# Spark 内存管理及缓存策略研究

孟红涛 余松平 刘 芳 肖 侬
(国防科学技术大学计算机学院 长沙 410072)

**摘 要** Spark 系统是基于 Map-Reduce 模型的大数据处理框架。Spark 能够充分利用集群的内存,从而加快数据的处理速度。Spark 按照功能把内存分成不同的区域:Shuffle Memory 和 Storage Memory,Unroll Memory,不同的区域有不同的使用特点。首先,测试并分析了 Shuffle Memory 和 Storage Memory 的使用特点。RDD 是 Spark 系统最重要的抽象,能够缓存在集群的内存中;在内存不足时,需要淘汰部分 RDD 分区。接着,提出了一种新的 RDD 分布式权值缓存策略,通过 RDD 分区的存储时间、大小、使用次数等来分析 RDD 分区的权值,并根据 RDD 的分布式特征对需要淘汰的 RDD 分区进行选择。最后,测试和分析了多种缓存策略的性能。

**关键词** 大数据,Spark 内存管理,RDD 缓存,缓存策略
**中图法分类号** TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.06.005

## Research on Memory Management and Cache Replacement Policies in Spark

MENG Hong-tao YU Song-ping LIU Fang XIAO Nong
(School of Computer, National University of Defense Technology, Changsha 410072, China)

**Abstract** Spark is a big data processing framework based on Map-Reduce. Spark can make full use of cluster memory, thus accelerating data processing. Spark divides memory into Shuffle Memory, Storage Memory and Unroll Memory according to their functions. These different memory zones have different characteristics. The features of Shuffle Memory and Storage Memory were tested and analyzed. RDD (Resilient Distributed Datasets) is the most important abstract in spark, which can cache in cluster memory. When the cluster memory is insufficient, Spark must select some RDD partitions to discard to make room for the new ones. A new cache replacement policies called DWRP (Distributed Weight Replacement Policy) was proposed. DWRP can compute the weight of every RDD partition based on the time of store in memory, size and frequency of use, and then select possible RDD partition to discard based on distribution features. The performance of different cache replacement policies was tested and analyzed at last.

**Keywords** Big data, Spark memory management, RDD cache, Cache replacement policies

大数据时代的到来使得数据的单机存储与处理变得越来越困难,很多并行存储与处理模型应运而生。Hadoop 是一种典型的大数据处理系统,使用 Map-Reduce 模型处理数据,通过 HDFS(Hadoop File System)来存储数据。Hadoop 可以运行在廉价的普通集群上,能够并行处理大量的数据,擅长批量处理和离线分析。Hadoop 在 Map-Reduce 模型下处理数据时,中间结果会写入 HDFS,从而引入磁盘 I/O 延迟。基于内存的分布式数据处理系统 Apache Spark,一方面充分利用了集群的内存:Spark 将计算过程中常用的数据集合缓存在集群的内存中,在迭代型机器学习、查询业务等数据复用率高的应用中能够大大加快整个任务的执行速度;另一方面实现了有向无环图 DAG(Directed Acyclic Graph),记录 RDD 之间的依赖关系,根据依赖关系快速恢复丢失的 RDD 分区。Spark 同时兼容 Hadoop,处理速度比 Hadoop 快 10～100 倍;并且,其还提供 Java,Scala,Python 和 R 的 API,更加易用。Spark 有结构化查询库 Spark SQL、流处理库 Spark Streaming、机器学习库 Spark MLlib 和图计算库 GraphX,这些处理能够在同一应用中做到无缝连接,通用性好,也很容易与现有的系统进行融合[1]。

集群内存的使用是研究的热点问题。Daniel Warneke 等[2]提出了一种数据中心的动态内存分配机制,使单个应用在占有已申请获得的内存的同时,可以再申请空闲内存,直到这些空闲内存再被其他应用申请。这种动态分配内存机制适合集群内同时运行多个应用的情况。而 Spark 集群中,内存使用问题主要集中于一个应用在分布式执行过程中的分配和管理。Li 等[3]提出一种新的分布式内存文件系统 Tachyon,

到稿日期:2016-11-11 返修日期:2017-01-02 本文受 863 计划"面向大数据的内存计算关键技术与系统"子课题"基于内存计算的并行处理系统与研究"资助。

孟红涛(1990－),男,硕士,主要研究方向为大数据系统、分布式系统,E-mail:mht0228@163.com;余松平 博士生,主要研究方向为分布计算、大数据、新型存储,E-mail:we.isly@163.com;刘 芳 副教授,硕士生导师,主要研究方向为网络存储、新型固态存储、移动计算;肖 侬 教授,博士生导师,主要研究方向为云计算与网络计算、大数据存储与处理、计算机系统结构。

