北京航空航天大學
BEIHANG UNIVERSITY

<<流式实时分布式计算框架Spark的研究与应用>>

需求规格说明书

北京航空航天大学

2016-04

版本变更历史

版本	提交日期	主要编制人	审核人	版本说明
V1.0	2016/3/23	阳艳红、武一杰、王铖成、于思民	于思民	初稿
V1.1	2016/3/24	阳艳红、武一杰、王铖成、于思民	于思民	二稿
V1.2	2016/4/1	阳艳红、武一杰、王铖成、于思民	于思民	三稿
V1.3	2016/4/1	阳艳红、武一杰、王铖成、于思民	于思民	四稿
V2.0	2016/4/3	阳艳红	于思民	五稿
V2.1	2016/4/4	阳艳红	于思民	六稿
V3.0	2016/4/7	阳艳红、武一杰、王铖成、于思民	于思民	七稿
V3.1	2016/4/8	阳艳红、武一杰、王铖成、于思民	于思民	七稿
V3.2	2016/4/9	阳艳红、武一杰、王铖成、于思民	于思民	七稿
V3.3	2016/6/25	于思民	于思民	八稿

目 录

1. 范围

1.1 标识

Spark 版本号：Spark1.2.1

需求报告版本：V3.3

1.2 系统概述

Apache Spark 是一种快速、通用、可扩展的大数据分析引擎。它是不断壮大的大数据分析解决方案家族中备受关注的明星成分，为分布式数据集的处理提供了一个有效框架，并以高效的方式处理分布式数据集。Spark 集批处理、实时流处理、交互式查询与图计算于一体，避免了多种运算场景下需要部署不同集群带来的资源浪费。

无论是工业界还是学术界，都已经广泛使用高级集群编程模型来处理日益增长的数据。这些系统将分布式编程简化为自动提供位置感知性调度、容错及负载均衡，使得大量用户能够在商用集群上分析超大数据集。

然而，集群环境对于编程来说带来了很多挑战，首先就是并行化：这就要求以并行化的方式重写应用程序，以便利用更大范围节点的计算能力。集群环境的第二个挑战就是对单点失败的处理，节点宕机以及个别节点计算缓慢在集群环境中非常普遍，这会极大影响程序的性能。最后一个挑战就是集群在大多数情况下都会被多个用户共享，那么动态地进行计算资源分配，也会干扰程序的执行。

因此，针对集群环境出现了大量的大数据框架。如 Google 的 Mapreduce、Storm、Impala 和 GrapLab。但是这些专有系统存在一些不足。

- 重复工作：许多专有系统存在同样的问题，比如分布式作业以及容错。举例来说，一个分布式的 SQL 引擎或者一个机器学习系统都需要实现并行聚合。这些问题在每个专有系统会重复的被解决。
- 组合问题：在不同系统之间进行组合计算是一件费力不讨好的事情。对于特定的大数据应用程序而言，中间数据非常庞大，而且移动的成本也非常高昂。

在目前的环境中，一般采取的措施就是将数据复制到稳定的存储系统中，以便在不同的计算引擎中进行分享。然而，这样的复制可能比真正的计算所花费的代价要大，所以以流水线的形式将多个系统组合起来效率并不高。

- 使用范围的局限性：如果一个应用不适合一个专有的计算系统，那么使用者只能关一个系统，或者重写一个新的计算系统。
- 资源分配：在不同的计算引擎之间进行资源的动态共享是比较困难的，因为大多数的计算引擎都会假设程序运行结束之前它们拥有相同的机器节点的资源。
- 管理问题：对于多个专有系统，需要花费更多的经历和时间来部署和管理。尤其是对于终端使用者而言，他们需要学习多种 API 和系统模型。

针对 MapReduce 及各种专有系统中的不足，伯克利大学退出了全新的统一大数据处理框架 Spark，创新性地提出了 RDD 概念，在某种程度上 Spark 是对 MapReduce 模型的一种扩展。Spark 在大数据分析处理平台中脱颖而出得益于以下主要优点。

- 速度快

与 Hadoop 的 MapReduce 相比，Spark 基于内存的运算要快上 100 倍以上；而基于硬盘的运算也要快 10 倍以上。Spark 实现了高效的 DAG 执行引擎，可以通过内存来高效处理数据流。

- 易用

Spark 支持 Java、Python 和 Scala 的 API，还支持超过 80 中高级算法，使用户可以快速构建不同的应用。而且 Spark 支持交互式的 Python 和 Scala 的 shell，这意味着可以非常方便地在这些 shell 中使用 Spark 集群验证解决问题的方法，而不像以前那样，需要打包、上传集群、验证等一系列操作。

- 通用性

Spark 提供了统一的解决方案。Spark 可以用于批处理、交互式查询（通过 Spark SQL）和图计算（通过 Spark GraphX）。这些不同类型的处理都可以在同一个应用中无缝使用。Spark 统一的解决方案非常具有吸引力，为公司统一平台方案减少了开发和维护的人力成本和部署平台的物力成本。作为统一的解决方案，Spark 并没有以牺牲性能为代价。相反，在性能方面，Spark 具有很大的优势。

- 可融合性

Spark 可以非常方便地与其它的开源产品进行融合。比如，Spark 可以使用 Hadoop 的 YARN 和 Apache Mesos 作为它的资源管理和调度器，并且可以处理所有 Hadoop 支持的数据集，包括 HDFS、HBase 和 Cassandra 等。这对已经部署 Hadoop 集群的用户特别重要，因为不需要做任何数据迁移就可以使用 Spark 的强大处理能力。Spark 也可以不依赖于第三方的资源管理和调度器，它实现了 Standalone 作为其内置的资源管理和调度框架，这样进一步降低了 Spark 的使用门槛，使得所有人都可以非常容易地部署和使用 Spark。此外，Spark 还提供在 EC2 上部署 Standalone 的 Spark 集群的工具。

