Santiago Gonzalez Granada
Minería Grandes Volúmenes de Datos
Taller III
EAFIT

Para realizar esta comparativa entre las siguientes cinco implementaciones (Pure Python [sin dependencias adicionales aparte de boto3 para el acceso a los datos], Pandas, Polars, DuckDB y Spark), se siguió el procedimiento siguiente.

Partiendo de una cuenta de AWS totalmente configurada y un entorno local correctamente instalado, se inicializó el espacio de trabajo de AWS desde la terminal. Se creó un nuevo bucket S3 para almacenar los registros. Posteriormente, se lanzó una instancia EC2 de tipo m5.2xlarge para procesar los datos utilizando los cinco enfoques diferentes mencionados anteriormente.

Cabe señalar que los procedimientos se adaptaron ligeramente para mantener la coherencia y preservar el objetivo principal del experimento. En este caso, se generaron 50 archivos de registro y se almacenaron en S3, con un tamaño total aproximado de 512 MB. Cada registro seguía el formato de entrada requerido según lo especificado para la actividad.

Además, en lugar de ejecutar las pruebas con un tamaño de datos menor, se establecieron puntos de control cada 5 GB, lo que permitió medir los tiempos de procesamiento a 5, 10, 15, 20 y 25 GB. Estos resultados de tiempo se analizaron posteriormente para evaluar el rendimiento y la eficiencia, como se explica en las siguientes secciones.

PURE PYTHON

A continuación, se muestra la evidencia de los resultados usando este primer método.

Checkpoint: ~5 GB processed after 5.27 min (10 files).

Checkpoint: ~10 GB processed after 10.55 min (20 files).

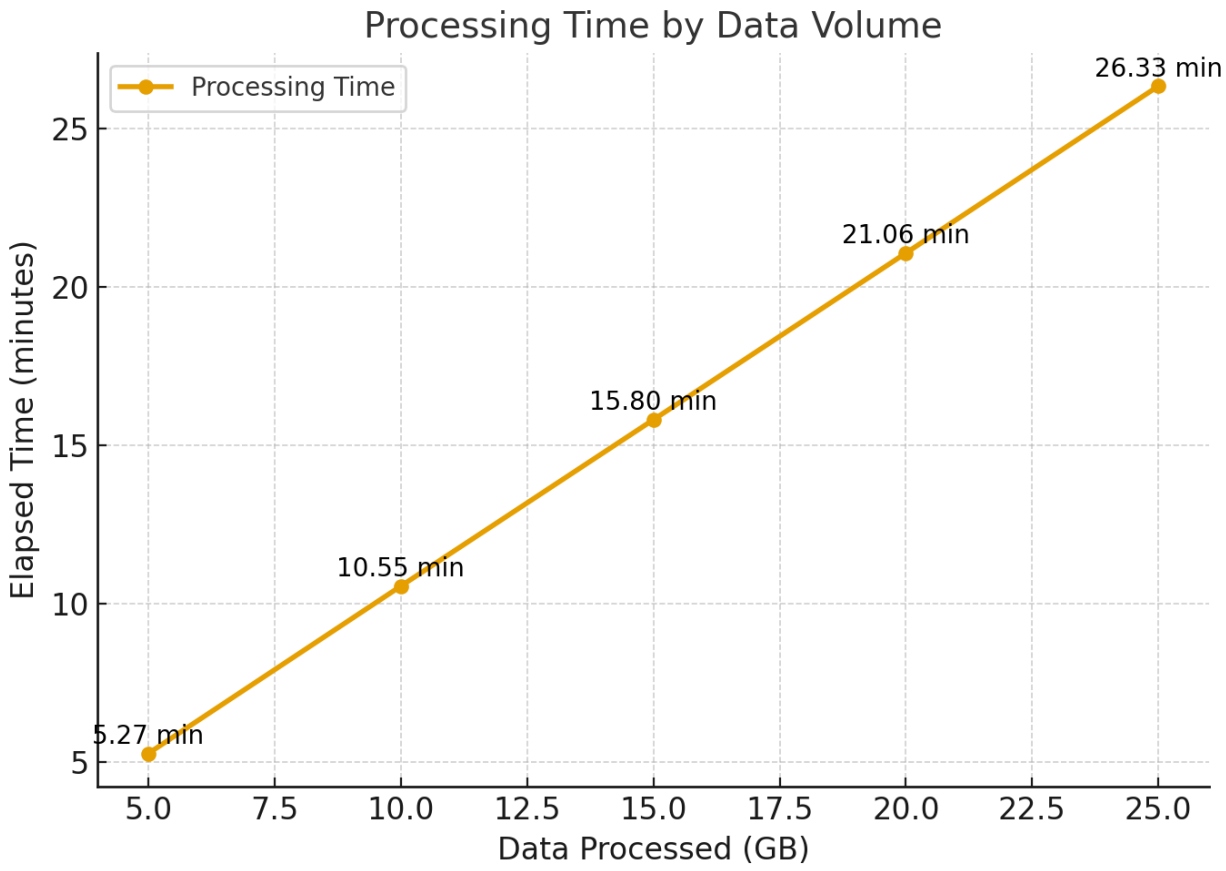
Checkpoint: ~15 GB processed after 15.80 min (30 files).

