https://doi.org/10.1051/jnwpu/20203830589

数据中心上异构资源的细粒度分配算法研究

汤小春, 符莹, 樊雪枫

(西北工业大学 计算机学院, 陕西 西安 710072)

摘 要:数据中心的出现,使得大数据分析任务被分散到不同的计算节点。随着 GPU 计算的广泛应用,如何为不同的计算框架合理分配异构计算资源是目前的研究热点。研究了传统大数据计算框架和 GPU 计算的特点,针对现有的集群资源管理和 GPU 管理模式,提出了一种集中式异构资源管理模型,计算节点负责本地资源管理和任务的执行和管理,资源管理中心统一管理各个计算框架。对于不同的计算框架,根据其使用 CPU 以及 GPU 资源的不同,设计并实现了一种混合主资源共享分配算法,通过计算不同框架对主资源的使用,优先从可用资源中为主资源使用率最小的框架分配资源,实现主资源在各个框架的公平共享,防止 CPU 任务过多而导致 GPU 资源"饥饿",或者反过来导致 CPU 资源"饥饿"的现象发生。通过实验验证,该分配算法在异构资源使用效率以及任务完成数量方面能提高 15%左右。

关 键 词:混合主资源;异构集群;资源共享;数据中心
中图分类号:TP31 **文献标志码**:A **文章编号**:1000-2758(2020)03-0589-07

由于从本地读取数据能够得到更大的性能提升,所以任务往数据存储节点迁移是大数据计算的一种趋势。然而,传统集群计算中资源分配采用静态方式,导致大量的数据需要远程访问,因而被越来越多的计算框架放弃。随着以 HDFS 为架构的数据存储中心的出现,各种各样的大数据计算框架不断涌现,如 MapReduce[1]、Dryad[2]、Spark[3]、Flink[4]等。但是,从现有的趋势看,由于无法使用一个通用的计算框架来满足所有的用户需求,这就造成了同一个企业或者组织内,存在同时运行多个计算框架的局面。为了减少基础设施的投入,一些企业或者组织开始使用资源管理系统,如 YARN[5]、Mesos[6]、Borg[7]、Omega[8]、Mercury[9]、Kubernetes[10]等,实现在各个计算框架之间共享集群计算资源。

另外,随着分布式环境下计算密集型计算任务以及视频数据处理需求的增加,如分布式机器学习领域等,这些新型计算框架通常采用 CPU 加速(GPU)的方式来提高性能。新型计算框架则需要一种包含 GPU 和 CPU 的集群计算资源,即异构集群资源。而现有的 Mesos、YARN 等集群资源管理主要考虑 CPU、内存和网络带宽资源的管理,对于 GPU,采用粗粒度资源分配方式,一次将整个 GPU 全部分配给任务。这种资源管理的缺点是,当任务只使用少量 GPU 资源(如,显存)时,由于任务对 GPU 的独占,其他任务就无法实现 GPU 资源的共享,浪费了部分 GPU 计算资源。因此,迫切需要设计和实现一种任务之间共享 GPU 的细粒度资源管理系统,提高 GPU 资源的使用效率。这种新的异构集群资源分配方法,它既能保证各种不同的计算框架在共享异构集群计算资源时,避免 CPU 任务分配过多而 GPU 资源"饥饿",或者 GPU 任务过多而 CPU 资源"饥饿",也能保证 GPU 计算资源在任务之间实现细粒度的共享,提高 GPU 计算资源的使用率。

论文的主要工作是提出了一种集中式异构集群细粒度资源分配方法,提高了集群计算资源的整体利用率。第一,设计了一种异构集群资源管理框架,满足不同计算框架对异构资源的共享;第二,提出了

收稿日期:2019-09-11
基金项目:科技部重点研发基金(2018YFB1003403)与西北工业大学硕士研究生创意创新种子基金(ZZ2019204)资助
作者简介:汤小春(1969—),西北工业大学副教授,主要从事大数据处理、云计算研究。

一种混合主资源公平(hybrid domain resource fairness,HDRF)分配模型,实现了 CPU 资源与 GPU 资源之间的合理分配;第三,通过 GPU 显存的分割,实现了 GPU 资源的细粒度分配,满足了 GPU 资源在不同任务之间的共享。

