

Trabajo Práctico

Fecha de Entrega: 8 de Octubre de 2025

Introducción

El modelo de computo concurrente fork-join permite el procesamiento de grandes volúmenes de datos de manera eficiente.

Para experimentar con ello, los alumnos deberán buscar y proponer un dataset estructurado que puedan obtener de sitios como Kaggle, con un volumen de datos significativo, al cual le apliquen algunas transformaciones; y donde puedan evaluar la performance del modelo al correr con diferente cantidad de workers.

Objetivo

Implementar una aplicación en Rust para procesamiento de información, aprovechando las ventajas del modelo Fork-Join. Para ello:

- Deberán buscar y proponer dataset de Kaggle, Hugging Face o cualquiera de su preferencia. Se recomienda que sea estructurado (con campos), aunque puede ser de texto. El tamaño mínimo deberá ser de 4GB.
- Deberán proponer un conjunto de transformaciones a aplicar, con agrupamientos. Por ejemplo en un dataset de reviews de hoteles, podrían buscar los 3 hoteles mejor puntuados por ciudad, y las 3 ciudades con más reviews por hotel. Tomar los TPs de cuatrimestres anteriores para mas ideas. Se deben aplicar por lo menos dos transformaciones.
- Deberán desarrollar el código en Rust para resolver estas transformaciones, utilizando las bibliotecas presentadas en clase para el modelo fork-join y/o herramientas de la librería standard.
- Deberán presentar un informe analizando cómo su código se comporta en la ejecución con distinta cantidad de procesadores, informando tiempos de ejecución y uso de memoria para al menos 3 ejecuciones distintas. Por ejemplo para ejecuciones con 1, 2 y 4 CPUs.

Restricciones

- Los datasets utilizados deberán ser diferentes entre los miembros del mismo grupo. Además, entre miembros de distintos grupos deberán intentar no repetir datasets. Si se encontraran datasets repetidos se validará exhaustivamente que los procesamientos y análisis sean efectivamente diferentes.
- El proyecto deberá ser desarrollado en lenguaje Rust, usando las herramientas de la biblioteca estándar.
- Se deberán utilizar las herramientas de concurrencia correspondientes al modelo fork-join.
- No se permite utilizar **crates** externos, salvo los explícitamente mencionados en este enunciado, en los ejemplos de la materia, o autorizados expresamente por los profesores. Para el procesamiento de JSON se puede utilizar el crate `serde_json`.
- El código fuente debe compilarse en la última versión stable del compilador y no se permite utilizar bloques unsafe.
- El código deberá funcionar en ambiente Unix / Linux.
- El programa deberá ejecutarse en la línea de comandos.
- La compilación no debe arrojar **warnings** del compilador, ni del linter **clippy**.
- Las funciones y los tipos de datos (**struct**) deben estar documentadas siguiendo el estándar de **cargo doc**.
- El código debe formatearse utilizando **cargo fmt**.
- Cada tipo de dato implementado debe ser colocado en una unidad de compilación (archivo fuente) independiente.

Entrega

La resolución del presente proyecto es individual.

La entrega del proyecto se realizará mediante Github Classroom. Cada estudiante tendrá un repositorio disponible para hacer diferentes commits con el objetivo de resolver el problema propuesto. Se recomienda iniciar tempranamente y hacer commits pequeños agreguen funcionalidad incrementalmente.

Se podrán hacer commits hasta el día de la entrega a las 19 hs Arg, luego el sistema automáticamente quitará el acceso de escritura.

El archivo README.md representará el informe de resolución y deberá incluir:

- Nombre y padrón del alumno.
- El link al dataset utilizado, junto a una descripción del mismo y la forma en que fue interpretado.

- Instrucciones para la correcta ejecución de la solución por los profesores y su parametrización para las distintas ejecuciones.
- Las transformaciones aplicadas al dataset y el formato del resultado esperado.
- Los análisis de performance.
- Conclusiones y posibles mejoras.

Evaluación

Principios teóricos y corrección de bugs

La evaluación se realizará sobre Github, pudiendo el profesor hacer comentarios en el repositorio y solicitar cambios o mejoras cuando lo encuentre oportuno, especialmente debido al uso incorrecto de herramientas de concurrencia.

Casos de prueba

Se someterá a la aplicación a diferentes casos de prueba que validen la correcta aplicación de las herramientas de concurrencia, por ejemplo, la ausencia de deadlocks.

Además la aplicación deberá mostrar algún incremento en performance cuando la ejecución de la misma se hace con varios hilos en un ambiente multiprocesador.

Informe

El informe debe estar estructurado profesionalmente y debe poder dar cuenta de las decisiones tomadas para implementar la solución.

Organización del código

El código debe organizarse respetando los criterios de buen diseño y en particular aprovechando las herramientas recomendadas por Rust. Se prohíbe el uso de bloques `unsafe`.

Tests automatizados

La presencia de tests automatizados que prueben diferentes casos, en especial sobre el uso de las herramientas de concurrencia es un plus.

Presentación en término

El trabajo deberá entregarse para la fecha estipulada. La presentación fuera de término sin coordinación previa con la cátedra influye negativamente en la nota final.