Spark

Spark 1
1. Spark生态圈 6
1.1. Spark Core：包含Spark的基本功能。 6
1.2. Spark
SQL：提供HiveQL与Spark进行交互的API。每个数据库表被当做一个RDD，Spark SQL查询被转换为Spark操作。 6
1.3. Spark Streaming：对实时数据流进行处理和控制。 6
1.4. MLlib：一个常用机器学习算法库，算法被实现为对RDD的Spark操作。 6
1.5. GraphX：控制图、并行图操作和计算的一组算法和工具的集合。 6
2. RDD（Resilient Distributes Dataset） 弹性分布式数据库 6
2.1. 基本概念 6
2.1.1. RDD，即Resilient Distributes
Dataset（弹性分布式数据集），是spark中最基础、最常用的数据结构。其本质是把input
source进行封装，封装之后的数据结构就是RDD，提供了一系列操作，比如map、flatMap、filter等。input
source种类繁多，比如hdfs上存储的文件、本地存储的文件，相应的RDD的种类也很多，不同的input source对应着不同的RDD类型。 6
2.2. 五个特征 6
2.2.1. （1）一个分片列表：partition list 7
2.2.2. （2）一个计算函数compute，对每一个split（分片）进行计算 7
2.2.3. （3）对其他RDD的依赖列表dependencies
list；依赖又分为宽依赖和窄依赖。（可容错） 7
2.2.4. （4）partitioner for key-value RDDs。比如说hash-partitioned
rdd（这是可选的，不是所有的rdd都有这个特征） 7
2.2.5. （5）对每一个split计算的优先位置Preferred
Location。比如对一个hdfs文件进行计算时，可以获取优先计算的block locations。 7
2.3. RDD的依赖关系 7
2.3.1. 窄依赖（Narrow
Dependencies）：窄依赖是指每个父RDD的一个Partition最多被子RDD的一个Partition所使用，例如map、filter、union等操作都会产生窄依赖。 7
2.3.2. 宽依赖（Wide
Dependencies）：宽依赖是指一个父RDD的partition会被多个子RDD的partition所使用，例如groupByKey、reduceByKey、sortByKey等操作都会产生宽依赖。 7

2.3.3. 依赖关系的特性 7
2.4. RDD的弹性 （指内存不够时可以与磁盘进行交换）.....8
2.4.1. （1）自动的进行内存和磁盘数据存储的切换 8
2.4.2. （2）基于Lineage（血统）的高效容错（第n个节点出错，会从第n-1个节点恢复，血统容错） 8
2.4.3. （3）Task如果失败会自动进行特定次数的重试（默认4次） 8
2.4.4.
（4）Stage如果失败会自动进行特定次数的重试（可以只运行计算失败的阶段），只计算失败的数据分片； 8
2.4.5. （5）check point和persist（check point是比较重量级的操作，RDD操作，一般每次都会产生新的RDD，除了最后一个action操作触发作业以外。但是有时候，链条比较长或者计算比较笨重，考虑把数据放到磁盘上，这就是Checkpoint。Persist是在内存或磁盘里复用。） 8
2.4.6. （6）数据弹性调度：DAG、Task和资源管理无关； 8
2.4.7. （7）数据分片的高度弹性（人工自由设置分片函数），repartition； 8
2.5. RDD的操作.....8
2.5.1. transformation.....8
2.5.2. action 8
2.6. RDD的来源.....9
2.6.1. （1）使用程序中的集合创建RDD（用于小量测试）； 9
2.6.2. （2）使用本地文件系统创建RDD（测试大量数据）； 9
2.6.3. （3）使用HDFS创建RDD； 9
2.6.4. （4）基于DB创建RDD； 9
2.6.5. （5）基于NoSQL创建RDD，例如HBase； 9
2.6.6. （6）基于s3创建RDD； 9
2.6.7. （7）基于数据流创建RDD； 9
2.7. RDD的容错性.....9
2.7.1.
血统容错，每个RDD都包含了他是如何由其它RDD变换过来的以及如何重建莫一块数据的信息。 9
2.7.2.
RDD的Lineage记录的是粗颗粒度的特定数据transformation操作（如filter、map、join等）行为，当这个RDD的部分分区数据丢失时，它可以通过血统机制获取足够的信息来重新运算和恢复丢失的数据分区。 9
3. 宽窄依赖 9
3.1. 概念.....9

