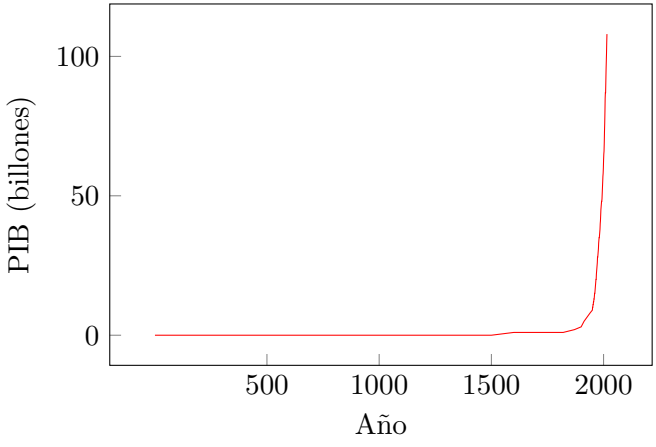
Revisión de literatura

Este capítulo está dividido en dos secciones.

1.1. Modelos teóricos

1.1.1. Hechos estilizados

Figura 1.1. PIB mundial de los últimos dos milenios



Fuente: elaboración propia con datos de Maddison (2010) y el Banco Mundial.

Capítulo 2

Marco teórico

El objetivo de este análisis es introducir al cómputo en paralelo y distribuido y describir las capacidades principales de los *frameworks* de *Big Data*: *Spark* y *Dask*.

2.1. Introducción al cómputo en paralelo

2.2. Apache Spark

Apache Spark es un motor de cómputo unificado con bibliotecas para procesamiento de datos en paralelo en clústeres de computadoras. Spark soporta múltiples lenguajes de programación como Python, Java, Scala y R. Entre sus aplicaciones más compunes están trabajos de SQL, *streaming* y *machine learning*.[5]

2.3. Arquitectura de Spark

Una aplicación de Spark consiste de un proceso *driver* y un conjunto de procesos llamados *executor*. Ambos tipos de procesos trabajan en conjunto durante la ejecución de la aplicación. El proceso *driver* es la parte central de la aplicación, ya que se encarga de mantener información sobre la aplicación, responder a las instrucciones del usuario y analizar, distribuir y calendarizar el trabajo entre los ejecutores. Este proceso existe en uno solo de los nodos y está en constante comunicación con los demás. [5]. Los *executors*, por otra parte, tienen el objetivo de realizar el trabajo que el *driver* les asigna. Estos se encargan de ejecutan el trabajo y guardan datos para la aplicación [**?**].

La abstracción de datos principal en Spark son los *Resilient Distributed Datasets (RDDs)* que son colecciones de objetos particionadas en un clúster y que se pueden manipular en paralelo a través de transformaciones. Además, estas transformaciones tienen un *lazy evaluation*, es decir que no realizan la acción inmediatamente, sino que la añaden a un plan eficiente de ejecución. Al llamar una acción, que es una instrucción para calcular un resultado a partir de las

transformaciones del plan [5].

2.4. Dask

Dask es una librería de Python para cómputo en paralelo que extiende herramientas populares en el análisis de datos como *NumPy*, *Pandas* y *Python iterators* a tareas que no caben en memoria o a ambientes distribuidos. Además, cuenta con un calendarizador dinámico de tareas y está desarrollado totalmente en *Python*. Se puede utilizar en modo *standalone* o en clústeres con cientos de máquinas. Dask ofrece una interfaz familiar ya que emula a *Numpy* y *Pandas*.

2.4.1. Cambios cerebrales y genéticos

Brizendine (2010) escribe que algunos científicos piensan que ciertas áreas del cerebro son como centros de actividad que mandan señales eléctricas a otras áreas del cerebro ocasionando un determinado comportamiento.[1]

> Mientras que la distinción entre los cerebros de niños y niñas empieza biológicamente, estudios recientes muestran que es *solo* el comienzo. La estructura cerebral no está escrita sobre piedra en el nacimiento ni al final de la infancia, como antes se creía, sino que continúa cambiando a lo largo de la vida. Más que ser inmutable, nuestros cerebros son mucho más plásticos y cambiables de lo que los científicos creían hace una década. El

[1] Por ejemplo, en el hombre la corteza del cíngulo anterior pesa opciones, detecta conflicto y motiva decisiones. La unión temporoparietal busca soluciones rápidas y ante situaciones estresantes toma en cuenta la perspectiva de otros individuos. La corteza cingulada anterior rostral se encarga de procesar los errores sociales, como la aprobación o desaprobación de otros.