1.3 文档概述

本文档是对 Spark1.2.1 版本的各模块需求分析和规格说明书，确定 Spark 系统的功能需求、非功能需求。

本文档不涉密，可自由用于学习交流。

1.4 术语和缩略词

编号	术语	英文	说明
1	Spark	Spark	Apache Spark 是一种快速、通用、可扩展的大数据分析引擎。
2	Spark 流处理库	Spark Streaming	Spark Streaming 基于 Spark Core 实现了可扩展、高吞吐和容错的实时数据流处理。
3	Spark 机器学习库	Spark MLlib	MLlib 是 Spark 对常用的机器学习算法的实现库，同时含有相关的测试和数据生成器，包括分类、回归、聚类、协同过滤、降维以及底层基本的优化元素。
4	Spark 交互式 SQL 查询	Spark SQL	Spark SQL 作为 Spark 平台获取数据的一个渠道。

	库		
5	Spark 图计算库	Spark GrapX	Spark GraphX 是 Spark 提供的关于图和图并行计算的 API。
6	弹性分布式数据集	RDD	Resilient Distributed Dataset,即弹性分布式数据集，是 Spark 中只读的、分区记录的集合。
7	分区	Partition	RDD 中的数据分片。
8	有向无环图	DAG	RDD 依赖关系形成的层次结构。
9	转换	Transformation	从父 RDD 到子 RDD 之间的转换。
10	动作	Action	从 RDD 到计算出结果执行的操作。
11	检查点	Checkpoint	未进行容错，而采取的记录系统某一时刻的状态。
12	阶段	Stage	RDD 依赖不同层次的分类。
13	作业	Job	用户程序实体。
14	任务	Task	用户程序划分的实际执行单元。
15	调度器	Scheduler	对资源及任务进行调度。
16	部署	Deploy	Spark 的部署模块。
17	本地	Local	Spark 的本地部署模式。
18	单机	Standalone	Spark 的单机部署模式。
19	Mesos	Mesos	Mesos 计算框架一个集群管理器,提供了有效的、跨分布式应用或框架的资源隔离和共享，可以运行 Hadoop、MPI、Hypertable、Spark。
10	YARN	YARN	Hadoop2.x 的资源管理器。
11	工作节点	Worker	执行任务 Task 的物理单元。
12	执行器	Executor	管理工作节点的单元。
13	Shuffle	Shuffle	将不同 Stage 的数据迁移到下一 Stage 的过程。
14	存储	Storage	Spark 的存储模块。

15	主节点	Master	与 Slave 节点相对应
16	从节点	Slave	与 Master 节点相对应
17	数据共享单元	Data Share Unit	数据共享单元就是，分布式计算式每个节点进行计算的数据单位，这些数据单位可以在各节点间进行共享。

2. 引用文档

无。

3. 功能需求

3.1 数据编程模型

在分布式计算过程中，有许多数据流模型在不适应应用场景时有个共同的特征：在计算过程中多需要高效率的数据共享。例如，迭代算法、聚类或者逻辑回归，都需要进行多次访问相同的数据集；交互式数据挖掘经常需要对于同一数据子集进行多个特定的查询；而流式应用则需要随时间对状态信息进行维护和共享。尽管数据流框架支持大量的计算操作运算，但是它们缺乏针对数据共享的高效原语。

为此，需要设计一种全新的数据编程模型，即数据抽象模式，使得用户可以直接控制数据的共享。将这种数据抽象模式，是一种弹性的分布式数据集，即 RDD（Resilient Distributed Dataset）。RDD 具有可容错性课并行数据结构特征，使得用户可以指定数据存储到硬盘还是内存，以及用户也可以控制数据的分区方法，并在数据集上进行种类丰富的操作。

在提供高效容错能力方面，RDD 提供一种粗粒度变换，可以将相同的操作应用到多个数据集上。而 RDD 通过记录创建数据集的变换，而不需要存储真正的数据，进而达到高效的容错性。

RDD 的数据编程模型的用例图如图 1 所示。

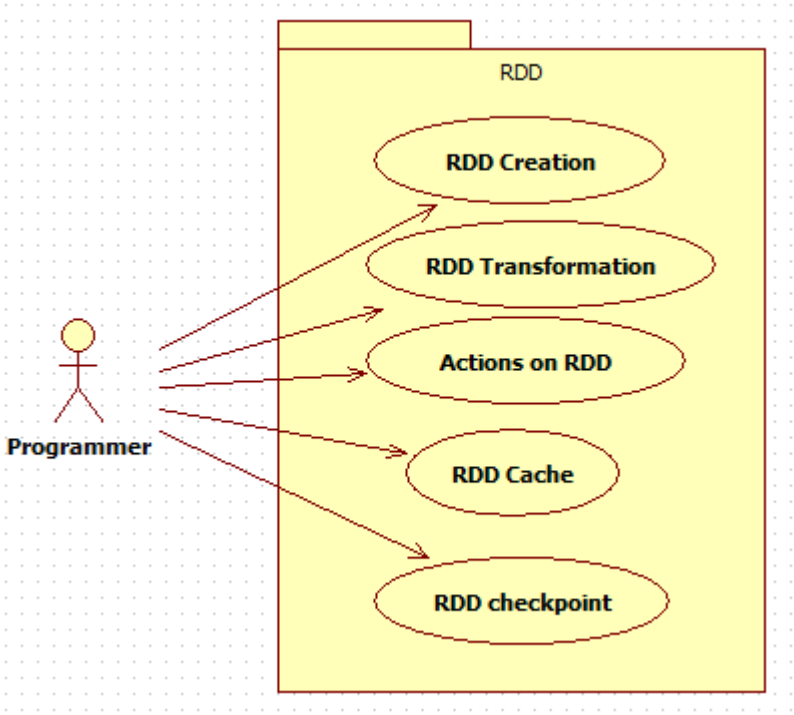


图 1 弹性分布式数据集用例图

3.1.1 RDD 的创建

开发人员可以通过从本地或者其他第三方分布式文件系统读取数据，然后生成 RDD；开发人员也可以通过对以后的 RDD 进行某种转换操作生成新的 RDD。开发人员从文件系统中读取数据，创建 RDD 的用例如图 2 所示。

