iFrame: An integrated data management layer for big data systems

Abstract

Big data systems are increasingly processing structured and semi-structured data. Currently, data processing and data management are in separate systems. For example, the processing engine such as Apache Spark reads in the data from databases such as Cassandra and HBase. This separation creates many problems. First, processing speed slows down because the data processing engine needs data locality for efficiently distribute tasks to nodes. However, the processing engine has no knowledge of where data stored in databases. Second, resources are not used efficiently. Modern databases aggressively caches data to optimize query and maintains indices. Data processing engine has to copy the data into its own subsystem. Resources used by databases and data processing engines are not shared. Third, the data read into data processing engine is subject to change. This complicates failure recovery and consistency.

To resolve these problems, we present iFrame, an integrated in-memory data management layer for big data systems. With iFrame, data does not have to be copied into data processing layer. Resources used by iFrame are shared and allocated by the same resource management system such as Mesos or Yarn. Since the data is managed and maintained by iFrame alone, this simplifies failure recovery and consistency. iFrame adopts a lineage-based mechanisms for fault tolerance.

1 introduction

Traditionally big data systems such as Hadoop, Spark performs batch processing. Analytics are generated in an offline fashion, e.g. on a daily basis. Data accumulated in a longer time period such as daily are analyzed. However, this is slow in incorporating analytics results into online service systems. Many applications require online learning and results affecting services in near real time. See Velox paper.

With the new requirements, there is a need to integrate data management with data processing so that data can be processed timely, efficiently and consistently.

2 Background

Similar to Tachyon, we target workloads with the following properties:

- Immutable data: Data is immutable once written. However, new data can keep arriving.
- Deterministic jobs: Many frameworks, such as MapReduce and Spark, use recomputation for fault tolerance within a job and require user code to be deterministic. We provide lineage-based recovery. Unlike Tachyon, our lineage tracks fine-grained database operations.
- Locality based scheduling: Many computing frameworks schedule jobs based on locality to minimize network transfers, so reads can be data-local.
- All data vs. working set: Even though the whole data set is large and has to be stored on disks, the working set of many applications fits in memory.
- Program size vs. data size: In big data processing, the same operation is repeatedly applied on massive data. Therefore, replicating programs is much less expensive than replicating data in many cases.

The key difference is that iFrame targets structured and semi-structured data.