> cerebro humano es también la máquina de aprendizaje más talentosa que conocemos. Así que nuestra cultura y el cómo nos enseñaron a comportarnos desempeñan un papel importante en el diseño y reestructura de nuestros cerebros (Brizendine 2010, 5-6).

Hipótesis 3. *La intensidad religiosa está relacionada negativamente con la innovación.*

Cuadro 2.1. Índices de modernidad y tradicionalismo

País	Índice de modernidad	Índice de tradicionalismo
Alemania	0.58	0.45
Austria	0.55	0.49
Bélgica	0.50	0.49
Canadá	0.61	0.50
Dinamarca	0.58	0.44
España	0.47	0.62
Estados Unidos	0.59	0.44
Finlandia	0.62	0.38
Francia	0.49	0.59
Holanda	0.58	0.49
Irlanda	0.54	0.59
Islandia	0.63	0.54
Italia	0.56	0.58
Japón	0.42	0.48
Noruega	0.53	0.44
Portugal	0.50	0.71
Reino Unido	0.56	0.54
Suecia	0.62	0.51
Promedio	0.58	0.51

Fuente: Bojilov y Phelps (2012).

2.5. Dask

Capítulo 3

Metodología

Este capítulo describe el proceso seguido para realizar la comparación entre Dask y Spark bajo ambientes y condiciones homologadas. Para tener una mayor visibilidad del desempeño de cada una de ellas bajo diferentes circunstancias, se crearon 7 procesos distintos que consisten en filtración, agrupación conteos y cálculo de estadísticas sobre los datos seleccionados. Los 7 procesos fueron escritos en PySpark y Dask utilizando funciones equivalentes en ambos *frameworks* de forma independiente. Cada proceso se ejecutó X veces en dos ambientes, uno local y otro en la nube. También se incorporo una variación en el tamaño de las muestras para observar el desempeño de cada herramienta con distintos volúmenes de datos. Cada ejecución fue registrada en una base de datos para su análisis posterior.

3.1. Frameworks

Para hacer la comparación se eligieron los *frameworks* Dask y Apache Spark que funcionan como herramientas para cómputo en paralelo sobre grandes volúmenes de datos. Las siguientes secciones detallan las características de cada uno y las versiones utilizadas.

3.1.1. Spark

Spark es un sistema de cómputo en

3.2. Procesos para comparación de desempeño

Para contrastar el desempeño de los herramientas, se implementaron 7 procesos que realizan distintas operaciones sobre el conjunto de datos. Cada uno busca responder una pregunta distinta sobre los vuelos en Estados Unidos, cuya respuesta requiere de distintas operaciones. En las siguientes secciones se explica de forma más detallada cada uno de los procesos y se listan las operaciones que los componen.

Para ejecutar los datos se creó un *script* que sigue el siguiente proceso para la ejecución de los algoritmos:

Algorithm 1 Ejecución de procesos

```
1: procedure EJECUCIONES
2:     n ← número de ejecuciones a realizar
3:     procesos ← lista de procesos que se van a ejecutar
4:     for proceso in procesos do
5:         framework ← lista de 0 y 1 en orden aleatorio de longitud 2n
6:         for x in framework do
7:             if x = 1 then
8:                 Ejecutar el proceso en Spark
9:             else
10:                Ejecutar el proceso en Dask
```

De esta forma, cada proceso se ejecuta n veces en cada *framework* y las ejecuciones entre Spark y Dask se alternan debido a que la lista *framework* tiene un orden aleatorio. La ejecución de procesos se realizó 10 veces, 5 de forma local con tamaños de muestra distintos y 5 en un clúster en la nube de con los mismos tamaños de muestra. Para iniciar la ejecución de un proceso es necesario que el anterior haya finalizado.

