Spark

1 论文介绍

主要阅读了三篇论文：Spark: Cluster Computing with Working Sets、Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing 和 An Architecture for Fast and General Data Processing on Large Clusters。前两篇论文的第一作者 Matei Zaharia 是 Spark 项目的创建者，也是其核心开发者。第三篇论文是 Matei Zaharia 的博士毕业论文。

这三篇论文的主要内容是：内存计算框架 Spark 中的核心概念：RDD(Resilient Distributed Datasets)-分布式内存的一种抽象，Spark 的设计与实现，构建在 Spark 之上的计算模型，Spark 适合的应用场景。

2 现有系统中存在的问题

集群计算需要解决的两个主要问题是并行和容错。为了方便程序员编写运行在集群上的并行应用，同时也为了有效地利用集群中的资源，众多的编程模型被设计出来。Google 提出的 MapReduce 是一种简单而通用的批处理模型，在这种模型中，计算过程被表示成一个有向无环图（Directed Acyclic Graph，DAG），节点表示计算任务。然而 MapReduce 在处理其他类型的计算问题时表现很差，主要包括迭代式、交互式和流式数据处理。为了解决这一问题，许多专用的计算模型被设计出来。例如，Google 开发的 Pregel 针对迭代式图算法提出了块同步并行（bulk-synchronous parallel，BSP）模型；F1 能够快速执行 SQL 查询，但没有容错机制；MillWheel 支持连续的流处理。

开发专用系统来解决特定领域的问题是一种很自然的想法，但这种方式存在一些弊端：

- 重复劳动：许多专用系统需要解决相同的底层问题，比如任务分配等。
- 组合：不同系统之间的协同很有必要但很难做到，主要问题是数据共享。
- 资源共享：因为大多数计算引擎假定自己在运行过程中独立地拥有同一组机器，所以很难做到共享集群资源。
- 管理：每个系统需要独立地部署和管理，管理员的工作量很大；用户需要学习多组 API 和执行模型，工作量也很大。

所以为集群计算提出一个统一的抽象可以获得很大的收益：既可以提高易用性，又可以提高性能，特别是对于复杂的应用和多用户环境来说。

3 Spark 解决上述问题的方法

Spark 扩展了 MapReduce，采用了 RDD-分布式内存的一种抽象，为集群环境提供了一个统一的编程抽象。

在 Spark 中，数据和中间结果以 RDD 的形式存储在内存中，便于数据重用，极大地提高了计算过程之间共享数据的效率。

可以在 Spark 之上构建已有的计算模型，Spark 为批处理（Spark Core），交互式数据分析（Shark），流式数据处理（Spark Streaming），机器学习（MLLib），图计算（GraphX）提供了一个统一的数据处理平台，使得不同计算模型之间的协作变得更加容易和高效，这相对于 Hadoop 具有很大优势。

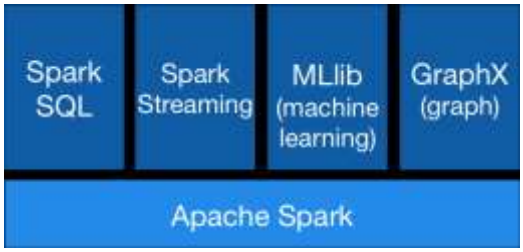

图 1 Spark 和构建在 Spark 上的开源工具

4 RDD

RDD 是分布式内存的一种抽象，它是一个只读的带分区的记录集合。可以在 RDD 之上执行两种类型的操作：转换和动作。转换生成一个新的 RDD，比如 map、filter、join 等；动作返回一个值或者将数据导出到文件系统中，比如 count、collect、save 等。

RDD 中保存着它自己是如何从其它的 RDD 通过哪些转换操作得来的所有信息，可以有效地重建丢失的分区，这是一种粗粒度的容错手段。

RDD 可以有效地表达许多集群计算模型：MapReduce、DryadLINQ、SQL、Pregel、Iterative MapReduce(HaLoop)、Batched Stream Processing，这其中很多是独立的计算框架。这个特性允许用户在同一个程序中组合应用多种计算模型（比如先用 MapReduce 来构造一个图，然后在这个图上运行 Pregel）和共享数据。