3.1.1. 窄依赖（Narrow Dependencies）：窄依赖是指每个父RDD的一个Partition最多被子RDD的一个Partition所使用，例如map、filter、union等操作都会产生窄依赖。 9
3.1.2. 宽依赖（Wide Dependencies）：宽依赖是指一个父RDD的partition会被多个子RDD的partition所使用，例如groupByKey、reduceByKey、sortByKey等操作都会产生宽依赖。 10
3.2. 特性.....10
3.2.1.
（1）窄依赖可以在某个计算节点上直接通过计算父RDD的某块数据计算得到子RDD对应的某块数据；宽依赖则要等到父RDD所有数据都计算完成之后，并且父RDD的计算结果进行hash并传到对应节点之后才能计算子RDD。 .10
3.2.2.
（2）数据丢失时，对于窄依赖只需要重新计算丢失的那一份数据来恢复；对于宽依赖则需要将祖先RDD中的所有数据块全部重新计算来恢复。所以在长血统链特别是有宽依赖的时候，需要在适当的时机设置数据检查点。10
3.2.3.
也是这两个特性要求对于不同依赖关系要采取不同的任务调度机制和容错恢复机制。 10
3.3. 算子：.....10
3.3.1. 窄依赖：map, filter, union, join(父RDD是hash-partitioned), mapPartitions, mapValues.....10
3.3.2. 宽依赖：groupByKey, join(父RDD不是hash-partitioned), partitionBy 10
3.4.
宽依赖会导致shuffle：在分布式计算中，每个阶段的各个计算节点只处理任务的一部分数据，若下一个阶段需要依赖前面阶段的所有计算结果时，则需要对前面阶段的所有计算结果进行重新整合和分类，这就需要经历shuffle的过程。 10
4. DAG.....10
4.1. job 10
4.1.1.
由一个rdd的action触发的动作，需要执行一个rdd的action时会生成一个job。 11
4.2. Stage.....11
4.2.1.
Spark任务会根据RDD之间的依赖关系，形成一个DAG有向无环图，DAG会提交给DAGScheduler，DAGScheduler会把DAG划分相互依赖的多个stage，划分stage的依据就是RDD之间的宽窄依赖。遇到宽依赖就划分stage,每个stage

包含一个或多个task任务。然后将这些task以taskSet的形式提交给TaskScheduler运行。 stage是由一组并行的task组成。 11
4.2.2. stage切割规则：从后往前，遇到宽依赖就切割stage 11
4.2.3. 计算模式：pipeline管道计算模式,pipeline只是一种计算思想，模式。 11
4.3. task 11
4.3.1.
stage下的一个任务执行单元，一般来说，一个rdd有多少个partition就会有多少个task，因为每一个task只是处理一个partition上的数据。 11
5. Shuffle 11
5.1. 概念： 11
5.1.1.
在分布式计算中，每个阶段的各个计算节点只处理任务的一部分数据，若下一个阶段需要依赖前面阶段的所有计算结果时，则需要对前面阶段的所有计算结果进行重新整合和分类，这就需要经历shuffle的过程。 11
5.2. 耗时原因 11
5.2.1.
（1）网络通信：shuffle操作需要将数据进行重新聚合和划分，然后分配到集群的各个节点进行下一个stage操作，这里会涉及集群不同节点间的大量数据交换。 11
5.2.2.
（2）数据读写：不同节点之间的数据通过网络进行传输时，需要先将数据写入磁盘，因此集群中每个节点均有大量的文件读写操作，从而导致shuffle操作十分耗时。 12
5.3. ShuffleManage模式 12
5.3.1. HashShuffleManage 12
5.3.2. SortShuffleManage 12
5.4. shuffle调优 12
5.4.1.
Shuffle阶段需要将数据写入磁盘，这其中涉及大量的读写文件操作和文件传输操作，因此对节点的系统IO有比较大的影响，因此可以通过调整参数减少shuffle阶段的文件数和IO读写次数来提高性能，具体参数主要有以下几个： 12
5.4.2. （1）spark.shuffle.manager
：设置Spark任务的shuffleManage模式，1.2以上版本的默认方式是sort,即shuffle
write阶段会进行排序，每个executor上生成的文件会合并成两个文件（包含一个索引文件）。 12

