Journal of Computer Applications
计算机应用,2017,37(6):1574 - 1579
ISSN 1001-9081
CODEN JYIIDU
2017-06-10
http://www.joca.cn

文章编号:1001-9081(2017)06-1574-06
DOI:10.11772/j.issn.1001-9081.2017.06.1574

Spark Streaming 动态资源分配策略

刘　备*,谭新明,曹文彬
(武汉理工大学 计算机科学与技术学院,武汉 430063)
(*通信作者电子邮箱 liubei1203@163.com)

摘　要:针对 Spark Streaming 作为混合大数据计算平台流处理组件时资源调整周期长和不能满足多应用多用户个性化需求的问题,提出了一种多应用下动态资源分配策略(DRAM)。该策略增加了应用全局变量来控制动态资源分配过程。首先,获取历史执行数据反馈和应用全局变量;然后,进行资源增减计算;最后,进行资源增减执行。实验结果表明,所提策略能够有效调整应用资源配额,且在稳定数据流和不稳定数据流两种情况下,其处理延时相比原 Spark 平台的 Streaming 策略和 Core 策略都有所降低;同时该策略也能够提高集群资源利用率。

关键词:Spark;实时数据流;多应用;动态资源分配
中图分类号:TP391.1　　**文献标志码**:A

Dynamic resource allocation strategy in Spark Streaming

LIU Bei*, TAN Xinming, CAO Wenbin
(*School of Computer Science & Technology, Wuhan University of Technology, Wuhan Hubei* 430063, *China*)

Abstract: The existing resource allocation strategy has long resource adjustment cycle and cannot sufficiently meet the individual needs of different applications and users when Spark Streaming is selected as stream processing component in hybrid large-scale computing platform. In order to solve the problems, a Dynamic Resource Allocation strategy for Multi-application (DRAM) was proposed. The global variables were added to control the dynamic resource allocation process in DRAM. Firstly, the historical data feedback and the global variables were obtained. Then, whether increasing or decreasing the number of resources in each application was determined. Finally, the increase or decrease of resources was implemented. The experimental results show that, the proposed strategy can effectively adjust the resource quota, and reduce the processing delay compared with the original Spark platform strategies such as Streaming and Core under both cases of the stable data stream and the unstable data stream. The proposed strategy can also improve the utilization rate of the cluster resources.

Key words: Spark; real-time data stream; multi-application; dynamic resource allocation

0　引言

随着针对数据流的研究,大规模动态数据集(也称为实时数据流)成为研究及工程人员争相探索的热点领域。大数据时代持续性流数据呈指数型增长,让实时数据流处理受到很大的关注。Spark[1]作为一个高效的分布式计算系统和顶级的内存计算技术,其组件 Spark Streaming[2]对于可靠的、高吞吐量和低延迟的流处理有着很好的支持。

目前 Spark 应用资源分配默认采取预分配的方式,资源量在程序提交时已经确定直到查询或者计算退出。当多个应用共享一个 Spark 集群时,集群资源总量是有限的,即多个应用的资源总量固定。而流处理中数据量往往表现出动态变化性,用户查询也具有随机性的特征,这样一个 Spark Streaming 应用需要的计算资源并不是恒定不变的,某个时间段可能需要的计算单元会陡然增加或者减少,静态配置应用程序资源不能满足资源合理利用需求,多用户下同一时间段不同应用程序有不同大小的计算任务,对资源的迫切程度也不一样。对资源动态分配既可以保证流处理的实时性和资源合理分配,又能够满足用户个性化需求。虽然 Spark Core 和 Spark Streaming 均提供动态资源分配机制来实现动态增减应用计算资源,但是现有的两种策略并不能保证处理的实时性,也不具备多应用的特征。分析结果表明,对于不稳定的输入流,流处理延迟时间呈现很大的波动性。

为了降低多用户下流处理延时,提高集群资源利用率,本文提出一种基于 Spark Streaming 流处理的多应用下动态资源分配策略(Dynamic Resource Allocation strategy of Multi-application, DRAM),以减少数据流频繁波动情况下处理延时和资源动态分配时间,并比较两种流处理场景下该方法的可行性和性能。

1　相关工作

企业级数据平台往往需要满足多种应用场景如流处理业务场景、海量批处理、迭代计算、图计算等。在一个项目中同时满足多种业务需求,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(Extract Transform Load, ETL),这无疑将增加系统的复杂程度和负担;另一方

收稿日期:2016-11-25;**修回日期**:2016-12-22。　**基金项目**:湖北省自然科学基金重点项目(2014CFA050)。
作者简介:刘备(1993—),男,湖北仙桃人,硕士研究生,主要研究方向:大数据应用、移动互联网;　谭新明(1961—),男,湖北荆州人,教授,博士,主要研究方向:软件工程方法、物联网技术及系统;　曹文彬(1991—),男,河南许昌人,硕士研究生,主要研究方向:移动互联网、大数据环境下处理平台。