5 Spark 的运行环境

Spark 运行在 Mesos 之上，可以看做 Mesos 上的一个应用。Mesos 是一个“集群操作系统”，它允许多个并行应用共享集群资源，并给应用提供了生成任务的 API。Spark 可以和其它的计算框架（如 Hadoop、MPI 等）运行在同一个集群上，共享集群资源，应用之间还可以共享数据。

Spark 使用 Scala 语言实现，可以通过一个修改过的 Scala 解释器来交互式地使用 Spark，它允许用户定义 RDD、函数、变量和类，然后交互式地操作这些元素。Spark 可以从 Hadoop 的数据源（如 HDFS 或 HBase）中读取数据。

6 Spark 的编程接口

Spark 为并行编程提供了两个主要的抽象：RDD 和作用在 RDD 上的并行操作。

Spark 中的每个 RDD 表示为一个 Scala 对象，有四个途径来构造 RDD：文件系统（如 HDFS 等）中的文件；并行化一个 Scala 集合，将集合分成许多片段后发送到多个节点上；在一个现有的 RDD 上执行转换操作；改变一个现有 RDD 的持久化状态。

在 RDD 上执行动作，返回一个值或者将数据导出到文件系统中。如 count 返回数据集中的条目数，collect 返回条目自身；save 将数据导出到文件系统中。另外，程序员可以通过调用 persist 方法来表明这些数据在将来的操作中会被重用，于是这些数据会驻留在内存中。

使用 Spark 时，开发者需要编写应用的高层控制流程，这段程序实际上构造了一个有向无环图（DAG），这个有向无环图由多个相互依赖的 RDD 构成。然后通过在 RDD 上执行动作将这个有向无环图作为一个 Job 提交给 Spark。

Spark 对有向无环图 Job 进行调度，确定阶段（Stage），分区（Partition），流水线（Pipeline），任务（Task）和缓存（Cache），优化后在 Spark 集群上运行。

7 举个例子-日志文件分析

假设一个 Web 服务出了问题，现在需要从 HDFS 中 TB 级的日志文件中找出错误的原因。使用 Spark 来完成这个任务时，只需要将错误信息加载到一组节点的内存中，然后进行交互式的查询。前面曾经提到过可以通过一个修改过的 Scala 解释器来交互式地使用 Spark。Scala 命令（代码）如下：

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
errors.persist()
```

第一行使用 HDFS 中的文件定义了一个 RDD，第二行使用 filter 操作从第一个 RDD 中过滤出包含出错信息的行并生成另一个 RDD，第三行将包含错误信息的 RDD 缓存在内存中，以便之后进行交互式地查询。进一步的查询分析如下：

