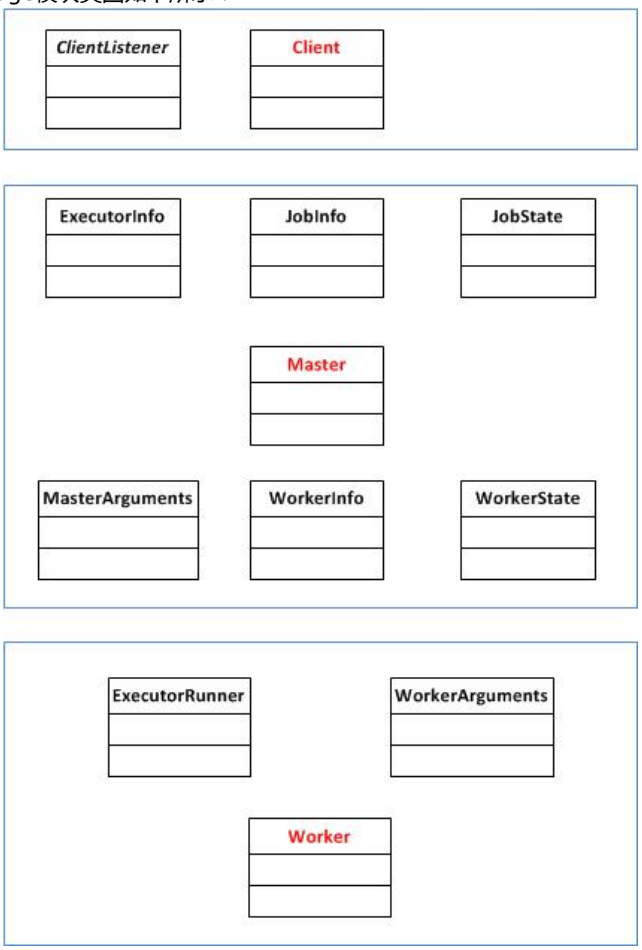
# 【转】Spark源码分析之-Storage模块

**原文地址：http://blog.csdn.net/aiuyjerry/article/details/8595991**

Storage模块主要负责数据存取，包括MapReduce Shuffle中间结果、MapReduce task中间stage结果、cache结果。下面从架构和源码细节上来分析Storage模块的实现。Storage模块主要由两大部分组成：

- BlockManager部分主要负责Master和Slave之间的block通信，主要包括BlockManager状态上报、心跳，add, remove, update block.
- BlockStore部分主要负责数据存取，Spark根据不同选择可以在Memory或(和)Disk中存储序列化数据.

Storage模块类图如下所示：




- SparkEnv创建时会实例化BlockManagerMaster对象和BlockManager对象。
- BlockManagerMaster对象会根据自己是master还是slave来创建BlockManagerMasterActor或是连接到BlockManagerMasterActor。
- BlockManager承担两种角色：
- 1. 负责向BlockManagerMaster上报block信息，保持心跳和接收block信息

昵称：vincent_hv
园龄：10个月
粉丝：7
关注：1
+加关注

< 2013年10月

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 29 | 30 | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | 31 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |

## 搜索

找找看
谷歌搜索

## 常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

## 最新随笔

1. linux解压zip乱码解决方案
2. 全能系统监控工具dstat
3. 【转】linux sar命令详解
4. 【原】gnome3增加自定义程序快捷方式
5. 【原】Ubuntu13.04安装、卸载Gnome3.8
6. 【原】安装、卸载、查看软件时常用的命令
7. 【原】中文Ubuntu主目录下的文档文夹改回英文
8. 【原】Ubuntu ATI/Intel双显卡 驱动装
9. 【原】Ubuntu 12.04 ATI显卡设置双屏显示
10. 【转】Hadoop vs Spark性能对比

## 随笔分类

Android(8)
Hadoop(2)
Java(20)
JVM(3)
Linux(23)
others(1)
Scala(5)
Spark(20)
数据结构与算法(2)

2. 负责通过BlockStore从Memory或Disk读取、写入block数据

- BlockManagerMessages封装与master传输的meta信息的具体格式。
- Slave通过BlockManager向BlockManagerMaster注册自己，在注册自己时会创建BlockManagerSlaveActor，用来Master向Slave通信，目前唯一request是请求Slave删除block。
- BlockManagerWorker则负责Slave之间的通信，包括get, put非本地的block
- BlockMessage类封装了与Master通信的block message的具体格式，而BlockMessageArray则是批处理接口。
- BlockStore提供持久化数据的接口，DiskStore和MemoryStore实例化了BlockStore接口，实现serialize, deserialize数据到Disk或Memory。

Spark Storage模块master和slave之间通信的信息包括：

- Slave --------> Master
  - RegisterBlockManager
  - HeartBeat
  - UpdateBlockInfo
  - GetLocations
  - GetLocationsMutipleBlockIds
  - GetPeers
  - RemoveExecutor
  - StopBlockManagerMaster
  - GetMemoryStatus
  - ExpireDeadHosts
  - GetStorageStatus
- Master ---------> Slave
  - RemoveBlock

Storage模块存取数据分析

MemoryStore：

Memory内部使用LinkedHashMap来作为block的存储结构，其中key是block id，value是Entry类，代码如所示:

```
  case class Entry(value: Any, size: Long, deserialized: Boolean, var dropPending: Boolean = false)
  private val entries = new LinkedHashMap[String, Entry](32, 0.75f, true)
```