面使用多套系统也增加了使用和维护的难度,使用 Spark 系统则可以适用于目前常用的各种大数据计算模式[3]。同时 Spark 在 Yarn 模式下的运行提供的资源隔离和资源弹性管理以及对传统批处理系统中文件存储的支持也方便企业对于历史数据的利用。故构建多用户下软件即服务(Software as a Service, SaaS)[4]模式平台计算中心采用 Spark 平台比较适用。

计算平台中常用的数据流可以来自股票市场的时序分析、企业交易、各种交互事件、Web 流量、点击流和传感器数据等,都是即时且带有时间戳的数据[5]。这些数据流需要及时处理以便于监测,例如异常检测、异常奇点、垃圾邮件、欺诈和入侵;也可提供基础的统计、计算和推荐。某些情形,总结性的汇总数据需要存储以备将来使用。计算平台中的数据流具有海量性、实时性和动态变化性的特点,所以数据平台的处理任务大小也具备动态变化特征,同样企业中对于数据流计算的查询也是动态变化的。

为保证数据处理实时性,资源需要动态变化,这样一方面提高了资源利用率,另一方面提高了实时性。已有的对于 Spark Streaming 实时性的改进更多的是对于微批处理大小、时间间隔长短等方面进行改进来保证输入流的稳定性。如文献[6]中研究微批处理大小对流处理的影响,从历史数据中得到信息修改 batch 间隔来保证输入流的平稳,但是对于周期性波动流数据依然有很大延迟。文献[7]中根据半个 batch 周期事件的平均值来控制生成任务数量,保证输入流平稳,但是缺少与 Spark 平台动态资源分配机制的结合。大数据计算中往往可以通过直接对资源动态分配来保证资源有效利用。文献[8]对云计算下虚拟机资源动态分配进行了研究。文献[9]对异构 Hadoop 计算资源动态分配进行了研究。这些对资源动态分配的研究大多集中在 Hadoop 等平台,目前 Spark 平台中除了在 Spark Core 和 Spark Streaming 提供相应的动态资源分配机制外,并未见到对动态资源分配机制的研究。

2　Spark Core 动态资源分配分析

2.1　Spark Core 运行过程分析

Spark 默认采用的是资源预分配和粗粒度的方式分配资源。所谓资源单位一般指的是 Executor, Executor 是某 Application 运行在 Worker Node 上的一个进程,该进程维持线程池运行 Task,并且负责将数据存在内存或者磁盘上。每个 Application 提交运行时将申请获取一组独立运行该 Application 中 Task 的 Executor 资源。直到运行结束,该程序将一直持有资源。提交程序时,通常使用 num-executors 来指定 Application 使用的 Executor 数量,而 executor-memory 和 executor-cores 分别用来指定每个 Executor 所使用的内存和虚拟内核数。Spark 基本运行过程如图 1 所示。

Spark 应用从用户提交应用到应用运行结束需要经过如下几个步骤:

a)用户启动客户端,向集群提交用户程序。

b)Resource Manager 遍历可用 Woker 节点,随机选取一个符合 SparkAppMaster 资源要求的 Worker 作为新建 SparkAppMaster 的节点,并向 Worker 中 NodeManager 发送新建 Executor 请求。

c)NodeManager 在选中的 Worker 节点上新建 Executor 用来运行 SparkAppMaster,并在 SparkAppMaster 中创建运行环境 SparkContext[1](启动 Driver 进程,创建 SparkContext 及 DAGScheduler、TaskSchduler 等)。

d)SparkAppMaster 向 ResourceManager 发送申请 Executor 资源的请求。

e)ResourceManager 分配 Executor 给 SparkAppMaster, SparkAppMaster 和相关的 NodeManager 通信,NodeManager 向 SparkAppMaster 中的 SparkContext 注册并等待分配 Task。

f)SparkContext 将 Applicaiton 代码序列化发送给 Executors,并且 SparkContext 解析代码构建运行逻辑的有向无环图(Directed Acyclic Graph, DAG),并提交给 DAGScheduler 分解成 Stage(Action 操作时,就会生成 Job 或 Job 集合;每个 Job 中含有 1 个或多个 Stage,Stage 产生在 RDD 宽依赖之间),然后将 Stage 中 TaskSet 提交给 TaskScheduler, TaskScheduler 负责将 Task 分配到应用中满足条件的 Executor 上执行;Executor 执行 Task 并向 SparkAppMaster 中的 SparkContext 汇报运行状况;Task 运行完毕,SparkContext 归还资源给 ResourceManager,并注销退出。

