基于 Spark 的大数据混合计算模型①

胡 俊[1], 胡贤德[1], 程家兴[1,2]

[1](安徽新华学院 信息工程学院, 合肥 230088)
[2](安徽大学 计算机科学与技术学院, 合肥 230031)

摘 要: 现实世界大数据应用复杂多样, 可能会同时包含不同特征的数据和计算, 在这种情况下单一的计算模式多半难以满足整个应用的需求, 因此需要考虑不同计算模式的混搭使用. 混合计算模式之集大成者当属 UCBerkeley AMPLab 的 Spark 系统, 其涵盖了几乎所有典型的大数据计算模式, 包括迭代计算、批处理计算、内存计算、流式计算(Spark Streaming)、数据查询分析计算(Shark)、以及图计算(GraphX). Spark 提供了一个强大的内存计算引擎, 实现了优异的计算性能, 同时还保持与 Hadoop 平台的兼容性. 因此, 随着系统的不断稳定和成熟, Spark 有望成为与 Hadoop 共存的新一代大数据处理系统和平台. 本文详细研究和分析了 Spark 生态系统, 建立了基于 Spark 平台的混合计算模型架构, 并说明通过 spark 生态系统可以有效地满足大数据混合计算模式的应用.

关键词: 大数据; 混合计算模式; spark; 弹性分布数据集

Big Data Hybrid Computing Mode Based on Spark

HU Jun[1], HU Xian-De[1], CHEN Jia-Xing[1,2]

[1](School of Information Engineering, Anhui Xinhua University, Hefei 230088, China)
[2](College of Computer Science and Technology, Anhui University, Hefei 230031, China)

Abstract: The use of big data in the real world was complicated. It may contain different characteristic of data and computing. In this case, the single computing mode was mostly difficult to met the application requirements. Therefore we need to consider different computing mode of mix use. The ultimate evolution of hybrid computing mode is spark system which invented by UCBerkeley AMPLab. It covers almost all the typical big data computing mode, including iterative computing, batch computing, memory computing, flow computing (Spark Streaming), data query analysis (Shark), and map computing (GraphX). Spark provides a powerful memory computing engine and implents computing with excellent performance, while maintaining compatibility with the Hadoop platform. Therefore, with the continuous stable and mature, Spark is expected to be colocalized with Hadoop and became a new generation of big data processing systems and platforms. The paper has studied and analyed the Spark ecosystem, and set up the hybrid computing model architecture based on Spark platform, which also has illustrated the spark ecosystem can meet the application of hybrid computing model.

Key words: big data; hybrid computing mode; spark; resilient distributed dataset

1 引言

目前的大数据处理可以分为如下三个类型: (1)复杂的批量数据处理(batch data processing), 通常的时间跨度在数十分钟到数小时之间. (2)基于历史数据的交互式查询(interactive query)通常的时间跨度在数十秒到数分钟之间. (3)基于实时数据流的数据处理(streaming data processing), 通常的时间跨度在数百毫秒到数秒之间[1]. 目前已有很多相对成熟的开源软件来处理以上三种情景, 我们可以利用 MapReduce[2]来进行批量数据处理, 可以用 Impala[3]来进行交互式查询, 对于流式数据处理, 我们可以采用 Strom[4]. 然而, 现实世界中的大数据处理问题复杂多样, 难以有一种单一的计

① 收稿时间:2014-07-19;收到修改稿时间:2014-08-25

算模式能涵盖所有不同的大数据计算需求. 研究和实际应用中发现, 由于 MapReduce 主要适合进行大数据线下批处理, 在面向低延迟和具有复杂数据关系和复杂计算的大数据问题时有很大的不适应性. 因此, 近几年学术界和业界在不断研究并推出多种不同的大数据计算模式.