1 研究现状

现有的集群计算资源管理系统,主要针对 CPU 和内存,使用集中或者分布式资源管理模型。YARN 采用集中式管理,提供 FIFO、容量以及公平分配 3 种资源调度方式。其资源仅包含 CPU 核以及内存,无法管理 GPU 资源。文献[11]扩展了 YARN 的 GPU 资源管理方式,采用任务独占 GPU 资源,其缺点是缺乏 CPU、GPU 资源的混合共享。Mesos 使用 DRF[6] 实现资源在不同计算框架的公平分配。针对不同的计算框架,设置 CPU 核或内存作为主资源,在调度过程中,选择主资源使用量最小的计算框架,向该计算框架提供资源,以期实现集群资源的公平分配。算法的缺点是只考虑 CPU 以及内存资源,GPU 资源采用粗粒度分配方式,不利于 GPU 资源的共享。另外,算法采用串行方式分配资源,存在延迟情况。Omega 解决了 Mesos 的串行分配的缺点,但是需要处理分配冲突,而且尚未发现有 GPU 资源的管理。分布式资源管理框架 Mercury 很好地解决了实时任务的延迟分配问题,但是,不支持 GPU 资源的管理。Kubernetes 虽然实现了 GPU 资源的管理,但是 GPU 资源亦采用独占方式。Borg 使用集中模型,尚未发现对 GPU 资源的管理。文献[11]、Mesos 以及 Kubernetes 等资源管理系统,由于只是按照 GPU 卡的个数来管理 GPU 资源,因此很难实现 GPU 资源的共享,即实现细粒度的 GPU 资源管理。本文中,通过将 GPU 显存引入集群资源管理范围,通过 GPU 可用显存的大小来调度任务,从而实现了 GPU 细粒度的资源分配。

针对单纯的 GPU 集群[12-13],现有资源管理系统主要研究如何构建 GPU 集群并管理 GPU 资源,这些系统按照日历或者静态方式分配 GPU 资源,无法满足不同计算框架之间的动态资源分配。而本文提出的异构集群资源管理策略,既可以管理 CPU 资源,又可以管理 GPU 资源,并且实现了不同计算框架之间的动态资源分配。

2 异构集群资源管理模型

本文主要针对 CPU-GPU 的异构集群,不涉及性能配置等异构的情况。首先给出异构数据中心上资源管理系统的整体结构。然后,介绍计算框架请求资源并提交任务的运行过程。最后给出了细粒度任务的共享资源分配算法。