Tachyon 能够充分发挥分布式集群的内存特性，其本质是一种文件系统，能够为 Spark 提供存储服务，没有针对性优化 Spark 运行过程中的内存管理。Ananthanarayanan 等[4]的 Pacman 集群内存分布式淘汰策略充分利用了集群内作业的分布式特征，能够把分布式作业的执行时间减少 50%左右。Pacman 是一种分布式作业内存使用的通用策略，没有对 Spark 的 RDD 缓存进行针对性优化。Duan 等[5]考虑 RDD 的特征，提出了一种 RDD 分区权值淘汰策略，在内存不足时，能够减少 Spark 应用的运行时间，但是他们没有考虑到 RDD 的分布式特征。Feng 等[6]实现了一种 RDD 自动缓存策略，在内存不足时，计算 RDD 分区权值，淘汰权值较低的 RDD 分区。这种策略能够提高 Spark 应用的性能，但是需要额外运行一次 Spark 应用以得到该应用运行时的信息，在正式运行 Spark 应用时再利用这些信息来针对性地优化内存的管理。

本文分析了 Spark 对集群内存的管理，提出了一种新的 RDD 缓存策略——分布式权值缓存策略，测试并分析了 Spark 对内存的使用特点和 Spark 应用在多种 RDD 缓存策略下的性能。本文第 1 节介绍了 Spark 对内存的管理机制；第 2 节通过不同的测试来进一步分析 Spark 的内存使用特征；第 3 节提出了一种新的 RDD 分布式权值缓存策略，并进行了初步的模拟实现和测试。

## 1 Spark 内存使用分析

### 1.1 RDD 介绍

Spark 提出了一种新的抽象弹性分布式数据集(Resilient Distributed Datasets，RDD)。RDD 是一个只读的、分区的记录的集合[7]。RDD 有两个特点：1) RDD 能够记录其本身是怎样通过其他数据集产生或者转换的，即 RDD 能够记录其“血统”信息。在某个 RDD 分区丢失时，能够根据“血统”信息恢复该分区，而不必重新计算所有分区。2) Spark 通过把计算过程中常用的 RDD 以分区的形式缓存在集群的内存中，在使用 RDD 时，首先在内存中查找该 RDD 是否有被缓存，如果有，则直接读取使用；如果没有，才需要重新计算。RDD 的这两个特点能够提升 Spark 应用的性能。

创建 RDD 的有方式两种：1) 从存储的数据源直接创建；2) 从其他 RDD 转换而来[8]。Spark 提供两种针对 RDD 的操作，一种是 Transformation 转换操作，一个 Transformation 转换操作会定义一个新的 RDD，Transformation 转换操作并不立即计算该 RDD 的实际数据；另一种是 Action 行动操作，Action 行动操作发生后，会提交本身的 Action 行动操作和所有的 Transformation 转换操作，开始实际的计算。Action 行动操作后会产生 Spark 程序的返回值或把 Spark 程序的结果写入到外存。

如图 1 所示，每次 Transformation 转换操作都会定义一个新的 RDD。原始数据定义 $RDD_0$，然后经过一个 Transformation 转换操作定义 $RDD_1$，最后一个 Transformation 转换操作定义新的 $RDD_N$，再经过一个 Action 行动操作提交并开始真正计算之前定义的所有 RDD，产生 Spark 程序的结果。


图 1 RDD 的 Transformation 和 Action 操作

### 1.2 Spark 内存管理

内存是 Spark 集群中非常重要的资源。在 Spark 集群中，内存的具体管理由集群中的节点负责，每一个节点按照不同的功能对其内存进行不同的管理。

Spark 作业由集群中每一个节点上的 Executor 负责具体执行。为了尽量避免内存溢出，Spark 通常只使用 Java Heap (Java 堆空间)的 90%作为 Safe Memory(安全内存)，这些内存就是 Executor 中可以直接使用的部分。Spark 把 Safe Memory(安全内存)空间按照功能在逻辑上划分成 3 个专门的区域(见图 2)：1) Shuffle Memory(Shuffle 内存)，它是 Shuffle 阶段数据的临时存储空间；2) Storage Memory(存储内存)，它是 RDD 分区缓存所使用的内存空间；3) Unroll Memory(展开内存)，它是 Storage Memory 中的一部分，专门用于 RDD 分区序列化和反序列化[9]。