所谓大数据计算模式, 即根据大数据的不同数据特征和计算特征, 从多样性的大数据计算问题和需求中提炼并建立的各种高层抽象(Abstraction)或模型(Model). 例如, MapReduce 是一个并行计算抽象, Spark 系统中的"分布内存抽象 RDD(RDD, a distributed memory abstraction)[5], CMU 著名的图计算系统 GraphLab 中的"图并行抽象"(Graph Parallel Abstraction)[6]等. 传统并行算法主要是从编程语言和体系结构层面定义了较为底层的模型, 但由于大数据处理问题具有很多高层的数据特征和计算特征, 因此大数据处理需要更多地结合这些高层特征考虑更为高层的计算模式.

根据大数据处理多样性的需求, 目前出现了多种典型和重要的大数据计算模式. 与这些计算模式相适应, 出现了很多对应的大数据计算系统和工具. 由于单纯描述计算模式比较抽象, 因此, 在描述不同计算模式时, 将同时给出相对应的典型计算系统和工具, 这将有助于对计算模式的理解以及对技术发展现状的把握, 并进一步有利于在实际大数据处理应用中对合适的计算技术和系统工具选择使用. 目前已有的计算模式和典型系统如表 1 所示:

表 1 典型大数据计算模式与系统

典型大数据计算系统	典型系统
大数据查询分析计算	Hbase, Hive, Cassandra, Impala, Shark, Hana 等
批处理计算	Hadoop MapReduce, Spark 等
流式计算	Scribe, Flume, Strom, S4, Spark Streaming 等
迭代计算	HaLoop, MapReduce, Spark 等
图计算	Pregel, Giraph, Trinity, PowerGraph, GraphX 等
内存计算	Dremel, Hana, Spark 等

因此, 如果要在一个项目中同时满足各种大数据需求, 需要使用多套特化系统. 一方面在各种不同系统之间避免不了要进行数据转储(ETL), 这无疑将增加系统的复杂程度和负担, 另一方面使用多套系统也增加了使用和维护的难度. 而使用 spark 系统则可以适用目前常用的各种大数据计算模式.

2 为什么spark适合混合计算模式

Spark 可以满足各种不同大数据计算模型的主要原因是提出了 RDD(Resilient Distributed Dataset, 弹性分布式数据集)这一抽象工作集. RDD 基于内存工作的, 由于内存存取的速度要远远快于磁盘的读取速度, 并且可以减少 I/O 操作的时间, 因此可以提升 spark 处理数据的速度. 同时 RDD 实现了容错, 传统上实现容错的方法主要有两种: 检查点和日志. 由于使用检查点容错机制会存在数据冗余并且会增加网络通信的开销, 因此 Spark 系统采用日志数据更新的方式进行容错.

2.1 **Resilient Distributed Dataset (RDD)弹性分布数据集**

利用其基于内存的特点, Spark 浓缩提炼出了 RDD 这一高度通用的抽象结构进行计算和容错[7]. 本质上是对分布式内存的抽象使用, 以类似于操作本地集合的方式来处理分布式数据集. RDD 是一串序列化的, 已被分区的数据集, 并且每次对 RDD 操作后的结果都可以 cache 到内存中, 以后的每个操作可以直接从内存读取上一个 RDD 数据集, 从而省去了对磁盘 IO 操作的过程. 这对于交互式数据挖掘和机器学习方法中使用频率较高的迭代计算而言大大提升了工作效率. RDD 创建的方法有两种: (1)直接从 HDFS(或者其它与 Hadoop 兼容的文件系统)加载数据集. (2)从父 RDD 转换得到新 RDD.