5.4.3. （2）spark.shuffle.sort.bypassMergeThreshold：设置启用bypass机制的阈值（默认为200），若Shuffle Read阶段的task数小于等于该值，则Shuffle Write阶段启用bypass机制。 12
5.4.4. （3）spark.shuffle.file.buffer （默认32M）：设置Shuffle Write阶段写文件时buffer的大小，若内存比较充足的话，可以将其值调大一些（比如64M），这样能减少executor的IO读写次数。 12
5.4.5. （4）spark.shuffle.io.maxRetries （默认3次）：设置Shuffle Read阶段fetches数据时的重试次数，若shuffle阶段的数据量很大，可以适当调大一些。 13
6. 数据倾斜 13
6.1. 概念 13
6.1.1. 并行处理数据集中，某一部分（如Spark中的一个partition）的数据显著多于其他部分，从而使得该部分的处理速度成为整个数据集处理的瓶颈。13
6.2. 原因及解决办法 13
6.2.1. 1.key本身分布不均匀 13
6.2.2. 2.key的设置不合理 13
6.2.3. 3.shuffle时的并发度不够 13
6.2.4. 4.计算方式有误 13

Spark是基于RDD，提供了一站式多维度的大数据计算框架。可以在一个技术栈里快速对数据进行批处理，即席查询、机器学习、图计算和准实时流处理。

1. Spark生态圈

1.1. Spark Core：包含Spark的基本功能。

1.2. Spark

SQL：提供HiveQL与Spark进行交互的API。每个数据库表被当做一个RDD，Spark SQL查询被转换为Spark操作。

1.3. Spark Streaming：对实时数据流进行处理和控制。

1.4. MLlib：一个常用机器学习算法库，算法被实现为对RDD的Spark操作。

1.5. GraphX：控制图、并行图操作和计算的一组算法和工具的集合。

2. RDD（Resilient Distributes Dataset）

弹性分布式数据库

2.1. 基本概念

2.1.1. RDD，即Resilient Distributes

Dataset（弹性分布式数据集），是spark中最基础、最常用的数据结构。其本质是把input

source进行封装，封装之后的数据结构就是RDD，提供了一系列操作，比如map、flatMap、filter等。input

source种类繁多，比如hdfs上存储的文件、本地存储的文件，相应的RDD的种类也很多，不同的input source对应着不同的RDD类型。

2.2. 五个特征

2.2.1. （1）一个分片列表：partition list

2.2.2. （2）一个计算函数compute，对每一个split（分片）进行计算

2.2.3. （3）对其他RDD的依赖列表dependencies list；依赖又分为宽依赖和窄依赖。（可容错）

2.2.4. （4）partitioner for key-value RDDs。比如说hash-partitioned rdd（这是可选的，不是所有的rdd都有这个特征）

2.2.5. （5）对每一个split计算的优先位置Preferred Location。比如对一个hdfs文件进行计算时，可以获取优先计算的block locations。

2.3. RDD的依赖关系

2.3.1. 窄依赖（Narrow Dependencies）：

窄依赖是指每个父RDD的一个Partition最多被子RDD的一个Partition所使用，例如map、filter、union等操作都会产生窄依赖。

2.3.2. 宽依赖（Wide Dependencies）：

宽依赖是指一个父RDD的partition会被多个子RDD的partition所使用，例如groupByKey、reduceByKey、sortByKey等操作都会产生宽依赖。