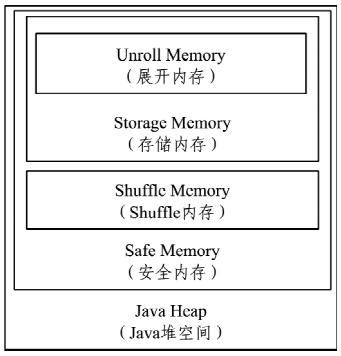


图 2 内存区域的划分

(1) Shuffle Memory

默认情况下，Shuffle Memory 占 Java 堆空间的 20%。Shuffle Memory 主要是在 Map-Reduce 的中间阶段由 Shuffle 使用。Map 阶段对数据进行分割，Reduce 阶段对所有节点上经过 Map 后的符合要求的数据进行合并。系统执行排序的过程(即将 Map 输出作为输入传给 Reduce)称为 Shuffle[10]。在 Reduce 阶段，首先会拉取相应的数据存储在 Shuffle Memory 中，当拉取完成后，按照 Key 值对这些数据进行排序。Shuffle Memory 暂时存储拉取的数据，拉取完成后，再在 Shuffle Memory 中对这些数据进行排序。在拉取过程中，如果 Shuffle Memory 的空间不够，需要把部分数据写入磁盘。当 Reduce 阶段完成后，释放其占有的 Shuffle Memory 空间。

(2) Storage Memory

Spark 分配给 Storage Memory 空间的大小为：

$$C_{Sto\_M}=C_{SM}\times memoryFraction \quad (1)$$

其中，$C_{Sto\_M}$ 代表 Storage Memory 的空间大小，$C_{SM}$ 代表 Safe Memory 的空间大小，$memoryFraction$ 指 Storage Memory 占 Safe Memory 空间的比例。$memoryFraction$ 是 Spark 集群的参数，可以在搭建集群时配置。以 Spark1.4.0 版本为例，其默认值为 0.6，因此 Storage Memory 空间的大小为：

$$C_{Sto\_M}=C_{SM}\times 0.6 \quad (2)$$

Storage Memory 用来存储缓存的 RDD 分区。用户通过调用 RDD 的 persist()和 cache()操作来缓存该 RDD。RDD 会以分区的形式缓存在集群节点 Storage Memory 中。在 Spark 作业执行过程中，需要用到某个 RDD 时，首先在Stora-

ge Memory中查找，如果存在，则直接使用；如果中不存在，则需要按照该 RDD 的依赖关系重新计算。

(3)Unroll Memory

RDD 分区可以以序列化和反序列化后的两种格式存储在内存中。序列化后可以减少其所占有的内存空间，便于网络传输，但是序列化后不能被直接访问和使用。当使用一个序列化后的 RDD 分区时，需要进行反序列化，Unroll Memory 就是进行反序列化时所需要的临时内存空间。

Spark 分配给 Unroll Memory 空间的大小 $C_{UM}$：

$$C_{UM}=C_{Sto\_M}\times unrollFraction \tag{3}$$

其中，$C_{UM}$ 代表 Unroll Memory 空间的大小，$C_{Sto\_M}$ 代表Storage Memory 空间的大小，$unrollFraction$ 指 Unroll Memory 占 Storage Memory 空间的比例。$unrollFraction$ 是 Spark 集群的参数，可以在搭建集群时配置。以 Spark1.4.0 为例，其默认值为 0.2，因此 Unroll Memory 空间大小默认为：

$$C_{UM}=C_{Sto\_M}\times 0.2 \tag{4}$$

当 Unroll Memory 空间不足时，可以临时借用 Storage Memory 空间。序列化和反序列化完成后，释放临时占用的空间。

### 1.3　Spark 缓存机制

Spark 是一种基于内存的分布式计算框架。内存是 Spark 集群中非常重要的资源，内存的有效利用是高效执行 Spark 应用的关键之一。因此 Spark 的缓存机制是 Spark 系统管理和使用内存资源的一个重要方面。

Spark 应用在执行过程中需要用到某个 RDD 时，需首先在集群的内存中查找该 RDD 是否已经被缓存，否则直接读取；如果未被缓存，则需要重新计算。重新计算时，按照其依赖关系在集群的内存中查找其依赖的父 RDD，如果集群内存中已缓存，直接使用父 RDD 计算得到该 RDD；否则继续逆推查找和计算。

Spark 程序员可以主动选择将哪些 RDD 缓存在内存中。这些已经缓存的 RDD 被再次使用时，可以直接从内存中读取，不用再次计算。当缓存的 RDD 复用率较高时，Spark 应用的整体性能得到了很大的提升。

RDD 分区在集群中的缓存如图 3 所示，$RDD_B$ 是由 $RDD_A$ 经过 Transformation 操作转换得到的，$RDD_B$ 在接下来的运行中有可能被再次使用，所以 Spark 通过 cache()操作主动缓存 $RDD_B$。假设 $RDD_B$ 包含 3 个分区 P1′，P2′和 P3′，集群中有 3 个节点，缓存时 P1′，P2′和 P3′分别存储在节点 1、节点 2 以及节点 3 的 Storage Memory 中。

