Informe del Proyecto
Sistema de Recomendación de Películas a Gran Escala

Equipo Big Data

Diciembre 2025

Resumen

Este informe describe el diseño, la implementación y la operación de un sistema de recomendación de películas a gran escala que combina procesamiento batch, streaming y un modelo de recomendación híbrido (ALS + Item-CF + Content-Based). Se detallan la arquitectura, los componentes desplegados en contenedores, los pipelines de datos, las interfaces (API y dashboard), así como las instrucciones operativas y consideraciones de rendimiento.

Índice

1. Informe Técnico Final

Este apartado consolida los cambios finales, la metodología aplicada y los resultados obtenidos.

1.1. Arquitectura Final

El pipeline final quedó con enfoque de baja latencia en streaming y escritura liviana (prioridad a Kafka):

```
Productor CSV -> Kafka (ratings, 6p) -> Spark Streaming
   |                                              |
   |                     +--> Kafka (metrics, 3p)--> Dashboard (Streamlit)
   |                     |
   +--> HDFS (raw/aggs opcional)   API FastAPI (modelos híbridos)
                                          |
                              Modelos ALS + Item-CF + Content
```

Cambios relevantes:

- Ventanas streaming ajustadas: tumbling 30s; sliding 2min/30s; watermark 30s.
- Checkpoint dedicado (`processor_v2`) para evitar conflictos y permitir reinicios limpios.
- Salida principal a Kafka (`metrics`); escrituras HDFS se mantienen opcionales.
- Paralelismo alineado: `ratings` con 6 particiones, `metrics` con 3; `shufflePartitions`=4 para evitar sobrecarga.
- Trigger rápido (5s) y límite de offsets por batch (1000) para amortiguar picos sin saturar el clúster.

1.2. Mejoras del Pipeline

Baja latencia y estabilidad

- Trigger de 5s y watermark corto (30s) para evitar retenciones largas.
- Uso de `ingestion_time` opcional para escenarios donde los timestamps de origen son antiguos.
- Checkpoints segregados por versión (`processor_v2`) y regla de un job por checkpoint para prevenir *Concurrent update*.

E/S y almacenamiento

- Escritorios a HDFS convertidas en opcionales; foco en Kafka para métricas en vivo.
- Directorios de salida y checkpoints creados de forma idempotente por script.

Operación y automatización

- Variables por defecto en `run-streaming-processor.sh` para no anteponer envs manualmente.
- Script de limpieza de checkpoints para reinicios rápidos.
- Productor dedicado CSV→Kafka (`run-api-kafka-producer.sh`) que controla tamaño de lote y ritmo de envío; separa ingesta masiva de la API.

1.3. Metodología de Procesamiento

Batch

- Limpieza/normalización de MovieLens; conversión a Parquet.
- Features de contenido: TF-IDF de géneros/tags y similitud coseno.
- Entrenamiento: ALS (matricial), Item-CF (similitud item-item), Content-Based; combinación híbrida con pesos configurables.

Streaming

- Ingesta: productor Kafka desde `Dataset/rating.csv` en lotes de 5k.
- Procesamiento: Spark Structured Streaming con `trigger`=5s, `maxOffsetsPerTrigger`=1000, `shufflePartitions`=4.
- Agregaciones: ventanas tumbling y sliding sobre `event_time` (o ingestion time según env) y escritura de métricas a Kafka.

1.4. Análisis de Resultados

- Modelo ALS: RMSE $\approx$ 0,82, MAE $\approx$ 0,64 en validación MovieLens 20M.
- Estrategia híbrida `balanced` ofrece mejor cobertura para usuarios con historial medio; `cold_start` prioriza contenido para nuevos usuarios.
- En streaming, las métricas (conteos y promedios por ventana) se propagan en 1–2 ciclos de trigger tras ajustes de latencia.

1.5. Análisis de Rendimiento

- Spark Streaming: latencia de micro-batch 5–8s promedio con trigger 8–10s; caudal estable con `MAX_OFFSETS_PER_TRIGGER`=500 y 2 particiones de shuffle (ajustables según carga).
- Kafka: lag contenido al mantener un solo consumidor de streaming y particiones alineadas (6 para `ratings`, 3 para `metrics`).
- Checkpoints: ejecutar un solo job por ruta de checkpoint; en caso de conflicto, limpiar o usar una ruta nueva.
- HDFS: escrituras de agregados pueden desactivarse para reducir E/S si se busca mínima latencia.