2.1 系统结构

异构数据中心资源管理系统由计算节点和资源管理中心组成,如图 1 所示。

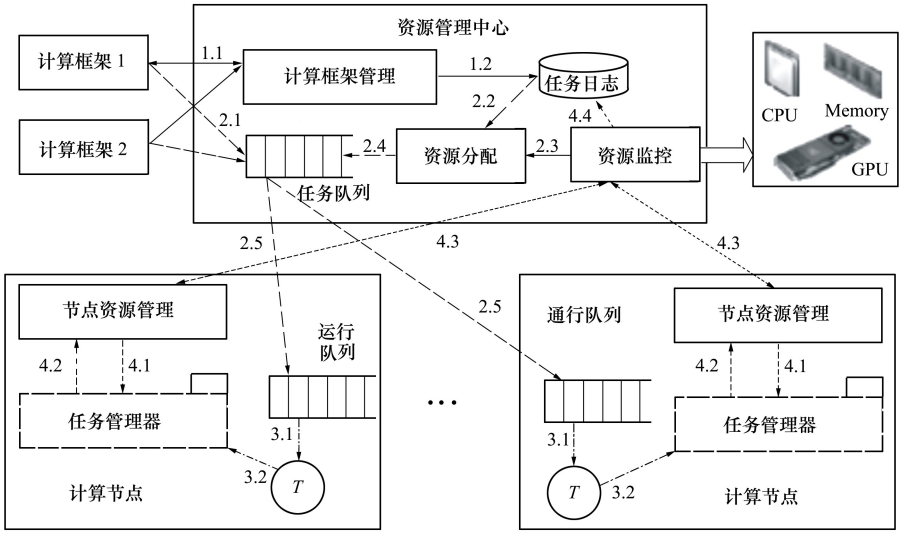

图 1 异构集群资源管理模型

计算节点由节点资源管理、任务管理器以及运行队列组成。它是任务执行的主体,负责进行节点的资源管理、任务执行调度和任务运行状态的控制,另外,计算节点向资源管理中心汇报任务的状态和节点资源信息。对于每个 GPU,按照 GPU 显存的大小对 GPU 进行细粒度分割。计算节点向资源管理器汇报可用显存的大小,资源管理中心可以将剩余的 GPU 显存分配给其他任务,实现任务对 GPU 的细粒度共享。

资源管理中心由计算框架管理、任务队列、任务日志、资源分配以及资源监控组成。计算框架管理会管理计算框架生命周期内的注册、日志登记、运行过程控制和退出的整个过程;任务日志完成计算框架对应的状态信息的记录,包括创建框架日志文件、所有任务的 DAG 图、任务的预想开始时间、预想结束时间、资源需求信息以及数据要求信息;任务队列中包含所有的排队状态的可执行任务,将任务提交到对应的计算节点运行队列;并从资源监控获得集群节点资源信息,然后将资源公平地分配给任务队列中的任务;资源监控收集并记录各个集群节点可使用的计算资源,主要包括 CPU 核数、节点可用内存、GPU 卡的编号以及对应的 GPU 显存大小、磁盘空间和网络带宽等信息。

2.2　资源管理运行过程

整个资源管理的运行过程按照以下步骤完成(以下的步骤序号与图 1 中序号对应)。

1) 计算框架注册:

步骤 1　计算框架运行后,其调度器向资源管理中心的计算框架管理器发送注册信息;

步骤 2　计算框架管理器将注册信息写入到日志记录中。

2) 任务的资源分配:

步骤 1　计算框架接收到注册回复消息后,向任务队列中提交自己的任务;

步骤 2　资源分配从日志记录中获得各个计算框架中已经结束的任务和正在执行的任务,计算出各个计算框架正在使用的资源信息;

步骤 3　从资源监控中获得可用资源信息;

步骤 4　将资源提供给任务队列中的任务;

步骤 5　任务队列根据资源分配结果,将任务转送到资源所对应的计算节点运行队列中。

3) 任务的执行:

步骤 1　运行队列负责将任务启动并执行;

步骤 2　任务的排队、开始、结束信息提供给计算节点的任务管理器。

4) 资源状态的更新:

步骤 1　节点资源管理启动任务管理器;

步骤 2　一旦任务的状态发生变化,任务管理器将信息汇报给节点资源管理器;

步骤 3　节点资源管理器将任务状态信息以及可用资源信息发送给资源监控;

步骤 4　资源监控更新日志记录中对应的任务的状态信息。

2.3　混合 DRF 资源分配算法

对于异构集群资源,本文提出混合 DRF 资源分配算法,该算法能够保证 CPU 计算框架和 GPU 计算框架共享计算资源。对于每个计算框架,除了计算其占用的 CPU 核以及内存的份额外,还需要单独计算其 GPU 资源使用。对于单纯的 CPU 计算任务,采用 CPU 的逻辑核作为分配单元;对于 GPU 计算,只需要较少的 CPU 资源完成初始化和启动,主要考虑 GPU 资源,为实现细粒度共享,任务需要的 GPU 显存大小需要确定,根据显存大小,就可以确定共享的任务的数量。综合考虑 GPU 资源显存份额,再考虑 CPU 以及内存的份额,实现这些资源份额在不同框架上的公平共享,即混合 DRF 算法。

集群中的资源包括 CPU 核数 V_C,内存大小 V_M,GPU 显存大小 V_G,集群全部资源表示为 $\boldsymbol{A}(V_C,V_M,V_G)$,已经使用的资源表示为 $\boldsymbol{R}(V_C,V_M,V_G)$,可用资源表示为 $\boldsymbol{A}-\boldsymbol{R}$。

