Big Data FlumeJava

Texto Base: Juan Esquivel Rodríguez
Luis Alexánder Calvo Valverde

FlumeJava

- Ejemplo de tecnología de procesamiento de datos a gran escala
- Evolución de MapReduce
- FlumeJava: Facilitar el desarrollo de data pipelines
  - Secuencia de operaciones para transformar datos
  - Bajo MapReduce, la complejidad para el programador era muy alta
- Los programadores deben ser capaces de desarrollar pipelines ágilmente
- Facilita la optimización distribuida de los pipelines

FlumeJava: Abstracciones

- PCollection<T>
  - Almacena colecciones de tamaño posiblemente enorme
  - Puede adoptar versiones ordenadas (PSequence) o no
- PTable<K,V>
  - Almacena un multi-mapa inmutable
  - Subclase de PCollection<Pair<K,V>>
  - Esta segunda nace por el uso común de llaves/valor en procesamiento distribuido

FlumeJava: Operaciones

- ParallelDo
  - Operación canónica
  - Recibe una clase "functor" que procesa la entrada, la transforma y retorna el resultado ajustado

FlumeJava: Operaciones

- DoFn<T, S>
  - Definen el mapeo entre las entradas T y las salidas S.
  - Conceptualmente estas operaciones se pueden pensar como secuenciales, aunque Flume se encarga de distribuir la carga de procesamiento como se requiera.
  - Es necesario indicar cómo se codificarán los datos de salida (e.g. collectionOf(strings()).
- Subclases DoFn
  - MapFn: Asume que la llave retornada es la misma.
  - FilterFn: Retorna los mismos valores excepto los que la función de filtro indique.

FlumeJava: Agregaciones

- Las transformaciones base en Flume cambian los objetos de entrada a objetos de salida en relación 1:N
- Se utilizan operaciones de "sintetización de datos"
- **groupByKey**
  - Juega el papel del paso shuffle en MapReduce
- **combineValues**
  - Caso especial de parallelDo
  - Recibe PTable<K, Collection<V>> y una función
  - Opera sobre objetos de tipo V y retorna PTable<K, V>
  - Esta función es la que nos permite unir múltiples valores en 1
  - El ejemplo más sencillo de esto es realizar un conteo.

FlumeJava: Modelo de ejecución

- Similar a Apache Spark
- Evaluación diferida (perezosa)
- Operaciones crean un plan de ejecución hasta que es necesario materializar
  - Grafo acíclico dirigido
- Optimizador actúa sobre el plan de ejecución
  - Integra múltiples pasos programados en pasos individuales
  - Determina localidad de ejecución y almacenamiento

FlumeJava: Ejemplo ejecución

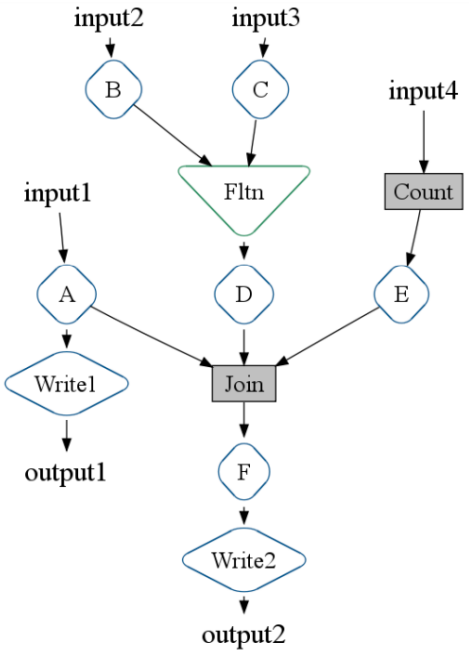


Figure 1. Initial execution plan for the SiteData pipeline.

