Pig on Storm Proposal

Table of Contents

Introduction

One of the prime philosophies of Pig is that "Pigs live anywhere." Although Pig currently only supports a Hadoop backend, it was never intended to be the only available backend. The Spork fork of Pig introduces Spark as a backend to Pig. This proposal discusses a strategy for implementing a Storm backend to Pig. For those unfamiliar with Storm, Storm is a free and open source distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. In order to come up with an implementation strategy, the core concepts of Storm must first be understood. Relevant concepts will be discussed briefly below. Full documentation can be read at https://github.com/nathanmarz/storm/wiki/Concepts

Storm Concepts

- **Topology** - A topology is a graph of computation. Each node in a topology contains processing logic, and links between nodes indicate how data should be passed around between nodes.
- **Tuple** - Storm uses tuples as its data model. A tuple is a named list of values (same as Pig), and a field in a tuple can be an object of any type.
- **Stream** - The core abstraction in Storm is the "stream". A stream is an **unbounded** sequence of tuples. Storm provides the primitives for transforming a stream into a new stream in a distributed and reliable way.
- **Spout** - A spout is a source of streams in a topology. Generally spouts will read tuples from an external source and emit them into the topology.
- **Bolt** - All processing in topologies is done in bolts. Bolts can do anything from filtering, functions, aggregations, joins, talking to databases, and more.
- **Stream Grouping** - Part of defining a topology is specifying for each bolt which streams it should receive as input. A stream grouping defines how that stream should be partitioned among the bolt's tasks.

blocked URL

Challenges

Pig relational operators with clear semantics in Storm

The following Pig operators have clear semantics in Storm and can be easily implemented. Since the semantics are decided, these should have little impact on overall functionality.

- **FILTER** - Filter has very clear semantics in Storm as Filter's operate on a single Tuple (as Bolt's do) and choose to emit the Tuple or swallow the Tuple.
- **FOREACH** - Foreach in Pig operates over a single Tuple from a Relation. This is analogous to a Bolt processing a single Tuple as they are streamed to it.
- **SPLIT** - Split has clear semantics in Storm since you can split a stream of tuples into multiple streams based on any number of criteria.
- **UNION** - Union can be easily implemented by simply merging the streams into a single stream.
- **SAMPLE** - Since sample decides to emit a Tuple randomly based on a probability, sample also has clear semantics in Storm.
- **DISTINCT** - Distinct can be easily implemented by ensuring that Tuples with the same content are grouped on to the same Bolt Task and emit only distinct Tuples. (Will be stateful).
- **LIMIT** - Limit has clear semantics because you can stop a stream after emitting the given number of tuples. (Will be stateful).

Pig relational operators with unclear semantics in Storm

The following Pig operators have unclear semantics and Storm. The type of operation you need will vary per application. For example, for some applications a join means, join all tuples for two streams over a finite window of time, whereas other applications expect exactly one tuple for each side of the join for each join field. Other applications may do the join completely differently. Deciding the semantics of these operators is critical to the implementation.

- **JOIN (inner/outer)** - A streaming join combines two or more data streams together based on some common field. Whereas a normal database join has finite input and clear semantics for a join, a streaming join has infinite input and unclear semantics for what a join should be.
- **GROUP** - Grouping relations has semi-clear semantics. A type of Stream Grouping called FieldsGrouping can be used to ensure that Tuples with the same group key will be sent to the same task. However, in Pig a group operator allows you to construct a row around the set of tuples that have the group key and allow you to work with the entire row as a single unit. In Storm, for a group key, you get an *unbounded stream* of Tuples.
- **CROSS** - Computing the cross product of unbounded streams doesn't make sense.
- **ORDER BY** - Sorting doesn't make sense on an unbounded stream of tuples. Order by is generally used in conjunction with limit to implement a top-n computation.
- **RANK** - Similar to order by and limit.

Other Pig operators

The following are Pig operators that are not relational. Deciding the semantics of the following should have little impact on overall functionality.

- **STREAM** - Stream has clear semantics in Storm. Storm has out-of-the-box support for sending Tuples to external processes via the use of ShellBolt.
- **MAPREDUCE** - MapReduce has unclear semantics as to how the output of a MapReduce job will integrate into a streaming workflow.
- **LOAD** - Load has unclear semantics in Storm, since Storm doesn't "load" data. Spouts are sources for streams of Tuples.
- **STORE** - Store also has unclear semantics in Storm as storm doesn't have to output any data, but it can do so. Sink Bolt's usually write data into a database or some other persistant storage.

