Power Hive with Spark

Marcelo Vanzin and Chao Sun

Cloudera, Inc.

What we'll talk about...

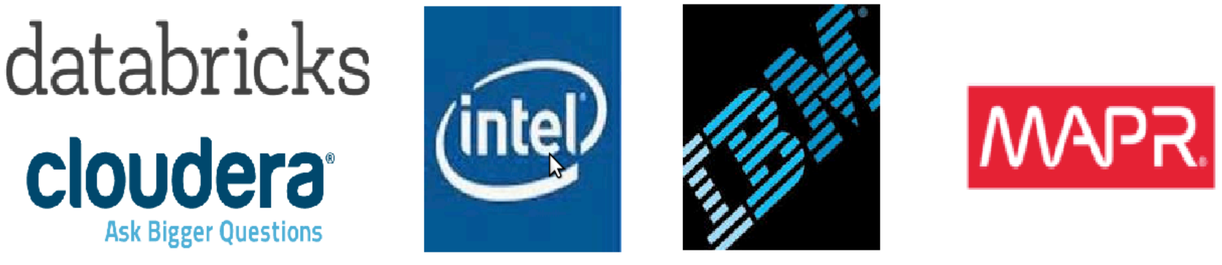
- Hive-on-Spark primer
- Spark additions for HoS
  - Dynamic Executor Allocation
  - Remote Spark Context
- What remains to be done

Hive-on-Spark Primer

- Hive is the “standard” engine for SQL in Hadoop
- HoS is a Spark-based execution engine for Hive
- Goal is twofold:
  - Run Hive on a modern distributed execution engine
  - Drive enhancements (performance and usability) for both
- Distinct from Spark SQL / HiveContext

HoS Primer (continued)

- Many organizations involved, plus individual contributors from both communities



- Part of Apache Hive 1.1
- Still a young project, lots left to be done

Spark Additions

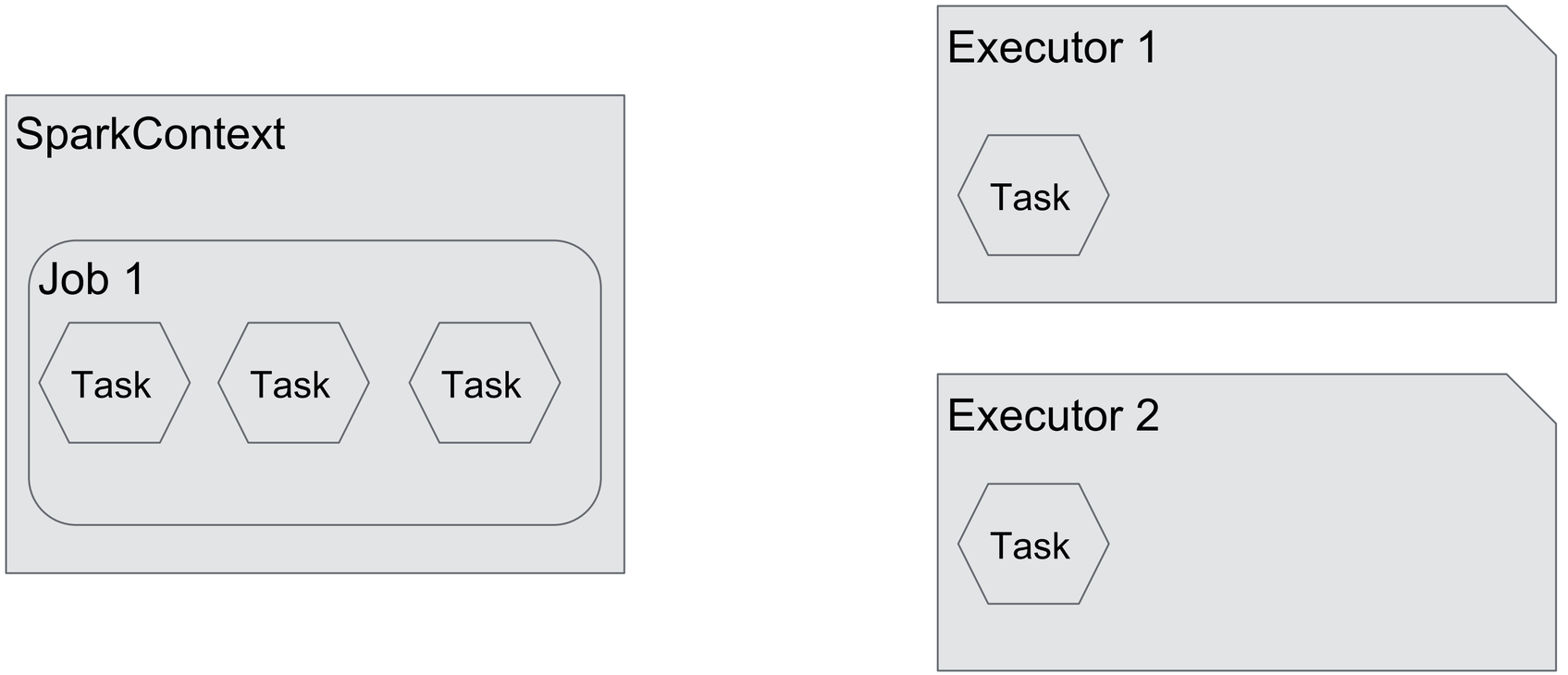
- Enhancements to Spark
  - repartitionAndSortWithinPartitions()
  - Recursive addFile()
  - Dynamic Executor Allocation

- Libraries built on top of Spark
  - Remote Spark Context

Dynamic Executor Allocation

- Problems:
  - Hard to know beforehand how many executors / cores will be needed to run a query
  - Interactive Spark sessions (e.g. Spark shell or Hive session) hold on to cluster resources while idle
- Solution: grow and shrink the Spark application as needed
- Currently available for YARN backend

DEA (continued)



DEA: Future Work

- Goal: default allocation mode for Spark on YARN
- Better heuristics (faster ramp up for large jobs)
- Data locality hints
- Support Spark's RDD caching mechanism

Remote Spark Context

- Problem: HiveServer2 services several users. Lots of issues ensue.
  - SparkContext uses non-trivial amount of memory
  - Spark doesn’t support multiple SparkContexts
  - `new SparkContext()` doesn’t support cluster mode
  - Need to isolate user’s sessions
  - Need to account for user’s resource usage

- Solution: allow HS2 to start Spark apps out-of-process and interact with them.

RSC (continued)

- Long-lived SparkContext tied to user's HS2 session
- On YARN, run with the user's credentials
- Remoting API allows fine-grained control of remote SparkContext instance

```
SparkClient client = SparkClientFactory.createClient(...);
try {
  JobHandle<Integer> handle = client.submit(rjc ->
    rjc.sc().parallelize(Arrays.asList(1,2,3))
      .reduce((i1, i2) -> i1 + i2));
  System.out.println("Result is: " + handle.get());
} finally {
  client.close();
}
```

HoS: What's Left

- Functionally complete
- Needs more testing and benchmarking
- Performance optimizations
  - Dynamic Partition Pruning
  - Table Caching
  - Multiple Outputs from same RDD
- HIVE-7292 tracks current and future work

Questions?