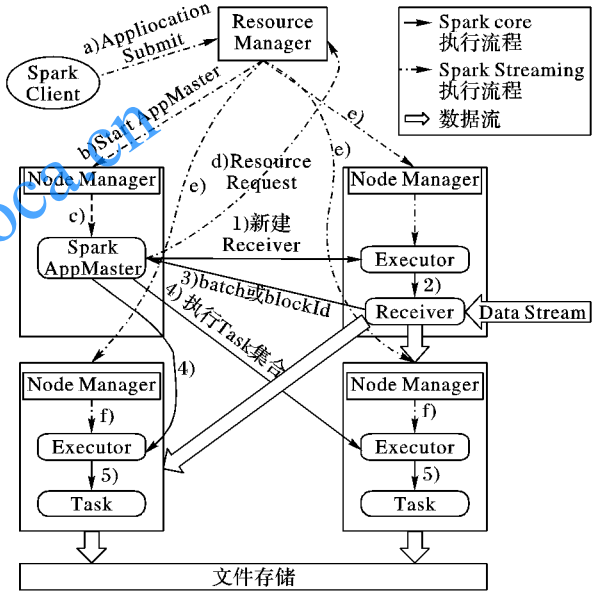

图 1　Spark 运行流程

Fig. 1　Operation flow of Spark

Spark Streaming 应用运行时会调用 Spark Core 中步骤 a)~e)新建 Spark AppMaster 并获取应用所需资源,然后执行如下步骤:

1)Spark AppMaster 选取一个或多个节点新建 Executor。

2)Node Manager 将新建的 Executor 作为数据接收器 Receiver,数据流由 Receiver 处理。

3)Receiver 将接收的数据按照应用时间间隔存储为 batch 或数据块并将数据块信息返回给 Spark AppMaster。

4)Spark AppMaster 按照 Spark Core 中步骤 f)中划分为 Task 集合,Spark AppMaster 将 Receiver 中 batch 信息和 Task 集合发送给各个节点中 Executor。

5)各节点中 Executor 根据数据块位置进行 Task 执行并返回处理结果。

2.2　Spark Core 动态资源分配分析

Spark 1.2 版本开始在 Spark-on-Yarn 模式下提供动态资

源分配机制。通过应用程序设置 dynamicAllocation. enabled = true 开启,同时可设置的可用属性有:最小分配 Executor 数 *minEs*、最大分配 Executor 数 *maxEs*、资源过期时间 *eit*、执行等待时间 *sbto*、申请资源时间间隔 *sto*。

基本思想是:Application 在 Job 中 Task 因没有足够资源被挂起的时候去动态申请资源,这种情况意味着现有的 Executor 数不能满足所有 Task 并行运行,所以需要向集群资源管理器申请更多资源,每隔一段时间申请一次,一直到申请足够的资源。当 Application 中分配到的 Executor 挂起或者等待 Task 超过过期时间(默认为 1 s)的时候,集群资源管理器会释放该 Executor 资源。

在算法 1 中,系统通过监听器监听集群中 Executor 状态信息和任务执行信息,新提交的 Task 集合需求的资源大于现有资源时,系统记录开始本次资源动态分配的时间 *at*,并会根据当前任务需要的最大的 Executor 数 *numN* 和当前已经有的 Excutor 数 *numT* 比较来增加和删除程序资源,起始当 $numN < numT$ 时,每次申请需要增加的 Excutors 数 $numEA = 1$,每次经过 *sto* 时间进行一次申请,每次申请的资源呈现指数增长,即 1、2、4、8 等采用指数增长,直到系统中 $numN = numT$ 时重置 $numEA = 1$。当 Excutor 没有任务运行并不存在 cache 时放入移除列表,经过 *eit* 时间如果 Executor 上没有任务运行则删除,否则从移除列表移除,每过 *sbo* 时间执行一次删除 Executor。

算法 1 Spark core 动态资源分配。

输入 Stage,最小/大资源量 *minEs*/*maxEs*,开始添加资源时间 *at*;

输出 资源增减。

1) 根据 Stage 生成 Task 和每个任务需要资源量,得到 *numN*;
2) 获取当前已获得资源量 *numT*,当前时间 *ct*,需要增加的资源数 *numEA* 以及 *at* 是否设置;
3) if ($numN \leqslant numT$) then
4) $at \leftarrow ct$; $numEA \leftarrow 1$;
5) $numT \leftarrow \max(numN, minNum)$
6) else if($ct \geqslant at$ and $at ==$ not set) then
7) addExecutor(*numN*);
8) end if
9) end if
10) addExecutor() 中根据 *numN*、*ct*、*at* 和 *sto* 计算 *numEA*,增加资源到 *numN*;
11) removeExecutor() 对放入删除列表的资源经过 *sbo* 时间执行删除。