RDD 支持两种操作: (1)转换(transformation)如: filter, map, join, union 等, 从现有的数据集创建一个新的数据集; (2)动作(actions)如: reduce, count, save, collect 等, 是将 transformation 的数据集进行叠加计算, 并将计算结果传递给驱动程序. 为了提高运行效率, Spark 中所有的 actions 都是延迟生成的, 也就意味着 actions 不会立即生成结果, 它只是暂时记住之前发生的转换动作, 只要当真正需要生成数据集返回给 Driver 时, 这些转换才真正执行. 例如: 在实际使用过程中我们先调用 map 生成一个新数据集, 然后在 reduce 中使用这个数据集, 最终 Spark 只会将 reduce 的结果返回给 driver, 而不是返回整个数据集. 通过这种运行方式提高了 Spark 的运行效率.

2.2 **Lineage(血统)**

Spark 是一种利用内存进行数据加载的系统, 但是与其它 In-Memory 类系统的主要区别是 Spark 提供了独特的分布式运算环境下的容错机制. 当节点失效/

数据丢失时，为了保证RDD数据的鲁棒性，Spark通过Lineage(血统)来记忆 RDD 数据集的演变过程并进行数据恢复. 相比于其它内存系统如 Dremel, Hana 依靠备份或者 LOG 进行细颗粒度的数据容错，Spark 中的 Lineage 是一种粗颗粒的容错机制，它只是记录特定数据转换操作行为过程. 当某个 RDD 数据丢失时，可以根据 Lineage 获取足够的信息，从而重新计算或者虎符丢失的数据. 这种粗颗粒度的数据模型减少了数据冗余和备份，也带来了性能的提升.

RDD 在 Lineage 依赖方面分为两种：Narrow Dependencies 与 Wide Dependencies 用来解决数据容错的高效性[3]. 如图 1 所示: Narrow Dependencies(窄依赖)是指父 RDD 的每个分区仅对应一个子 RDD 分区，而一个子 RDD 分区可以使用一个或者多个父 RDD 分区. Wide Dependencies(宽依赖)是指父 RDD 的每个分区可以对应多个子 RDD 分区，而一个子 RDD 分区也可以使用父 RDD 的多个分区.

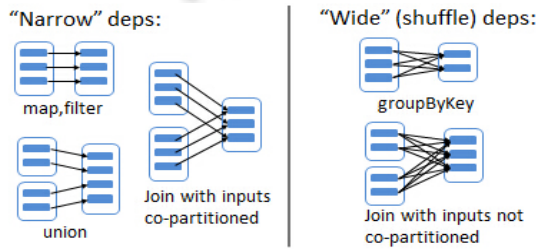

图 1　窄依赖和宽依赖示例

当一个节点宕机时，如果运算是窄依赖，因为丢失的数据不依赖其它节点，那么只需要根据 Lineage 记录的操作过程，重新计算 RDD 分区即可. 如果运算是宽依赖，由于 Lineage 会对之前每一步计算操作进行物化，保持中间结果，这样根据前面操作的物化结果重新计算数据集即可. 如果上一步中间的物化结果也丢失了或者中间节点也宕机时，这时候需要在向上判断其祖先节点的所有输入是否有效(这就是 lineage, 血统的意思)，如果有效则根据 Lineage 记录的操作重新计算. 因此可以看出，宽依赖情况下对于数据重算的开销要远大于窄依赖.

2.3 Spark 执行过程分析

Spark 执行过程主要分为两个阶段：第一阶段生成RDD数据集、增量构建DAG图(Direct Acyclic Graph, 有向无环图)、Lineage 记录变换算子序列；第二阶段 Task Scheduler 通过 Cluster manager 如(Mesos, Yarn 等)将 DAG 中的任务集发送到集群中的节点上执行. 图 2 显示了 Spark 程序的运行场景.

