Data

Alex Sverdlov
alex@theparticle.com

1 Introduction

AI models are trained on data. Data often represents logs of what happened.

Often such logs are structured in some way (have a defined layout, such as timestamp, event type, etc.).

When there is no structure (e.g. unstructured text), then part of building a model often involves learning/defining a structure (e.g. turning each word into a feature vector of numbers).

1.1 Events vs State

Data often comes in two varieties: records of events (what happened), and records of state (what is). Events are the things that alter the state.

Building models from events has often been viewed as more difficult than building it on the state. For example, a state often compresses multiple sequence of events into a single fixed-width state feature vector.

1.2 Volume and Sharding

More data often leads to more accurate AI models. When datasets get too big for a single machine, the data is often sharded: split up into non-overlapping chunks. Each chunk can often be processed independently of other chunks, and often in parallel.

A logical file turns into a directory, where each file may be written/read by a worker thread, independently of other worker threads.

Many distributed file systems implement this splitting at the datablock level, where a single logical file is automatically split into datablocks for consumption by multiple workers in parallel.

1.3 HDFS and Hadoop

TODO: brief outline

1.4 Data vs Compute

Cloud computing has introduced another idea: separation of data and compute. These are two separate services provided by cloud vendors. Data at rest only requires payment for storage, often per gigabyte-per-month.

Processing data requires renting computers to read/write data, and is paid for by compute-hours. Users can pick and choose between fast-and-expensive, and slow-and-cheaper compute.

Fast interconnects, and independent evolution of networking and storage technology (networking tech advanced quite a bit, while storage tech mostly lagged behind), means that reading a remote file (e.g. from S3) is very comprable (speed wise) as reading a local disk.

1.5 SQL

While large datasets of interest to this class are not "stored in a database", the separation of storage and compute meant the emergence of SQL engines that operate on remote data.

Engines such as Trino, pulling data from S3 is a popular way of getting at the data, and applying SQL-based transformations/analytics.

2 Spark

Often straight SQL is too restrictive, and we need to write procedural code to process data (often to re-create state from event-level-data).

Spark is a distributed compute engine that can operate on data stored anywhere. The system is written in scala, with availability of the standard Java library. There are bindings for Python (PySpark).

The central abstraction in spark is the RDD, which is a collection of data partitions, potentially stored on different machines. Operations on RDDs often run in parallel on each partition.