3 Spark Streaming 动态资源分配分析

3.1 Spark Streaming 运行机制分析

Spark Streaming 作为 Spark 计算平台的组件之一,充分利用了 Spark 的核心架构。同时作为流处理功能的入口点 Streaming Context[2],它构建在 SparkContext[1] 之上。集群管理器将至少单独分片一个工作节点作为接收器,这是一个长时间运行的任务执行器来处理进入的流数据。执行器创建 Discretized Streams[10] 或者从输入数据流中得来的一组弹性分布式数据(Resilient Distributed Dataset, RDD)[11] 集合 DStreams, DStream 默认为另一个 Worker Node 的缓存。接收器服务于输入数据流,多个接收器提升了并行性,产生多个 DStreams,Spark Streaming 用它操作 RDD。流处理程序中 Application 的 action 操作生成 Job 集合提交给 Spark 内核,每个 Job 生成 Task 集合给 Executor 进程执行。

如图 2 所示,SparkAppMaster 将接收器 Receiver 作为一个 Task 提交给一个 Executor, Receiver 启动会按照时间间隔 batch interval 读入时间长度为 batch Duration 的流数据,生成数据块 block;Job 生成模块 Job Generator 根据 batch 生成相应的 Job,Job 提交给 Job 执行模块 Job Pocessor,Job Pocessor 在集群中寻找空闲 Executor 执行 Job 中 Task 集合。

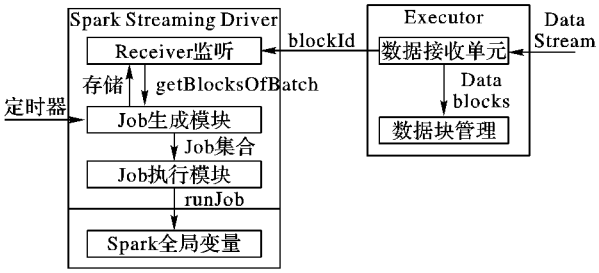

图 2 Streaming 运行流程

Fig. 2 Operation flow of Streaming

Spark Streaming 通过 Receiver 以生产者生产数据的速率接收数据,计算过程中会出现 batch processing time > batch interval 的情况,其中:batch processing time 为实际计算一个批次花费时间, batch interval 为 Streaming 应用设置的批处理间隔。这意味着 Spark Streaming 的数据接收速率高于 Spark 从队列中移除数据的速率,也就是数据处理能力低,在设置间隔内不能完全处理当前接收速率接收的数据。如果这种情况持续过长的时间,会造成数据在内存中堆积,导致 Receiver 所在 Executor 内存溢出等问题。

流处理中每个时间切片 batch Duration 中数据量大小差别很大,导致要求的资源差别很大,如果几个周期内还没调整完资源,可能导致任务挂起或者执行延时较大,所以 Spark Core 中动态资源分配算法在流处理中并不适用。

3.2 Spark Streaming 动态资源分配分析

Spark Streaming 组件将数据输入流拆分为一个个 batch 去处理,batch 中每个记录处理的总时间都不一样。一个 batch 中早到达的记录会有一个长的延迟时间。假设时间切片为 T_w 即 batch 时间长度,最坏的情况下一个 batch 中记录的总执行时间为 $T_t = T_w + T_e$,T_e 代表记录 *e* 总延迟时间,而在 Spark Streaming 中 T_e 分为事件调度时间 T_s 和事件运行时间 T_p,故 $T_t = T_w + T_s + T_p$。文献[7]中通过对比实验得出,当 batch 间隔太小时即 T_w 太小时会导致 batch 中数据量太小产生的 Job 数量会陡然增加,从而导致大部分小 Job 在等待分配资源队列中,T_s 时间几乎可以代表总执行时间 T_t,当获取到资源之后当 Executer 不足时会延长 T_p 时间,所以 T_p 时间长短代表资源充足与否。

Spark 1.5 版本推出的 Spark Streaming 动态资源分配机制获取前一个 batch 的 T_p 时间来判断是否需要增减资源。即算法 2 中从监听器获取前一 batch 总运行时间 T_p 和事件总数 *bpc*,同时从 Streaming Context 中获取参数上延时比例 *sur* 和下延时比例 *sdr*。比较事件平均处理时间 $T_{avg} = T_p / bpc$ 和时间窗口 Duration,得到两者比例 *Ratios*,$Ratios \geqslant sur$ 则请求资源,请求资源数为 max(round(*Ratios*),1),$Ratios \leqslant sdr$ 则减少资源,减少资源时需要确认 Executor 不存在 Receiver。为了保证

之前的资源调整完毕,每隔 *sis* 时间调整一次,*sis* 由应用程序设置。

算法 2　Spark Streaming 动态资源分配。

输入　最小/大资源量 *minEs/maxEs*, batch 周期时间 *bd*, 上 / 下延时比例 *sur/sdr*,动态管理时间间隔 *sis*。

输出　资源增减。

1) 获取执行时间总量 T_p 和微批处理总个数 *bpc*,计算平均批处理执行时间 *t*1。
2) *Ratios* = *t*1/*bd*;
3) if *Ratios* ⩾ *sur* then
4) *NumEA* ← max(round(*t*1),1);
5) RequestExecutor(*NumEA*);
6) else if *Ratios* ⩽ *sdr* then
7) KillExecutor();
8) end if
9) end if