Trident

Introduction

Trident is a high-level abstraction for doing realtime computing on top of Storm. It allows you to seamlessly intermix high throughput (millions of messages per second), stateful stream processing with low latency distributed querying. For pig users, the concepts of Trident will be very familiar – Trident has joins, aggregations, grouping, functions, and filters. In addition to these, Trident adds primitives for doing stateful, incremental processing on top of any database or persistence store. Trident has consistent, exactly-once semantics, so it is easy to reason about Trident topologies.
Trident processes the stream as small batches of tuples. For example, an incoming stream of sentences might be divided into batches like so:

blocked URL

Generally the size of those small batches will be on the order of thousands or millions of tuples, depending on your incoming throughput.

Trident provides a fully fledged batch processing API to process those small batches. The API is very similar to what you see in high level abstractions for Hadoop like Pig or Cascading: you can do group by's, joins, aggregations, run functions, run filters, and so on. Of course, processing each small batch in isolation isn't that interesting, so Trident provides functions for doing aggregations across batches and persistently storing those aggregations – whether in memory, in Memcached, in Cassandra, or some other store. Finally, Trident has first-class functions for querying sources of realtime state. That state could be updated by Trident (like in this example), or it could be an independent source of state.

State

A key problem to solve with realtime computation is how to manage state so that updates are idempotent in the face of failures and retries. It's impossible to eliminate failures, so when a node dies or something else goes wrong, batches need to be retried. The question is – how do you do state updates (whether external databases or state internal to the topology) so that it's like each message was only processed only once?

This is a tricky problem, and can be illustrated with the following example. Suppose that you're doing a count aggregation of your stream and want to store the running count in a database. If you store only the count in the database and it's time to apply a state update for a batch, there's no way to know if you applied that state update before. The batch could have been attempted before, succeeded in updating the database, and then failed at a later step. Or the batch could have been attempted before and failed to update the database. You just don't know.

Trident solves this problem by doing two things:

1. Each batch is given a unique id called the "transaction id". If a batch is retried it will have the exact same transaction id.
2. State updates are ordered among batches. That is, the state updates for batch 3 won't be applied until the state updates for batch 2 have succeeded.

Execution of Trident topologies

Trident topologies compile down into as efficient of a Storm topology as possible. Tuples are only sent over the network when a repartitioning of the data is required, such as if you do a groupBy or a shuffle. So if you had this Trident topology:
blocked URL
It would compile into Storm spouts/bolts like this:
blocked URL

Q & A

Would a physical plan be translated to a Storm job (or jobs)? (Alan Gates on Pig mailing list)

A physical plan can most likely be directly translated into a Storm topology.

Would it need a different physical plan? (Alan Gates on Pig mailing list)

I don't think a different physical plan would be needed but I'm not sure.

Would you just have the connection at the language layer and all the planning separate? (Alan Gates on Pig mailing list)

This seems like the simplest solution since we can connect to Trident at the language layer and have Trident handle the planning.

Do you envision needing extensions/changes to the language to support Storm? (Alan Gates on Pig mailing list)

Yes. Pig currently only has support for LoadFunc and StoreFunc and these are heavily integrated into the hadoop Input/OutputFormat concepts. There will also need to be additional concepts that Pig is missing right now, mainly a DataSource (or something like that) concept for identifying stream's of data. The parallelism concept in Pig will probably have to be extended/modified to allow users to be able to control parallelism for the Storm topologies.

Thoughts

Any approach to a streaming extension to Pig should be done with care to allow for other streaming implementations (such as S4).

Resources

1. S4 - S4 is a general-purpose, distributed, scalable, fault-tolerant, pluggable platform for processing continuous unbounded streams of data.
2. HStreaming - HStreaming is a real-time data analytics platform to analyze, process, store and archive streaming data on Hadoop.
3. Summingbird - Summingbird is a platform for streaming map/reduce used at Twitter to build aggregations in real-time or on hadoop.
4. Spark - Spark is an open source cluster computing system that aims to make data analytics fast — both fast to run and fast to write.
5. Spork - Pig on Spark
6. Jubatus - Jubatus is a distributed processing framework and streaming machine learning library.

Acknowledgements

Much of the information about Storm and Trident has been copied (shamelessly) from the original documentation that can be found at https://github.com/nathanmarz/storm/wiki. I'd like to thank the Storm team for providing great documentation!