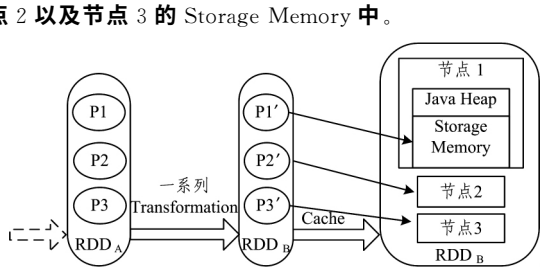


图 3　RDD 在集群中的缓存

随着 Spark 应用的执行，节点的 Storage Memory 中存储了很多个 RDD 的分区。Storage Memory 空间有限，不可能存储所有需要缓存的 RDD 分区。当出现新的 RDD 缓存需求而 Storage Memory 空间已满时，就需要淘汰已存储在该节点 Storage Memory 中的某些 RDD 分区。进行淘汰时，每个节点的 Storage Memory 管理本节点内的 RDD 分区。Spark 目前采取的是 LRU(Least Recently Used)策略，出现新的 RDD 缓存需求时，需要存储该 RDD 某些分区的 Storage Memory，并按照 LRU 策略淘汰符合条件的 RDD 分区。

## 2　Spark 内存使用测试与分析

内存是集群的重要资源，对内存的有效利用可以大大加快 Spark 应用的执行速度。本文对 Spark 内存中的 Shuffle Memory 和 Storage Memory 的使用特点进行了测试和分析。使用中科院大数据测试 BigDataBench[11] 中的 Sort 和 PageRank进行测试。测试环境和使用的数据集如表 1 所列。

表 1　Spark 内存使用测试实验环境和数据集

| 硬件环境 | | 软件环境 | | 数据集 | |
|---|---|---|---|---|---|
| 集群内 3 个节点 | | 操作系统 | ubuntu14.04LTS | PageRank | cit-Patent |
| 处理器 | Intel® Xeon (R) CPUE5-2620v3 @ 2.40 GHz * 12 | Spark | 1.4.0 | | web-Google |
| 内存 | 54.9GiB | Java | 1.7.0_79 | Sort | BigDataBench 自动产生 |
| 操作系统类型 | 64-bit | Scala | 2.10.4 | | |

### 2.1　Shuffle Memory 使用特点测试

在 Shuffle 阶段，Reduce 任务把数据拉取到本节点的内存中时需要一部分内存存储这些数据，在拉取完成后，进行聚合操作及排序，在这个过程中使用的内存就是 Shuffle Memory。Reduce 任务拉取上一个 Map 阶段的输出，拉取来的数据首先存储到 Shuffle Memory 中，当 Shuffle Memory 空间不足时，会将拉取到的数据写入本地磁盘，并释放其占有的内存空间以供 Reduce 任务继续拉取数据，这种把拉取到的数据写入本地磁盘的操作称为 Spill 操作。当 Spill 操作发生后，相应的 Shuffle Memory 空间将被释放。接下来，Reduce 任务继续拉取数据、排序、Spill、释放空间。Spill 到本地磁盘的数据以数据块的形式存储在本地磁盘上，当拉取完数据时，最后通过归并排序算法把所有 Spill 到本地磁盘的数据合并为一个大的文件，并将其作为 Reduce 阶段的输出。

Spill 操作会产生磁盘 I/O，Spill 之后的归并也会产生磁盘 I/O。如果 Shuffle Memory 空间充足，则不会引发 Spill 操作，也不会导致额外的磁盘 I/O。当 Shuffle Memory 空间不足时，Spill 操作产生磁盘 I/O，从而影响 Shuffle 的执行效率。

排序应用 Sort 在 Spark 中的具体处理流程如下：1)对所有源数据进行 map 操作，将其转换为(Key，Value)形式；2)再进行 Key 值排序操作，按照字典序对所有的 Key 值对数据进行排序；3)对按照 Key 排序好的(Key，Value)数据对进行 map 操作，将其转换为仅仅包含 Key 的数据。最后存储结果。

在 Key 值排序时产生了 Shuffle 操作，map 后的数据拉取到 reduce 端后，如果 Shuffle Memory 空间不足，则会产生

Spill。如图 4 所示，当 Shuffle Memory 空间逐渐增大时（图 4 中横轴为 Shuffle Memory 占 Safe Memory 的比例），Spill 到磁盘的总数据量变少，Spill 的次数变少；当 Shuffle Memory 空间完全满足一次 Reduce 所需要的空间（图 4 中 Shuffle Memory 占 Safe Memory 的比例大于 0.5）后，不会产生 Spill。