3.3　多应用下的动态资源分配策略

Spark Core 中动态资源策略在面对 Spark Streaming 流处理中 batch 前后差距很大的情况时,需要几个周期去资源调整,所以流处理中并不适用。现有的 Spark Streaming 中动态资源分配算法仅仅考虑了前一 batch 的执行时间,当数据流中事件呈现周期性波动性很大时,会造成系统频繁地去增减资源造成系统抖动,同样资源调整需要花费较多的周期数,造成 T_p 时间延迟,所以更多的时候需要结合控制输入流流入速率机制来保证输入流的稳定性以达到要求。多用户多应用提交应用程序到集群处理时,对任务处理实时性期望不一样,数据流量也会不同,当需要增减资源配额时,高优先级用户可能需要下一个周期就能满足资源需求,而低优先级用户则可以通过几个周期来调整。多用户的需求体现在数据的稳定性和资源需求的迫切性,所以需要增加流处理应用程序参数来区分各个应用的调整比例和最小资源保有情况,实现多应用下的动态资源分配策略(DRAM)。

3.3.1　动态资源分配模型

根据 Spark Streaming 应用运行机制和现有 Spark Streaming 动态资源分配机制分析得到 Spark Streaming 动态资源分配模型,如图 3 所示。

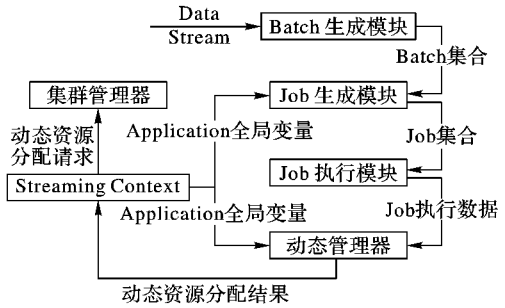

图 3　动态资源分配模型

Fig. 3　Dynamic resource allocation model

从图 3 中可以看出,数据流经过 batch 生成模块 Batch Generator 拆分为一个个 batch,提交给 Job 管理器 Job Generator;Job Generator 结合 Application 中全局参数和任务处理步骤生成对 RDD 的操作即 Job 集合,然后提交到集群中由 Job Processor 运行;监视器器周期性从 Job Processor 中获取历史 Job 数据;动态管理器 Allocation Controller 获得监听器历史数据反馈并结合 Application 环境变量,计算资源调整结果;Allocation Controller 将资源增减结果发送给 Streaming Context,Streaming Context 向集群管理器提交增减任务资源请求;集群管理器增减 Job 资源量并发送给 Job 执行器。所以动态资源分配模型可分解为获取历史数据和应用程序全局变量、资源增减计算和资源增减执行三个部分。

3.3.2　历史数据和全局变量获取

在开始动态资源分配前,需要获得输入参数。针对多个用户任务的实时性要求不同,在全局变量中添加了任务重要比例 *ar*、参考 batch 数 *rbs*、资源减少周期数 *rr*、减少资源时保有比例 *rra*;同时设置上 / 下延时比例 *sur/sdr*、动态调整时间 *sis* 和最小 / 大资源量 *minEs/maxEs*。历史数据需要获取任务已有资源量 *ctn*、Job Processor 中前 *rbs* 个 batch 中事件平均执行时间 $t1_i(i = 1,2,\cdots,rbs)$、batch 切片时间 *bd*。全局变量中,*ar* 代表增加资源坡度,区间为[0,1],默认值为 1;减少资源方面,*rr* 表示资源减少时需要在 *rr* 个 *sis* 时间内减少资源,默认值为 1。为了防止资源频繁增减造成系统震荡,设置了最终需要保有的比例值 *rra*。

3.3.3　资源增减计算

当开启动态资源分配并获得历史数据和全局变量后,Allocation Controller 首先要判断是否需要增减资源。这里需要参考 *rbs* 个 batch 中事件平均执行时间 $t1_i$ 计算动态资源分配所参考的 batch 中事件平均执行时间 *T*1,并计算它与 batch Duration 的比例 *Ratios*:

$$T1 = \sum_{i=1}^{rbs} t1_i \Big/ rbs \tag{1}$$

$$Ratios = T1/bd \tag{2}$$

当 *Ratios* 处于区间[*sdr*,*sur*] 时,动态管理器不进行资源调整,直接进入下一个 *sis* 周期。当 *Ratios* ⩾ *sur* 时,表明处理时间大于 batch 切片间隔,则任务现有资源不能满足处理需求,需要给任务添加资源,添加的资源量为:

$$NumEA = (maxEs - ctn) * ar \tag{3}$$

当 *Ratios* ⩾ *sdr* 时说明当前用户任务资源过剩,需要减少任务资源避免资源浪费,这里需要分多个周期来减少并能保有一定比例来防止系统震荡,总的要减少的资源量为:

$$NumPr = \text{round}(ctn * ((bd - T1)/bd - rra)) \tag{4}$$

当前周期内实际需要减少的资源量为:

$$NumArn = NumPr/bd \tag{5}$$

3.3.4　资源增减执行

应用程序设置参数 *sis* 来保证资源调整时间的,*sis* 默认 60 s。当 Allocation Controller 向 Streaming Context 提交资源调整请求后,Streaming Context 将请求发送给集群管理器来增减任务资源。增加资源后,资源量不高于 *maxEs*, 当空闲资源不足时会等待;减少资源时首先需要确定资源中没有 Receiver,减少的资源 Executor 将不会再分配任务,并交给集群管理器去异步减少。

算法 3　多应用动态资源分配算法。

输入　最小资源量 *minEs*,最大资源量 *maxEs*,任务重要比例 *ar*,资源减少周期数 *rr*,保有比例 *rra*,参考周期数 *rbs*,batch 周期时间 *bd*, 上 / 下延时比例 *sur/sdr*;

输出 资源增减。

1) 获取当前资源量 *ctn*,得出前 *rbs* 个周期平均执行时间 *T*1;
2) *Ratios* = *T*1/*bd*;
3) if *Ratios* ≥ *sur* then
4) *NumEA* ← (*maxNum* − *ctn*) * *ar*;
5) RequestExcutor(*NumEA*);
6) else if *Ratios* ≤ *sdr* then
7) *numPr* ← round(*ctn* * ((*bd* − *t*1)/*bd* − *rra*));
8) *arn* = *NumPr*/*rr*;
9) KillExecutor(*arn*);
10) end if
11) end if

4 实验结果与分析

4.1 测试环境配置

实验中采用 8 台虚拟机模拟物理机器搭建 Spark 集群,集群配置情况总的有 28 个 CPU 核数、24 GB 内存,CPU 主频为 2.20 GHz,Linux 版本是 32 bit Ubuntu14.04,Spark 版本是 1.6.1,集群中有一台服务器作为 Master,其余七台作为 Slave,每个应用程序能请求到的最大的 CPU 核数为 4,集群运行模式为 Spark on Yarn,集群中最多允许提交的任务数为 4。

实验程序选取 Spark 中常用的应用实例作为应用程序提交:第一个是 WordCount,用来统计数据流中单词出现的次数,每一次微批处理相当于一组接收自网络流接口的单词组。第二个是 Grep,应用于输入数据流中匹配目标字符串并计算字符串出现的个数。为了测试多种数据下的性能表现,选取不同的数据流类型模拟输入流,数据流类型分别为平滑的数据流、不平滑的数据流。平滑的输入流大部分时间都是稳定的,但为了测试资源动态分配导致的执行时间变化也设置了峰值,平滑数据流 batch 间隔为 1 s,batch 中 event 个数在 2000 上下浮动 500 个 event;不平滑的输入流则设置了周期性大量变化的时间以对系统进行性能测试,每隔 50 个 batch 会有一次尖锐的峰值,batch 中 event 数量增加到 5 000 以上,然后 event 数量稳定到 1 300。Spark Streaming 利用 Spark 内核去执行流处理任务,而 Spark 的优势在于利用内存来缓存中间结果以及储存,所以 Spark 集群中内存利用率通常会很高,选取内存利用率作为资源利用率的参考不具备代表性。所以这里比较三种策略在相同条件下 batch 总执行时间和 CPU 利用率的情况。实验结果通过查看日志文件得到。实验结果中,Core 代表 Spark Core 动态资源分配策略下实验情况,Streaming 表示 Spark Streaming 动态资源动态分配策略下实验情况,DRAM 即为多应用动态资源分配策略下实验结果。

4.2 结果分析

4.2.1 稳定数据流分析

将 Grep 和 WordCount 分别提交到 Spark 集群中,分别设置在 Spark Core 动态资源分配策略、Spark Streaming 动态资源动态分配策略和多应用动态资源分配策略(DRAM)下运行。Spark Core 动态资源分配策略和 Spark Streaming 动态资源动态分配中程序参数设置均设置为系统默认值,多用户动态资源分配策略中任务重要比例均设置为 0.5,参考周期设置为 1,资源保有比例设置为 0.2,上下延时比例设置为 0.9/0.3。实验中考虑到影响集群运行的因素有很多,为了保证实验代表性,所以在相同的稳定输入流下运行了 500 个 batch,经过反复实验后发现系统前 50 个 batch 需要经历初始化过程,流处理延时波动会比较大,故丢弃前 50 个 batch,取系统稳定阶段平均处理时间。统计了三种策略下平均任务完成时间如图 4 所示。