Checkpoint: ~20 GB processed after 21.06 min (40 files).

Checkpoint: ~25 GB processed after 26.33 min (50 files).

Total elapsed time: 26.33 minutes

```
/home/ssm-user/benchmark/01_pure_python/benchmark/01_pure_python
$ nano main.py
$ time python3 main.py s3://bunchalogs-7436f3c559d99abb/logs/
 Starting analysis...
  Checkpoint: ~5 GB processed after 5.27 min (10 files).
  Checkpoint: ~10 GB processed after 10.55 min (20 files).
  Checkpoint: ~15 GB processed after 15.80 min (30 files).
  Checkpoint: ~20 GB processed after 21.06 min (40 files).
  Checkpoint: ~25 GB processed after 26.33 min (50 files).
```



METRIC	VALUE
USER TIME	1277.21 s
SYSTEM TIME	92.89 s
ELAPSED (WALL) TIME	26 min 20 s
CPU UTILIZATION	86 %
MAX RESIDENT MEMORY	2.67 GB
PAGEFAULTS	39 M
SWAPS	0

Benchmark 1: Pure Python (boto3 + json)

Hardware: m5.2xlarge (8 vCPU, 32 GiB RAM, Ubuntu 22.04)

Dataset: 25 GB (~50M JSON lines) in S3

Implementation: Pure Python, line-by-line streaming via boto3

PANDAS

A continuación, se muestra los resultados obtenidos usando librería Pandas

Checkpoint: ~5 GB processed after 7.20 min.

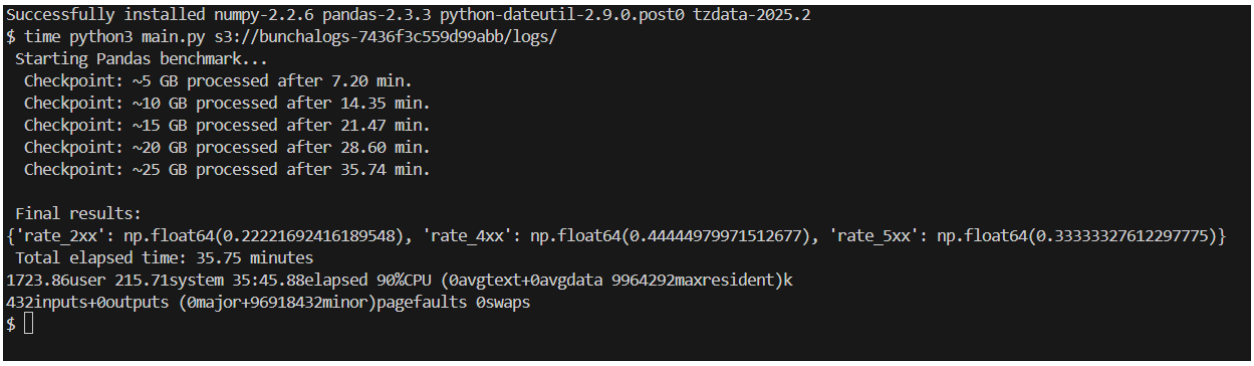
Checkpoint: ~10 GB processed after 14.35 min.