3.2.1. Proceso 1: Cálculo del tamaño de la flota por aerolínea

Este proceso calcula el número de aviones activos con el que cuenta una aerolínea en un tiempo específico. La agregación se hace por día, mes, trimestre y año. Para obtener el número de equipos únicos se cuenta el número de matrículas distintas correspondientes a cada aerolínea. El proceso se compone de las siguientes operaciones:

- Lectura de datos desde `parquet`.
- Borrado de duplicados.

- Conteo.
- Agregación de datos agrupando por diferentes columnas.
- Escritura de datos a SQL.

3.2.2. Proceso 2: Demoras por aerolínea

Este proceso tiene como objetivo obtener estadísticas sobre las demoras de cada aerolínea. Este proceso realiza las siguientes operaciones principales:

- Lectura de datos desde `parquet`.
- Conteo.
- Cálculo de promedio.
- Agregación de datos agrupando por diferentes columnas.
- Escritura de datos a SQL.

3.2.3. Proceso 3 y 4: Demoras por aeropuerto de origen y destino

Estos procesos son muy similares ya que ambos calculan estadísticas de demoras de vuelos de acuerdo al aeropuerto. El proceso 3 hace el cálculo agrupando por el aeropuerto de orgien mientras que el 4 hace la agregación respecto al aeropuerto de destino. Los procesos ejecutan las siguientes operaciones:

- Lectura de datos desde `parquet`.
- Conteo.

- Cálculo de promedio.
- Agregación de datos agrupando por diferentes columnas.
- Escritura de datos a SQL.

3.2.4. Procesos 5 y 6: Demoras por ruta de aeropuerto y *market id*

Estos procesos tienen como objetivo obtener información sobre las rutas recorridas entre aeropuertos y ciudades identificadas por el *market id*. El primer paso es obtener la ruta que se recorre en cada vuelo y después calcular las estadísticas correspondientes. El proceso 5 calcula la ruta de acuerdo al aeropuerto de origen y destino y el 6 lo hace de acuerdo al *market id*.

- Lectura de datos desde `parquet`.
- Concatenación de columnas.
- Conteo.
- Cálculo de promedio.
- Agregación de datos agrupando por diferentes columnas.
- Escritura de datos a SQL.

3.2.5. Proceso 7: Cálculo de la menor ruta en tiempo utilizando el algoritmo de ruta mínima *Dijkstra*

El último proceso consiste en calcular la ruta mínima (minimizando el tiempo de vuelo) entre dos aeropuertos cualesquiera. Para esto se hizo una implementación de *Dijkstra* con algunas

variaciones que permiten descartar vuelos que dejan de ser factibles por el horario. Además el algoritmo considera que debe existir un tiempo mayor a 2 horas cuando se requiera hacer una conexión entre vuelos. Para calcular la ruta mínima se requiere un aeropuerto de salida, uno de llegada y una fecha deseada de salida. La ruta resultante, en caso de que exista, se buscará dentro de un periodo de una semana a partir del día especificado por el usuario.

El proceso se compone de las siguientes operaciones principales:

- Lectura de datos desde `parquet`.
- Filtrado de datos.
- Conteo de datos.
- Borrado de duplicados.
- Cálculo de registro mínimo
- Ordenamiento de datos.
- Suma de columnas a un escalar.
- Concatenación de *DataFrames*.
- Escritura de datos a SQL.

3.2.6. Registro de tiempo y resultados

Al finalizar la ejecución de cada proceso, se obtiene el tiempo de ejecución del mismo (*duration*). Después, esta información se escribe en una base de datos SQL junto con los detalles del ambiente en que se ejecutó (*description* y *resources*), la fecha de inicio (*start_ts*) y término de la ejecución (*end_ts*) en tiempo UNIX, la fecha de inserción del

registro (*insertion_ts*) y el nombre del proceso que se ejecutó (*process*). La siguiente tabla muestra un ejemplo de la información almacenada en la base de datos.

process	demoras_ruta_mktid_dask
start_ts	1615231907.4139209
end_ts	1615231916.5688508
duration	9.154929876327515
description	Ejecucion de prueba en equipo local.
resources	16 GB y 12 nucleos.
insertion_ts	2021-03-08 13:31:56

La información de la ejecución se registra en una de dos tablas dependiendo del *framework* elegido. Los resultados, por otro lado, se almacenan en una tabla específica que se sobre escribe cada que finaliza la ejecución de ese proceso.