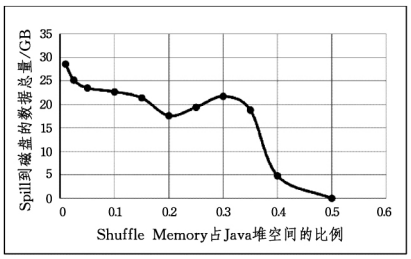


图 4 Spill 总数据量和 Shuffle Memory 空间的关系

PageRank 则对 Shuffle Memory 没有太多需求。测试显示，在使用 Google-Web 标准数据集来测试 PageRank 时，使用默认配置不会发生 Spill。所以，计算密集型应用需要配置较大的 Shuffle Memory 空间。

### 2.2 Storage Memory 使用特点测试

Spark 是一种充分利用内存的大数据计算框架，可以把接下来将要使用的 RDD 缓存在集群的内存中，当需要使用时可以从内存中直接读取，节省了重新计算 RDD 的开销。当 Spark 系统缓存了复用率较高的 RDD 时，Spark 应用的性能能够得到很大的提升。

在 Spark 系统中，选择哪些 RDD 进行缓存不是由 Spark 系统自动进行的，而是由 Spark 应用程序员通过调用 cache() 或者 persist() 这两个 API 进行的。cache() 会把 RDD 分区缓存到内存中，persist() 需要指定 RDD 分区缓存的存储级别，可以指定缓存到内存和磁盘。这种缓存方式非常灵活，但是也带来了很多不确定性，由于缓存 RDD 能够带来性能提升，因此 Spark 应用程序员可能会倾向于把很多 RDD 都缓存在集群节点的内存中，但是测试表明，缓存太多的 RDD 反而会影响 Spark 的性能。

图 5 示出 PageRank 运行过程中 RDD 的转换及依赖关系。可以看出，在每次迭代过程中都需要 links 这个 RDD，在 Page-Rank 中，与其他 RDD 相比，links 是需要缓存的重要 RDD。

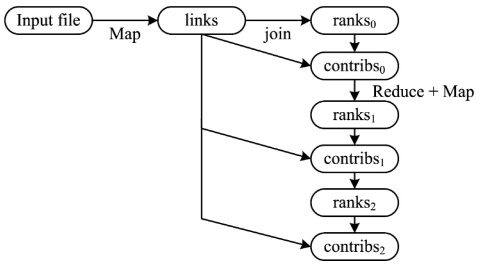


图 5 PageRank 中 RDD 的依赖关系

使用 web-Google 标准数据集和 cit-Patents 标准数据集，分别只缓存 links 和 contribs（缓存方式 2），或者缓存 links，contribs 和 Ranks（缓存方式 1），来测试 PageRank 的执行时间。对 web-Google 和 cit-Patents 进行 PageRank，Java 堆空间分别配置为 600MB 和 1GB。图 6 中的结果显示，采用缓存方式 1 时，PageRank 的性能平均下降 28%。

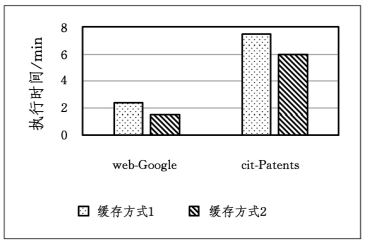


图 6 不同缓存方式下 PageRank 的执行时间

基于实验结果，在内存有限的情况下，缓存过多的 RDD 反而可能会影响 Spark 应用的性能。原因有两点：1）缓存相对不重要的 RDD 会占用有限的 Storage Memory 空间，当内存不足时，根据 LRU 淘汰策略，重要的 RDD 可能会被淘汰出内存。例如在 PageRank 中，如果 links 被淘汰出内存，当下一次迭代时，则需要根据原始数据重新产生 links，这将导致性能严重下降。2）RDD 分区在内存中的存储由 Java 虚拟机管理，在 Storage Memory 存储了较多的 RDD 分区后，空间不足时会引发 GC（Garbage Collection）。在任务执行过程中，GC 会占用一部分时间，从而导致整个 Spark 应用性能的下降。在 Spark 集群上运行占用内存资源较大的应用时，如果缓存过多的 RDD，反而有可能会影响程序的性能。