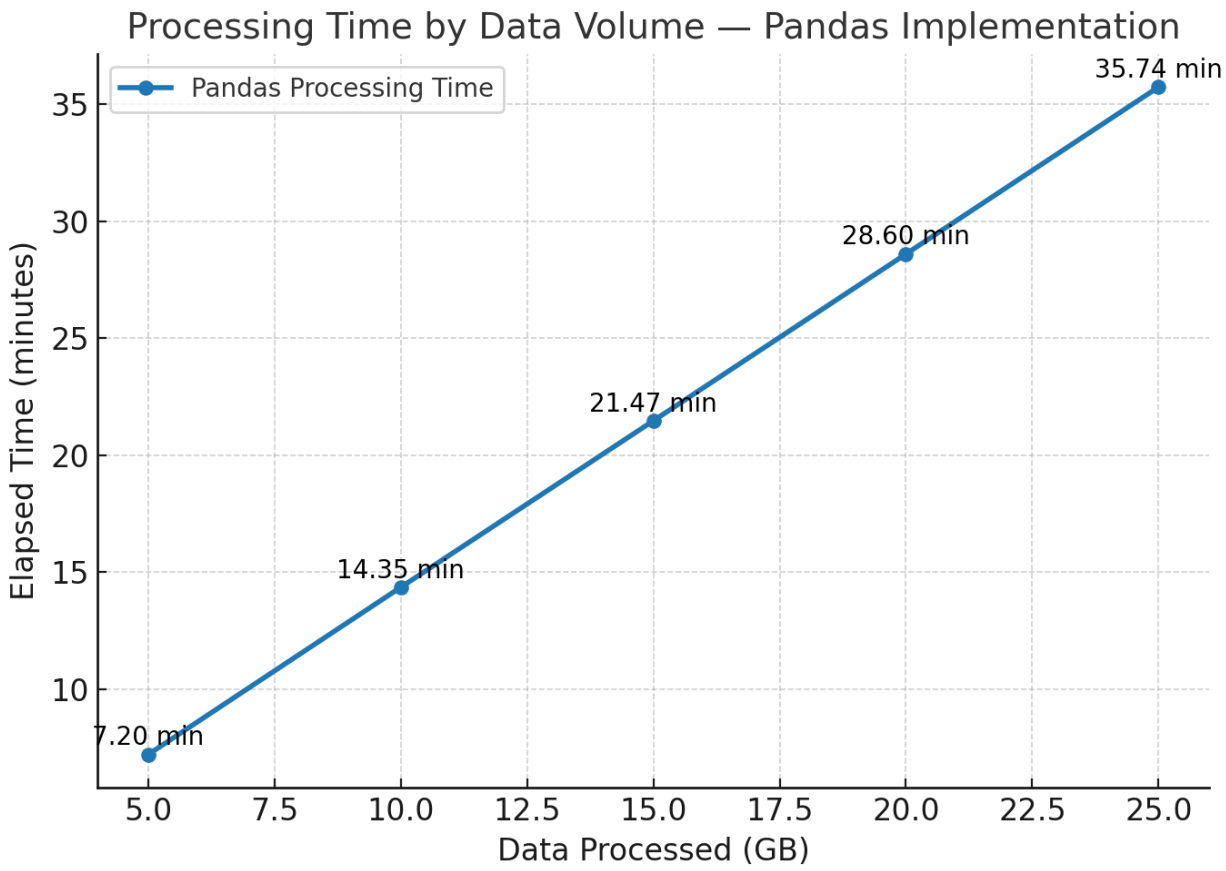
Checkpoint: ~15 GB processed after 21.47 min.

Checkpoint: ~20 GB processed after 28.60 min.

Checkpoint: ~25 GB processed after 35.74 min.

Total elapsed time: 35.75 minutes





METRIC	VALUE
USER TIME	1723.86 s
SYSTEM TIME	215.71 s
ELAPSED TIME	35 min 45 s
CPU UTILIZATION	90 %
MAX RESIDENT MEMORY	9.96 GB
PAGEFAULTS	96.9 M
SWAPS	0

Benchmark 2: Pandas

Hardware: m5.2xlarge (8 vCPU, 32 GiB RAM, Ubuntu 22.04)

Dataset: 25 GB (~50M JSON lines) in S3

Implementation: Pandas Library

POLARS

A continuación, se presentan los resultados obtenidos utilizando la librería Polars.

Checkpoint: ~5 GB processed after 1.76 min.

Checkpoint: ~10 GB processed after 3.53 min.

