doi:10.11887/j.cn.201506021 http://journal.nudt.edu.cn

分布式流体系结构及其编程模型与资源管理*

李 鑫[1,2,3],杨学军[1,2],徐新海[1,2]
(1.国防科技大学 计算机学院,湖南 长沙 410073;
2.国防科技大学 高性能计算国家重点实验室,湖南 长沙 410073;
3.中国人民解放军总参谋部第六十三研究所,江苏 南京 210007)

摘 要:利用互联网资源提供大数据计算服务面临着资源异构性、动态性与通信长延迟等方面的挑战,现有分布式计算模型仍存在一些不足。运用流计算模型提出分布式流体系结构,包括分布式流编程模型与资源管理等,能够高效支持多种并行执行模式。在10个CPU-GPU异构结点上实现了原型系统,仿真实验验证了7个不同的测试用例。实验结果表明,与本地串行计算相比,分布式流体系结构可以平均提高39倍计算性能,具有较大的应用潜力。

关键词:流体系结构;大数据;编程模型;分布式计算
中图分类号:TP338.8 **文献标志码**:A **文章编号**:1001-2486(2015)06-110-06

Programming model and resource management of distributed stream architecture

LI Xin[1,2,3], *YANG Xuejun*[1,2], *XU Xinhai*[1,2]
(1. College of Computer, National University of Defense Technology, Changsha 410073, China;
2. State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China;
3. The 63rd Research Institute of PLA General Staff Headquarters, Nanjing 210007, China)

Abstract: While providing big data computing services using Internet resources, there remains a big challenge to researchers, including heterogeneity of Internet resources, dynamics of Internet resources and long latency of Internet communication. Current influent distributed computing models still have some shortage. A novel distributed stream computing model was proposed based on the traditional stream computing model, including the distributed stream programming model and resource management can efficiently support multiple parallel execution modes. The prototype system implemented on the 10 CPU-GPU heterogeneous nodes. Seven different benchmarks used in the simulation experiment. The experimental result shows that the distributed stream architecture can achieve the speedup of at least on average over the local serial computing, with significant potential for applications.

Key words: stream architecture; big data; programming model; distributed computing

近年来,“大数据”已经成为国际社会普遍关注的热点,在金融、军事、电信等领域引起了人们的高度重视。大数据具有数据量大、种类繁多、处理速度快与价值密度低等4V特征。利用互联网资源构建面向大数据计算的运行环境具有较好的发展前景,这种方法不仅能降低计算成本与提高资源利用率,还能提供可扩展的计算能力,但是却需要应对互联网资源异构性、动态性、通信长延迟与有限带宽等挑战。

目前,主流的分布式计算模型仍然存在一些不足。云计算[1]主要基于虚拟化技术等提供弹性可扩展的快速服务部署能力,提供大数据架构基础设施的运行环境。Hadoop等主流大数据技术能应对多种应用场景,如MapReduce[2]模型等特别适合松耦合的大规模数据处理应用的批处理过程;Spark[3-4]等适合于机器学习等内存迭代计算;Spark Streaming[5]等比较适合于数据规模庞大且不可预知或者实时动态产生的流式数据处理;Dremel[6]与Impala[7]等交互式处理技术主要是在数据量非常大的情况下提供实时或准实时的数据查询分析能力;PowerGraph[8]等图计算技术则是面向互联网社交网络等大规模图相关的应用场

* 收稿日期:2015-09-06
基金项目:国家自然科学基金资助项目(61221491,61303071)
作者简介:李鑫(1984—),男,安徽安庆人,博士研究生,E-mail:xinli@nudt.edu.cn;
杨学军(通信作者),男,教授,博士,博士生导师,E-mail:xjyang@nudt.edu.cn

景,它们主要支持非结构化数据与半结构化数据的大规模处理,通常运行在数据中心较稳定的大规模同构资源上,但是在支持互联网资源异构性以及动态性上还存在一些不足。网格计算[9]采用类 MPI 编程模型,在支持动态负载均衡上还有待改进。P2P 计算模型[10]提供的大数据处理方式较为简单,计算任务并行性较好,通常处理流程较简单。

