Comments on Streaming Deep Learning with Flink

Dean Wampler, Ph.D.
dean@lightbend.com
@deanwampler

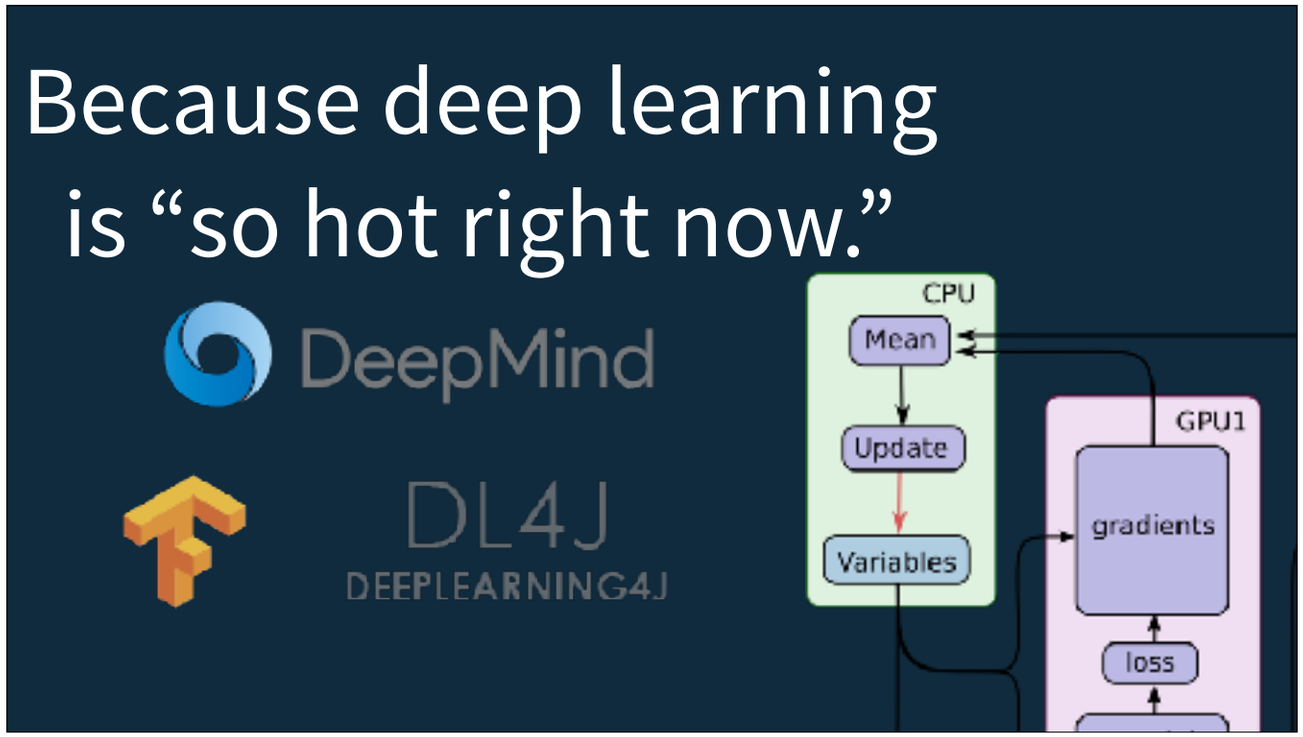
Flink Forward San Francisco, April 11, 2017

© Lightbend, 2014–2017. All rights reserved.

- Why DL for Streaming? Why DL for Flink?
- Model Training vs. Inference
- Practical Approaches with Flink
- The Future of DL with Flink

Outline - This is a status report of work in progress at Lightbend. It's far from complete and definitely not "battle tested", but this talk reflects our current thinking and approach.

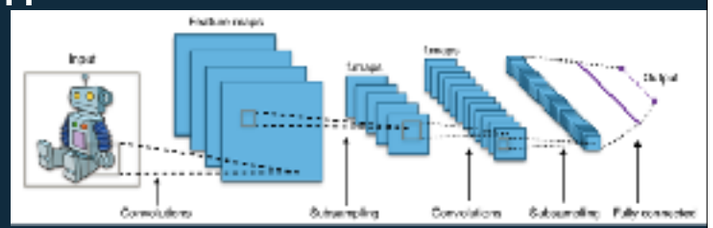
Why DL for Streaming?
Why DL for Flink?



Images: https://github.com/tensorflow/models/tree/master/inception, https://deeplearning4j.org, https://github.com/tensorflow, and https://deepmind.com/

What Is deep learning?