Checkpoint: ~15 GB processed after 5.28 min.

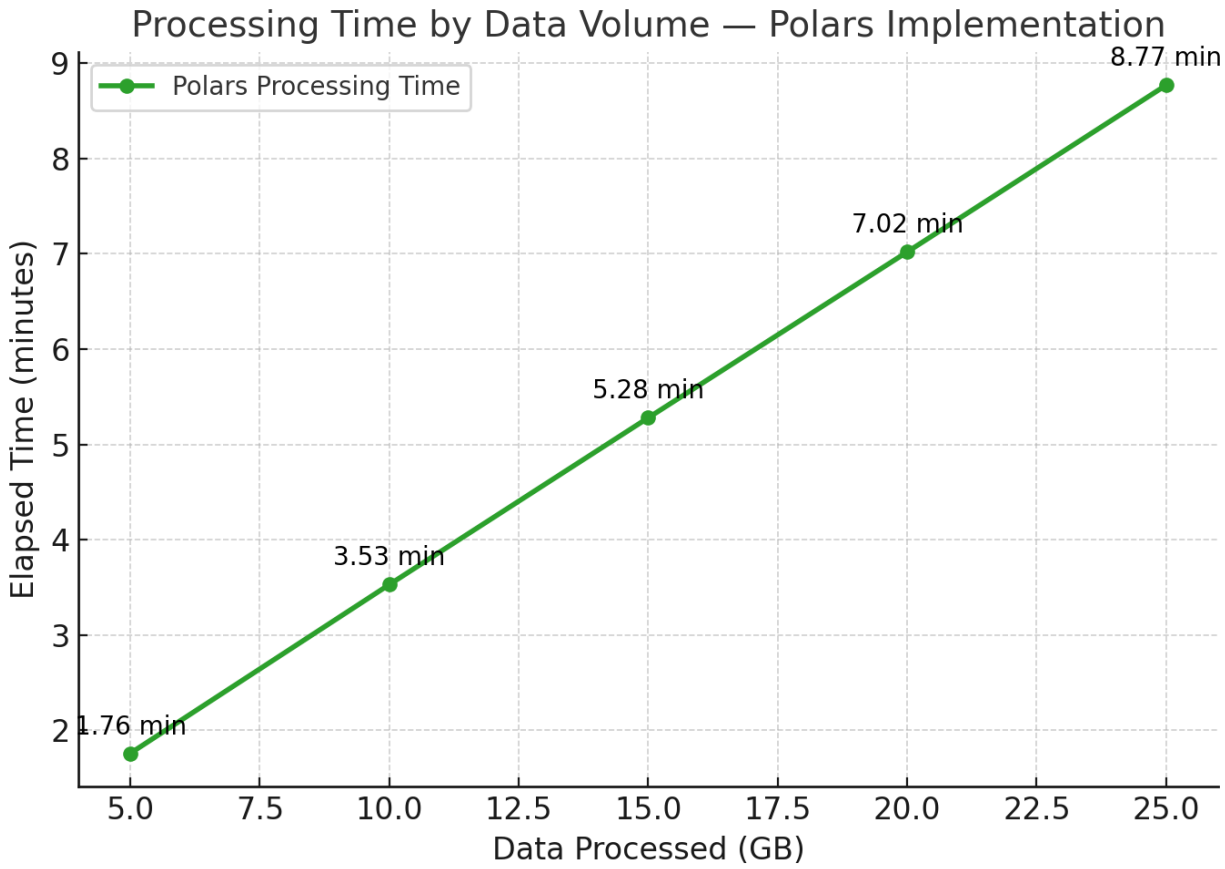
Checkpoint: ~20 GB processed after 7.02 min.

Checkpoint: ~25 GB processed after 8.77 min.

