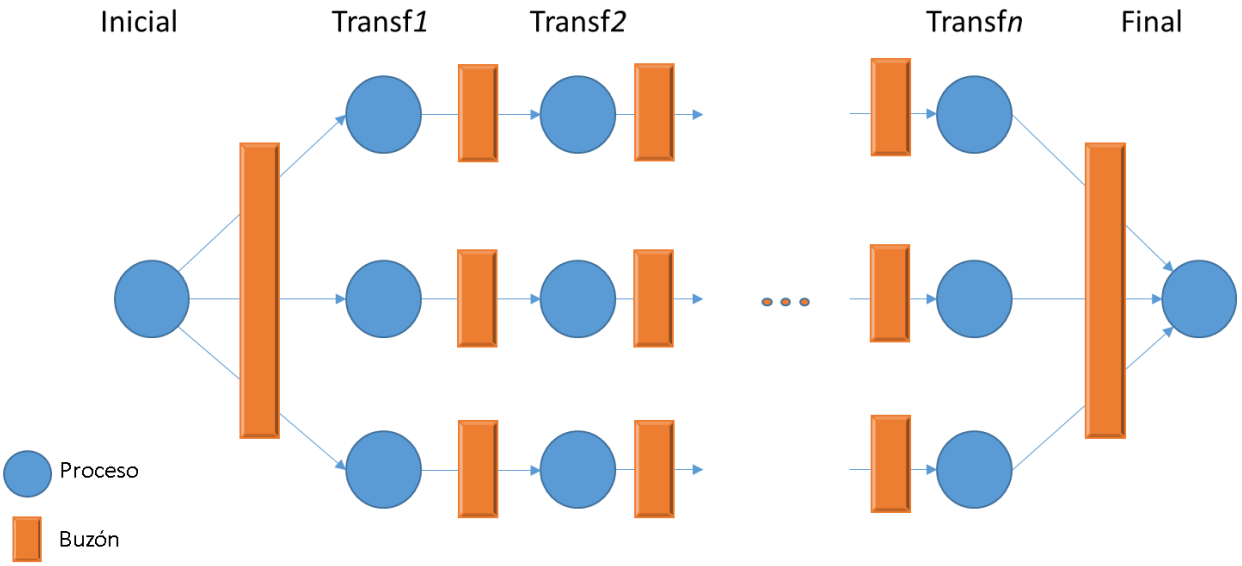
Universidad de los Andes
Departamento de Ingeniería de Sistemas y Computación
ISIS 2203 Infraestructura Computacional
202220

Caso 1
Manejo de la Concurrencia

MapReduce es un modelo de programación ampliamente utilizado para la ejecución de programas paralelos que utilizan grandes cantidades de datos. En este modelo, el conjunto de datos se divide en varios subconjuntos para ser tratados y transformados en paralelo hasta que se vuelven a integrar para producir el resultado final. La gráfica ilustra este modelo. Aquí, un conjunto de datos de entrada va sufriendo transformaciones en paralelo hasta lograr el resultado final:



Para este caso, vamos a definir que los datos iniciales se dividen en N subconjuntos y cada subconjunto es transformado 3 veces (3 niveles) antes de entregar los resultados para su integración. Cada nivel de transformación está a su vez conformado por 3 procesos independientes. Tenemos así 11 procesos en total: 1 proceso inicial que tiene los datos originales y los divide en N subconjuntos que va entregando para su procesamiento, 9 procesos que representan 3 niveles de transformación de 3 procesos cada uno y un proceso final que recibe las transformaciones y las integra.

La comunicación entre los procesos se hará siempre por buzones, para ello cada proceso de transformación tiene asociado un buzón de entrada y uno de salida. Para los procesos de transformación, cada proceso del nivel **i** tiene un buzón de salida que es el mismo buzón de entrada del correspondiente proceso del nivel **i+1**. El proceso inicial tiene un único buzón de salida que es conocido por todos los procesos del primer nivel de transformación y el proceso final tiene un único buzón de entrada que es conocido por todos los procesos del último nivel de transformación.

Objetivo

Diseñar un mecanismo de comunicación para implementar la arquitectura descrita. Para este caso, los procesos serán *threads* en la misma máquina (en realidad debería ser un sistema distribuido; este es solo un prototipo). La

comunicación siempre pasa a través de los buzones. Por ejemplo, para comunicar el proceso A con el proceso B, el proceso A debe almacenar el mensaje en su buzón de salida y el proceso B debe retirar el mensaje de su buzón de entrada (que debe ser el mismo buzón de salida de A).

El sistema recibe (por consola) el número de subconjuntos a generar (N), el tamaño de los buzones intermedios y el tamaño de los buzones de los extremos. Todos los buzones intermedios son del mismo tamaño, al igual que los buzones del proceso inicial y final tienen la misma capacidad, pero este último valor puede ser distinto al tamaño de los buzones intermedios. Una vez implementada la arquitectura, los subconjuntos empiezan a viajar desde el proceso inicial, se transforman en los procesos de transformación y llegan al proceso final para ser integrados. El proceso inicial distribuye los N subconjuntos entre los procesos del nivel de transformación 1 simplemente depositando los N mensajes en su buzón de salida; los procesos de transformación de nivel 1 competirán por tomar estos mensajes y transformarlos.