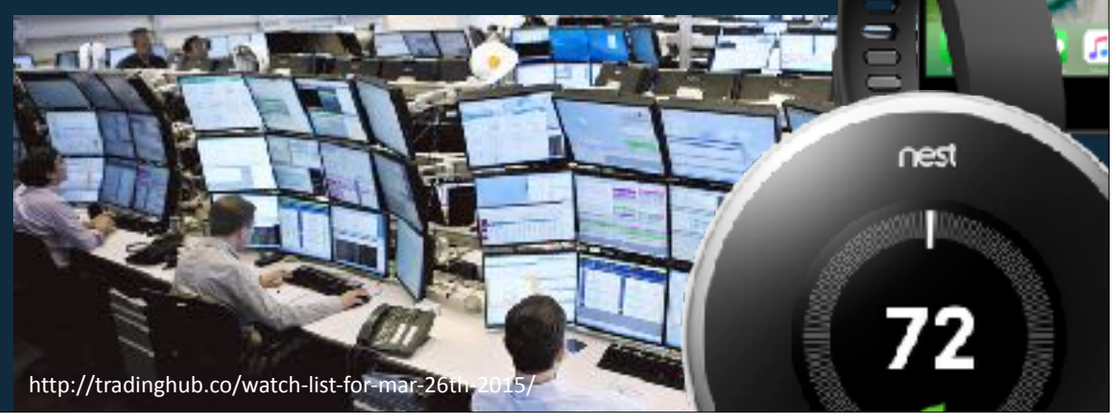
- A very sophisticated non-linear model
- A way to discover features, rather than engineer them



https://en.wikipedia.org/wiki/Convolutional_neural_network

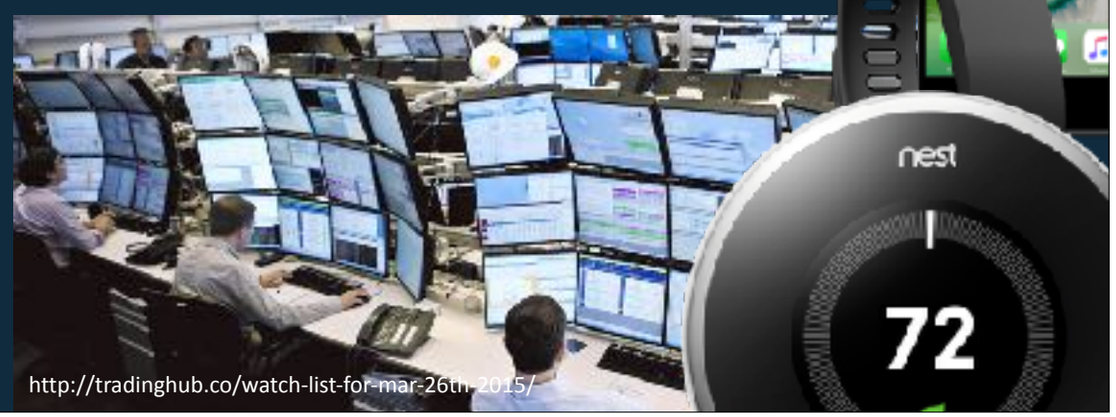
I won’t explain DL, but offer a few points. It’s not mysterious. It’s a sophisticated non-linear modeler. It’s also a tool for learning the “features” (the characteristics of the data), rather than engineering them explicitly.

Because streaming is “so hot right now.”



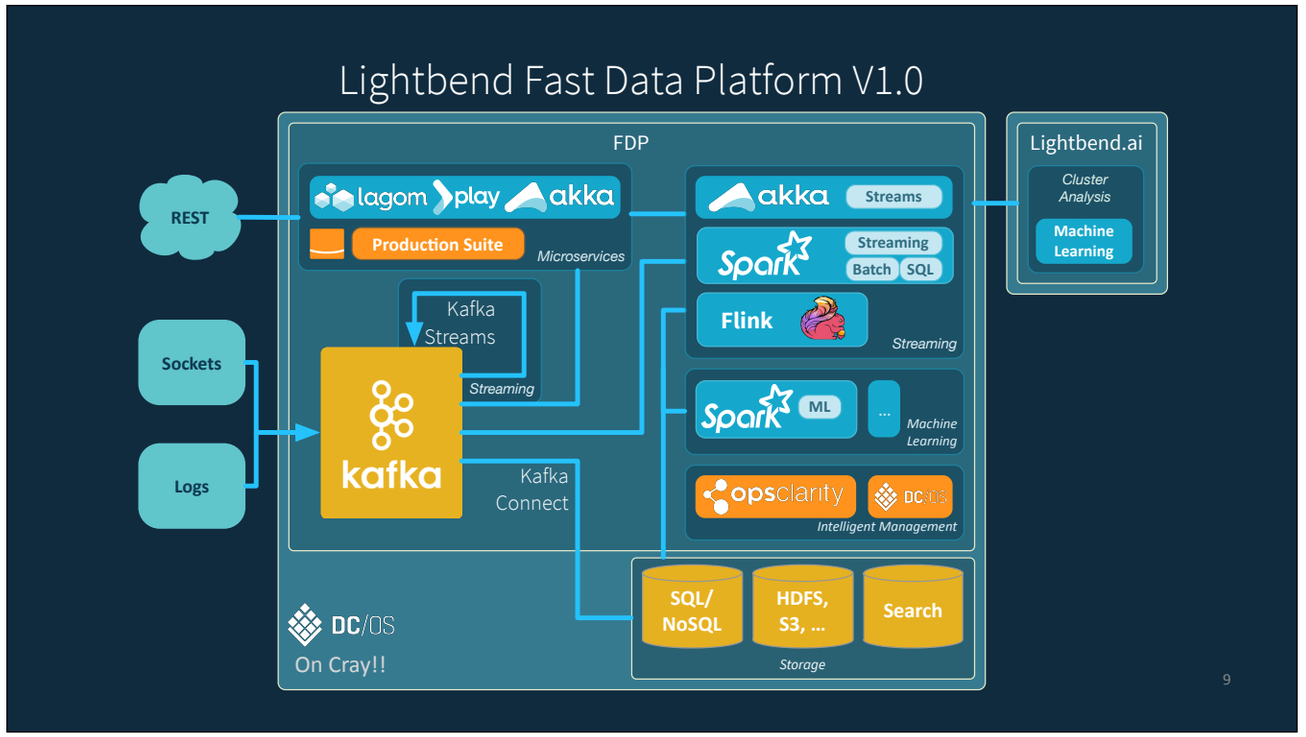
http://tradinghub.co/watch-list-for-mar-26th-2015/