```
errors.filter(_.contains("HDFS"))
.map(_.split('\t')(3))
.collect()
```

过滤出错误信息中包含“HDFS”的行并取出这些行中的时间域（假设是第三列，日志文件不同的域采用tab分隔）。第三次查询时的linage图如图2所示。

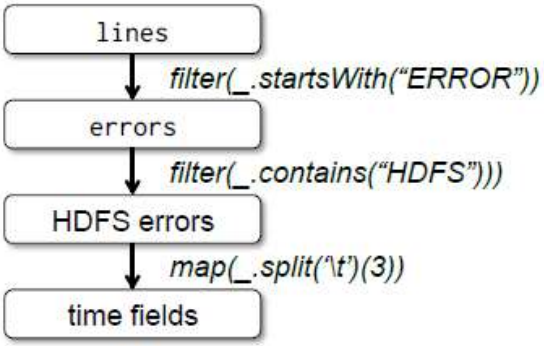

图2 第三次查询时的linage图

8 Spark 适合的应用场景

Spark 特别适合那些在多次操作之间需要重用数据集的应用：迭代式算法和交互式分析。迭代式算法：许多常见的机器学习算法在同一个数据集上多次执行相同的操作来优化一个参数。交互式分析：重复查询感兴趣的数据集。

9 总结

Hadoop 解决了海量数据的可靠存储和处理。包括 HDFS 和 MapReduce 两个核心组件。

HDFS 为海量的数据提供了存储。HDFS 可以在普通 PC 机组成的集群上提供高可靠的文件存储，通过保存块的多个副本来解决服务器或硬盘坏掉的问题。

MapReduce 为海量的数据提供了计算。MapReduce 通过简单的 Mapper 和 Reducer 抽象提供一个编程模型，可以在一个几十上百台 PC 组成的不可靠集群上并发地、分布式地处理大量的数据集，而把并发、分布式（如机器间通信）和容错等计算细节隐藏起来。各种复杂的数据处理任务都可以分解成多个 Job（包含一个 Mapper 和一个 Reducer）组成的有向无环图（DAG），然后将每个 Mapper 和 Reducer 放到 Hadoop 集群上执行，就可以得到结果。

Hadoop 中存在的问题：

- 抽象层次低，只提供两个操作，Map 和 Reduce，表达能力欠缺；
- 一个 Job 只有 Map 和 Reduce 两个阶段，复杂的计算需要大量的 Job 完成，Job 之间的依赖关系由开发者自己管理，处理逻辑隐藏在代码细节中，没有整体逻辑；
- Job 之间共享数据需要通过 HDFS，延时很大；
- 适用于大规模数据的批处理，对于交互式数据处理、实时数据处理和迭代式数据处理的支持不够。

Spark 相对于 Hadoop 最大的改进在于采用了 RDD 这种分布式内存抽象，可以充分利用集群中的内存资源。主要表现在：计算的中间结果不需要写回文件系统，有效地减少了 I/O 操作上的时间消耗；可以将数据集缓存在内存中，便于多次操作之间数据集的重用，特别适合迭代式算法和交互式分析。

其次，Spark 提供了 linage 这种粗粒度的容错手段。RDD 中保存着它自己是如何从其它的 RDD 通过哪些转换操作得来的所有信息，可以有效地重建丢失的分区。

另外，借助于 RDD，Spark 为许多集群计算模型提供了一个统一的数据处理平台。RDD 可以有效地表达 MapReduce、DryadLINQ、SQL、Pregel、HaLoop 等计算模型。

Hadoop 和 Spark 之间的比较

Hadoop	Spark
中间结果存放在 HDFS 中	中间结果存放在内存中
时延高，适合于海量数据的批处理，对于实时数据处理的支持不够	通过将流拆成小的 batch 提供 Discretized Stream 处理流式数据
对于迭代式算法和交互式分析支持不够	通过在内存中缓存数据集，极大地提高了迭代式算法和交互式分析的性能
处理逻辑隐藏在代码中，没有整体逻辑	提供处理逻辑的整体视图，代码中不包含具体操作的实现细节，逻辑更清晰
抽象层次低，需要手工编写许多代码	基于 RDD 的抽象，实现数据处理逻辑的代码比较简短
只提供两个操作：Map 和 Reduce，表达能力欠缺	提供很多转换和动作，如 map、filter、join、count、collect、save 等
一个 Job 只有 Map 和 Reduce 两个阶段（Phase），复杂的计算需要大量的 Job 完成，Job 之间的依赖关系由开发者自己管理	一个 Job 包含 RDD 上的多个操作，调度时生成多个阶段（Stage），如果多个操作中 RDD 的分区不变，可以放在同一个 Task 中运行
ReduceTask 需要等待所有的 MapTask 都完成后才可以开始	分区相同的转换构成流水线放在一个 Task 中运行，分区不同的转换需要 Shuffle，被划分到不同的 Stage 中，需要等待前面的 Stage 完成后才可以开始