Batch to Flink Conversion Processing

To convert batch processing to Flink, you essentially move your existing batch job logic into Flink's DataStream API. This involves configuring the stream execution environment to use batch mode and then adapting your data operations to work with Flink's streaming data processing model.

Flink v2 batch processing workflow breakdown:

1. Configure the Execution Environment:

- Use `StreamExecutionEnvironment.getExecutionEnvironment()` to obtain the execution environment.
- Set the runtime mode to `RuntimeExecutionMode.BATCH` using `env.setRuntimeMode(RuntimeExecutionMode.BATCH)`.
- You can also configure this via command line: `./bin/flink run -Dexecution.runtime-mode=BATCH <jarFile>`.

2. Adapt Your Data Source:

- Ensure your data source is bounded, meaning the amount of data is known beforehand.
- Use Flink's DataStream API to read your data. For example, you can read from a file using `readCsvFile`, `readTextFile`, or `readFileOfPrimitives`.

3. Implement Your Transformations:

- Use Flink's DataStream API operations (e.g., `map`, `filter`, `flatmap`, `groupBy`, `reduce`, `join`) to perform your desired transformations.
- These operations are similar to those in other batch processing frameworks, like Informatica, Apache Hadoop, Apache Spark, AWS Batch, and Google Cloud Dataflow, but they are designed to work with streams.

4. Define Your Sink:

- Specify where you want to write the results based on the existing batch programming. Common sinks include writing to files, databases, or other data stores.

5. Consider Data Exchange (Shuffle):

- In batch mode, Flink offers different shuffle modes (e.g., Blocking Shuffle, Hybrid Shuffle) to manage data exchange between operators.

- You can configure the shuffle mode using `execution.batch-shuffle-mode`.
  Note: Hybrid Shuffle mode is experimental and not something you can use in production and has limitations. Essentially, Blocking means don't do anything until you have read all the data, while the newer Hybrid mode is trying to process the data in sub-batches as it's consumed.

6. Test and Optimize:

- Thoroughly test your Flink job to ensure it produces the same results as your original batch job.
- Optimize your job by tuning parameters like parallelism, memory usage, and shuffle mode.

Existing Workflow:

Batch job that reads a CSV file, performs some filtering and aggregation, and then writes the result to another file.

Flink program for batch processing

Java

```
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.streaming.api.datastream.DataStream;
import
org.apache.flink.streaming.api.environment.StreamExecutionEnviro
nment;

public class FlinkBatchExample {

    public static void main(String[] args) throws Exception {
        // 1. Configure the execution environment
        StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
        env.setRuntimeMode(RuntimeExecutionMode.BATCH);

        // 2. Define your data source (replace with your actual
CSV file)
        DataStream<String> inputData =
env.readTextFile("path/to/your/input/file.csv");

        // 3. Implement your transformations (example: filter
and count)
        DataStream<String> filteredData = inputData
                .map(new MapFunction<String, String>() {
```

```
                    @Override
                    public String map(String value) throws
Exception {
                        // Replace with your actual filtering
logic
                        if (value.contains("someCondition")) {
                            return value;
                        }
                        return null;
                    }
                })
                .filter(s -> s != null);

        // 4. Define your sink (example: write to a file)

filteredData.writeAsText("path/to/your/output/file.txt");

        // 5. Execute the job
        env.execute("Flink Batch Example");
    }
}
```