… and Flink provides large scale and low latency!

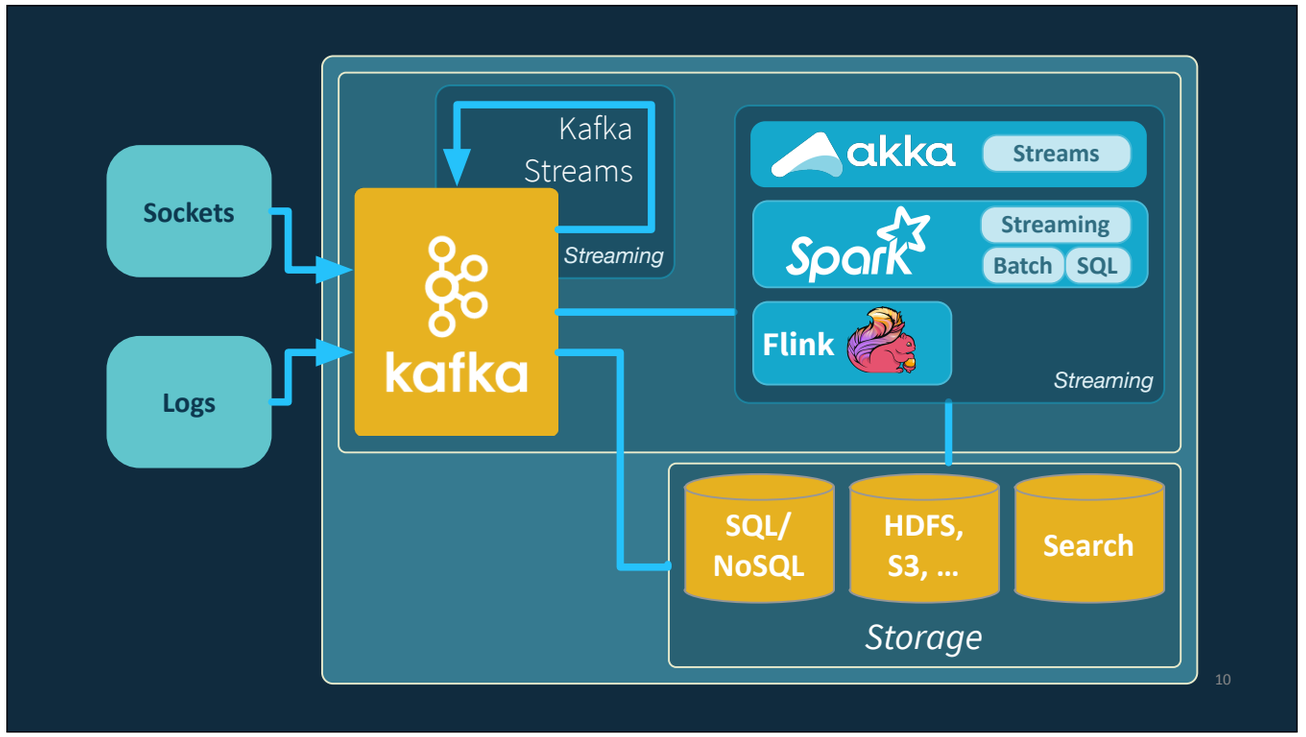


http://tradinghub.co/watch-list-for-mar-26th-2015/

Why Lightbend likes Flink

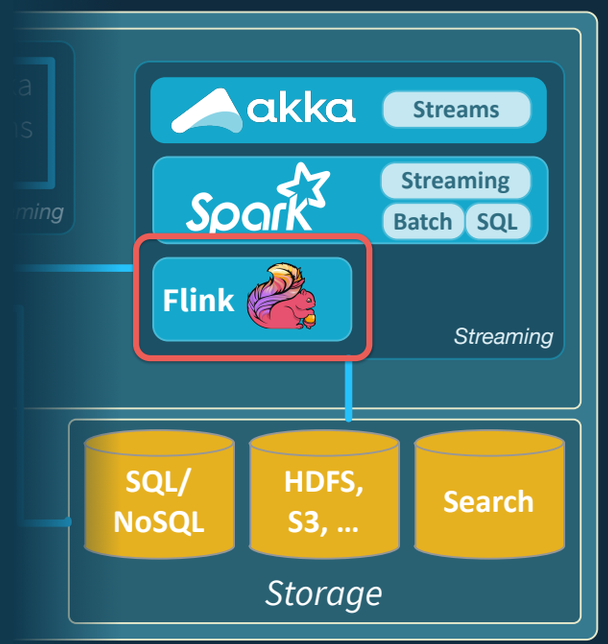


The Fast Data Platform that we're building as a commercial, streaming data processing stack. I won't go through the details (but I'll show a link at the end for more information). Rather, I'll focus on the streaming core…

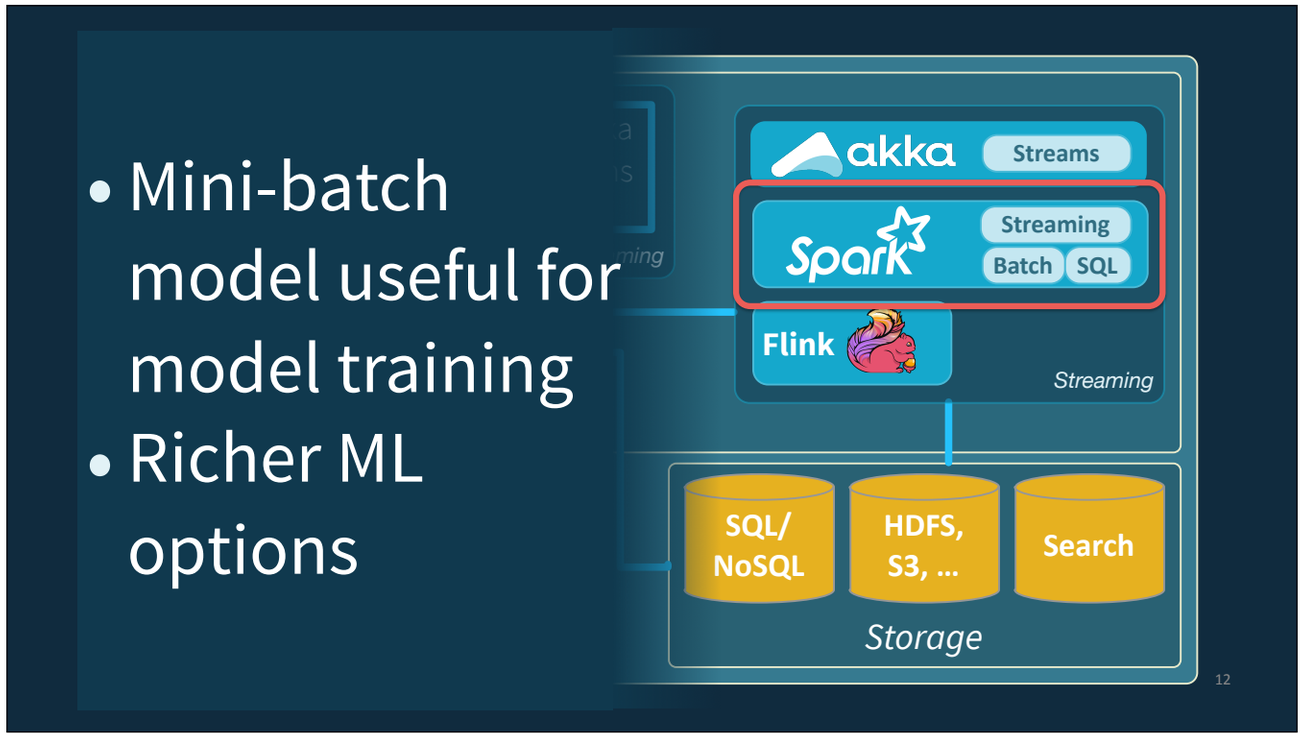


The core: It requires a backplane designed from streaming - Kafka here. To meet the full spectrum of streaming needs, we're bundling 4 stream processing engines, because no single engine meets all needs. Finally, persistence is required.

- Low latency
- High volume
- Apache Beam Data flows (correctness)



Flink contrasts mostly with Spark. It fixes two (current) limitations: 1) low latency instead of medium latency from the current mini-batch streaming model in Spark, 2) Flink provides state of the art semantics for more sophisticated stream processing, especially when high accuracy (as opposed to approximate values) is important. These semantics are defined by Apache Beam (a.k.a., Google Dataflow). Flink can run Beam data flows (The open source Apache Beam requires a third-party runner). Both Spark and Flink provide excellent, scalable performance at high volumes.