## 3 一种分布式权值缓存策略

在 Spark 系统中，RDD 以分区的形式缓存在集群节点的 Storage Memory 中。当某个 Storage Memory 空间不足时，选择淘汰掉本节点的某些 RDD 分区。目前 Spark 使用 LRU 策略淘汰 RDD 分区。LRU 选择近期最少使用的 RDD 分区，适用于局部性较好的内存页面淘汰管理。内存页面与 RDD 分区有很大的不同：首先，内存页面大小固定；其次，所有需要的数据都必须由内存页面来提供；最后，内存页面的数据是通过 I/O 来产生的，产生所有内存页面的代价一致。因此，需要基于 RDD 的特征来设计 RDD 分区的缓存替换策略。

### 3.1 分布式权值缓存策略分析

RDD 分区具有 4 个特征：1）RDD 包含多个 RDD 分区，存储在集群中多个节点的 RDD 分区组成整个 RDD，因此 RDD 具有分布式特征。Storage Memory 管理的是 RDD 分区，并不是整个 RDD。进行 LRU 策略淘汰时，仅仅利用了本节点的时间特征，并没有考虑到 RDD 的分布式特征。2）RDD 分区的存储会占据一定的存储空间。一般情况下，某个节点的 Storage Memory 中存储的不同 RDD 的分区大小不一致。一个 RDD 的多个分区的大小有很小的差别。当发生数据偏斜时，一个 RDD 的多个分区大小会有很大的差别，因此，RDD 分区具有空间大小特征。在其他条件相同的情况下，应该优先淘汰大的 RDD 分区，因为淘汰大的 RDD 分区后，会释放更多的内存资源。3）不同 RDD 分区具有不同的被访问频率。一般情况下，缓存的 RDD 都是在接下来的计算中需要用到的。RDD 分区缓存在节点的 Storage Memory 后会被再次访问。缓存这些 RDD 能够加速算法执行。如 Page-Rank 中的 links，每次迭代都需要访问。因此，缓存的 RDD 分区被访问的次数在一定程度上能够反映 RDD 的重要程度。RDD 分区

被访问的次数越多，说明该 RDD 在 Spark 应用中越重要。在内存不足时，应该尽量保留被访问次数多的 RDD 分区，淘汰被访问次数较少的 RDD 分区。4）RDD 的访问具有时间特征。在 Spark 应用运行过程中，需要缓存的 RDD 很多。某些 RDD 只在一个计算阶段内被频繁使用，而下一个计算阶段会频繁访问另一个 RDD。RDD 访问的时间特征使得存储在 Storage Memory 中的 RDD 分区的访问也具有时间特征。淘汰 RDD 分区时，也需要考虑 RDD 分区被访问的时间特征。

本文基于 RDD 分区的 4 个特征，提出了一种充分利用 RDD 特征的分布式权值缓存策略。在选择淘汰 RDD 分区时，需首先考虑该 RDD 的所有分区是不是全部被缓存，优先淘汰没有完全缓存在集群内的 RDD 分区，再根据 RDD 分区的其他特征计算其权值，优先淘汰权值较低的 RDD 分区。具体过程如下。

（1）分布式特征的利用。RDD 是 Spark 的核心数据结构，通过 RDD 的依赖关系形成 Spark 的调度顺序。通过对 RDD 的操作形成整个 Spark 程序[12]。在操作过程中，一组相同的 Task 负责处理一个 RDD。每一个 Task 负责处理一个 RDD 分区。当所有的 Task 处理完之后，再进入下一个阶段。Task 在处理 RDD 分区时，如果该 RDD 的所有分区都缓存在 Stora-ge Memory 中，该阶段的所有 Task 的处理速度都很快。但如果该 RDD 的某一个分区被淘汰了，处理这个分区的 Task 则要首先按照 RDD 的依赖关系重新计算该 RDD 分区，然后再处理该 RDD 分区。这样的话，一个 RDD 分区的重新计算会引入延迟，从而影响该 RDD 整个阶段的处理速度。

如图 7 所示，假设 RDDA 有分区 P1，P2，P3 分别缓存在节点 1、节点 2、节点 3 的 Storage Memory 中，当 P1 由于节点 1 上的 LRU 策略被淘汰出节点 1 的 Storage Memory 后，P2 和 P3 还分别缓存在节点 2 和节点 3 中。P1 由于不在内存中，因此需要重新计算。P2 和 P3 处理完成后 P1 还在处理中，处理 RDDA 的整个阶段的速度被 P1 的处理速度拉低。从整体上来看，缓存 P2 和 P3 并没有加快整个阶段的速度。因此，当节点 2 和节点 3 的 Storage Memory 空间不足时，应该首先淘汰 P2 和 P3。这就是 RDD 的分布式特征所导致的问题，即只有当缓存了所有 RDD 分区时，对该 RDD 处理的整个阶段才会被加速。

