Other Big Data Tools

CMPT 732, Fall 2018

A Look Back

We have had a decent look at HDFS, MapReduce, Cassandra, and Spark tools.

That's a drop in the bucket. See:

- Big-Data Ecosystem Table.
- Apache "big data" projects.
- Amazon Web Services in Plain English.

What Else Is There?

We can't hope to really see a significant fraction of these in a semester (or in a degree, or even in a career). What follows is a summary of some more.

Proposed take-away messages:

- There are tools for big data outside the Hadoop universe.
- The big data world is evolving extremely fast.
- There are many things out there, and you should know them enough to think "I'm pretty sure I heard about a tool for that already…".

The Plan…

Various technologies, organized by category, with some explanation of what they are for. Trying to balance a reasonable number of technologies with completeness.

Part 1: We Have Seen An Example

Doing Computation

The goal: get the compute work to some computer with processor/memory to do it, and get the results back.

- Apache YARN: Hadoop's resource manager.
- Mesos: resource manager more closely aligned with Spark.

- Amazon EC2: get VMs to do work with.
- Amazon EMR: EC2 + Hadoop set up automatically.
- Google Compute Engine: get VMs from Google.
- Amazon Lambda: give Amazon functions; it runs them when you call.
- Google AppEngine: give Google functions; it runs them when you call.

This could also extend to tools like Kubernetes or AWS Elastic Container Service that can get compute work running and scale up/down.

Expressing Computation

The goal: Describe your computation in a way that can be run on a cluster.

- MapReduce.
- Spark.
- Flink: competitor to Spark. Scala/Java. "Streaming first".
- Hive: take varied data and do SQL-like queries.
- Pig: high-level language to analyze data. Produces MapReduce jobs.
- Programming. Distributed systems.

Data Warehousing

The goal: Take lots of data from many sources, and do reporting/analysis on it.

- Spark DataFrames.
- Hive: take varied data and do SQL-like queries.
- Apache Impala: massively-parallel SQL queries on Hadoop, against varied inputs.
- Apache Drill: take data from many sources (SQL, NoSQL, files, …) and query it.
- Google BigQuery: Google's data warehouse tool.
- Amazon RedShift: Amazon's data warehouse tool.

Storing Files

The goal: Store files or file-like things in a distributed way.

- HDFS.
- Amazon S3: Amazon's file storage.
- Gluster: distributed network filesystem.

- Alluxio: in-memory distributed storage system.
- Ceph: distributed filesystems and object store.
- Files. Filesystems. Disks. NAS.

Databases

The goal: store records and access/update them quickly. I don't need SQL/relations.

- Cassandra: Good clustering. Secondary keys, but no joins.
- HBase: Good clustering, fast. Otherwise very manual.
- MongoDB. Clustered, but questionable reliability. Suggest not using for primary data storage. **
- Amazon SimpleDB and DynamoDB.

The goal: store records and access/update them quickly. I want SQL and/or relations.

- Amazon Aurora: Amazon's scalable relational database.
- Other NewSQL databases.
- PostgreSQL, MySQL, etc.

Serialization/Storage

The goal: read/write data efficiently for memory/disk/network.

- Parquet: efficient columnar storage representation. Supported by Spark, Pandas, Impala.
- HDF5: on-disk storage for columnar data.
- CSV, JSON: well-understood interchange formats.
- Arrow: in-memory representation for fast processing. Available in Spark 2.3+.

Streaming

The goal: deal with a continuously-arriving stream of data.

- Spark Streaming (DStreams for RDDs, Structured Streaming for DataFrames).
- Apache Storm.
- Apache Flume.
- Amazon Kinesis.

ML Libraries

The goal: use machine learning algorithms (at scale) without having to implement them.

- Spark MLlib.
- Apache Mahout.
- Amazon Machine Learning.

Or at smaller scale, scikit-learn, PyTorch, etc.

Part 2: New (to us) Categories

Visualization

The goal: take the data you worked so hard to get, and let people understand it and interact with it.