Use Case Specification	
Use Case Name	RDD Creation
Brief Description	创建一个弹性分布式数据集RDD。
Precondition	None
Primary Actor	Programmer
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Normal" ▼	Steps	
	1	用户设置数据的uri。
	2	将uri对应的数据读取到Spark应用程序。
	Postcondition	RDD创建成功。

图 2 RDD 创建的用例

开发人员首先要提供数据所在的 URI，然后将该 URI 对应的数据读入到

Spark，而这个过程也就在 Spark 中创建了一个 RDD。

当然用户也可以从已有的 RDD 中，执行某种操作，从而创建新的 RDD。

3.1.2 RDD 的转换

开发人员根据实际业务逻辑对已有的数据形式对应的 RDD 做出某种转换操作，从而生成新的数据形式的 RDD。开发人员对 RDD 进行转换的用例如图 3 所示。

Use Case Name	RDD Transformation
Brief Description	开发人员对已有的RDD对应的数据采用某种转换操作，将一种数据形式的RDD转化成另外一种数据形式的RDD。
Precondition	None
Primary Actor	Programmer
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow	Steps	
"Normal"▼	1	获取一个RDD。
	2	在此RDD上进行数据转换操作。
	3	返回新的数据形式对应的RDD。
	Postcondition	RDD转换成功。

图 3 RDD 转换的用例

开发人员获取已有的 RDD，并在该 RDD 上调用代表某种数据转换的函数操作，操作完成后，该 RDD 会转换成新的 RDD。

3.1.3 RDD 的操作

开发人员在对 RDD 进行一系列的数据形式转换后，需要将最终形式的数据存储到文件系统，可能是本地文件系统，或者第三方的分布式文件系统。对 RDD 进行操作（Action）的用例如图 4 所示。

Use Case Specification	
Use Case Name	Action on RDD
Brief Description	开发人员在经过一系列的数据形式的转换操作后，会对最终的RDD中的数据形式，如存储到文件系统，如提供给其他系统等。
Precondition	None
Primary Actor	Programmer
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	获取最终的RDD。
	2	在该RDD上调用某种Action的函数调用。
	Postcondition	数据操作成功。

图 4 RDD 操作用例

开发人员通过一系列的 RDD 转换操作，将最初的数据形式转化为所需的数据形式后，会对最终的数据形式执行某种操作，如存储到文件系统，如提供给其他系统进行处理。

3.1.4 RDD 的缓存

由于 RDD 可能会被用户应用程序重复使用，所以一种高效的读取 RDD 的方式就是将该 RDD 缓存起来，以提高效率。开发人员缓存 RDD 的用例如图 5 所示。

Use Case Specification	
Use Case Name	RDD Cache
Brief Description	开发人员将重要的RDD缓存起来，以备后续重复使用，这样可以提高效率。
Precondition	None
Primary Actor	Programmer
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Noraml" ▼	Steps
	1 获取要进行缓存的RDD。
	2 IF 该RDD未被缓存
	3 THEN 将该RDD缓存。
	Postcondition None

Specific Alternative Flow "Alternative" ▼	RFS 2
	1 ELSE 该RDD已被缓存
	2 不采取任何操作，返回。
	Postcondition None

图 5 缓存 RDD 的用例

用户在进行缓存 RDD 时，首先会检查该 RDD 是否已经被缓存。如果该 RDD 尚未被缓存，则会进行缓存该 RDD 操作；否则，不采取任何操作。

3.1.5 RDD 检查点

RDD 的缓存在第一次完成计算后，将计算结果保存到内存、本地文件系统中。通过缓存，Spark 可以避免了 RDD 上的重复计算，能够极大地提升计算速度。如果缓存丢失了，则需要重新计算。如果计算特别复杂或者计算耗时特别大，那么缓存丢失对整个作业的影响都是不可小觑的。为避免缓存丢失而进行重复计算的开销，Spark 设计了检查点（checkpoint）机制。检查点的用例如图 6 所示。

Use Case Specification

Use Case Name	RDD chekpoint
Brief Description	开发人员为计算特别复杂后者计算耗时特别大的RDD进行容错，而采取的机制就是为该RDD创建检查点。当该RDD丢失时，可以根据该检查点恢复RDD，而不需要进行重新计算。
Precondition	None
Primary Actor	Programmer
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Normal" ▼	**Steps**
	1 获取需要创建检查点的RDD。
	2 **IF** 尚未为该RDD创建检查点
	3 **THEN** 为该RDD创建检查点。
	Postcondition None

Specific Alternative Flow "Alternative" ▼	**RFS** 2
	1 **ELSE** 已经为该RDD创建检查点
	2 **THEN** 不进行任何操作。
	Postcondition None

图 6 检查点创建的用例

首先，开发人员在为某 RDD 创建检查点时，Spark 会检测该 RDD 是否已经存在检查点。如果该 RDD 不存在检查点，则为当前 RDD 创建检查点；否则，不进行任何操作。