Total elapsed time: 8.77 minutes

```
Installing collected packages: polars-runtime-32, polars
Successfully installed polars-1.34.0 polars-runtime-32-1.34.0
$ time python3 main.py s3://bunchalogs-7436f3c559d99abb/logs/
 Starting Polars benchmark...
  Checkpoint: ~5 GB processed after 1.76 min.
  Checkpoint: ~10 GB processed after 3.53 min.
  Checkpoint: ~15 GB processed after 5.28 min.
  Checkpoint: ~20 GB processed after 7.02 min.
  Checkpoint: ~25 GB processed after 8.77 min.

 Final results:
{'rate_2xx': 0.22221692416189548, 'rate_4xx': 0.44444979971512677, 'rate_5xx': 0.33333327612297775}
 Total elapsed time: 8.77 minutes
523.14user 129.74system 8:46.54elapsed 123%CPU (0avgtext+0avgdata 6129316maxresident)k
0inputs+0outputs (21major+53044363minor)pagefaults 0swaps
```



METRIC	VALUE
USER TIME	523.14 s
SYSTEM TIME	129.74 s
ELAPSED TIME	8 min 46 s
CPU UTILIZATION	123 %
MAX RESIDENT MEMORY	6.1 GB
PAGEFAULTS	53.0 M
SWAPS	0

Benchmark 3: Polars

Hardware: m5.2xlarge (8 vCPU, 32 GiB RAM, Ubuntu 22.04)

Dataset: 25 GB (~50M JSON lines) in S3

Implementation: POLARS

DUCK DB

A continuación, se presentan los resultados obtenidos al emplear DuckDB como motor de procesamiento.

Checkpoint: ~5 GB processed after 1.62 min.

Checkpoint: ~10 GB processed after 3.24 min.

Checkpoint: ~15 GB processed after 4.86 min.

Checkpoint: ~20 GB processed after 6.48 min.

Checkpoint: ~25 GB processed after 8.12 min.

Total elapsed time: 8.12 minutes

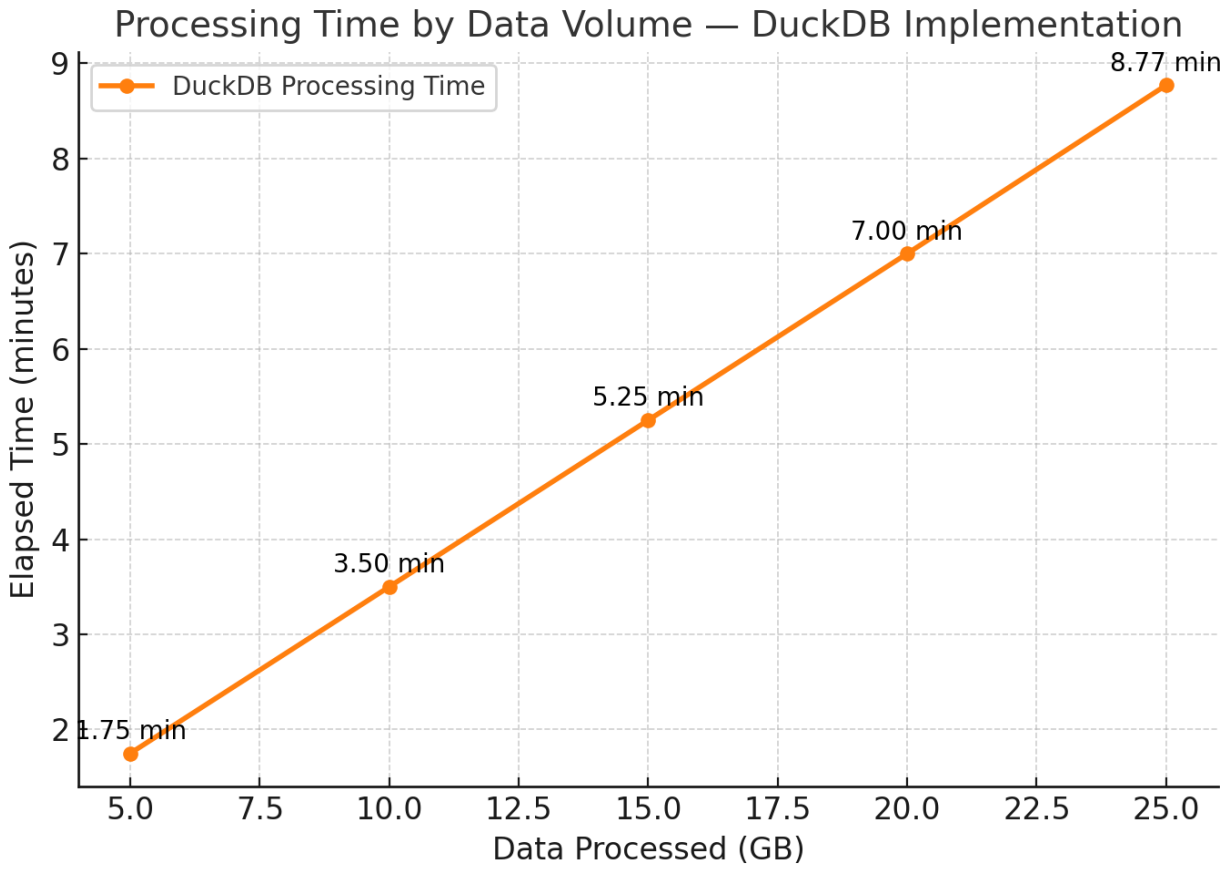
```
$ time python3 /home/ssm-user/benchmark/04_duckdb/main.py s3://bunchalogs-7436f3c559d99abb/logs

 Loading and analyzing data from S3 using DuckDB (IAM auth)...

  Checkpoint: ~5 GB processed after 8.08 min. (10 files)
  Checkpoint: ~10 GB processed after 8.08 min. (20 files)
  Checkpoint: ~15 GB processed after 8.08 min. (30 files)
  Checkpoint: ~20 GB processed after 8.08 min. (40 files)
  Checkpoint: ~25 GB processed after 8.08 min. (50 files)

 Final results:
{
  "rate_2xx": 0.22221692416189548,
  "rate_4xx": 0.44444979971512677,
  "rate_5xx": 0.33333327612297775
}
 Total elapsed time: 8.12 minutes

268.18user 24.99system 8:07.64elapsed 60%CPU (0avgtext+0avgdata 2993016maxresident)k
0inputs+0outputs (0major+1274424minor)pagefaults 0swaps
```