Spark has medium latency (~0.5 seconds and up), optimized for excellent, scalable performance at high volumes, but it's not a true streaming engine. We would rather use Flink for low-latency model application (scoring), but training models incrementally through is one option we'll discuss. Also, the Spark ecosystem is ahead of Flink's in terms of integrations with ML tools.

Challenges Using Flink for Deep Learning

Use a Static Model for Inference?

- This is (relatively) straightforward. You load a model at startup and apply it (score data) in the stream.
- See Eron Wright's talk!

First, if you have a static model and you want to use it from Flink ("as a map function" - Eron Wright), this is relatively straightforward.

Train the Model *and* Do Inference?

- May not be reasonable to do training per record with low latency
- But see Swaminathan Sundararaman's talk

Harder problem is training the model on the stream and doing inference

Train the Model and Do Inference

- If too slow, could train mini-batches in windows with longer latency
  - Periodically update the model
  - Do inference with low latency using “old” model

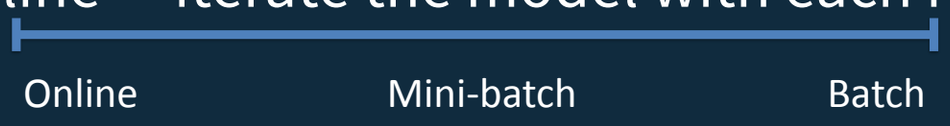
Distributed Training

- You have to train on each data partition separately, then merge the model.
- Actually, DL4J does this, then averages the model parameters.

Model Training vs.
Model Serving (Inference)

Kinds of Training

- Batch - all data at once
- Mini-batch - sample batches (with replacement!), iterate the model with each batch
- “Online” - iterate the model with each record



The “with replacement” means that by design, you keep sampling the same data set, so each record will appear in 0, 1, or more mini-batches.

Mini-batch Training

- Originally developed for offline use, where it's too expensive to train with all the data.
- Would "mini-batches" of stream data work?

That is, mini-batch training was originally invented as a more efficient, if potentially less accurate way to train offline, where each mini-batch samples the same data set repeatedly and the model improves with each iteration (this does work). What if the "mini-batch" is actually a group of N new records instead? By this definition, Online is a mini-batch where N = 1.

Online Training

- Go to Trevor Grant's talk next!

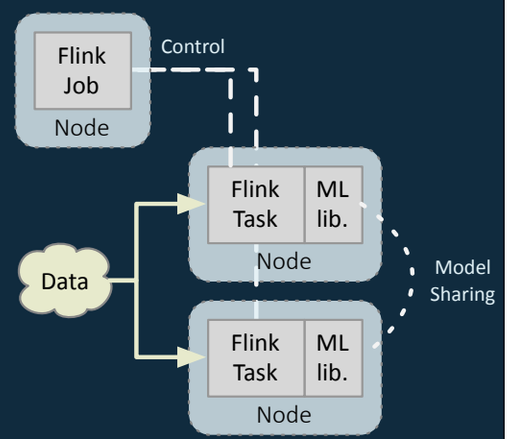
Trevor is going to discuss online learning scenarios in depth.

Practical Approaches with Flink

Design Approaches

1. Same Job - Call Library

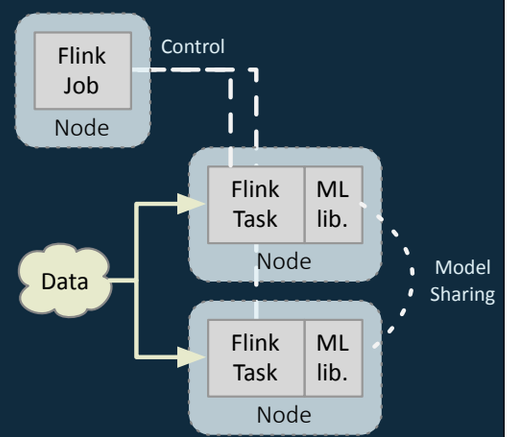
- Training and scoring using the same ML library, in the same job.



The main problem is how to share the model across the tasks running across a cluster. A model/parameter server is one answer.

1. Same Job - Call Library

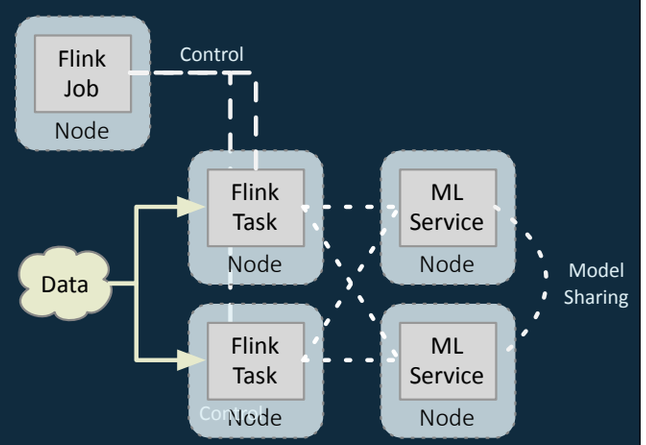
- But what does that mean in a distributed system like Flink?
- Must share model across the cluster & tasks.