3.2 数据存储

在数据计算的过程中，往往会产生大量的新的数据。分布式计算框架需要将这些数据存储起来，存储到硬盘或者内存中。

通常数据存储包括一下两个方面。

第一个方面就是存储数据信息本身，也就是计算中产生的数据。

第二个方面就是存储数据的元信息。元信息描述的数据的存储位置、访问权限等信息。

正是数据存储有这两类特征，使得数据存储十分适合主从（Master/Slave）架构。

Master 负责管理用户应用程序数据的元信息；而 Slave 负责具体数据的存储，并将元信息的更新状态同步到 Master，同时也要接受来自 Master 的命令，对数据执行操作增、删、改、查等操作。

数据存储模块的用例图如图 7 所示。

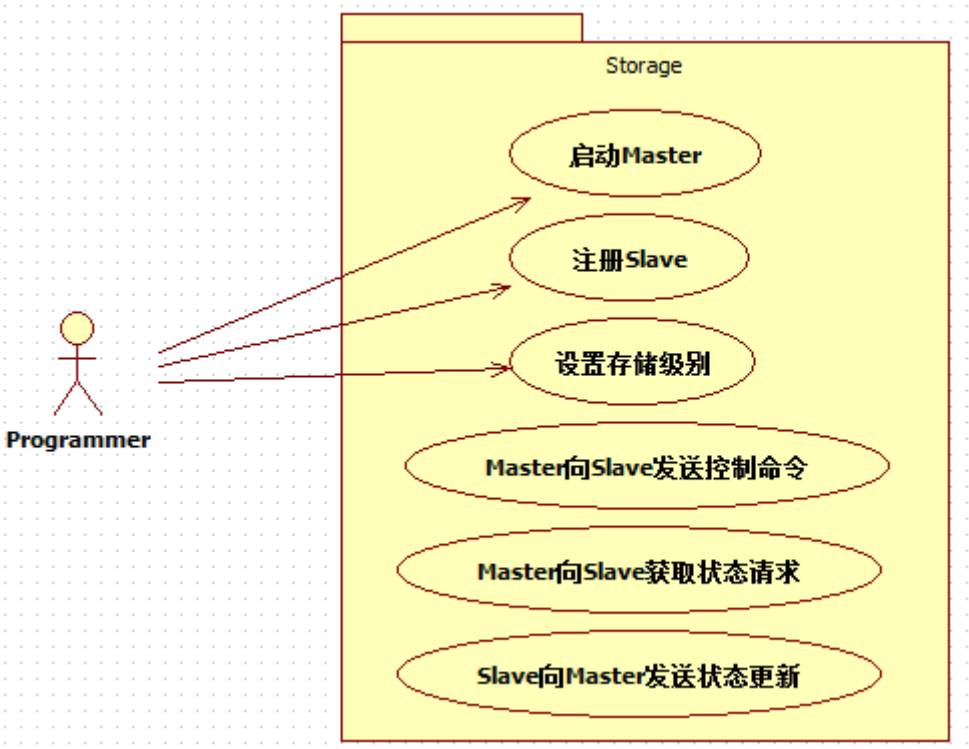


图 7 数据存储模块用例图

3.2.1 启动 Master

Spark 集群启动时，同时也要启动主从结构的数据存储节点（进程）。其中启动 Master 的用例如图 8 所示。

Use Case Specification	
Use Case Name	启动Master
Brief Description	开发人员启动存储主节点（主进程）
Precondition	None
Primary Actor	Programmer
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Normal" ▼	Steps	
	1	开发人员设置Master节点配置信息
	2	向Spark集群发送启动Master命令
	3	获取VALIDATES THAT Master启动信息
	Postcondition	None

Specific Alternative Flow (Untitled) ▼	RFS 3	
	1	Master启动出现错误。
	Postcondition	None

Specific Alternative Flow (Untitled) ▼	RFS 3	
	1	Master成功启动。
	Postcondition	None

图 8 Master 启动用例图

3.2.2 注册 Slave

在启动主节点后，Slave 节点会启动，并向 Master 节点注册，成为其从节点。Slave 向 Master 注册的用例图如图 9 所示。

Use Case Specification

Use Case Name	注册Slave
Brief Description	Slave节点向Master节点注册
Precondition	None
Primary Actor	None
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Normal" ▼	**Steps**
	1 读取Slave节点配置信息。
	2 启动Slave节点。
	3 读取Master节点信息。
	4 获取状态更新。
	5 向Master发送状态更新，进行注册。
	6 **IF** Master是否响应注册 **THEN**
	7 **VALIDATES THAT** Master响应数据
	8 **ELSE**
	9 提示注册失败。
	Postcondition None

Specific Alternative Flow (Untitled) ▼	**RFS** 7
	1 响应数据为成功注册数据。
	Postcondition None

Specific Alternative Flow (Untitled) ▼	**RFS** 8
	1 响应数据为注册失败数据。
	Postcondition None

图 9 Slave 注册用例图

3.2.3 Master 向 Slave 发送控制命令

Master 节点可以向 Slave 发送控制信息，如删除数据块、获取 RDD 相关的 RDD、广播变量相关的数据等。Master 向 Slave 发送控制命令的用例图如图 10 所示。

Use Case Specification

Use Case Name	Master向Slave发送控制命令
Brief Description	Master向Slave发送具体的控制命令
Precondition	None
Primary Actor	None
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Normal" ▼	**Steps**
	1 读取Slave节点配置信息。
	2 向Slave发送某控制命令请求。
	3 **IF** Slave是否响应注册 **THEN**
	4 **VALIDATES THAT** Slave响应数据
	5 **ELSE**
	6 提示注册失败。
	Postcondition None

Specific Alternative Flow (Untitled) ▼	**RFS** 7
	1 响应数据为操作成功数据。
	Postcondition None

Specific Alternative Flow (Untitled) ▼	**RFS** 8
	1 响应数据为操作失败数据。
	Postcondition None