近年来,流计算模型已经成功应用在高性能计算、媒体类应用等领域[11-13],并取得了"天河"等广泛而有影响力的应用成果[14-15]。流计算模型具有计算资源普适性、高度数据并行性、延迟计算绑定特性以及流水线并行性等特点,有很大的潜力应对上述互联网多方面的挑战。因此,李鑫等基于流计算模型首次提出了一种新型的分布式流体系结构(Distributed Stream Architecture,DSA)以试图解决互联网环境下大数据计算模型的科学问题,并设计实现了分布式流体系结构编程模型 Brook#与资源管理。

1　分布式流体系结构

分布式流体系结构首次将流处理思想引入分布式领域,扩展了传统流计算模型的概念,将可用的软硬件计算对象定义为 Kernel(计算核心),同时将计算数据与控制状态数据定义为 Stream(流或数据流),其基本概念如下:

控制数据流(ControlStream):控制计算流程的数据或状态数据;

计算数据流(ComputeStream):封装计算核心并行处理的数据;

软计算核心(SoftKernel,SK):封装计算核心程序信息的对象,其元信息包括软件共享库名称、网络位置等;

硬计算核心(HardKernel,HK):封装结点内可用计算结点硬件资源信息的对象,其元信息包括网络地址、处理器类型、线程数目等;

应用计算核心(ApplicationKernel,AK):封装应用程序中主程序代码相关信息的对象,负责申请获取资源,管理与监控计算任务运行;

客户管理计算核心(Client Management Kernel,CMK):提供用户查询和请求服务的接口;

资源管理计算核心(Resource Management Kernel,RMK):提供命令解释器与执行器的功能,负责向 SMK 注册本地资源信息;

服务管理计算核心(Service Management Kernel,SMK):提供应用服务等功能,负责维护服务(查询、添加、删除、更新等)、Kernel(HK,SK,AK 与 RMK)与用户(CMK)的元信息,并负责调度软硬件资源。

如图 1 所示,分布式流体系结构的资源管理采用了主从架构,SMK 负责统一维护资源元信息与资源调度,RMK 负责命令解释与执行,CMK 用于提交应用程序请求。运行一个完整的分布式流应用程序通常采用 1 个 host 结点用于管理执行环境与多个 device 结点用于执行计算核心程序。其中:host 结点上运行主程序(AK),负责管理整个应用执行流程;device 结点上运行普通计算核心程序(SK)。AK 与 SK 均由 RMK 负责启动执行。MPEG2 编码应用的复杂处理流程共包括 1 个 AK 与 7 个 SK 计算核心。

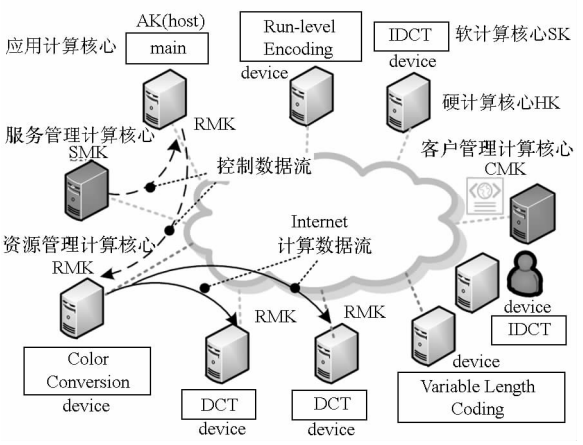

图 1　分布式流体系结构基本概念与 MPEG2 编码应用

Fig. 1　Basic concepts of DSA and MPEG2 encoder application

host 结点上的 RMK 负责启动一个线程来管理应用主程序(AK)。当该线程执行到一个计算核心时,它会根据编译指导命令划分原计算任务为若干子任务并行执行,创建管理该任务及其子任务的线程,并向 SMK 申请执行该计算核心的软硬件资源,通知计算结点 RMK 下载代码与数据,计算完毕后由主程序(AK)更新维护数据一致性。主程序(AK)如此推进计算过程直至完成整个任务。