The main problem is how to share the model across the tasks running across a cluster. A model/parameter server is one answer.

2. Same Job - Call Service

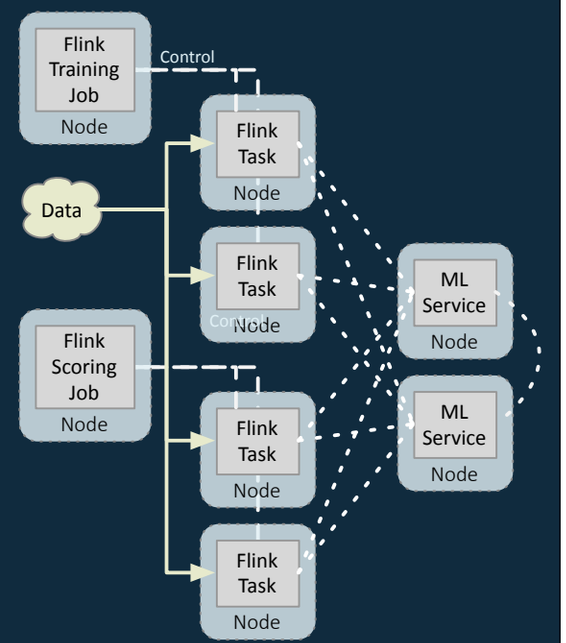
- Training and scoring are done using Async I/O to a separate service, but from the same Flink job.



This service might itself be distributed. It may or may not run on separate nodes from Flink tasks.

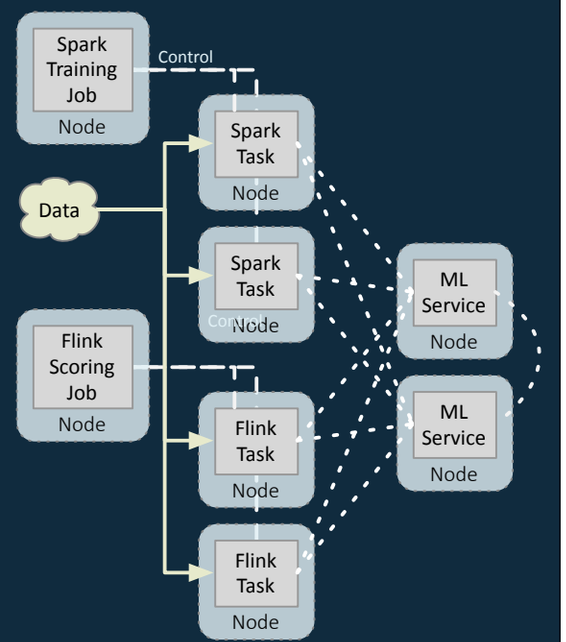
3. Two Flink Jobs

- Training and scoring using separate Flink jobs.
- Could either use a ML library or service.



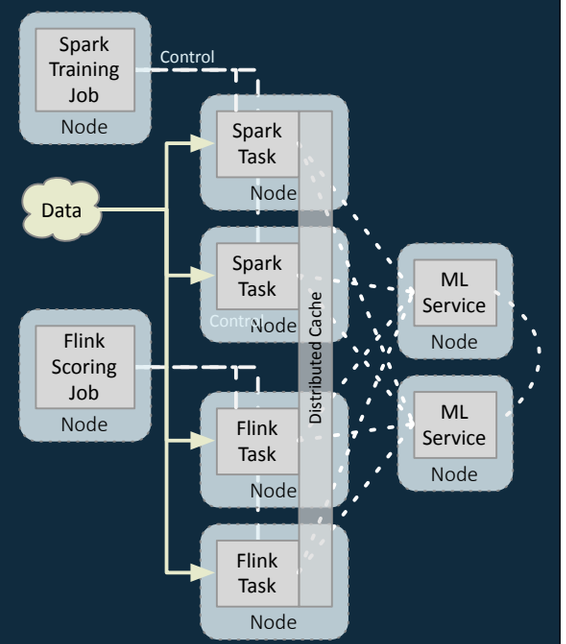
4. Two Different Systems

- Training with Spark Streaming or Batch (for example) and scoring with Flink.