计算框架表示为 f,计算框架中的任务表示为 t,每个任务使用的资源表示为 r。假如集群中有 n 个计算框架,可表示为 $\boldsymbol{F}=\{\boldsymbol{f}_1,\boldsymbol{f}_2,\cdots,\boldsymbol{f}_n\}$,其中 $\boldsymbol{f}_i$ 代表第 i 个计算框架。如果 $\boldsymbol{f}_i$ 中有 k 个任务,表示为 $\boldsymbol{T}^i=\{t_1^i,t_2^i,\cdots,t_k^i\}$,其中 t_j^i 表示计算框架 f_i 的第 j 个任务。任务 t_j^i 需要的计算资源表示为 $r_j^i=<V_C,V_M,V_G>$。

如果计算框架 $\boldsymbol{f}_i$ 中,有 p 个任务正在运行,那么其使用的资源总和表示为 $R_i(V_C,V_M,V_G)$,其中 CPU 核数为 $R_{i,V_C}=\sum_{l=1}^{p}r_{l,V_C}^i$;内存的使用量为 $R_{i,V_M}=\sum_{l=1}^{p}r_{l,V_M}^i$;GPU 显存总和为 $R_{i,V_G}=\sum_{l=1}^{p}r_{l,V_G}^i$。

正在运行的任务使用的资源量与系统总的资源量的比值称为资源比,用 s 表示。计算框架 $\boldsymbol{f}_i$ 的 CPU、内存和显存的资源比分别为 $s_c^i=R_{i,V_C}/\boldsymbol{A}_{V_C}$,

$s_m^i = R_{i,V_M}/\boldsymbol{A}_{V_M}$ 和 $s_g^i = R_{iV_G}/\boldsymbol{A}_{V_G}$。资源比中最大的记为主资源 s_i,即 $s_i = \max\{s_c^i, s_m^i, s_g^i\}$。

当系统的可用资源为 $\boldsymbol{A} - \boldsymbol{R}$ 时,资源分配器首先计算各个计算框架的主资源,选择主资源最小的框架作为优先分配可用资源的计算框架,然后再根据计算框架中未执行的任务的资源需求与可用资源的大小决定哪个任务获得计算资源。假如计算框架 f_i 的主资源 s_i 最小,计算框架 f_i 获得资源分配权力,从自己未执行的任务中选择一个可以符合以下要求的任务 t_j^i 执行。对于 CPU 任务,即,① 任务 t_j^i 的 CPU 核数需求小于 $\boldsymbol{A} - \boldsymbol{R}$ 中的 CPU 核数,$t_{j,V_C}^i \leqslant (\boldsymbol{A} - \boldsymbol{R})_{V_C}$;② 任务 t_j^i 的内存需求小于 $\boldsymbol{A} - \boldsymbol{R}$ 的内存大小,$t_{j,V_M}^i \leqslant (\boldsymbol{A} - \boldsymbol{R})_{V_M}$;对于 GPU 任务,即,③ 任务 t_j^i 的 GPU 显存需求小于 $\boldsymbol{A} - \boldsymbol{R}$ 的显存大小,$t_{j,V_G}^i \leqslant (\boldsymbol{A} - \boldsymbol{R})_{V_G}$;当前计算框架没有任务满足要求时,可根据主资源选择下一个计算框架。

为了避免某些资源要求高的任务的“饥饿”问题,使用资源预约机制。在资源分配器中提前设置了一个超时时间 O_T,而对于每个任务都有一个等待资源的时间,记作 W_T,这个时间从任务提交到任务队列上开始计时,当任务等待资源的时间达到了超时时间,就为这个任务进行资源预留。一旦集群的可用资源满足该任务的预留资源要求,就分配资源。在资源分配时,资源分配器检查是否存在延迟超过的任务,如果有,该资源保留,直到为任务 t_j^i 预留到需求的资源为止。

算法 1:HDRF

输入:计算框架集合 $\boldsymbol{F}$,可用资源容量 $\boldsymbol{R}$,超时时间 $\boldsymbol{O}_T$

输出:计算框架的任务分配的资源容量

过程:

S1 $\boldsymbol{A} = <V_C, V_M, V_G>$ // 全部资源容量

S2 $\boldsymbol{R} = <V_C, V_M, V_G>$ // 已经使用的资源容量

S3 $\boldsymbol{F} = \{\boldsymbol{f}_1, \boldsymbol{f}_2, \cdots, \boldsymbol{f}_n\}$ // n 个计算框架