- Tableau.
- Qlik.
- Power BI.
- Programming and plotting/drawing libraries.

Extract-Transform-Load

The goal: Extract data from the source(s); transform it into the format we want; load it into the database/data warehouse.

- Apache Sqoop.
- Amazon Data Pipeline.
- NiFi
- MapReduce, Spark, programming.

Message Queues

The goal: pass messages between nodes/processes and have somebody else worry about reliability, queues, who will send/receive, etc.

- Apache Kafka.
- RabbitMQ.
- ZeroMQ/ØMQ.
- Amazon SQS.

All designed to scale out and handle high volume.

The idea:

- Some nodes *publish* messages into a *queue*.
- The message queue makes sure that they are queued until they can be processed; ensures each message is processed once (depending on the tool).
- Some nodes *subscribe* to the queue(s) and *consume* messages.

Or other interactions with the queues. Freely switch languages between publisher/consumer too.

These things are *fast*: RabbitMQ Hits One Million Messages Per Second.

Realistic streaming scenario: Spark streaming takes in the data stream, filters/processes minimally, and puts each record into a queue for more processing. Then many nodes subscribe to the queue and handle the data out of it.

Or without Hadoop, just generate a bunch of work that needs to be done, queue it all, then start consumer processes on each computer you have.

Either way: you can move around the bottleneck (and hopefully then fix it).

Message passing example with RabbitMQ:

- rabbit-receiver.py
- rabbit-source.py
- rabbit-source.rb

Let's try it...

```
window1> python3 rabbit-receiver.py
window2> python3 rabbit-receiver.py
window3> python3 rabbit-source.py
window4> ruby rabbit-source.rb
# kill/restart some and see what happens
```

Message passing example with Kafka:

- kafka-producer.py
- kafka-consumer.py

Let's try it...

```
window1> python3 kafka-consumer.py
window2> python3 kafka-consumer.py
window3> python3 kafka-producer.py
```

Task Queues

The goal: get some work on a distributed queue. Maybe wait for results, or maybe don't.

- Celery (Python).
- Resque, Sidekiq (Ruby).
- Google AppEngine Task Queues (Python, Java, Go, PHP).
- Amazon Simple Workflow Service.
- Any message queue + some code.

With a task queue, you get to just call a function (maybe with slightly different syntax). You can then retrieve the result (or just move on an let the work happen later).

Where the work happened is transparent.

A task with Celery: tasks.py.

Let's try it...

```
window1> celery -A tasks worker --loglevel=info --hostname=worker1@%h
window2> celery -A tasks worker --loglevel=info --hostname=worker2@%h
window3> ipython3
```

```python
from tasks import add
result = add.delay(4, 4)
result.get(timeout=1)
```

Need a lot of work done without Hadoop? Run task queue workers on many nodes; make all the asynchronous calls you want; let the workers handle it.

Need nightly batch analysis done? Have a scheduled task start a Spark task.

Have a spike in usage? Let tasks queue up and process as possible. Or add more workers.

Text Search

The goal: index lots of data so you (or your users) can search for records they want.

- Apache Solr/Apache Lucene.

- Elasticsearch.
- Amazon CloudSearch.

All of these are designed to scale out across many nodes.

Indexing and searching with Elasticsearch:

- elastic-index.py
- elastic-search.py

Let's try it… (See also CourSys search when an instructor.)

```
python3 elastic-index.py
python3 elastic-search.py
curl -XGET 'http://localhost:9200/comments/_search?q=comment' \
  | python3 -m json.tool
```

Hadoop Distributions

The goal: Get a Hadoop cluster running without becoming an expert on Hadoop configuration.

- Cloudera: what is running our cluster.
- Hortonworks HDP.
- MapR.
- Amazon EMR: EC2 + Hadoop set up automatically.

Learning More

Places to learn more about more:

- Spark Summit.
- Strata + Hadoop World.
- Meetups on Big Data, Big Data Analytics, Data Analytics, Data Science.