2　分布式流体系结构编程模型 Brook#

2.1　基本概念

Brook#在计算执行过程中流与计算核心的并行度划分为四种 Kernel 执行模式,如图 2 所示。

单个计算核心单个流(Single Kernel Single Stream,SKSS):即在一个计算结点上执行一个 Kernel 计算核心任务,处理单一流,依靠开发结点内处理器的并行性来提升计算能力;

单个计算核心多个流(Single Kernel Multiple

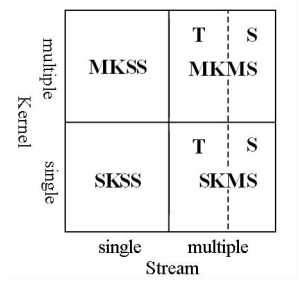

图2 Kernel 并行执行模式划分图

Fig. 2 Parallel execution modes of Kernel

Streams,SKMS):即多个计算结点执行相同的 Kernel 代码以完成一个计算核心任务,并处理不同的流,通过空间并行(SKMS-S)或时间并行(SKMS-T)的方式提高性能;

多个计算核心单个流(Multiple Kernels Single Stream,MKSS):即多个计算结点上执行多个 Kernel 以流水线方式处理同一个流,通过时间并行的方式隐藏通信延迟,从而提高性能;

多个计算核心多个流(Multiple Kernels Multiple Streams,MKMS):即多个计算结点上同时执行不同的 Kernel 代码且处理不同的流,包括空间并行性(MKMS-S)与时间并行性(MKMS-T)两种并行方式。MKSS 是 MKMS 的一种特例。

2.2 编译指导语句

Brook#继承了传统流编程模型 AMD ATI Stream SDK 1.4 Beta 的 Brook + 编程模型全部语法规范,并开发扩充了具有分布式语义的语法元素。见表1,Brook#允许程序员使用编译指导语句显式的指明代码区域 Kernel 执行模式及相关属性,采用 C 和 C + + 标准提供的 pragma 机制,并提供三类形式:parallel_mode,distribute 与 barrier,同时无须关心资源异构性、动态负载均衡等底层细节。

表1 Brook#编译指导语句表

Tab. 1 Brook# compilation directives

序号	编译指导语句
1	#pragma brs **skms_s**(***n***) start/finish
2	#pragma brs **skms_t**(***m***) start/finish
3	#pragma brs **distribute**[**clause**①, …]
4	#pragma brs **skms_t**(***n***) | **skms_s**(***m***) **distribute** [**clause**, …]
5	#pragma brs **mkss** start/finish
6	#pragma brs **mkms_t** start/finish
7	#pragma brs **mkms_s** start/finish
8	#pragma brs **barrier all** | **kernel**

①**clause** 语法形式是 in/out | streamName[(BLOCK/*(*n*),…),BLOCK/CYCLE(*n*)]|。

2.2.1 parallel_mode 语句

程序员可以使用两个 parallel_mode 编译指导语句包围住代码区域,并指明其单个或多个 Kernel 的并性执行模式 parallel_mode,即

#pragma brs parallel_mode(*n*) start

Kernel/Multiple Kernels

#pragma brs parallel_mode finish

其中,*n* 表示子任务数。

2.2.2 distribute 语句

由于子任务映射到输入流与输出流上的数据在维度上可能是不同的,所以,每个子任务映射的数据划分方式也可能不一样。程序员需要根据经验与实际情况灵活配置流的划分方式与任务映射方式,达到高效计算的目的,即

#pragma brs distribute [**clause**, …]