S4 $R_1, R_2, \cdots, R_n$ // n 个计算框架使用的资源

S5 $s_1, s_2, \cdots, s_n$ // n 个计算框架的主资源

S6 while($F \neq$ null) {

S6.1 computing s_c^i // 计算 CPU 比值

S6.2 computing s_m^i // 计算内存比值

S6.3 computing s_g^i // 计算显存比值

S6.4 computing s_i // 计算主资源

S6.5}

S7 if($\exists (t_{j,W_T}^i > O_T) \wedge t_{j,r}^i \geqslant \boldsymbol{A} - \boldsymbol{R}$) { // 存在超时任务

S7.1 reserve t_j^i;

S7.2 return;

S7.3}

S8 $f_l = \min\{s_1, s_2, \cdots, s_n\}$ // 选择主资源最小的计算框架

S9 if($\exists (\max(t_{j,W_T}^l) \wedge t_j^l \in f_l \wedge t_{j,r}^l \leqslant \boldsymbol{A} - \boldsymbol{R})$) { // 满足任务的资源要求

S9.1 update R, R_l

S9.2}

算法的 S1 到 S5 为设置初始值。S6 计算各个计算框架正在使用的资源比值。S7 检查延迟超时任务,S8 根据各个计算框架的主资源值,选择主资源最小的计算框架。S9 选择最等待资源时间最长以及资源需求小于可用资源的任务,并更新系统的使用资源量以及当前计算框架的资源使用量。

3 系统评价

系统在一个集群上进行试验,集群中包含 8 台 NF5468M5 服务器,每台服务器 2 颗 Xeon2.1 处理器,每个处理器包含 8 个核,32 GB DDR4 内存,2 块 RTX2080TI GPU 卡,10 GB 显存,总的资源容量为:128 核,256 GB 内存,160 GB 显存。1 台 AS2150G2 磁盘阵列,用于存储数据。

3.1 资源使用率的评价

测试时,使用了 2 个计算框架 JOB1 和 JOB2,JOB1 是字符串统计程序,其每个任务使用<2 core,2 GB内存>,其主资源为 CPU;JOB2 是完全 GPU 实现的矩阵乘法,其每个任务使用<1 core,4 GB 内存,4 GB 显存>,其主资源为 GPU。

图 2a)是 JOB1 中的任务随着时间变化而分配得到的 CPU 和内存资源曲线,图 2b)是 JOB2 中的任务随着时间变化而分配得到的 CPU、内存和 GPU 显存资源曲线,图 2c)是 JOB1 和 JOB2 随着时间变化而分配得到的主资源曲线。在前 1 min,JOB1 的主资源是 CPU,JOB2 的主资源为 GPU 内存,在后续的时间里面,每个作业都能够使用 60%左右自己的主资源,高于平均分配资源给不同的作业的方式。所以,从测试的数据看,使用混合主资源分配方法,可以很好地利用 CPU 资源和 GPU 资源。