图 10 Master 向 Slave 发送控制命令用例图

3.2.4 Master 向 Slave 获取状态请求

Master 有时还需要获取数据块的状态等信息，这时 Master 需要向 Slave 发送获取状态的请求。Master 向 Slave 获取状态请求的用例图如图 11 所示。

Use Case Specification

Use Case Name	3.2.4Master向Slave获取状态请求
Brief Description	Master需要获取数据块的状态等信息，这时Master需要向Slave发送获取状态的请求。
Precondition	None
Primary Actor	None
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Normal"	Steps
	1 读取Slave节点配置信息。
	2 向Slave发送获取状态的请求。
	3 IF Slave是否响应注册 THEN
	4 VALIDATES THAT Slave响应数据
	5 ELSE
	6 提示注册失败。
	Postcondition None

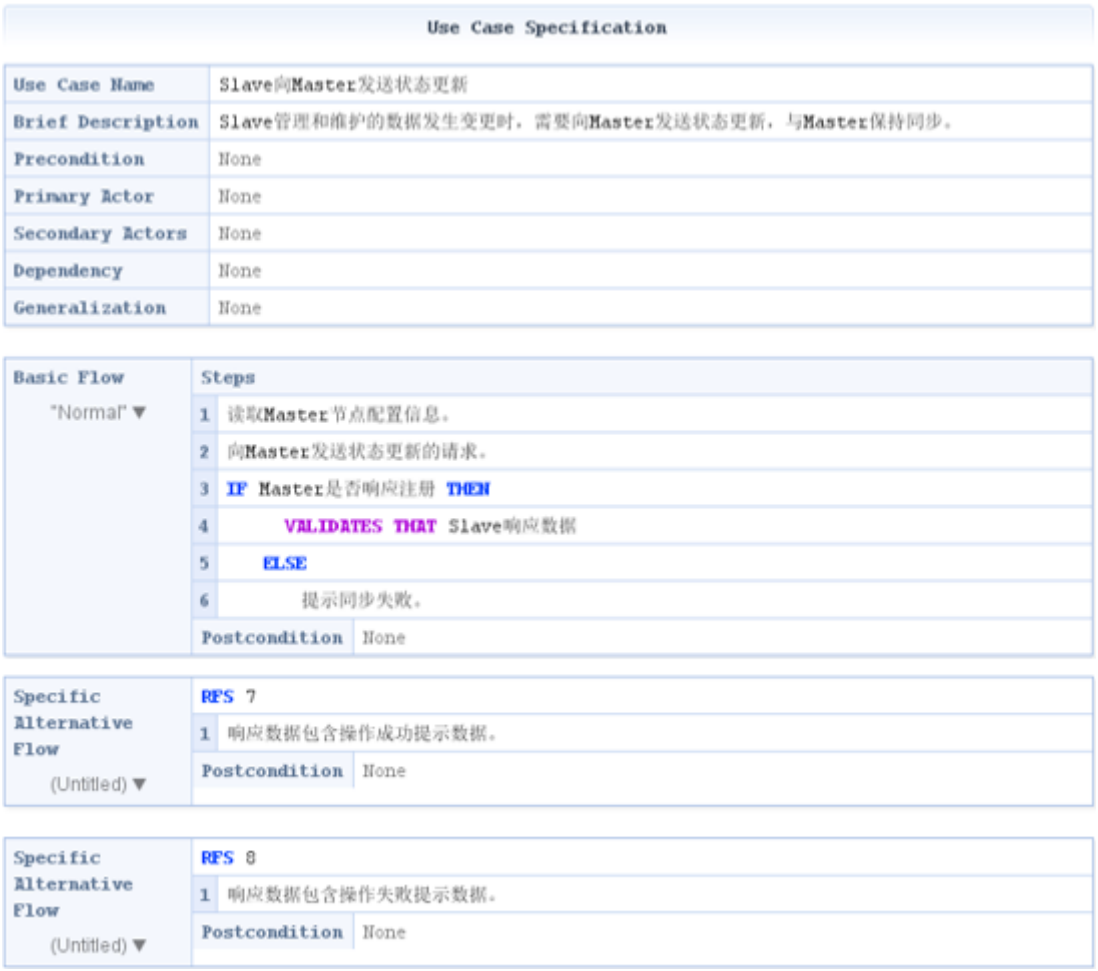
Specific Alternative Flow (Untitled)	RFS 7
	1 响应数据包含操作成功提示数据。
	2 从响应数据中获取状态数据。
	Postcondition None

Specific Alternative Flow (Untitled)	RFS 8
	1 响应数据包含操作失败提示数据。
	Postcondition None

图 11 Master 向 Slave 获取状态请求的用例图

3.2.5 Slave 向 Master 发送状态更新

Slave 管理和维护的数据发生变更时，需要向 Master 发送状态更新，与 Master 保持同步。Slave 向 Master 发送状态更新的用例如图 12 所示。



Use Case Specification

Use Case Name	Slave向Master发送状态更新
Brief Description	Slave管理和维护的数据发生变更时，需要向Master发送状态更新，与Master保持同步。
Precondition	None
Primary Actor	None
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Normal" ▼	Steps
	1 读取Master节点配置信息。
	2 向Master发送状态更新的请求。
	3 IF Master是否响应注册 THEN
	4 VALIDATES THAT Slave响应数据
	5 ELSE
	6 提示同步失败。
	Postcondition None

Specific Alternative Flow (Untitled) ▼	RFS 7
	1 响应数据包含操作成功提示数据。
	Postcondition None

Specific Alternative Flow (Untitled) ▼	RFS 8
	1 响应数据包含操作失败提示数据。
	Postcondition None

图 12 Slave 向 Master 发送状态更新的用例图

3.3 集群部署

Spark 是一个快速数据处理引擎或者平台。Spark 需要庞大的集群节点来支撑其快速的数据处理。由于已经存在比较成熟的集群部署框架，Spark 可以使用这些已有的集群，在这些集群上运行 Spark 作业。

