《Shark SQL and Rich Analytics at Scale》

5 Implementation部分

**Memory-based Shuffle:** Both Spark and Hadoop write map output files to disk, hoping that they will remain in the OS buffer cache when reduce tasks fetch them. In practice, we have found that the extra system calls and file system journaling adds significant overhead. In addition, the inability to control when buffer caches are flushed leads to variability in shuffle tasks. A query’s response time is determined by the last task to finish, and thus the increasing variability leads to long-tail latency, which significantly hurts shuffle performance. We modified the shuffle phase to materialize map outputs in memory, with the option to spill them to disk.

思考：总运行时间取决于最后一个task的完成。尽可能的减少不必要的shuffle操作，能提升性能

**Temporary Object Creation:** It is easy to write a program that creates many temporary objects, which can burden the JVM’s garbage collector. For a parallel job, a slow GC at one task may slow the entire job. Shark operators and RDD transformations are written in a way that minimizes temporary object creations.

思考：因为scala是依赖于java的JVM的，那其垃圾回收会依赖于GC。要减少不必要的中间变量，spark的transformation步骤尽量多“窄依赖”少“宽依赖”。

**Bytecode Compilation of Expression Evaluators:** In its current implementation, Shark sends the expression evaluators generated by the Hive parser as part of the tasks to be executed on each row. By profiling Shark, we discovered that for certain queries, when data is served out of the memory store the majority of the CPU cycles are wasted in interpreting these evaluators. We are working on a compiler to transform these expression evaluators into JVM bytecode, which can further increase the execution engine’s throughput.

思考：这是否是序列化的好处？ 由“**Memory-based Shuffle”**提到，spark也会将map阶段的中间结果存入磁盘的，如果中间结果却是够多只能存入磁盘，那么是否已序列化的方式进行，从而提高性能？

《Resilient Distributed Datasets A Fault-Tolerant Abstraction for In-Memory Cluster Computing》

Abstract

We present Resilient Distributed Datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. RDDs are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on coarse-grained transformations rather than fine-grained updates to shared state. However, we show that RDDs are expressive enough to capture a wide class of computations, including recent specialized programming models for iterative jobs, such as Pregel, and new applications that these models do not capture.We have implemented RDDs in a system called Spark, which we evaluate through a variety of user applications and benchmarks.

思考：从整体介绍了spark

1 Introduction

思考：介绍了其他分布式系统的不足——对分布式内存的利用，对中间结果的利用

《http://jerryshao.me/architecture/2014/01/04/spark-shuffle-detail-investigation/》

Shuffle是必须的，但是sort就不一定要做。

《http://dongxicheng.org/framework-on-yarn/apache-spark-shuffle-details/》

里面提到了shuffle的改进