3.3. Infraestructura

Los *frameworks* fueron comparados en dos ambientes distintos para probar su funcionamiento en modo *standalone* y distribuido en un clúster. El primer ambiente es el local y el segundo el clúster en la nube. Las siguientes secciones explicarán las características de los ambientes y las ejecuciones en ellas.

3.3.1. Ambiente local

A pesar de que las computadoras personales se quedan cortas para realizar grandes cargas de trabajo de *Big Data*, siguen siendo una herramienta importante para hacer análisis previo y realizar cargas de trabajo no muy grandes. Tanto *Dask* como *Spark* tienen la capacidad

de funcionar en una sola máquina y aprovechar los recursos para realizar las cargas de trabajo de forma más rápida que otras herramientas. Por esta razón fue importante probar ambos *frameworks* en un ambiente local.

Para ello se utilizó un equipo con las siguientes características:

Procesador	Intel Core i7-10750H CPU @ 2.60GHz
Núcleos	12
Memoria	15.5 GiB
Almacenamiento	1.1 TB
Sistema Operativo	Ubuntu 20.04.2 LTS

Los procesos fueron ejecutados mediante el proceso descrito en 1 y almacenados en una base de datos en MySQL.

Además, se utilizó un ambiente de *Anaconda* con *Python* 3.8.5. Este ambiente se utilizó para las ejecuciones con la versión 2021.1.1 de *Dask* y la versión 2.4.7 de *Spark*.

3.3.2. Ambiente en la nube

El ambiente en la nube tiene como objetivo probar el desempeño de *Dask* y *Spark* en un ambiente distribuido. Para esto, se utilizaron instancias **X** con las siguientes características:

Procesador	X
Núcleos	X
Memoria	X
Almacenamiento	X
Sistema Operativo	X

3.4. Datos

Todas las operaciones se hicieron sobre la base de datos: *Reporting Carrier On-Time Performance (1987-present)*, disponible en [3]. Este conjunto de datos está compuesto de 109 columnas que contienen información de demoras de vuelos dentro de Estados Unidos en el que cada registro corresponde a un vuelo único. Los datos tienen las siguientes columnas:

- YEAR: (Entero) Año en el que sucedió el vuelo.
- QUARTER: (Entero entre 1 y 4) Trimestre en el que sucedió el vuelo.
- MONTH: (Entero entre 1 y 12) Mes en el que sucedió el vuelo.
- DAY_OF_MONTH: (Entero entre 1 y 31) Día en el que sucedió el vuelo.
- DAY_OF_WEEK: (Entero entre 1 y 7) Día de la semana en en el que sucedió el vuelo. El 1 corresponde al Lunes.
- FL_DATE: (*string* en formato *yyyy-mm-dd*) Fecha en la que sucedió el vuelo.
- X...X

Los datos utilizados corresponden al periodo del 1^{ro} de enero de 2008 al 30 de Noviembre de 2020 y se componen de 80,345,298 registros.

3.4.1. Tratamiento de los datos

Estos datos fueron obtenidos en formato `csv` y convertidos a `parquet` para reducir su dimensión y acelerar el tiempo de consulta para ambos

frameworks . Una vez almacenado en 4096 *parquets*, los datos tienen un tamaño de 4.774 GB.

Para evitar que la partición de los datos beneficiara a alguno de las herramientas, se crearon dos copias de los datos y cada una fue particionada y almacenada de acuerdo a los lineamientos especificados en la documentación de la herramienta correspondiente.