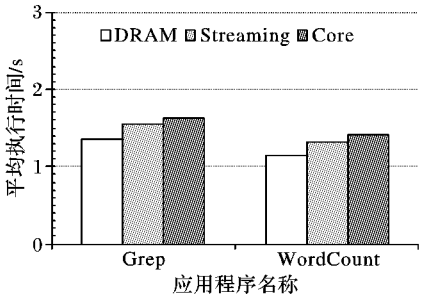

图 4 稳定数据流下 Grep 和 WordCount 平均执行时间

Fig. 4 Average execution time of Grep and WordCount with stable data stream

从图 4 中可以看到,Grep 应用 batch 平均处理时间方面,DRAM 比 Streaming 策略降低了接近 15%,同样,DRAM 比 Core 策略降低了 17%。在使用 DRAM 的 Spark Streaming 平台中 WordCount 应用 batch 处理时间比使用 Core 和 Streaming 两种策略的 batch 处理时间也有相应的降低。

4.2.2 不稳定数据流分析

动态资源分配策略使用场景为:当数据流中数据量增长时,实时增加程序完成任务缺少的资源,减少任务等待时间;在数据流中处理事件较少时,能够平滑地减少资源数,做到计算资源合理利用。在不稳定数据流的测试中,每隔一段时间 batch 中的处理事件(这里的处理事件数是指由某一时间段数据流生成的处理任务集)呈现周期性的尖锐峰值增长,应用程序设置与稳定数据流下一致,程序运行 500 batch 之后查看日志下某一个 batch 的执行时间并记录表格。图 5 表示 WordCount 应用程序在三种不同策略下的 batch 总处理时间(total time)和 batch 中事件的个数(event number)。

由图 5 可以看出,在有尖锐峰的不稳定数据流下,DRAM 相对于 Core 和 Streaming 动态资源分配策略,其 batch 总执行时间波动较小,且平均时间有所减低,表明 DRAM 能够在不稳定数据流中对任务资源动态分配,降低系统延时。

在处理过程中,集群资源的利用率越高代表资源空闲时间越短,集群资源利用越有效。图 6 为应用程序处理过程中各个节点平均 CPU 资源利用率。从图 6 中可以看出,DRAM 的平均节点利用率高于另外两种策略,这说明 DRAM 集群资源空闲时间更短,资源利用率更高。

5 结语

针对当前 Spark 计算平台应用于多应用 SaaS 平台中面临的数据流实时波动情景下,现有动态资源分配机制响应不够及时并需要结合控制数据速率来提高实时性的问题,本文通过对现有机制进行分析,提出动态资源分配模型,并针对多用户下多数据流变化的特点,根据系统反馈任务资源信息变化,增加了任务重要比例参数和减少周期、比例的基础上,实时调整应用程序资源,以更好应对突发计算任务。实验结果表明,

本文所提方法能够有效保证计算任务的实时执行,优化了 Spark Streaming 动态资源分配,提高了集群资源利用率。未来将继续研究测试方法中参数值的合理化设置以及动态资源分配机制与控制流入速率方面的结合。

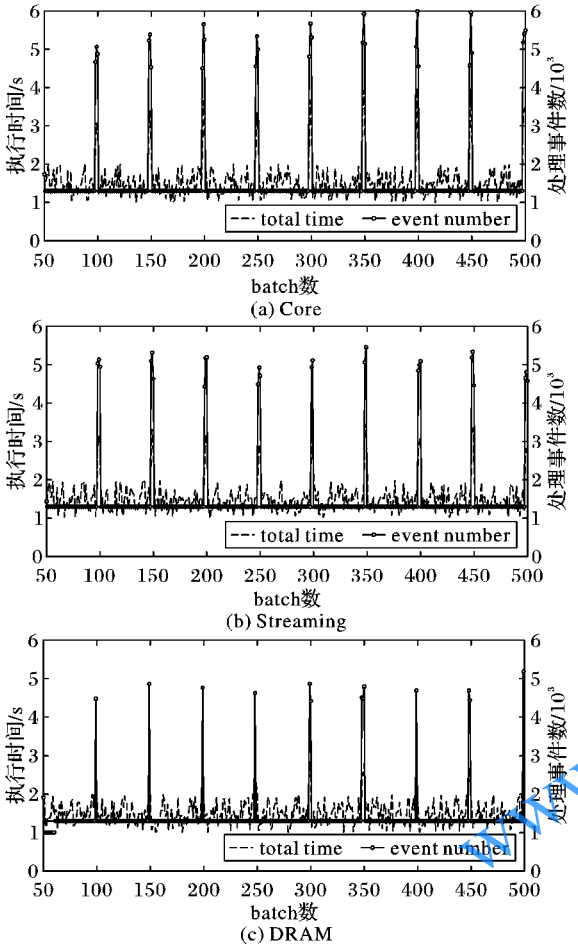

图 5　不同策略在不稳定数据流下执行时间

Fig. 5　Execution time of different strategies with unstable data stream

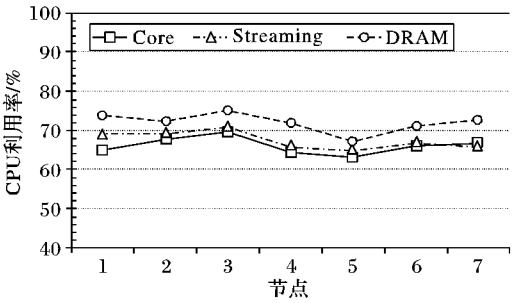

图 6　各节点平均 CPU 资源利用率

Fig. 6　Average CPU resource utilization rate of the nodes

参考文献 (References)