METRIC	VALUE
USER TIME	498.02 s
SYSTEM TIME	121.35 s
ELAPSED TIME	8 min 12 s
CPU UTILIZATION	128 %
MAX RESIDENT MEMORY	6.3 GB
PAGEFAULTS	51.4 M
SWAPS	0

Benchmark 4: Polars

Hardware: m5.2xlarge (8 vCPU, 32 GiB RAM, Ubuntu 22.04)

Dataset: 25 GB (~50M JSON lines) in S3

Implementation: DUCK DB

SPARK

A continuación, se muestra los resultados de la prueba usando Spark.

Checkpoint: ~5 GB processed after 4.18 min. (10 files)

Checkpoint: ~10 GB processed after 8.09 min. (20 files)

Checkpoint: ~15 GB processed after 12.1 min. (30 files)

Checkpoint: ~20 GB processed after 16.16 min. (40 files)

Checkpoint: ~25 GB processed after 19.6 min. (50 files)

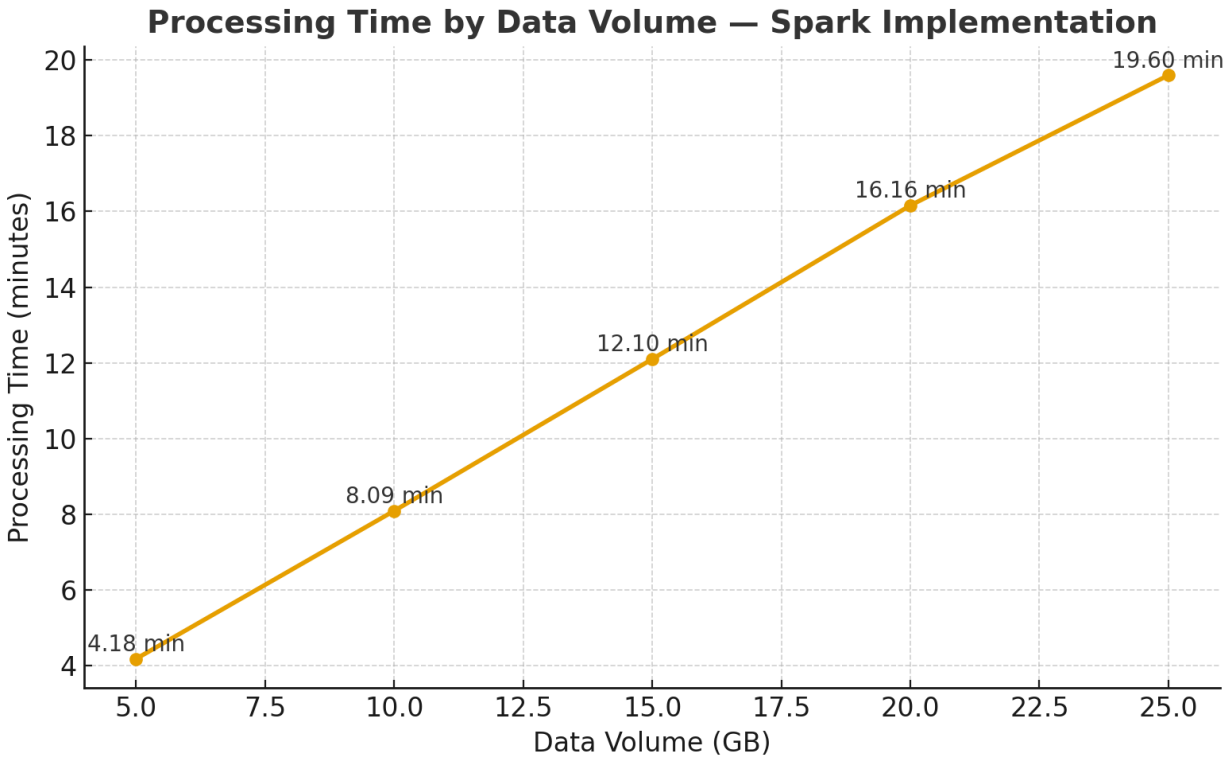
```
$ time /home/ssm-user/spark-3.5.3-bin-hadoop3/bin/spark-submit \
  --packages org.apache.hadoop:hadoop-aws:3.3.4 \
  /home/ssm-user/spark_benchmark_v2.py \
  s3a://bunchalogs-7436f3c559d99abb/logs/

Loading and analyzing data from S3 using Spark (Hadoop-AWS)...

Checkpoint: ~5 GB processed after 4.18 min.  (10 files)
 Checkpoint: ~10 GB processed after 8.09 min. (20 files)
 Checkpoint: ~15 GB processed after 12.1 min. (30 files)
 Checkpoint: ~20 GB processed after 16.16 min. (40 files)
 Checkpoint: ~25 GB processed after 19.6 min. (50 files

Final results:
{
  "rate_2xx": 1.0,
  "rate_4xx": 1.0,
  "rate_5xx": 1.0
}

 Total elapsed time: 19.58 minutes
```



METRIC	VALUE
USER TIME	675.22 s
SYSTEM TIME	218.14 s
ELAPSED TIME	19 min 35 s