Spark 为适应已有的集群，对各类集群框架进行适配。Spark 集群有一下部署方式：Local、Standalone、Hadoop YARN、Mesos、EC2 等方式。不同的部署方式就有不同的作业调度模式。

3.3.1 Local 部署模式

Spark 可以部署在单机节点上，也就是说，Local 部署模式下，没有集群的概念，所有的工作节点都以进程或者线程的方式出现。Local 部署模式又可以据此

而继续细分。

（1） local

使用一个工作线程来运行计算任务，不会重新计算失败的计算任务。

（2） local[N]/local[*]

对于 local[N]，使用 N 个工作线程；对于 local[*]，工作线程的数量取决于本机的 CPU Core 的数目，保证逻辑上一个工作线程可以使用一个 CPU Core。和 local 一样，不会重新计算失败的计算任务。

（3） local[threads,maxFailures]

threads 设置了工作线程的数目；maxFailures 则设置了计算任务最大的失败重试次数。

（4） local-cluster[numSlaves,coresPerSlave,memoryPerSlave]

伪分布式模式，本机将运行 Master 和 Worker。其中 numSalves 设置了 Worker 的数目；coresPerSlave 设置了 Worker 所能使用的 CPU Core 的数目；memoryPerSlave 设置了每个 Worker 所能使用的内存容量。

3.3.2 Standalone 部署模式

对于 Local 模式下，主节点和工作节点都以进程的方式存在于单一节点。这些都不是分布式的集群，或者只能称之为伪分布式的进群部署。Spark 还应提供真正的分布式部署模式，即 Spark Standalone。

在 Spark Standalone 模式下，Spark 集群有 Master 节点和 Worker 节点构成，用户程序通过与 Master 节点交互，申请所需的资源，Worker 节点负责具体 Executor 的启动运行。

3.3.3 Mesos 部署模式

Apache Mesos 采用了 Master/Slave 的架构。主要由四部分组成：Mesos Master、 Mesos Slave、Framework（Mesos Application）和 Executor。

为使得 Spark 的计算可以运行在 Apache Mesos 之上。Spark 还要对 Apache Mesos 进行适配，即 Spark 应有一套 Mesos 部署模式。

Spark 集群运行在 Mesos 模式下，需要先部署 Mesos 到所有的工作节点上。

在 Mesos 模式中，资源的调度可以分为粗粒度调度和细粒度调度两种，根据这两种资源的颗粒度，Spark 分别使用粗粒度的 Mesos 资源分配器和细粒度的 Mesos 资源分配器来配合任务调度器工作。

3.3.4 YARN 部署模式

同样，Spark 也为 Hadoop 集群适配了 YARN 部署模式。而这种部署可以分为两种，一种是 YARN Client 模式和 YARN Cluster 模式。

a) YARN Cluster 模式

YARN Cluster 模式，就是要通过 Hadoop YARN 来调度 Spark 应用程序所需要的资源。在 YARN Cluster 模式下，Spark 的所有计算都应在 YARN 的集群节点上运行。

b) YARN Client 模式

YARN Client 模式与 YARN Cluster 模式的区别在于用户提交的 Application 在 Spark-Context 是在本机上运行，适合 Application 本身需要在本地进行交互的场景。

3.4 作业调度

Scheduler（任务调度）模块是 Spark Core 的核心模块之一，其主要分为两大部分，即 DAGScheduler 和 TaskScheduler，他们负责将用户提交的计算任务按照 DAG 划分为不同的阶段并且将不同阶段的计算任务提交到集群进行最终的计算。

3.4.1 用户作业提交

用户编写的应用程序，即 Job 最终会调用 DAGScheduler 的 runJob 方法，而该方法又会调用 submitJob 来提交作业。其中作业提交的用例如图 13 所示。

Use Case Specification	
Use Case Name	Job Submission
Brief Description	用户向Spark集群提交的作业，由SparkContext进行提交，DAGScheduler负责创建作业实体。
Precondition	用户应用程序App
Primary Actor	SparkContext
Secondary Actors	DAGScheduler
Dependency	None
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	用户向Spark集群提交作业（从用户角度提交）；
	2	SparkContext为将该作业提交给DAGScheduler；
	3	DAGScheduler为该作业生成Job ID；
	4	DAGScheduler为该作业生成Job Waiter来监听job的执行情况；
	5	DAGScheduler为将改作业的实体提交给eventProcessActor.
	Postcondition	None

图 13 用户作业提交的用例图

3.4.2 DAG 创建

用户作业（Job）在 Spark 集群中执行的过程，其实可以简化为，数据集经过一系列的转化（RDD 转换过程）操作最后执行某种操作（RDD 动作过程），并得到最终的计算结果。在转换的过程中，可能有些重复的转化操作、前后依赖的转化操作等情况。Spark 将一对一的转化操作称之为“窄依赖”；而将非一对一的转化操作称之为“宽依赖”。为了使运行在 RDD 上任务最大化并行操作，Spark 将这些 RDD 划分阶段，形成一个有向无环图的 RDD 转化流程。DAG 生成的用例如图 14 所示。

Use Case Specification	
Use Case Name	DAG Creation
Brief Description	为用户应用程序生成RDD转换的有向无环图。
Precondition	INCLUDE USE CASE RDD Creation,RDD Transformation
Primary Actor	DAGScheduler
Secondary Actors	None
Dependency	None
Generalization	SparkContext

Basic Flow "Normal" ▼	Steps
	1 获取计算任务；
	2 获取RDD信息；
	3 根据用户程序中RDD的转换操作，生成RDD转换的有向无环图。
	Postcondition None