[1] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: cluster computing with working sets [C/OL]// HotCloud'10: Proceedings of the 2010 2nd USENIX Conference on Hot Topics in Cloud Computing. Berkeley, CA: USENIX Association, 2010. [2016-10-25]. https://www.usenix.org/legacy/events/hotcloud10/tech/full_papers/Zaharia.pdf.

[2] 夏俊鸾, 邵赛赛. Spark Streaming: 大规模流式数据处理的新贵[J]. 程序员, 2014(2): 44 – 47. (XIA J L, SHAO S S. Spark Streaming: the upstart of large-scale streaming data processing [J]. Programmer, 2014(2): 44 – 47.)

[3] 胡俊, 胡贤德, 程家兴. 基于 Spark 的大数据混合计算模型[J]. 计算机系统应用, 2015, 24(4): 214 – 218. (HU J, HU X D, CHENG J X. Big data hybrid computing mode based on Spark [J]. Computer Systems & Applications, 2015, 24(4): 214 – 218.)

[4] 王舜燕, 黄芬, 刘万春. 基于 SaaS 模式的软件设计方法探讨[J]. 计算机与数字工程, 2008, 36(10): 102 – 105. (WANG S Y, HUANG F, LIU W C. Software design based on SaaS [J]. Computer & Digital Engineering, 2008, 36(10): 102 – 105.)

[5] 彭宏, 刘洋, 邓维维, 等. 股票数据流的相关性计算方法[J]. 华南理工大学学报(自然科学版), 2006, 34(1): 86 – 89. (PENG H, LIU Y, DENG W W, et al. Computing method of correlation of stock data streams [J]. Journal of South China University of Technology (Natural Science Edition), 2006, 34(1): 86 – 89.)

[6] DAS T, ZHONG Y, STOICA I, et al. Adaptive stream processing using dynamic batch sizing [C]// SOCC'14: Proceedings of the 2014 ACM Symposium on Cloud Computing. New York: ACM, 2014: 1 – 13.

[7] LIAO X Y, GAO Z W, JI W X, et al. An enforcement of real time scheduling in Spark Streaming [C]// IGSC'15: Proceedings of the 2015 Sixth International Green and Sustainable Computing Conference. Washington, DC: IEEE Computer Society, 2015: 1 – 6.

[8] 吴杰谦, 严然, 栾钟治, 等. 云计算环境下资源动态分配方法研究[C/OL]//2013 全国高性能计算学术年会论文集. 桂林: 中国计算机学会, 2013: 677 – 680. [2016-10-25]. http://www.docin.com/p-1205736858.html. (WU J Q, YAN R, LUAN Z Z, et al. Research on dynamic resource allocation in cloud [C/OL] // Proceedings of the 2013 China High Performance Computing Annual Meeting. Guilin: China Computer Federation, 2013: 677 – 680. [2016-10-25]. http://www.docin.com/p-1205736858.html.)

[9] 李锋刚, 魏炎炎, 杨龙. 基于和声算法异构 Hadoop 集群资源分配优化[J]. 计算机工程与应用, 2014, 50(9): 98 – 102. (LI F G, WEI Y Y, YANG L. Computing resource optimization in heterogeneous Hadoop cluster based on harmony search algorithm [J]. Computer & Digital Engineering, 2014, 50(9): 98 – 102.)

[10] ZAHARIA M, DAS T, LI H Y, et al. Discretized streams: fault-tolerant streaming computation at scale [C]// SOSP'13: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. New York: ACM, 2013: 423 – 438.

[11] KANG W, KAPITANOVA K, SANG H S. RDDS: a real-time data distribution service for cyber-physical systems [J]. IEEE Transactions on Industrial Informatics, 2012, 8(2): 393 – 405.

This work is partially supported by the Key Projects of Hubei Province Natural Science Foundation (2014CFA050).

LIU Bei, born in 1993, M. S. candidate. His research interests include big data application, mobile Internet.

TAN Xinming, born in 1961, Ph. D., professor. His research interests include software engineering method, Internet of things technology and system.

CAO Wenbin, born in 1991, M. S. candidate. His research interests include mobile Internet, processing platform of big data environment.