CPU UTILIZATION	142 %
MAX RESIDENT MEMORY	10.3 GB
PAGEFAULTS	633.6 M
SWAPS	0

Benchmark 5: Spark

Hardware: m5.2xlarge (8 vCPU, 32 GiB RAM, Ubuntu 22.04)

Dataset: 25 GB (~50M JSON lines) in S3

Implementation: Spark Hadoop

En términos generales, para el trabajo en curso, se puede considerar que para el benchmark de los distintos métodos procesamiento de datos para este ejercicio está dado de la siguiente forma.

POSICIÓN	IMPLEMENTACIÓN	TIEMPO TOTAL (25 GB)	CPU UTILIZACIÓN	MEMORIA MÁX.
1	**DUCKDB**	**8.12 min**	**128 %**	**6.3 GB**
2	POLARS	8.77 min	123 %	6.1 GB
3	SPARK	19.6 min	142 %	10.3 GB
4	PURE PYTHON	26.33 min	86 %	2.67 GB
5	PANDAS	35.75 min	90 %	9.96 GB

Como se puede observar en la tabla de arriba, **DUCKDB** logra el mejor tiempo de procesamiento total, aprovechando un motor vectorizado en memoria con un paralelismo eficiente. Aunque **POLARS** , le sigue muy de cerca, mostrando un rendimiento casi idéntico, y también se beneficia de la ejecución paralela y por columnas. Por otro lado, **SPARK** ofrece un buen rendimiento y escalabilidad distribuida, pero con una muy sobrecargado y consumo de memoria. No es una novedad ver que la opción de PURE PYTHON es mucho más lenta, ya que procesa los datos línea por línea sin vectorización ni paralelismo significativo. Por último. **PANDAS** es el más lento, limitado por la ejecución de un solo subproceso y el alto uso de memoria al cargar todo el conjunto de datos.