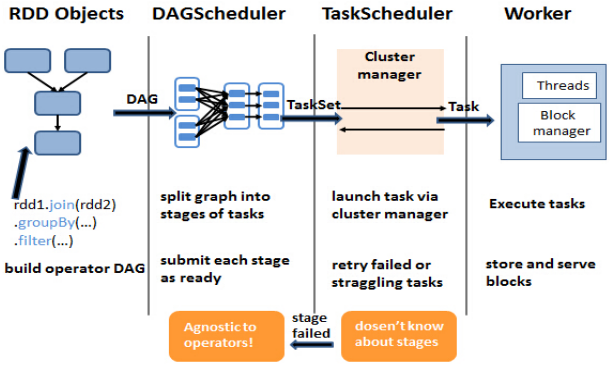

图 2　Spark 程序运行过程

总的来说，每一个 Spark 应用都通过驱动程序(driver program)调用用户的 main 函数，在集群上并行的执行 RDD. RDD 可以是从 HDFS 中创建或者通过已经存在的 RDD 计算生成，用户可以在内存中保留 RDD 以方便重复使用，最后当 RDD 丢失时也可以通过 Lineage 自动恢复.

2.4 Spark 生态系统

正因为有了 RDD 这个抽象数据结构，Spark 立足于内存计算，提供了流计算，迭代计算，图计算等一系列计算范式的解决方案. 可以说目前 Spark 是大数据领域内罕见的"全能选手". Spark 的生态系统如图 3 所示:

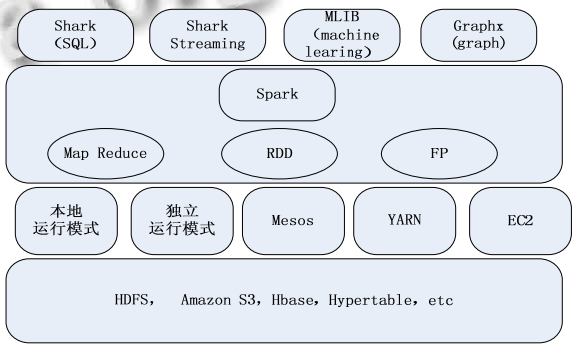

图 3　spark 生态系统

目前最新发布的 Spark1.0.0 版本主要支持的组件有：(1)用于大数据查询分析计算组件 Shark[8,9]. 对于 Spark 来说，Shark 的作用就类似于 Hive 在 Hadoop 系统中的作用. Shark 提供了一系列的命令接口，通过配置参数可以缓存 Spark 中特定的 RDD，并对数据集进

行检索. 此外, Shark 可以调用 UDF(用户自定义函数)将数据分析与 SQL 查询相结合并实现数据重用, 从而提高计算速度. (2)用于流式计算组件 Spark Streaming[10]. 它的基本原理是将流数据分割成非常小的数据片段封装到RDD分区中, 然后以类似批处理的方式来处理这些小数据. 利用 Spark 基于内存的特点, 可以保证计算的低延迟性(100ms 左右), 以及兼容批处理和实时数据处理的算法, 另外通过 Lineage 也能保证数据的容错处理. (3)用于图计算的 GraphX; Spark 的 Graphx 提供了对图操作的 API, 它通过 Resilient Distributed Property Graph 从 Spark 的 RDD 继承. 在最新的 spark1.0.0 中, GraphX 在图加载、边反转和邻接计算方面对通信的要求更低, 产生的 RDD 图更简单, 从而在性能方面得到了很大提升. 利用 GraphX 框架可以很方便的实现多种图算法. (4)用于机器学习 MLIB 组件. MLIB 提供了机器学习算法的实现库, 目前支持聚类、二元分类、回归以及协同过滤算法, 同时也提供了相关测试和数据生成器.

Spark 既可以在本地单节点运行(开发调试用)也可以集群运行. 在集群运行的情况下需要通过集群管理系统如 Mesos, Yarn 等将计算任务分发到分布式系统中的各个节点上运行.

Spark 中的 RDD 数据来源可以由 HDFS(或者其它类似文件系统如 Hbase, Hypertable, 本地文件)等生成.