Para simular los subconjuntos vamos a manejar mensajes (un texto que puede ser un consecutivo que identifique el mensaje) y, para simular las transformaciones, cada proceso agrega una marca que permita saber quién realizó la transformación. Por ejemplo, si los procesos de transformación los identificamos como NM siendo N el nivel de transformación y M un distintivo entre 1 y 3 para los 3 procesos de un mismo nivel, un mensaje inicial "M1" que es procesado por el primer proceso del primer nivel de transformación, podría representar sus transformaciones llegando a destino como "M1T11T21T31", si cada proceso agrega "TNM" al texto que recibe.

El proyecto debe ser realizado en java, usando *threads*. Para la sincronización solo pueden usar directamente las funcionalidades de Java vistas en clase: `synchronized, wait, notify, notifyAll, sleep, yield, join` y las `CyclicBarrier`.

Funcionamiento

El thread que representa al proceso inicial envía los N mensajes iniciales a su buzón de salida desde donde los tomarán los procesos de transformación de nivel 1. Una vez ha terminado de enviar los N mensajes (N > 3), envía mensajes de "FIN" para indicar a los procesos en los niveles de transformación que deben terminar. Note que se deben enviar tantos mensajes de FIN como procesos intermedios existan en cada nivel y que este mensaje NO debe ser transformado para que todos lo reciban sin alteración. El proceso final recibe todos los mensajes, incluidos 3 mensajes de FIN y así sabe que debe integrar todos los resultados.

Los procesos de transformación ejecutan un ciclo recibiendo, trasformando y enviando los mensajes. Una vez reciben el mensaje de "FIN" terminan su ejecución.

El acceso a los buzones debe ser controlado para evitar incoherencias en el sistema. Un buzón que está lleno no puede aceptar más mensajes. Un buzón que está vacío no puede entregar mensajes. Todos los buzones de comunicación son de tamaño 3.

- El proceso inicial envía de manera semiactiva (si el buzón donde va a depositar está lleno, cede el procesador, pero vuelve a solicitarlo inmediatamente).
- El proceso final recibe de manera semiactiva (si el buzón de donde va a retirar está vacío, cede el procesador, pero vuelve a solicitarlo inmediatamente).
- Los procesos de transformación envían y reciben de manera pasiva (si no puede completar la acción de envío y/o recepción, el proceso espera a que le avisen que ya puede realizar la acción).

La transformación de un mensaje debe agregar un tiempo de espera para simular esta operación. Esto es, además de efectivamente modificar el mensaje, un thread debe esperar un tiempo aleatorio de entre 50 y 500 milisegundos antes de intentar enviar el mensaje.

El programa debe generar mensajes suficientemente claros para evidenciar el buen funcionamiento de este. En particular debe ser claro que todos los mensajes llegaron a destino, los nodos que intervinieron en la transformación de cada mensaje, las demoras que se aplicaron en cada mensaje para simular su transformación y que no hay

mensajes por recibir en el nodo final. Recuerde que su programa será analizado por un tercero que no lo conoce y debe entender fácilmente los resultados.

Condiciones de entrega

- En un archivo .zip entregar el código fuente del programa, y un documento Word explicando el diseño (diagrama de clase) y funcionamiento del programa, así como la validación realizada. En particular, para cada pareja de objetos que interactúan, explique cómo se realiza la sincronización, así como el funcionamiento global del sistema. El nombre del archivo debe ser: caso1_*login1*_*login2*_*login3*.zip
- El trabajo se realiza en los grupos definidos en el curso para este caso. No debe haber consultas entre grupos.
- El grupo responde solidariamente por el contenido de todo el trabajo, y lo elabora conjuntamente (no es trabajo en grupo repartirse puntos o trabajos diferentes).
- Habrá una coevaluación del trabajo en grupo. Cada miembro del grupo evaluará a sus dos compañeros y las notas recibidas por un estudiante ponderarán la nota final de caso para ese estudiante.
- En el parcial se incluirá una pregunta sobre el desarrollo de alguna de las funcionalidades del caso. La nota obtenida en esa pregunta puede afectar la nota de todos los miembros del grupo.
- El proyecto debe ser entregado por Bloque Neón por uno solo de los integrantes del grupo. **Al comienzo del documento Word, deben estar los nombres y carnés de los integrantes del grupo**. Si un integrante no aparece en el documento entregado, el grupo podrá informarlo posteriormente, sin embargo, habrá una penalización: la calificación asignada será distribuida (dividida de forma equitativa) entre los integrantes del grupo.
- **Se debe entregar por BNe a más tardar el lunes 5 de septiembre a las 23:55 (p.m.)**

RUBRICA:

Tengan en cuenta que el código debe correr en los casos de prueba definidos para la sustentación. El funcionamiento y calidad del programa entregado será evaluado con los siguientes ítems.

- Entradas y salidas claras del programa.
- Arquitectura de comunicación.
- Comunicación activa.
- Comunicación pasiva.
- Informe.