而内部存储会调用如下代码：

```
  private def tryToPut(blockId: String, value: Any, size: Long, deserialized: Boolean): Boolean = {
    putLock.synchronized {
      if (ensureFreeSpace(blockId, size)) {
        val entry = new Entry(value, size, deserialized)
        entries.synchronized { entries.put(blockId, entry) }
        currentMemory += size
        if (deserialized) {
          logInfo("Block %s stored as values to memory (estimated size %s, free %s)".format(
            blockId, Utils.memoryBytesToString(size), Utils.memoryBytesToString(freeMemory)))
        } else {
```

```
          logInfo("Block %s stored as bytes to memory (size %s, free %s)".format(
            blockId, Utils.memoryBytesToString(size), Utils.memoryBytesToString(freeMemor
y)))
        }
        true
      } else {
        // Tell the block manager that we couldn't put it in memory so that it can drop i
t to
        // disk if the block allows disk storage.
        val data = if (deserialized) {
          Left(value.asInstanceOf[ArrayBuffer[Any]])
        } else {
          Right(value.asInstanceOf[ByteBuffer].duplicate())
        }
        blockManager.dropFromMemory(blockId, data)
        false
      }
    }
  }

  private def ensureFreeSpace(blockIdToAdd: String, space: Long): Boolean = {
    logInfo("ensureFreeSpace(%d) called with curMem=%d, maxMem=%d".format(
      space, currentMemory, maxMemory))

    if (space > maxMemory) {
      logInfo("Will not store " + blockIdToAdd + " as it is larger than our memory limit"
)
      return false
    }

    if (maxMemory - currentMemory < space) {
      val rddToAdd = getRddId(blockIdToAdd)
      val selectedBlocks = new ArrayBuffer[String]()
      var selectedMemory = 0L

      entries.synchronized {
        val iterator = entries.entrySet().iterator()
        while (maxMemory - (currentMemory - selectedMemory) < space && iterator.hasNext)
{
          val pair = iterator.next()
          val blockId = pair.getKey
          if (rddToAdd != null && rddToAdd == getRddId(blockId)) {
            logInfo("Will not store " + blockIdToAdd + " as it would require dropping ano
ther " +
              "block from the same RDD")
            return false
          }
          selectedBlocks += blockId
          selectedMemory += pair.getValue.size
        }
      }

      if (maxMemory - (currentMemory - selectedMemory) >= space) {
        logInfo(selectedBlocks.size + " blocks selected for dropping")
        for (blockId <- selectedBlocks) {
          val entry = entries.synchronized { entries.get(blockId) }
          // This should never be null as only one thread should be dropping
          // blocks and removing entries. However the check is still here for
          // future safety.
          if (entry != null) {
            val data = if (entry.deserialized) {
              Left(entry.value.asInstanceOf[ArrayBuffer[Any]])
            } else {
              Right(entry.value.asInstanceOf[ByteBuffer].duplicate())
            }
            blockManager.dropFromMemory(blockId, data)
          }
        }
        return true
      } else {
        return false
      }
    }
    return true
  }
```

tryToPut会调用ensureFreeSpace来淘汰掉一些block，为此block的存储释放新的空间，而tryToPut会将其添加到LinkedHashMap中。如果ensureFreeSpace无法获得足够的空间去存储此block，tryToPut会调用dropFreeMemory来drop此block。

DiskStore：

Spark会根据配置项spark.local.dir在本地建立目录，所有的block都会依照不同路径存储到此目录下，当spark.local.dir中配置了多个path时，Spark会根据hash将block存储到不同的path下

- 首先，Spark会根据spark.local.dir的配置在所有配置目录下建立localDir，localDir命名为spark-local-%s-%04x,其中%s是格式化后的当前时间(yyyyMMddHHmmss)，%d是一个小于65535的随机16进制数字。
- 其次，每当要存储block时，Spark会根据blockId在localDir下建立子目录和相应的文件，block存储目录的选择规律是：
- 1. 根据blockId的hash值计算出dirId和subDirId
  2. 取出或创建subDir
  3. 在subDir下面以blockId为名字创建文件

```
    val subDirsPerLocalDir = System.getProperty("spark.diskStore.subDirectories", "64").t
oInt
    val subDirs = Array.fill(localDirs.length)(new Array[File](subDirsPerLocalDir))

    // Figure out which local directory it hashes to, and which subdirectory in that
    val hash = math.abs(blockId.hashCode)
    val dirId = hash % localDirs.length
    val subDirId = (hash / localDirs.length) % subDirsPerLocalDir

    // Create the subdirectory if it doesn't already exist
    var subDir = subDirs(dirId)(subDirId)
    if (subDir == null) {
      subDir = subDirs(dirId).synchronized {
        val old = subDirs(dirId)(subDirId)
        if (old != null) {
          old
        } else {
          val newDir = new File(localDirs(dirId), "%02x".format(subDirId))
          newDir.mkdir()
          subDirs(dirId)(subDirId) = newDir
          newDir
        }
      }
    }

    new File(subDir, blockId)
```

- 最后，根据压缩和序列化方式选择将block存储到文件中

分类: Spark





« 上一篇：【转】弹性分布式数据集：一种基于内存的集群计算的容错性抽象方法

» 下一篇：【转】Spark源码分析之-deploy模块

posted @ 2013-09-23 13:35 vincent_hv 阅读(17) 评论(0) 编辑 收藏