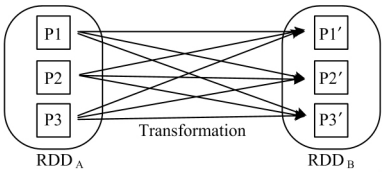


图 7　RDD 的处理

如果一个 RDD 的所有分区都缓存在集群节点的内存中，则称之为完整 RDD；完整 RDD 的分区称为完整 RDD 分区。如果一个 RDD 有部分分区被淘汰出节点的内存，则称之为不完整 RDD；其分区称为不完整 RDD 分区。

当节点的 Storage Memory 空间不足时，需考虑 RDD 的分布式特征，应该优先淘汰不完整 RDD 分区，保留完整 RDD 分区。

（2）权值的计算。在制定缓存策略时，还应该考虑 RDD 分区占用内存空间的大小特征、访问频率特征和时间特征。RDD 的大小特征、逻辑特征和时间特征可以分别用 RDD 分区大小（$S$）、RDD 分区在内存中被使用的次数（$N$）、RDD 分区存入内存的时间（$T$）3 个变量来表示。分析 RDD 分区的这 3 个特征：1）在其他条件相同的条件下应该优先淘汰占据内存空间较大的 RDD 分区。因为淘汰出较大的 RDD 分区后，会释放更多的内存资源。2）RDD 分区被使用的次数越多，说明该 RDD 在 Spark 应用中越重要。在内存不足时，应该尽量保留被访问次数多的 RDD 分区，淘汰被访问次数较少的 RDD 分区。3）某个 RDD 分区存入内存的时间越久，说明它是相对“旧”的 RDD 分区，在其他条件相同的情况下，优先淘汰这些“旧”的数据。因此，RDD 分区的权值计算公式为：

$$P=\frac{N}{S\times T} \tag{5}$$

在内存空间不足时，优先淘汰权值 $P$ 较低的 RDD 分区。

### 3.2　实验结果与分析

为测试并分析分布式权值缓存策略的性能，本文修改了 Spark1.4.0 core 部分，实现了 DWRP 策略、权值缓存策略和随机策略 3 种缓存策略。实验的软、硬件环境如表 1 所列，数据集使用 web-Google。实验测试了配置不同节点内存时 Page-Rank在每一种缓存策略下的执行时间。其中，LRU 策略选择淘汰掉 Storage Memory 中最近最少被访问的 RDD 分区。随机策略 RANDOM 随机选择 Storage Memory 中的某个 RDD 分区。DWRP 策略是本文提出的分布式权值缓存策略。

实验结果如图 8 所示，图中横坐标为集群内节点的可用内存，纵坐标是 PageRank 迭代 10 次的执行时间。实验结果显示，在内存充足的情况下（图 8 中内存大于 2048MB），各种缓存策略的结果基本一致。在内存不足的情况下，缓存策略决定了 PageRank 的执行时间。RANDOM 随机选择 RDD 分区进行淘汰，相对于其他策略，其性能不稳定。LRU 策略在内存不足的情况下，性能表现优于 DWRP 策略。DWRP 策略的性能稍低，原因在于：1）DWRP 策略实现复杂；2）DWRP 策略需要考虑 RDD 的分布式特征，分布式特征的获取需要进行集群内节点之间的网络通信，网络通信导致了性能的下降。

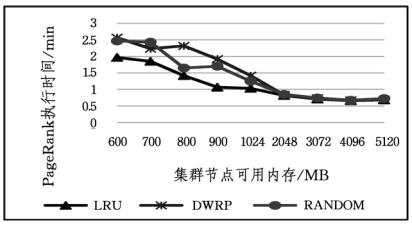


图 8　多种缓存策略下 PageRank 的执行时间

结束语　本文测试并分析了 Spark 系统的两个内存使用特点：1）计算密集型应用更倾向于使用 Spark 节点的 Shuffle Memory；2）在 Spark 系统中，适度缓存 RDD 能够加快 Spark 应用的执行速度，但缓存过多 RDD 反而可能影响 Spark 应用的整体性能。基于 RDD 的多个特征，本文提出了一种新的分布式权值缓存策略，在淘汰 RDD 分区时综合考虑了 RDD 的多个特点，最后测试了采用多种缓存策略时 PageRank 的执

（下转第 74 页）

prod/collateral/…/ CiscoEMSWhitePaper. pdf.

[5] WANG H, GARG A, BERGMAN K. Design and demonstration of an all-optical hybrid packet and circuit switched network platform for next generation data centers[C]// OFC 2010, OTuP3. 2010: 1-3.

[6] SINGLA A, SINGH A, Ramachandran K, et al. Feasibility study on topology malleable data center networks (DCN) using optical switching technologies[C]// Optical Fiber Communication Conference and Exposition (OFC) and the National Fiber Optic Engineers Conference(NFOEC). 2011: 1-3.