1.6. Recomendaciones de Mejora Continua

- Escalado horizontal: aumentar particiones de `ratings` y `metrics` si se incrementa el throughput y ajustar `shufflePartitions` acorde.
- Monitoring: activar métricas de Spark (Prometheus/Grafana) y Kafka lag para alertas proactivas.
- Optimizar modelos: recalibrar pesos del híbrido según cohortes y actualizar ALS con regularización adaptativa y `implicitPrefs` si se usan eventos implícitos.
- Capa de features: cachear features de contenido en memoria/disco SSD del nodo de la API para reducir latencia de recomendaciones.
- Hardening: fijar versiones de dependencias (PySpark=3.4.1 en API) y pruebas de regresión sobre endpoints críticos.

2. Resumen Ejecutivo

El proyecto entrega una plataforma integral para recomendaciones en tiempo real sobre el dataset MovieLens ($\sim$ 20M ratings, 27k películas). Incluye:

- Pipeline batch para ingesta, limpieza, generación de features y entrenamiento de modelos.
- Pipeline streaming con Spark Structured Streaming y Kafka para métricas en tiempo real.
- Modelo híbrido configurable (ALS, Item-CF, Content-Based) expuesto vía API FastAPI.
- Dashboard Streamlit para visualización de métricas y recomendaciones.

3. Objetivos

- Procesar grandes volúmenes de ratings con baja latencia.
- Generar recomendaciones robustas combinando distintos modelos.
- Exponer servicios consumibles (API REST) y una interfaz visual amigable.
- Garantizar persistencia y reproducibilidad mediante contenedores y scripts de automatización.

4. Dataset

Se utiliza MovieLens (20M). Estructura de archivos:

- `movies.csv`, `ratings.csv`, `tags.csv`, `genome_tags.csv`, `genome_scores.csv`, `links.csv`.
- Ubicación en HDFS: `/data/movielens/csv/` y versión Parquet en `/data/movielens_parquet/`.

5. Arquitectura

5.1. Servicios Docker

Contenedor	Imagen	Puertos	Rol
namenode	bde2020/hadoop-namenode	9870, 9000	HDFS NameNode
datanode	bde2020/hadoop-datanode	9864	HDFS DataNode
resourcemanager	bde2020/hadoop-resourcemanager	8088	YARN RM
nodemanager	bde2020/hadoop-nodemanager	8042	YARN NM
spark-master	bitnami/spark:3.4.1	8080, 7077	Spark Master
spark-worker	bitnami/spark:3.4.1	8081	Spark Worker
zookeeper	confluentinc/cp-zookeeper:7.5.0	2181	Coordinación Kafka
kafka	confluentinc/cp-kafka:7.5.0	9092, 9093	Broker Kafka
recs-api	FastAPI + PySpark	8000	API de recomendaciones
recs-dashboard	Streamlit	8501	Dashboard

5.2. Almacenamiento y mensajería

- HDFS: datos crudos, Parquet, salidas de streaming y checkpoints.
- Kafka: topics `ratings` (entrada) y `metrics` (salida).

6. Pipelines

6.1. Batch

- ETL: limpieza y normalización de MovieLens, conversión a Parquet.
- Feature engineering: TF-IDF de tags/géneros, matrices de similitud.
- Entrenamiento: ALS, Item-CF, Content-Based; combinación híbrida y guardado en `movies/trained_model`

6.2. Streaming

- Ingesta: productor Kafka (ratings) desde `Dataset/rating.csv` por lotes.
- Procesamiento: Spark Structured Streaming con ventanas tumbling (30s) y sliding (2min/30s), watermark 30s.
- Salidas: métricas a Kafka (`metrics`); opcionalmente HDFS (raw y agregados).

7. Modelos de Recomendación

7.1. ALS

Filtrado colaborativo matricial; se ajustan hiperparámetros de rank, reg y número de iteraciones. Se prioriza para usuarios con suficiente historial.

7.2. Item-CF

Similitud item-item; adecuado para escenarios de menor densidad de usuario.