图 14 DAG 生成的用例图

3.4.3 Stage 划分

为了提高作业执行的并行化效率，Spark 根据 RDD 转换的关系，即依赖关系，将 RDD 进行不同阶段的划分。同一 Stage 阶段的 RDD 可以并行执行，而不同阶段的 Stage 可能需要进行同步顺序执行。Spark 在创建 Job 的实体后，会由 DAGScheduler 对 RDD 构成的 DAG 进行划分。Stage 的划分用例如图 15 所示。

Use Case Specification	
Use Case Name	Stage Creation
Brief Description	对不同的RDD进行分阶段划分，以提高RDD上任务执行的并发度。
Precondition	Spark已创建该作业的实体
Primary Actor	DAGScheduler
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Normal" ▼	Steps
	1 获取触发Action（动作）操作的RDD1；
	2 获取该RDD所依赖的父RDD2；
	3 将该RDD1与其有窄依赖关系的父RDD2划分为同一Stage；
	4 将该RDD1与其有宽依赖关系的父RDD2划分为不同的Stage；
	5 对于父RDD2重复2-4操作，直到遍历完所有的RDD；
	6 最终生成不同阶段Stage的RDD有向无环图。
	Postcondition None

图 15 Stage 划分的用例图

3.4.4 TaskSet 生成

在划分完 Stage 阶段后，DAGScheduler 要取得需要计算的 Partition，在判断哪些 Partition 需要计算后，就会为每个 Partition 生成 Task，然后将这些 Task 封装成 TaskSet，即任务集，最后将任务集提交给 TaskSchduler。 任务集生成的用例如图 16 所示。

Use Case Specification

Use Case Name	TaskSet Generation
Brief Description	为需要计算的Partition生成任务集。
Precondition	DAGScheduler完成Job实体创建、Stage划分。
Primary Actor	DAGScheduler
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow "Normal" ▼	**Steps**	
	1	判断需要进行计算的Partion；
	2	依次为每个Partition创建对应的Task；
	3	将这些Task封装成TaskSet任务集；
	4	将TaskSet任务集提交给TaskScheduler
	Postcondition	None

图 16 任务集生成的用例图

3.4.5 Task 调度

DAGScheduler 完成了 Stage 的划分，以及为 Partition 生成 TaskSet 任务集后，任务的调度操作则是由 TaskScheduler 进行。TaskScheduler 只是 Spark 提供的任务调度的抽象，具体的任务操作可以由 YAEN/Mesos 等第三方资源管理器进行管理。而 Spark 为 TaskScheduler 提供了简单的实现 TaskSchedulerImpl。由该类对 Task 进行调度。TaskSchedulerImpl 调度任务，将任务提交 Executor 执行的用例如图 17 所示。

Use Case Specification	
Use Case Name	Task Scheduler
Brief Description	对具体的Task进行调度
Precondition	DAGScheduler将任务集提交到TaskScheduler
Primary Actor	TaskScdeduler的具体实现对象
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	// 提交任务
	2	TaskSchedulerImpl#submitTasks
	3	// TaskSetManager为Task分配计算资源，监控Task的执行状态并采取必要措施
	4	SchedulerBuilder#addTaskSetManager
	5	// 为每个Task具体分配资源
	6	TaskSchedulerImpl#resourceOffers
	7	// 将tasks发送到Executor
	8	DriverActor#launchTasks
	9	// Executor执行作业
	10	Executor#launchTask
	Postcondition	None

图 17 Task Scheduler 用例图

3.5 任务执行

Executor 模块负责运行 Task 计算任务，并将计算结果回传给 Driver。Spark 支持多种资源调度框架，这些资源框架在为计算任务分配资源后，作为都会使用 Executor 模块完成最终的计算。

每个 Spark 的 Application 都是从 SparkContext 开始，它通过 Cluster Manager 和 Worker 上的 Executor 简历联系，由每个 Executor 完成 Application 的部分计算任务。不同的 Cluster Master，即资源调度框架的实现模式会有区别，但任务的划分和调度都是由运行 SparkContext 端的 Driver 完成的，资源调度框架在为 Application 分配资源后，将 Task 分配到计算的物理单元 Executor 去处理。

在 Standalone 模式下启动集群时，Worker 会向 Master 注册，使得 Master 可以感知进而管理整个集群；Master 通过借助 Zookeeper，可以简单实现高可用性；而应用方通过 SparkContext 完成 Application 的注册，由 Master 为其分配 Executor；

在应用方创建了 RDD 并且在这个 RDD 上进行多次转换后出发 Action，通过 DAGScheduler 将 DAG 划分为不同的 Stage，并将 Stage 转换为 TaskSet 交给 TaskSchedulerImpl；再有 TaskSchedulerImpl 通过 SparkDeploySchedulerBackend 的 reviveOffers，最终向 ExecutorBackend 发送 LaunchTask 的消息；ExecutorBackend 收到消息后，启动 Task 并在集群中启动计算。

Use Case Specification

Use Case Name	创建executor
Brief Description	根据资源分配结果来创建executor
Precondition	Master的资源分配结果
Primary Actor	Driver
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	**Steps**
	1 Driver向集群申请资源
	2 资源分配调度
	3 启动创建executor
	Postcondition None

Use Case Specification

Use Case Name	接收代码和文件
Brief Description	Driver将代码和文件传给executor
Precondition	executor启动完毕
Primary Actor	Driver
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	**Steps**
	1 executor接收代码和文件
	Postcondition None

Use Case Specification	
Use Case Name	计算task任务
Brief Description	进行各种运算完成task任务
Precondition	构建taskSet Manager，然后将task提交给executor运行。
Primary Actor	None
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 将task提交给executor
	2 executor进行运算
	Postcondition None

Use Case Specification	
Use Case Name	结果返回
Brief Description	运行完之后将结果返回给driver
Precondition	executor将task运算完成
Primary Actor	Driver
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 executor运算task完成
	2 结果返回Driver
	Postcondition None