FlumeJava: Fusión ParallelDo

- Composición de funciones $g(f())$
  - Cuando se tiene una operación f y el resultado es consumido por otra operación g
- Fusión productor-consumidor
  - Si el resultado de f no es utilizado afuera de una composición, no se materializa
  - Se puede agregar las instrucciones de f a la operación g
- Fusiones operaciones hermanas
  - Cuando múltiples ParallelDo leen la misma fuente de datos

FlumeJava: Fusión ParallelDo

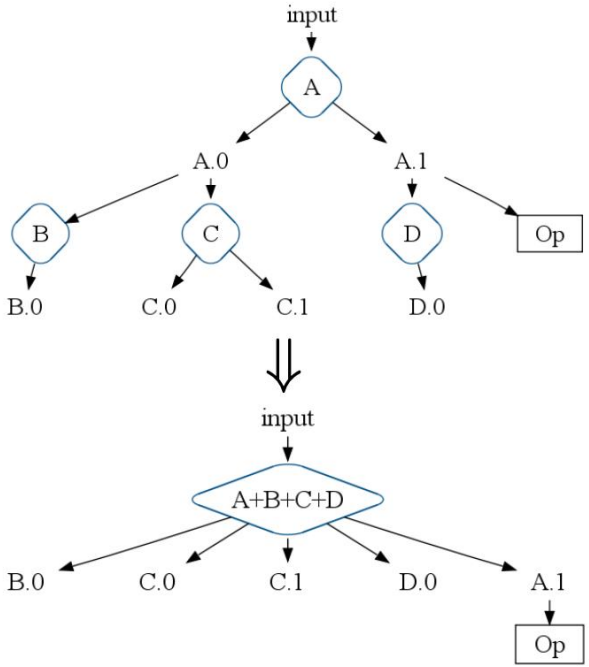


Figure 2. ParallelDo Producer-Consumer and Sibling Fusion.

FlumeJava: MSCR

- MapShuffleCombineReduce
- Convertir siempre tratará de convertir combinaciones ParallelDo, GroupByKey, CombineValues y Flatten en un MapReduce sencillo.
- Para cumplir ese objetivo se realiza un paso intermedio
  - Hay M canales de entrada y R canales de reducción, o salida.
  - Entre ambos M y R se puede tener, de manera opcional, una operación de Shuffle y una de Combinación.
  - Se relajan las restricciones estándar de MapReduce que un reductor debe producir salidas con la misma llave que las entradas.

FlumeJava: MSCR

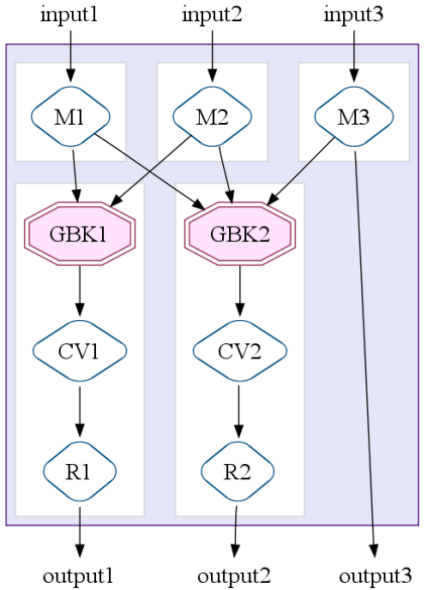


Figure 3. A MapShuffleCombineReduce (`MSCR`) operation with 3 input channels, 2 grouping output channels, and 1 pass-through output channel.

FlumeJava: Fusión MSCR

- Orientada hacia agrupamientos de datos que comparten informacion entre canales a través de operaciones ParallelDo.
- Evita la necesidad de múltiples operaciones flatten. Realiza una única operación MSCR.