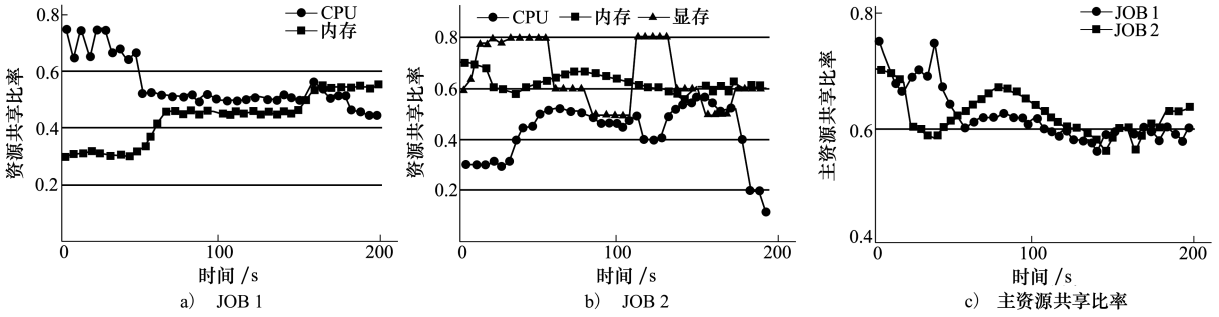

图 2　资源共享使用比率

3.2　任务执行数量的评价

接下来,本文评价了使用混合 DRF 资源分配方法与其他资源分配方法的比较,一个是先来先服务,另外一个是只使用 CPU 和内存的 DRF 方法。

实验中采用大小 2 种规模任务,小任务包含的计算资源:CPU 任务<1 core,0.5 GB>,GPU 任务<2 core,2 GB 内存,2 GB 显存>;大任务包含的计算资源:CPU 任务<2 cores,2 GB>,GPU 任务<1 core,4 GB内存,4 GB 显存>。同时运行 2 个计算框架,每个计算框架中包含任务数量为 80,一半是 GPU 任务,一半 CPU 任务。实验中 1 个计算框架完成后,继续提交相同类型任务,实验持续时间 10 min。实验结束后计算完成的任务数量,结果如表 1 所示。

表 1　不同分配模式下完成任务数量

任务数	大任务			小任务		
	FIFO	DRF	HDRF	FIFO	DRF	HDRF
CPU	2 700	2 140	2 749	5 401	5 013	5 421
GPU	2 683	2 858	2 753	5 212	5 109	5 463
总数	5 383	4 998	5 502	10 613	10 122	10 884

从表 1 的数据看,不管是大任务还是小任务,FIFO 调度方式中,CPU 任务完成的数量与 GPU 完成的大致相同,这是由于 FIFO 调度时不考虑任务对主资源的要求,2 种不同类型的任务数同样对待的结果。对于 DRF 来说,完成的 CPU 任务数明显比完成的 GPU 任务数少,这说明把 GPU 任务按照 CPU 或者内存来计算主资源时,它获得的计算机会较多,所以其完成的任务数较多。对于混合 DRF,完成的 GPU 任务数量与完成的 CPU 任务数大致相同,原因在于各自按照 CPU 和 GPU 作为自己的主资源,每种主资源占比相同,所以其完成的任务数大致相当。从总任务来看,虽然 FIFO 以及混合 DRF 中完成的 CPU 与 GPU 任务数大致相同,但是由于混合 DRF 考虑各自的主资源,使得每种主资源都得到最大的分配,故混合 DRF 中,总的完成任务数明显比 FIFO 中总任务数多出许多。这说明混合 DRF,使得不同类型的任务最大化自己的主资源,因而系统的总体资源得到更高效的利用。

3.3　粗细粒度 GPU 资源使用率的比较

针对实验中的任务,计算框架分别按照粗粒度和细粒度方式分配 GPU 计算资源,实验中 1 个计算框架完成后,继续提交相同类型的任务,实验持续时间 75 min。实验中的计算框架与任务与 3.2 中的相同,实验结束后,图 3 给出了粗细粒度资源调度下 GPU 资源使用率曲线。

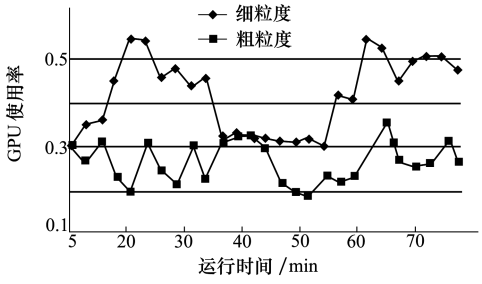

图 3　粗细粒度 GPU 使用率比较

从图 3 可以看出,当任务独占 GPU 时,GPU 利用率大约只有 30%左右;当根据 GPU 显存大小来进行细粒度 GPU 分配时,由于同一个 GPU 上存在多个任务对 GPU 的共享,数据从主机内存拷贝到显存以及从显存拷贝到主机内存是并行运行,间接提高了 GPU 中 kernel 函数的执行效率,GPU 使用率平均

达到 45%左右。因此,使用 GPU 资源的细粒度共享,可以提高 GPU 资源利用率。

4 结 论

在异构集群的细粒度资源分配算法中,计算资源的合理分配非常重要,对于 CPU 和 GPU 混合的计算框架,采用混合 DRF 是一种简单有效的资源分配方式。另外,GPU 与 CPU 之间的解耦至关重要,可保证异构集群的整体资源使用效率。