Kernel

其中,clause 的语法形式如表1中注释①所示,它表示输入输出流的基本属性,包括流输入或输出方向、流名称、流划分方式以及与子任务的映射方式。

Brook#中流划分方式包括块分布(BLOCK)和缺省不划分(*)两种方式,其中,块分布(BLOCK)是指将流按照指定的维度均匀划分为 *n* 块。子任务映射方式包括块分布(BLOCK)与循环分布(CYCLE),其中块分布(BLOCK)是指同一个数据块映射到 *n* 个连续的子任务上,循环分布(CYCLE)则是将数据块依次循环映射到下标递增的子任务上。如图3所示,假设输入流的划分方式为 BLOCK(4),程序员指定启动8个子任务并行执行该计算核心。若流映射任务的方式为 BLOCK(2),则第1个数据块映射到子任务1与子任务2,说明这两个子任务的计算都需要输入流的第1个数据块,依次类推。若流映射任务的方式为 CYCLE,则说明子任务1与子任务5的计算需要第1个数据块。数据块映射子任务的方式是根据不同的程序执行特点而灵活指定的,需要

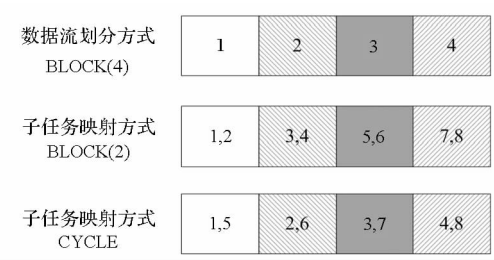

图3 流划分方式与子任务映射方式示例

Fig. 3 Example of data division method and sub-tasks mapping method for analyzing input streams

程序员了解应用程序的执行特点来配置完成。

2.2.3　barrier 语句

程序员可以使用 barrier 指定程序执行的同步点,实现对单个或多个 Kernel 的同步操作,以确保该同步点之前所有 Kernel 或指定 Kernel 必须执行完毕后才能继续执行程序,即

ProgramCode

#pragma brs barrier all | kernel

2.3　Brook#编译器运行时

分布式流体系结构编译器运行时的整体组织结构示意图如图 4 所示。当 RMK 接收到启动应用程序 AK 计算核心(host 主程序)的请求后,RMK 会创建执行线程(executor thread)负责执行 AK 程序,并调用编译器运行时库执行已经被翻译成具有指定 Brook#语义的多线程代码序列。执行线程负责创建任务工作线程(worker thread)与子任务工作线程(subworker thread)用于管理每个计算核心任务及其子任务的执行流程,与远程计算结点 RMK 进行交互,如流与代码的下载与发送、任务启动等。远程 device 结点上 RMK 接收到启动任务请求后,创建执行线程(SK 代码)启动子任务,调用本地设备(CPU 或 GPU 等)的 Kernel 函数代码进行计算,一旦结束就通知 host 结点上本任务工作线程(AK 程序)更新全局列表信息,以保证数据一致性。

图 4 中分布式流程序包括 matsum 与 matmul 两个计算核心,并分别被划分为 4 个子任务与 2 个子任务,并采用 MKMS-T 执行模式以流水线并行的方式执行。因此,编译器运行时在 host 结点上创建了 1 个 executor thread、2 个 worker thread 与 6 个 subworker thread 以及在 device 结点上创建了 6 个 subworker thread。

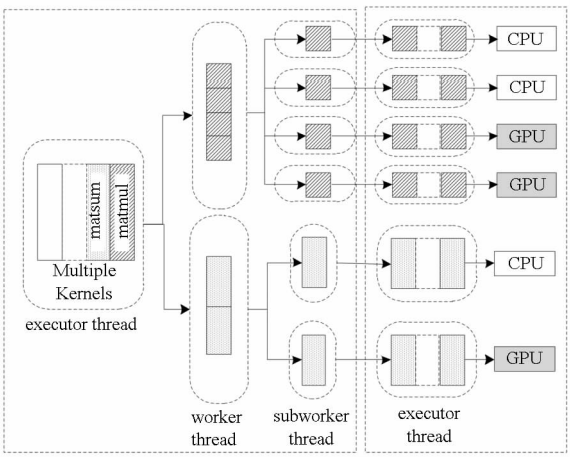

图 4　编译器 Brook#运行时整体组织结构示例

Fig. 4　Example of entire organization struture of Brook# runtime

3　分布式流体系结构资源管理

分布式流体系结构资源管理主要包括 SMK,RMK,CMK,AK,SK 等组件,如图 5 所示,其采用了主从架构以适应大规模资源的管理,具有良好的可扩展性,支持多任务独立运行。

