Spark Streaming

Summary by Lucy Yu
Motivation

• Most of “big data” happens in a streaming context
  – Network monitoring, real-time fraud detection, algorithmic trading, risk management

• Current Model: Continuous Operator Model
  – Fault tolerance achieved via replication or upstream backup
Replication
Upstream Backup
D-Streams

• “Instead of managing long-lived operators, the idea ...is to structure a streaming computation as a series of stateless, deterministic batch computations on small time intervals.”

• Use a data structure: Resilient Distributed Datasets (RDDs)
  – keeps data in memory
  – can recover it without replication (track the lineage graph of operations that were used to build it)
D-Streams

https://spark.apache.org/docs/1.2.0/streaming-programming-guide.html
D-Streams

• The data received in each interval stored reliable across the cluster to form an input dataset for that interval

• Do batch operation to get another RDD, which acts as a state or output
D-Stream API

• Users register one or more streams using a functional API

• **Transformations** create a new D-Stream from parent stream(s)
  – Stateless: map, reduce, groupBy, join
  – Stateful: window operations

• **Output operations** let the program write data to external systems (e.g. save)
## Transformations

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map(func)</code></td>
<td>Return a new DStream by passing each element of the source DStream through a function <code>func</code>.</td>
</tr>
<tr>
<td><code>flatMap(func)</code></td>
<td>Similar to <code>map</code>, but each input item can be mapped to 0 or more output items.</td>
</tr>
<tr>
<td><code>filter(func)</code></td>
<td>Return a new DStream by selecting only the records of the source DStream on which <code>func</code> returns true.</td>
</tr>
<tr>
<td><code>reduce(func)</code></td>
<td>Return a new DStream of single-element RDDs by aggregating the elements in each RDD of the source DStream using a function <code>func</code> (which takes two arguments and returns one). The function should be associative so that it can be computed in parallel.</td>
</tr>
<tr>
<td><code>updateStateByKey(func)</code></td>
<td>Return a new &quot;state&quot; DStream where the state for each key is updated by applying the given function on the previous state of the key and the new values for the key.</td>
</tr>
</tbody>
</table>
Window Operations

https://spark.apache.org/docs/1.2.0/streaming-programming-guide.html
# Window Operations

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>window(windowLength, slideInterval)</code></td>
<td>Return a new DStream which is computed based on windowed batches of the source DStream.</td>
</tr>
<tr>
<td><code>countByWindow(windowLength, slideInterval)</code></td>
<td>Return a sliding window count of elements in the stream.</td>
</tr>
<tr>
<td><code>reduceByWindow(func, windowLength, slideInterval)</code></td>
<td>Return a new single-element stream, created by aggregating elements in the stream over a sliding interval using <code>func</code>. The function should be associative so that it can be computed correctly in parallel.</td>
</tr>
<tr>
<td><code>reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])</code></td>
<td>When called on a DStream of <code>(K, V)</code> pairs, returns a new DStream of <code>(K, V)</code> pairs where the values for each key are aggregated using the given reduce function <code>func</code> over batches in a sliding window.</td>
</tr>
<tr>
<td><code>reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval, [numTasks])</code></td>
<td>A more efficient version of the above <code>reduceByKeyAndWindow()</code> where the reduce value of each window is calculated incrementally using the reduce values of the previous window.</td>
</tr>
</tbody>
</table>
Fault Recovery

• Parallel recovery of a lost node’s state.
  – When a node fails, each node in the cluster works to recompute part of the lost node’s RDDs, resulting in significantly faster recovery than upstream backup without the cost of replication.

• In a similar way, D-Streams can recover from stragglers using speculative execution

• Checkpoint state RDDs periodically