3 Spark混合计算模型架构

如图 4 所示: 一般大数据架构首先确定数据源, 经过一个处理过程, 把它存储下来, 经过计算(Mapreduce, spark 等), 将计算结果存放到可以实时查询的数据库中或者存储介质上去. 为节省查询和计算的时间, 可以把提前将一些数据结果作为 web 接口或者报表展示给用户. 根据一般数据操作流程本文抽象出了一种基于 spark 系统的大数据架构, 具体如图 5.

首先将不同数据源数据通过分布式系统的每个 Agent 进行移动和收集, 通过 Staging/Distributing 模块进行数据分发. 分发以后的数据会分为两种流向. 一种流向是通过 ETL 存入 HDFS 上, 然后经过 spark 处理, 也可以编写一些 shark 程序进行批处理. 因为系统是运行在 yarn 上, 所以即使以前的 MapReduce 或者 hive 来不及替换, 也可以继续运行. 另一种数据流向是直接走流处理路线. 走流处理流程会阶段性的将计算数据放到 RDBMS 或者 NoSQL 中, 然后一些实时应用会去调用已经计算好的中间结果, 减少操作时间. 该架构既包含了大数据批处理和实时处理的需求, 同时对原有的 MapReduce 架构有良好的兼容性, 我们可以在使用过程中逐步将系统中的 Mapreduce|Hive 逐渐替换为 Spark|Shark 架构.

图 4 数据操作流程

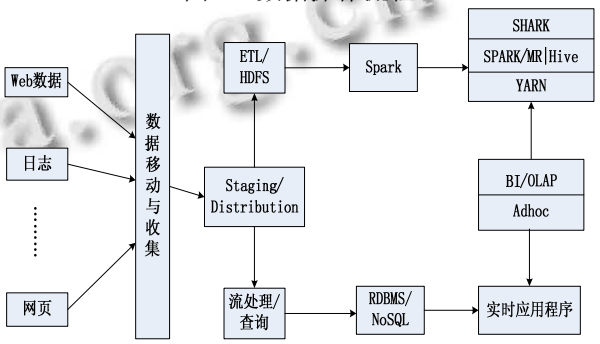

图 5 Spark 混合计算模型架构

4 结语

从长远来看, Spark 系统的优势是可以使用一个框架来满足多种应用场景, 包括批处理计算、流计算、图分析、机器学习等等. 本文详细研究和分析了 Spark 架构和核心原理. 在此基础之上设计了一种用于混合计算模型的架构, 按照目前的发展趋势, Spark 有望成为继 Hadoop 之后又一个大数据处理的热门框架.

参考文献

1 夏俊鸾,邵赛赛.Spark Streaming: 大规模流式数据处理的新贵. http://www.csdn.net/article/2014-01-28/2818282-Spark-Streaming-big-data. 2014

2 Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2008, 3(51-1): 107–113.

3 耿益锋,陈冠诚.Impala:新一代开源大数据分析引擎. http://www.csdn.net/article/2013-12-04/2817707-ImpalaBig-Data-Engine. 2013.12

4 Strom. http://storm.incubator.apache.org/. 2014

5 Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstration for in-memory cluster computing. Proc. of the 9th USENIX Conference on

NetWorked System Design and Implementation. 2012. 2–16.

6 Gonzalez J, Low Y, Gu H. PowerGraph: Distributed garph–parallel computation on natural graphs. Proc. of the 10th USENIX Symposium on Operating Systems Design and Implementatin. 2012. 17–30

7 Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster Computing with Working Sets. Technical Report No. UCB/ EECS- 2010-53 May 7, 2010

8 Xin R, Rosen J, et al. Shark: SQL and Rich Analytics at Scale. [Technical Report] UCB/EECS. 2012.11

9 Engle C, Lupher A, et al. Shark: Fast Data Analysis Using Coarse-grained Distributed Memory. SIGMOD 2012. May 2012.

10 Zaharia M, Das T, Li HY, Shenker S, Stoica I. Discretized streams: An efficient and fault-tolerant model for stream. Proc. on Large Clusters. HotCloud 2012. June 2012.