2.3.3. 依赖关系的特性

（1）窄依赖可以在某个计算节点上直接通过计算父RDD的某块数据计算得到子RDD对应的某块数据；宽依赖则要等到父RDD所有数据都计算完成之后，并且父RDD的计算结果进行hash并传到对应节点之后才能计算子RDD。

（2）数据丢失时，对于窄依赖只需要重新计算丢失的那一份数据来恢复；对于宽依赖则需要将祖先RDD中的所有数据块全部重新计算来恢复。所以在

长血统链特别是有宽依赖的时候，需要在适当的时机设置数据检查点。也是这两个特性要求对于不同依赖关系要采取不同的任务调度机制和容错恢复机制。

2.4. RDD的弹性

（指内存不够时可以与磁盘进行交换）

2.4.1. （1）自动的进行内存和磁盘数据存储的切换

2.4.2. （2）基于Lineage（血统）的高效容错（第n个节点出错，会从第n-1个节点恢复，血统容错）

2.4.3. （3）Task如果失败会自动进行特定次数的重试（默认4次）

2.4.4. （4）Stage如果失败会自动进行特定次数的重试（可以只运行计算失败的阶段）；只计算失败的数据分片；

2.4.5. （5）check point和persist（check point是比较重量级的操作，RDD操作，一般每次都会产生新的RDD，除了最后一个action操作触发作业以外。但是有时候，链条比较长或者计算比较笨重，考虑把数据放到磁盘上，这就是Checkpoint。Persist是在内存或磁盘里复用。）

2.4.6. （6）数据弹性调度：DAG、Task和资源管理无关；

2.4.7. （7）数据分片的高度弹性（人工自由设置分片函数），repartition；

2.5. RDD的操作

2.5.1. transformation

惰性执行，需要具体的action触发才会去执行

2.5.2. action

每个action触发都会提交一个job

2.6. RDD的来源

2.6.1. （1）使用程序中的集合创建RDD（用于小量测试）；

2.6.2. （2）使用本地文件系统创建RDD（测试大量数据）；

2.6.3. （3）使用HDFS创建RDD；

2.6.4. （4）基于DB创建RDD；

2.6.5. （5）基于NoSQL创建RDD，例如HBase；

2.6.6. （6）基于s3创建RDD；

2.6.7. （7）基于数据流创建RDD；

2.7. RDD的容错性

2.7.1. 血统容错，每个RDD都包含了他是如何由其它RDD变换过来的以及如何重建莫一块数据的信息。

2.7.2. RDD的Lineage记录的是粗颗粒度的特定数据transformation操作（如filter、map、join等）行为，当这个RDD的部分分区数据丢失时，它可以通过血统机制获取足够的信息来重新运算和恢复丢失的数据分区。

3. 宽窄依赖

3.1. 概念

3.1.1. 窄依赖（Narrow Dependencies）：窄依赖是指每个父RDD的一个Partition最多被子RDD的一个Partition所使用，例如map、filter、union等操作都会产生窄依赖。

3.1.2. 宽依赖（Wide Dependencies）：宽依赖是指一个父RDD的partition会被多个子RDD的partition所使用，例如groupByKey、reduceByKey、sortByKey等操作都会产生宽依赖。

3.2. 特性

3.2.1. （1）窄依赖可以在某个计算节点上直接通过计算父RDD的某块数据计算得到子RDD对应的某块数据；宽依赖则要等到父RDD所有数据都计算完成之后，并且父RDD的计算结果进行hash并传到对应节点之后才能计算子RDD。

3.2.2. （2）数据丢失时，对于窄依赖只需要重新计算丢失的那一份数据来恢复；对于宽依赖则需要将祖先RDD中的所有数据块全部重新计算来恢复。所以在长血统链特别是有宽依赖的时候，需要在适当的时机设置数据检查点。

3.2.3. 也是这两个特性要求对于不同依赖关系要采取不同的任务调度机制和容错恢复机制。

3.3. 算子：

3.3.1. 窄依赖：map, filter, union, join(父RDD是hash-partitioned), mapPartitions, mapValues