4. 数据需求

暂无。

5. 非功能需求

5.1 运行速度

Spark 创建伊始就是不仅要摆脱 MapReduce 这一单一计算模型，而且也要解决 MpaReduce 计算过程中影响运行速度的问题。Spark 是基于内存的分布式框架，

相比基于硬盘运算要快数十，甚至近百倍。而且采用高效的 DAG 执行引擎，可以高效处理数据流。

5.2 易用性

Spark 支持 Java/Python 和 Scala 的 API,还支持超过 80 种高级算法，使用户可以快速构建不同的应用。

5.3 通用性

Spark 为大数据分析提供统一的解决方案。Spark 可以用于批处理、交互式查询（通过 Spark SQL）、实时流处理（通过 Spark Streaming）、机器学习（通过 Spark MLlib）和图计算（通过 Spark Graphx）。这些不同类型的处理可以在同一个应用中无缝使用。

5.4 容错性

Spark 采用高效的数据编程模型，即弹性分布式数据集 RDD（Resilient Distributed Dataset）。RDD 的容错性表现在，将数据的一系列转换操作形成一个有向无环图，当计算任务失败后，可以快速地从其父 RDD 任务进行转换，而无需重新进行全局计算。还有就是，Spark 还为 RDD 提供了更进一步的容错机制，即检查点。考虑到有些 RDD 的转换操作，计算可能特别负载或者计算耗时特别大，这时可以将 RDD 进行缓存，然后为该 RDD 创建检查点，不仅提高效率，也能进一步容错。

5.5 可融合性

Spark 可以非常方便地与其他的开源产品进行融合。比如，Spark 可以使用 Hadoop 的 YARN 和 Apache Mesos 作为它的资源管理和调度器，并且可以处理所有 Hadoop 支持的数据，包括 HDFS、HBase 和 Cassandra 等。

5.6 安全机制

Spark 支持共享密钥的认证方式。Spark 从以下三个方面保证系统的安全性。

a) Web UI 安全

b) 事件审计安全

c) 网络端口安全

6. 运行需求

6.1 硬件接口

- CPU: 主频至少 2.20GHz
- 内存：2G 以上（官方建议 8G 以上）
- 硬盘：50G 以上
- 网络：10G 以上

6.2 软件接口

- 操作系统：CentOS6/7
- JDK：1.7 版本及以上
- Scala：2.1 版本及以上
- Spark：1.2.1 版本

7. 应用场景

本实验从需求出发，对 Spark 系统存在的普遍问题进行改进完善。当应用场景出现以下情况时，可对根据给出的解决方案，提升 Spark 对应用场景的支持。

(1) 应用场景中出现单条记录消耗过大

Spark 的分布式计算是基于 RDD 上的,其主要的操作是进行 RDD 的转换操作，而最终操作的单元是分区或记录。当应用场景中，对于每条记录的操作比较复杂

时，会出现单条记录上任务执行消耗过大，从而影响任务并行及推进。这时需要将 RDD 的操作单元从单条记录转换为以分区为单元的操作。这就要求用户程序中尽量使用 mapPartition 替代 map 操作，这是由于 mapPartition 是对每个 Partition 进行计算，而 map 是对 partition 中的每条记录进行计算。

(2) 应用场景中出现很多空任务或小任务

一些重复冗余或者无效的 RDD 转换会产生许多空任务或者大量的小任务，这些空任务或小任务会增加系统在任务调度上的开销。所以在用户程序存在重复冗余或者低效的 RDD 转换时，应当适时地使用使用 coalesce 或 repartition 去减少 RDD 中 partition 数量，从而减少空任务或者小任务的数量，减少任务调度的开销。

(3) 应用场景中经常进行 Shuffle 操作而导致 IO 时间过长

有些分布式计算过程中需要 Shuffle 操作，而大量的 Shuffle 操作会增加系统时延。一种可行的方法就使用多个 IO 进行传输，即设置 spark.local.dir 为多个磁盘，并设置磁盘为 IO 速度快的磁盘，通过增加 IO 来优化 shuffle 性能。

(4) 应用场景中出现任务执行速度倾斜

任务执行速度倾斜由两种情况导致。第一种情况时数据倾斜，这一般是由于用户选取的分区关键字不恰当。这时，用户应考虑数据倾斜的情形，并以并行的处理方式，在数据操作中间添加 aggregation 操作。第二种倾斜是 Worker，即工作节点倾斜。工作节点倾斜一般不可控，但是系统可以检测在工作节点上任务执行。要想应对工作节点倾斜，应当将 spark.speculation=true，这样系统对监督工作节点上任务执行的速度，如果任务执行过慢，系统就任务该工作节点发生倾斜（异常），将该工作节点上的任务取消执行，并将任务调度到其他工作节点上运行。

(5) 应用场景中任务数不合理，要么任务开销大，要么任务运行缓慢

需根据实际情况调节默认配置，调整方式是修改参数 spark.default.parallelism。通常，reduce 数目设置为 core 数目的 2 到 3 倍。数量太大，造成很多小任务，增加启动任务的开销；数目太少，任务运行缓慢。

8. 参考文献

1. Spark 官方指导文档翻译：《spark-grogramming-guide-zh-ch.pdf》

2. Spark 官方开发者文档翻译：《spark-developer-guide.pdf》

3. 网络 Spark 运维实战文档：《spark-operation-maintenance-management.pdf》

4. 网络 Apache Spark 设计与实现文档：《SparkInternals.pdf》

5. 张安站 《Spark 技术内幕：深入解析 Spark 内核架构设计与实现原理》，机械工业出版社. 2015

6. 耿嘉安《深入理解 SPARK：核心思想与源码分析》，机械工业出版社. 2016