SMK 负责维护结点资源元信息,包括对硬件、软件、服务以及用户等元信息的查询、添加、删除、更新等操作。如图 5 所示,RMK 与 CMK 启动后主动注册到 SMK 上,包括本地可用硬件信息,用户则将作业的计算核心代码(AK 与 SK)以及数据上传到资源结点并注册到 SMK。此外,SMK 还负责管理作业的生命周期过程,对资源请求进行合理的资源调度,以实现不同作业的安全隔离运行。

RMK 资源管理计算核心负责解释与执行请求的消息命令,是本地结点的资源管理器与任务执行器,管理本地可用硬件资源、作业文件资源与数据资源,并提供资源请求服务,同时管理与监控本地计算任务,周期性地向 SMK 汇报并更新本结点运行状态。

CMK 客户管理计算核心提供客户端的功能,一般部署在用户结点上,负责将程序代码(AK 或 SK)以及数据提交到资源结点上,并将应用程序或作业注册到 SMK,请求查询作业运行状态,并从资源结点上接收结果数据。

AK 应用计算核心是封装了作业主程序代码信息的对象,负责每个应用程序的具体任务执行过程,并采用了一种中间列表法的方式来维护数据一致性。AK 会维护记录输入输出流结点信息的 KernelList 与记录最近更新流的 Kernel 名的 StreamList。当主程序执行线程(AK)执行到一个计算核心时,AK 会主动向 SMK 申请资源分配给子任务,通过 StreamList 查询更新输入流的 Kernel 名,并在 KernelList 中查找其所在的结点信息,将这些信息发送给计算结点,通知 RMK 启动计算并监控任务状态。当任务完毕后,AK 更新 KernelList 中输出流的结点信息和 StreamList 中对应流的 Kernel 名,从而维护数据一致性。

4　实验验证

整个实验评估过程是在 10 个结点组成的互连网络(千兆以太网)上完成的,每个结点由 1 个六核 Intel Xeon X5670 与 1 个 AMD Radeon HD 4870 × 2 GPU 组成,操作系统为 64 位 Red Hat

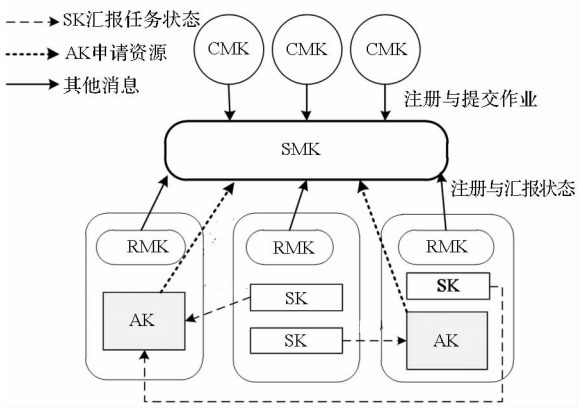

图 5　分布式流体系结构资源管理架构

Fig. 5　Framework of resource management of distributed stream architecture

Enterprise Linux 5.4,内存容量为 24GB。

选取 7 个典型测试用例(输入规模,用例来源),包括 MatrixMul(16 384 × 16 384 矩阵规模,AMD)、Blackscholes (30 000 000 个期权,PARSEC)、BinomialOption (8 192 000 个期权,AMD)、N-Body(499 968 个体,AMD)、MRI-Q(64 组 Large 数据,Parboil)、CP(100 000 个原子,Parboil)与 MatrixMulAdd(16 384 × 16 384 矩阵规模,AMD),并采用 Brook#将其移植到分布式流体系结构原型系统上,分别简记为 MM,BL,BO,NB,MQ,CP 与 MMA,均采用单精度浮点数。