3.3.2. 宽依赖：groupByKey, join(父RDD不是hash-partitioned), partitionBy

3.4. 宽依赖会导致shuffle：在分布式计算中，每个阶段的各个计算节点只处理任务的一部分数据，若下一个阶段需要依赖前面阶段的所有计算结果时，则需要对前面阶段的所有计算结果进行重新整合和分类，这就需要经历shuffle的过程。

4. DAG

4.1. job

4.1.1. 由一个rdd的action触发的动作，需要执行一个rdd的action时会生成一个job。

4.2. Stage

4.2.1. Spark任务会根据RDD之间的依赖关系，形成一个DAG有向无环图，DAG会提交给DAGScheduler，DAGScheduler会把DAG划分相互依赖的多个stage，划分stage的依据就是RDD之间的宽窄依赖。遇到宽依赖就划分stage,每个stage包含一个或多个task任务。然后将这些task以taskSet的形式提交给TaskScheduler运行。 stage是由一组并行的task组成。

4.2.2. stage切割规则：从后往前，遇到宽依赖就切割stage

4.2.3. 计算模式：pipeline管道计算模式,pipeline只是一种计算思想，模式。

4.3. task

4.3.1. stage下的一个任务执行单元，一般来说，一个rdd有多少个partition就会有多少个task，因为每一个task只是处理一个partition上的数据。

5. Shuffle

5.1. 概念：

5.1.1. 在分布式计算中，每个阶段的各个计算节点只处理任务的一部分数据，若下一个阶段需要依赖前面阶段的所有计算结果时，则需要对前面阶段的所有计算结果进行重新整合和分类，这就需要经历shuffle的过程。

5.2. 耗时原因

5.2.1. （1）网络通信：shuffle操作需要将数据进行重新聚合和划分，然后分配到集群的各个节点进行下一个stage操作，这里会涉及集群不同节点间的大量数据交换。

5.2.2.（2）数据读写：不同节点之间的数据通过网络进行传输时，需要先将数据写入磁盘，因此集群中每个节点均有大量的文件读写操作，从而导致shuffle操作十分耗时。

5.3. ShuffleManage模式

5.3.1. HashShuffleManage

5.3.2. SortShuffleManage

普通模式

bypass模式

5.4. shuffle调优

5.4.1. Shuffle阶段需要将数据写入磁盘，这其中涉及大量的读写文件操作和文件传输操作，因此对节点的系统IO有比较大的影响，因此可以通过调整参数减少shuffle阶段的文件数和IO读写次数来提高性能，具体参数主要有以下几个：

5.4.2.（1）spark.shuffle.manager
：设置Spark任务的shuffleManage模式，1.2以上版本的默认方式是sort,即shuffle
write阶段会进行排序，每个executor上生成的文件会合并成两个文件（包含一个索引文件）。

5.4.3.（2）spark.shuffle.sort.bypassMergeThreshold
：设置启用bypass机制的阈值（默认为200），若Shuffle
Read阶段的task数小于等于该值，则Shuffle Write阶段启用bypass机制。

5.4.4.（3）spark.shuffle.file.buffer （默认32M）：设置Shuffle
Write阶段写文件时buffer的大小，若内存比较充足的话，可以将其值调大一些（比如64M），这样能减少executor的IO读写次数。

5.4.5. （4）spark.shuffle.io.maxRetries （默认3次）：设置Shuffle Read阶段fetches数据时的重试次数，若shuffle阶段的数据量很大，可以适当调大一些。

6. 数据倾斜

6.1. 概念

6.1.1. 并行处理数据集中，某一部分（如Spark中的一个partition）的数据显著多于其他部分，从而使得该部分的处理速度成为整个数据集处理的瓶颈。

6.2. 原因及解决办法

https://www.jianshu.com/p/06b67a3c61a9

6.2.1. 1.key本身分布不均匀

6.2.2. 2.key的设置不合理

6.2.3. 3.shuffle时的并发度不够

6.2.4. 4.计算方式有误