[7] BALIGA J, HINTON K, TUCKER R S. Energy Consumption of the Internet[C]// Proc. Joint International Conference on Optical Internet and the 32nd Australian Conference on Optical Fibre Technology (COIN-ACOFT 2007). Melbourne, Australia, 2007: 1-3.

[8] GAO S, ZHOU J, AYA T, et al. Reducing network power consumption using dynamic link metric method and power off links [C]// IEICE Communications. 2009.

[9] ZHANG M, YI C, LIU B, et al. GreenTE: Power-aware traffic engineering[C]// 2010 18th IEEE International Conference on Network Protocols (ICNP). Kyoto, 2010: 21-30.

[10] CHABAREK J, SOMMERS J, BARFORD P, et al. Power awareness in network design and routing[C]// Proc. IEEE INFOCOM. April 2008: 457-465.

[11] TSENG P K, CHUNG W H. Near optimal link on/off scheduling and weight assignment for minimizing IP network energy consumption[J]. Computer Communications, 2012, 35(6): 729-737.

[12] CUOMO F, ABBAGNALE A, CIANFRANI A, et al. Keeping the connectivity and saving the energy in the internet[C]// 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). Shanghai, China, 2011: 319-324.

[13] WANG N, MICHAEL C, HO K H. Disruption-Free Green Traffic Engineering with NotVia Fast Reroute[J]. IEEE Communications Letters, 2011, 15(10): 1123-1125.

[14] ZHOU B, ZHANG F, WANG L, et al. HDEER: A Distributed Routing Scheme for Energy-Efficient Networking[J]. IEEE Journal on Selected Areas in Communications, 2016, 34(5): 1713-1727.

[15] AMALDI E, CAPONE A, GIANOLI L G. Energy-aware IP traffic engineering with shortest path routing[J]. Computer Networks, 2013, 57(6): 1503-1517.

[16] Network Group at UESTC. topo_tm_files[OL]. https://bitbucket.org/netlab-uestc/greentm/src.

(上接第 35 页)

行时间，分析了 Spark 系统在不同缓存策略下的性能。DWRP 在内存不足时的性能略低于 LRU 策略，这是因为 DWRP 策略需要考虑 RDD 的分布式特征，在选择淘汰 RDD 分区时需要进行通信，会产生网络通信开销。随着网络性能的不断提高和网络通信延迟的降低，DWRP 策略将充分发挥其优势，提高 Spark 系统的性能。

## 参 考 文 献

[1] ZAHARIA M, CHOWDHURY M, FRANKLINM J, et al. Spark: cluster computing with working sets[C]// Usenix Conference on Hot Topics in Cloud Computing. 2010: 10.

[2] WARNEKE D, LENG C. A Case For Dynamic Memory Partitioning in Data Centers[C]// The Workshop on Data Analytics in the Cloud. 2013: 41-45.

[3] LI H, GHODSI A, ZAHARIA M, et al. Tachyon: Reliable, memory speed storage for cluster computing frameworks[C]// Proceedings of the ACM Symposium on Cloud Computing. ACM, 2014: 1-15.

[4] ANANTHANARAYANAN G, GHODSI A, WANG A, et al. PACMan: coordinated memory caching for parallel jobs[C]// Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, 2012: 20.

[5] DUAN M, LI K, TANG Z, et al. Selection and replacement algorithms for memory performance improvement in Spark[J]. Concurrency and Computation: Practice and Experience, 2015, 28(8): 2473-2486.

[6] FENG L. Research and Implementation of Memory Optimization Based on Parallel Computing Engine Spark[D]. Beijing: Tsinghua University, 2013. (in Chinese)
冯琳. 集群计算引擎 Spark 中的内存优化研究与实现[D]. 北京：清华大学，2013.

[7] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing: UCB/EECS-2011-82[R]. EECS Department, University of California, Berkeley, 2011.

[8] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]// Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2.

[9] GRISHCHENKO A. Spark Architecture: Shuffle [EB/OL]. (2015-08)[2016-09]. https://0x0fff.com/spark-architecture-shuffle.

[10] WHITE T. Hadoop: The Definitive Guide, 3E. [M]. California: O'Reilly Medis, 2012: 226-227.

[11] WANG L, ZHAN J, LUO C, et al. Bigdatabench: A big data benchmark suite from internet services[C]// 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2014: 488-499.

[12] GAO Y J. DataProcessing with Spark, Technology, Application and Performance Optimization [J]. Beijing: China Machine Press, 2014: 38-39. (in Chinese)
高彦杰. Spark 大数据处理技术，应用与性能优化[M]. 北京：机械工业出版社，2014: 38-39.