参考文献:

[1] DEAN J, GHEMAWAT S. MapReduce: Simplified Data Processing on Large Clusters[J]. Communication of the ACM, 2008, 51(1): 107-113

[2] ISARD M, BUDIU M, YU Y, et al. Dryad: Distributed Data-Parallel Programs Form Sequential Building[C]//Proceedings of the 2nd ACM SIGOPS/Euro Sys European Conference on Computer Systenm, 2007: 59-72

[3] ZAHARIA M, N. CHOWDHURY N M M, FRANKLIN M, et al. Spark: Cluster Computing with Working Sets[R]. Technical Report UCB/EECS-2010-53, 2010

[4] PARIS Carbone, ASTERIOS Katsifodimos, STEPHAN Ewen, et al. Apache Flink: Stream and Batch Processing in a Single Engine[J]. IEEE Data Engineering Bulletin, 2015, 38(4): 28-38

[5] HADOOP Project. Hadoop Capacity Scheduler[EB/OL].(2019-08-23)[2019-08-25]. http://hadoop.apache.org/common/docs/r1.2.9/capacity_scheduler.html

[6] HINDMAN B, KONWINSKI A, ZAHARIA M, et al. Mesos: a Platform for Fine-Grained Resource Sharing in the Data Center[C]//Implementation Berkety, CA: USENIX Association, 2011: 22-35

[7] VERMA A, PEDROSA L, KORUPOLU M, et al. Large-Scale Cluster Management at Google with Borg[C]//Tenth European Conference on Computer Systems, New York, 2015: 18-34

[8] SCHWARZKOPF M, KONWINSKI A, ABD-EL-MALEK M, et al. Omega: Flexible, Scalable Schedulers for Large Compute Clusters[C]//Proc of 8th ACM European Conference on Computer Systems, New York, 2013: 351-364

[9] KARANASOS K, RAO S, CURINO C, et al. Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters[C]//2015 USENIX Annual Technical Conference, Berkeley, 2015: 485-497

[10] ACURA P. Deploying Rails with Docker, Kubernetes and ECS[M]. Berkeley, Apress, 2016

[11] FUKUTOMI D, IIDA Y, AZUMI T, et al. GPUhd: Augmenting YARN with GPU Resource Management[C]//International Conference on High Performance Computing in Asia-Pacific Region, 2018

[12] MYEONGJAE J, SHIVARAM V, AMAR P, et al. Multi-Tenant GPU Clusters for Deep Learning Workloads: Analysis and Implications[EB/OL].(2018-05-13)[2019-08-10]. http://www.microsoft.com/en-us/research/publication/multi-tenant-gpu-clusters-deep-learning-workloads-analysis-implications/

[13] VOLODYMYR V, KINDRATENK O, JEREMY J, et al. GPU Clusters for High-Performance Computing[C]//2009 IEEE International Conference on Cluster Computing and Workshops, New Orleans, USA, 2009: 1-8

Fine-Grained Allocation Algorithm for Sharing Heterogeneous Resources in Data Center

TANG Xiaochun, FU Ying, FAN Xuefeng

(School of Computer Science and Technology, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: Data in a data center are stored dispersively. The data-oriented task computing disperses big data analysis tasks to different computing nodes. The extensive use of graphics processing unit (GPU) makes it urgent and important to study how to reasonably assign heterogeneous resources to different computing frameworks. We investigate the existing big data computing framework and the GPU computing. Based on the existing cluster resource management model and the GPU management model, we propose a hybrid heterogeneous resource management model that combines CPU resources with GPU resources. The computing nodes manage local resources and implement tasks; the resource management center concertedly manage various computing frameworks. We design and implement a hybrid domain resource sharing and allocation algorithm, which allocates the hybrid domain resources to computing frameworks according to the coordinated use of them so as to fairly share the hybrid domain resources among various computing frameworks and prevent the CPU from too many tasks but the GPU or CPU from resource "hunger". The experimental results show that the allocation algorithm can increase the use of heterogeneous resources and the number of completed tasks by around 15%.

Keywords: hybrid domain resource; allocation algorithm; heterogeneous resource; resource sharing; data center