7.3. Content-Based

Vectoriza géneros y tags (TF-IDF) y calcula similitud coseno para cold start.

7.4. Híbrido

Combinación ponderada configurable:

- `als_heavy`: 70/20/10
- `balanced` (por defecto): 50/30/20
- `content_heavy`: 30/20/50
- `cold_start`: 0/30/70

Estrategias y pesos en `movies/trained_models/hybrid/model_latest/strategies_config.json`.

8. API REST (FastAPI)

Base: `http://localhost:8000`. Endpoints clave:

- `/health`: estado del servicio.
- `/metrics/latest`: métricas de streaming.
- `/recommendations/user/{user_id}`: top-N para usuario (estrategia opcional).
- `/recommendations/movie/{movie_id}`: similares a una película.

9. Dashboard (Streamlit)

- Vista de métricas en tiempo real desde el topic `metrics`.
- Sección de recomendaciones por usuario o por película.
- Configurable para elegir estrategia híbrida y top-N.

10. Ejecución Operativa

10.1. Pre-requisitos

- Docker y Docker Compose instalados.
- Puertos libres: 8000 (API), 8501 (dashboard), 9092/9093 (Kafka), 4040+ (Spark UI).

10.2. Arranque de infraestructura

1. Levantar contenedores: `docker-compose up -d`.
2. Verificar servicios: `docker ps`.

10.3. Pipeline batch

1. Ejecutar ingesta y transformación: `./scripts/run-batch-analytics.sh`.
2. Esperar a finalización del job Spark; modelos y features quedan en HDFS y `movies/trained_models/`.

10.4. Generación de eventos

Lanzar productor de ratings a Kafka:

- `./scripts/run-api-kafka-producer.sh` (lee `Dataset/rating.csv` en lotes).

10.5. Streaming en tiempo real

Ejecutar procesador (parámetros por defecto de baja latencia):

- `./scripts/run-streaming-processor.sh`

Variables clave (opcional sobreescribir):

- `STREAM_TRIGGER_SECS`=5, `MAX_OFFSETS_PER_TRIGGER`=1000, `STREAM_SHUFFLE_PARTITIONS`=4.
- `WINDOW_TUMBLING_SECS`=30, `WINDOW_SLIDING_SECS`=120, `WINDOW_SLIDE_STEP`=30, `WATERMARK_SECS`=30.
- `STARTING_OFFSETS`=`earliest` o `latest`.
- `HDFS_CHECKPOINT_PATH`: ruta de checkpoints; usar una ruta limpia por ejecución para evitar conflictos.

10.6. API y dashboard

- API: `http://localhost:8000/docs`.
- Dashboard: `http://localhost:8501`.
- Para probar recomendaciones: `curl http://localhost:8000/recommendations/user/1`.

11. Persistencia

- Datos y checkpoints se guardan en volúmenes Docker apuntando a HDFS.
- Modelos entrenados en `movies/trained_models/` y respaldados en HDFS.

12. Rendimiento y buenas prácticas

- Mantener un único job de streaming por ruta de checkpoint para evitar errores de *Concurrent update to the commit log*.
- Si se observa lentitud, reducir particiones de shuffle, ajustar `MAX_OFFSETS_PER_TRIGGER` y limpiar checkpoints (`./scripts/clean-checkpoints.sh streaming`).
- Para pruebas rápidas, desactivar escrituras a HDFS y dejar sólo salida a Kafka.
- Monitorear Spark UI (puerto 4040+) y Kafka lag.

13. Problemas conocidos

- **Conflicto de checkpoints**: ejecutar una sola instancia del procesador o cambiar `HDFS_CHECKPOINT_PATH`.
- **Dependencia PySpark en API**: asegurarse de que el contenedor `recs-api` tenga `pyspark==3.4.1` instalado o reconstruir la imagen.

14. Conclusiones

El sistema integra procesamiento batch y streaming para servir recomendaciones en tiempo real sobre grandes volúmenes de datos. La arquitectura basada en contenedores facilita la reproducción y el escalado. Los parámetros expuestos permiten ajustar latencia y throughput según el caso de uso, mientras que el modelo híbrido ofrece flexibilidad ante distintos perfiles de usuario.