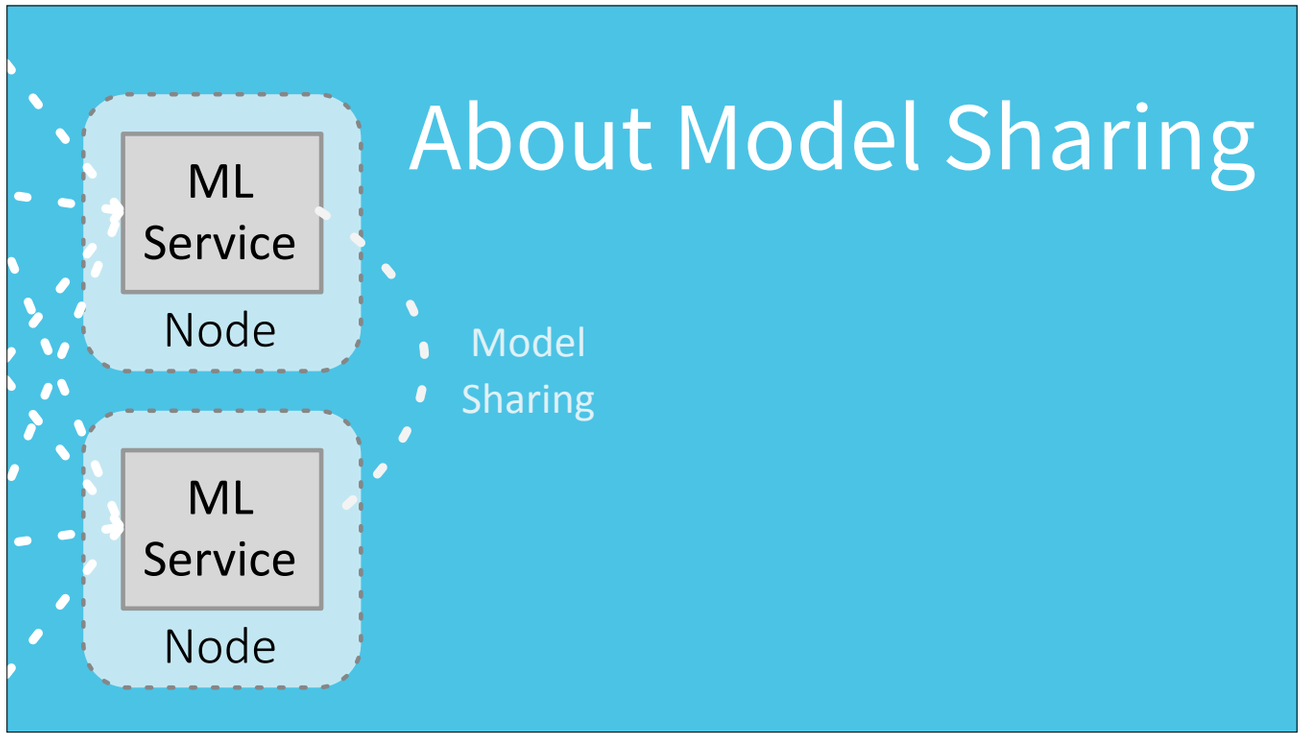


5. Cache is King!

- Use a distributed cache
  - Alluxio (Tachyon)
  - Hazelcast
  - …



About Model Sharing



Parameter Servers

- Scalable, low-latency.
- Serve both training and scoring engines.
  - ND4J parameter server
  - TensorFlow serving system
  - Clipper - prediction serving system
  - PredictionIO - ML server

ND4J provides multi-dimensional arrays for Java, targeting CPUs, GPUs, etc. Mimics Numpy, Matlab, and Scikit-learn.
ND4J parameter server: https://github.com/deeplearning4j/nd4j/tree/master/nd4j-parameter-server-parent
TensorFlow parameter server:
Clipper: https://github.com/ucbrise/clipper

Deeplearning4J

- JVM-based.
- Discussed at last year's Flink Forward.
- Use Flink Dataset to load data into DL4J's DataVec format, then feed to DL4J service.

https://www.slideshare.net/FlinkForward/suneel-marthi-deep-learning-with-apache-flink-and-dl4j

Leveraging Flink Features

Side Inputs

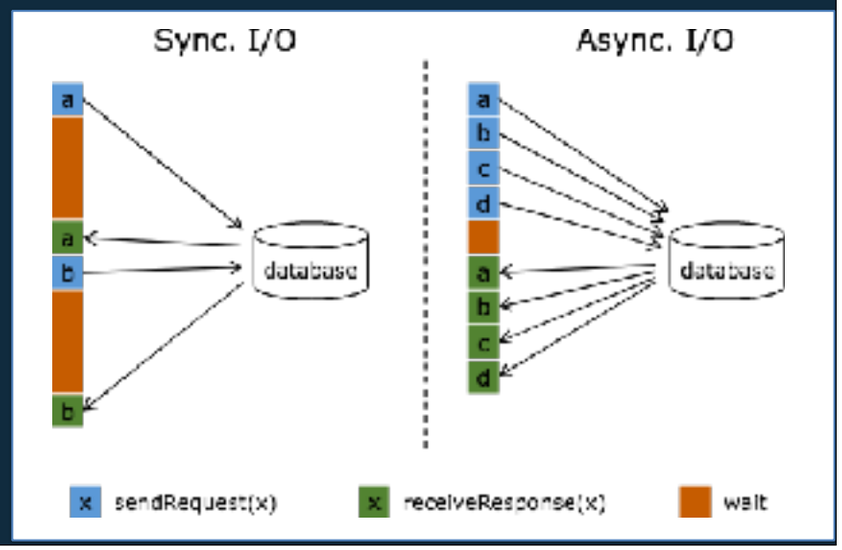
- Use case: Join a stream with slowly changing data.
  - So, for ML, what do you join with what?
  - Keyed or broadcast?
  - Windowed or not (“global window”)