FlumeJava: Fusión MSCR

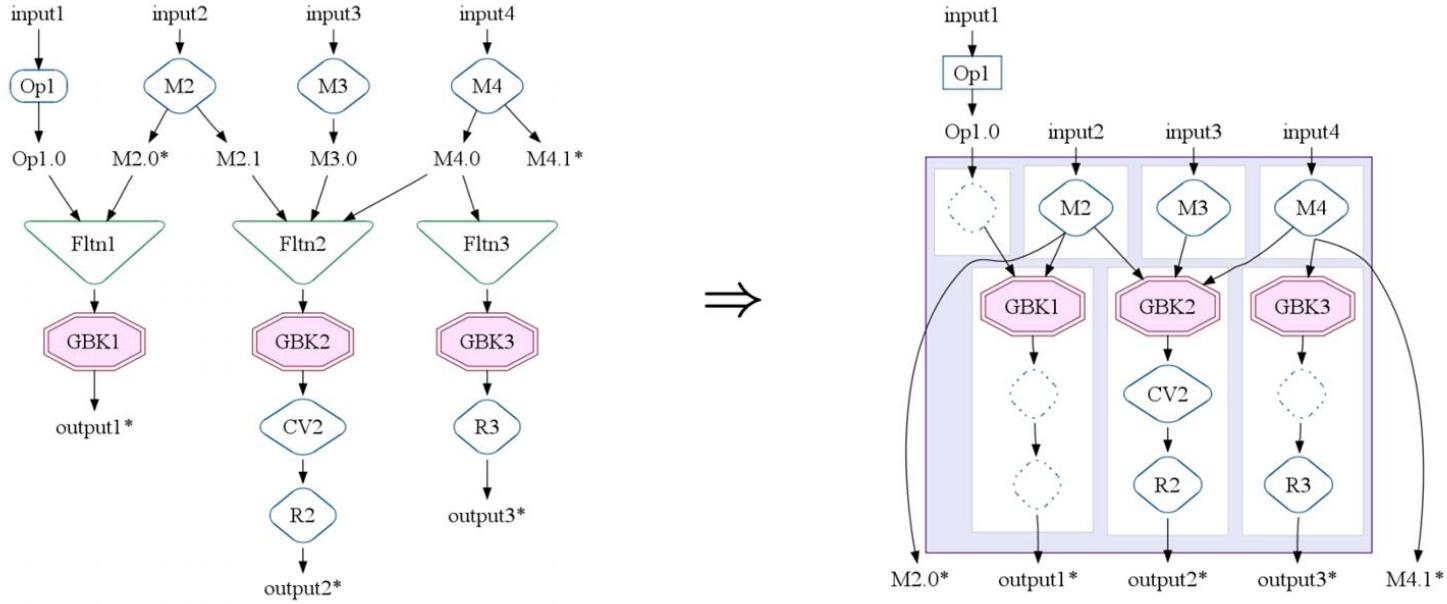


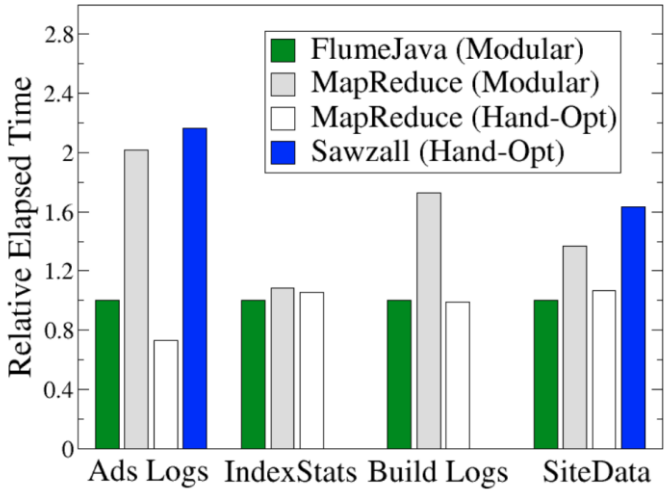
Figure 4. An example of `MSCR` fusion seeded by three `GroupByKey` operations. Only the starred `PCollections` are needed by later operations.

FlumeJava: Ventajas

- No es necesario saber cómo ni dónde se almacena la información
  - Puede ser en memoria, en disco o algún servicio (e.g. Bigtable)
- No es necesario definir cómo se ejecutan las operaciones
  - Local en la máquina con un ciclo
  - Ejecución distribuida
  - Consulta a un servicio externo
- Las pruebas pueden realizarse en memoria local
  - Favorece Test Driven Development, permitiendo a los programadores crear pequeños casos de prueba en forma de colecciones.
- Se puede implementar la ejecución utilizando evaluación diferida
  - El optimizador opera sobre el grafo de ejecución sin intervención

FlumeJava: Benchmark

- Hipótesis: Rendimiento de data pipelines en FlumeJava es comparable a un(a) buen(a) programador(a) de Map Reduce
  - No necesariamente experto(a) pero experimentados(as)
  - Debido a la considerable curva de aprendizaje en MapReduce, es suficiente



Referencias

- Chambers, C; Raniwala, A; Perry, F; Adams, S; Henry, R; Bradshaw, R; Weizenbaum, N. FlumeJava: Easy, Efficient Data-Parallel Pipelines. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/35650.pdf