En el caso de Dask, se siguieron los siguientes lineamientos que se especifican en [4]:

- Las particiones deben caber en memoria, preferiblemente ser menores a 1 GB.
- Se deben tener pocas particiones.
- Las particiones deben tener un tamaño aproximado de 100 MB.

Además, se creó el archivo `_metadata` que permite acelerar la lectura de los archivos.

Por otro lado, para Spark, el conjunto de datos fue almacenado particionado por año y mes para acelerar la consulta sobre estos periodos.

3.4.2. Muestras de datos

Para contrastar las herramientas bajo distintas condiciones, los procesos fueron ejecutados en 5 muestras de datos de los siguientes tamaños:

Muestra	Número de registros	Tamaño (MB)
10K	10,000	8
100K	100,000	16
1M	1,000,000	71
10M	10,000,000	2281
Total	80,345,298	4774

La obtención de las muestras se realizó utilizando la función `sample` de *Spark* sin reemplazo y con la semilla aleatoria 22102001.

Capítulo 4

Resultados

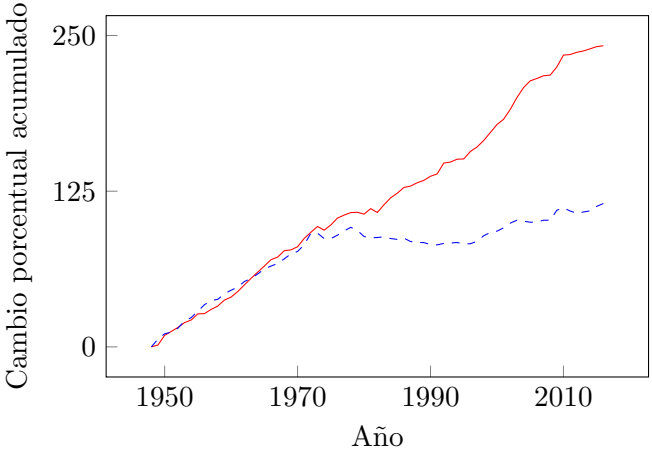
En este último capítulo se analiza el caso mexicano en dos partes: la dimensión nacional y la internacional.

4.1. Dimensión nacional

4.1.1. ¿Por qué no crecemos?

El Grupo Huatusco se ha reunido por más de quince años intentando responder la pregunta de por qué México no crece a mayores tasas.

Figura 4.1. Brecha entre productividad y remuneración laboral en el mundo (1948-2016)



Fuente: elaboración propia con datos del Economic Policy Institute.

Conclusiones

Hegel (1953) escribe en su obra *Lecciones sobre la filosofía de la Historia Universal* un par de conceptos que son útiles para comprender de manera abstracta la esencia detrás de esta investigación.

Apéndice A

Cuadros anexos

Referencias

Abramovitz, Moses. 1957. «Resources on Output Trends in the United States since 1870.» *The American Economic Review* 46 (2): 5–23.

Acemoglu, Daron. 2009. *Introduction to Modern Economic Growth.* Princeton: Princeton University Press.

On-Time : Reporting Carrier On-Time Performance (1987-present), $https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ = FGJ$.

Dask - Best Practices, $https://docs.dask.org/en/latest/dataframe - best - practices.html$.

Chambers, B. & Zaharia, M. (2018). Spark: The definitive guide. Beijing: O'Reilly.

Cluster mode overview. (n.d.). Retrieved March 24, 2021, from $https://spark.apache.org/docs/2{,}4{,}0/cluster - overview.html$.

Zahaira, M., Xin, R. S., Wendell, P., Das, T., Ambrust, M., Dave, A., . . . Stoica, I. (2016, November). Apache Spark: A Unified Engine for Big Data Processing. Retrieved March 24, 2021, from $https://people.eecs.berkeley.edu/\ alig/papers/spark - cacm.pdf$

Confianza institucional, inclusión social,
intensidad religiosa y percepción
tecnológica como factores de innovación
para el crecimiento económico,
escrito por Farid Hannan,
se terminó de imprimir en diciembre de 2018
en los talleres de Tesis Matozo.
Campeche 156, colonia Roma,
Ciudad de México.