https://cwiki.apache.org/confluence/display/FLINK/FLIP-17+Side+Inputs+for+DataStream+API

```
// Side Input sketch
// Example adapted from https://cwiki.apache.org/confluence/
display/FLINK/FLIP-17+Side+Inputs+for+DataStream+API

val dataStream: DataStream[Record] = …
val modelStream: DataStream[Params] = …

val modelInput = new SingletonSideInput(modelStream)

dataStream
  .map { record =>
    val params = getRuntimeContext().getSideInput(modelInput)
    val score  = model.withParams(params).score(records)
    (score, record)
  }
  .withSideInput(modelInput);
```

Async I/O

- Callback response handler



Async I/O

- Might be better for periodically retrieving model parameter updates.
- Calling to a model server for scoring could be too inefficient?

```
// Async I/O sketch
// Example adapted from https://ci.apache.org/projects/flink/
flink-docs-release-1.2/dev/stream/asyncio.html

/**
 * Params is a type that encapsulates the notion of a
 * sequence of model parameters, each of which knows its
 * location in the model, etc.
 */
case class Params(...) extends Iterable[Param] {...}

/**
 * Handle asynchronous requests for the entire set of model
 * parameters, where a String key is used to specify the
 * model.
 */
```

```
/**
 * Handle asynchronous requests for the entire set of model
 * parameters, where a String key is used to specify the
 * model.
 */
class AsyncMLModelParameterRequest
  extends AsyncFunction[String, Params] {

 /**
  * An ML Model-specific client that can issue concurrent
  * requests with callbacks. Would be customized for
  * DL4J, TensorFlow, etc.
  */
 lazy val client = MLModelClient(host, port, ...)

 /** The Flink context used for the future callbacks. */
 implicit lazy val executor =
  ExecutionContext.fromExecutor(Executors.directExecutor())
```

```
  override def asyncInvoke(
   key: String,
   asyncCollector: AsyncCollector[Params]): Unit = {

   // Issue the asynchronous request, receive a
   // future for the result
   val paramsFuture: Future[Params] = client.getParams(key)

   // Set the callback to be executed once the request by the
   // client is complete. The callback simply forwards the
   // params to the collector, exploiting the fact that Params
   // implements Iterable.
   paramsFuture.onSuccess {
    case params: Params => asyncCollector.collect(params);
   }
  }
 }
```

The Future of
Deep Learning with Flink

Current Flink ML Roadmap

- Today: Offline batch learning
- New: Google Doc investigating:
  - Offline training with Flink Streaming
  - Online training
  - Low-latency model serving

Flink ML: Do

- Focus on 3rd-party integrations:
  - DL4J (like the Spark integration)
  - TensorFlow
  - H2O
  - Mahout
  - …

Flink ML: Do

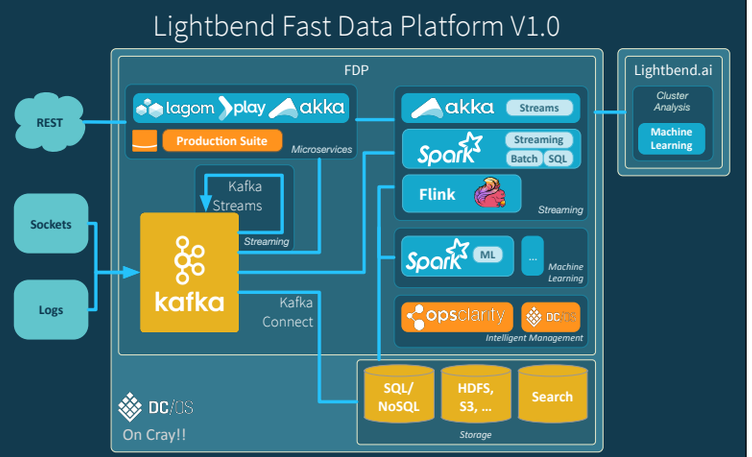
- Complete useful core capabilities, e.g.,
  - FLINK-5782 (GPU support)
  - FLINK-6131 (Side inputs)
- Remove anything in today's ML library that isn't "solid".

Flink ML: Don’t

- PMML - doesn’t work well enough. Not really that useful?
- Samoa - appears dead
- Reinvent the wheel…

Thank You!

Dean Wampler, Ph.D.
dean@lightbend.com
@deanwampler



lightbend.com/fast-data-platform

For more information on the Fast Data Platform.