实验采用的基准时间是本地 1 个 CPU 程序版本的串行执行时间,实验的对比程序设置了三组测试。第一组测试使用本地 1 个 GPU 的程序版本(LOCAL-1G),第二组与第三组测试都使用移植到分布式流计算体系结构上远程运行的程序版本,其中,第二组测试采用多 GPU 程序版本(DSA-G),第三组测试使用多 CPU 程序版本(DSA-C)或混合使用多 CPU 与多 GPU 的程序版本(DSA-C/G)。通过对比各组程序相对基准程序时间的加速比来评估在互联网模拟通信延迟与有限带宽等环境下分布式流体系结构的有效性。结点内的通信方法主要是基于 PCIE 协议与 GPU 等设备进行数据传输,结点间的通信方法是基于 TCP/IP 协议进行互联网模拟通信,其中,国际互联网延时采用 Internet Traffic Report 网站统计的 2015 年五大洲延迟时间平均值 100ms,国际互联网带宽采用 Speedtest 网站在 2013 年 186 个国家与地区测量带宽的 30 天移动平均值 13.98Mbps。

实验结果如图 6 与图 7 所示,图 6 显示了三组测试相对本地 1 个 CPU 的加速比实验结果,图 7 显示了第二组与第三组测试中测试用例通信时间、计算时间和其他时间开销与通信开销的归一化时间统计情况。

第二组测试中的 DSA-C 或 DSA-C/G 版本平均加速比达到 39 倍,第三组测试中的 DSA-G 版本平均加速比达到 58 倍,分布式流体系结构可以支持模拟互联网环境下测试用例的运行,能够利用异构资源与并行计算来提升性能,其中,第三组测试中的 MMA 采用了 MKMS-T 与 SKMS-S 执行模式的 DSA-C/G 版本,其余测试用例都采用了 SKMS-S 执行模式的 DSA-C 或 DSA-C/G 版本,以挖掘程序潜在的线程级并行性与任务级并行性。

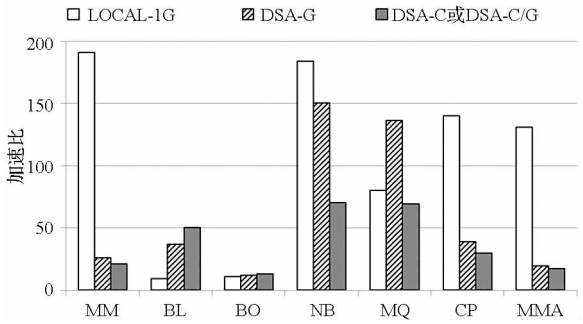

图 6　相对本地 1 个 CPU 执行时间的加速比实验结果

Fig. 6　Speedups of experimental results over execution time of a local CPU

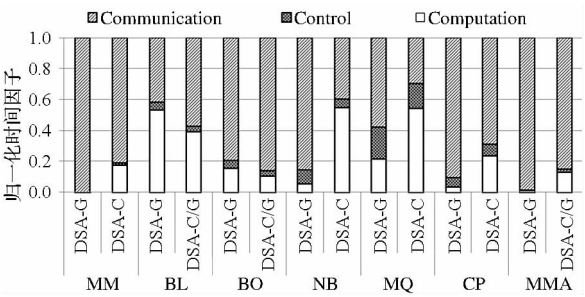

图 7　测试用例通信时间、计算时间以及其他时间的归一化实验结果

Fig. 7　Normalized experimental results of communication time, computing time and other time of benchmarks

基于多 GPU 的 DSA-G 或 DSA-C/G 版本的加速效果并不比 LOCAL-1G 版本显著,这是由互联网中通信问题的复杂性等造成的。从实验结果看,DSA-G 或 DSA-C/G 由于是远程运行的应用程序,在互联网模拟环境有限带宽和长延迟等情况下,计算过程必然带来不可忽视的通信开销,使得通信开销成为影响性能的因素。因此,这使得加速效果不如本地 LOCAL-1G 版本。

如图 6 所示,7 个典型测试用例在三组测试上的结果表现出三类特点:

1)一般情况下,应用加速比呈现下降趋势,

如 MM,NB,CP 与 MMA。其中,MM,NB 与 MMA 加速比下降得较快。这是由于分布式环境引入了大量不可忽视的通信与控制开销,其中通信开销平均达到 82%,显著增加了执行时间,这使得并行计算方式带来的性能提升不足以补偿通信开销造成的性能损失,从而造成加速比下降。

2)加速比呈上升趋势,如 BL 与 BO。其中,BL 的 DSA-G 与 DSA-C/G 加速比分别增长了 3.2 倍与 4.8 倍,BO 的 DSA 版本与本地 GPU 版本具有相当的加速比。虽然通信带来较大开销,但在这些应用中引入的通信开销与计算开销相当,图 7 中 BL 的通信开销比例平均达到 50%,异构资源强大的并行计算能力使得加速比保持上升。虽然 BO 通信开销比例较高,但是相对其他测试组并不大,因此,使得加速比保持缓慢上升。

3)MQ 加速比在 DSA-G 版本中上升,但在 DSA-C 版本中下降。这是由于前者利用 GPU 并行计算能力带来的性能优势能够补偿通信开销造成的损失,而 DSA-C 版本则不足以补偿这些性能损失,增大的程序控制开销也影响了性能,造成加速比下降。

由此可见,在分布式流体系结构中应用程序的计算开销、通信开销与控制开销之间的优化对于程序运行性能有至关重要的影响。

5　结论

分布式流体系结构能够较好地适应互联网资源特点,提供高效的分布式编程模型与资源管理,支持多种计算核心并行执行模式,而无须关心资源异构性、动态资源绑定等细节,具有较大的大数据计算应用潜力。

参考文献(References)

[1] Mell P M, Grance T. The NIST definition of cloud computing[R]. NIST, 2011.

[2] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [C]// Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation, 2004: 147 - 152.

[3] 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.

[4] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets [C]// Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. USENIX Association, 2010,10:10.

[5] Zaharia M, Das T, Li H, et al. Discretized streams: fault-tolerant streaming computation at scale [C]// Proceedings of the 24th ACM Symposium on Operating Systems Principles, ACM, 2013: 423 - 438.

[6] Melnik S, Gubarev A, Long J J, et al. Dremel: interactive analysis of web-scale datasets [C]// Proceedings of the VLDB Endowment, 2010, 3(1): 330 - 339.

[7] Erickson J. Impala: a modern SQL engine for Hadoop [R]. Tech Report, 2013.

[8] Gonzalez J, Low Y, Gu H. Power graph: distributed graph-parallel computation on natural graphs [C]// Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation, USENIX Association, 2012: 17 - 30.

[9] Foster I, Kesselman C. The grid 2: blueprint for a new computing infrastructure [M]. 2rd ed. USA: Morgan Kaufmann Publishers Inc., 2003.

[10] Kamvar S D, Schlosser M T, Garcia-Molina H. The EigenTrust algorithm for reputation management in P2P networks [C]// Proceedings of the 12th International World Wide Web Conference, ACM, 2003: 640 - 651.

[11] Yang X J, Yan X B, Xing Z C, et al. A 64-bit stream processor architecture for scientific applications [C]// Proceedings of the 34th Annual International Symposium on Computer Architecture, San Diego, USA, ACM, 2007: 210 - 219.

[12] Kapasi U J, Dally W J, Rixner S, et al. The imagine stream processor [C]// Proceedings of the 20th IEEE International Conference on Computer Design: VLSI in Computers and Processors, IEEE, 2002: 282 - 288.

[13] Ye Y, Li K L, Wang Y, et al. Parallel computation of Entropic Lattice Boltzmann method on hybrid CPU-GPU accelerated system [J]. Computers & Fluids, 2015, 110: 114 - 121.

[14] Xue W, Yang C, Fu H H, et al. Ultra-scalable CPU-MIC acceleration of mesoscale atmospheric modeling on Tianhe-2[J]. IEEE Transactions on Computers, 2015, 64(8): 2382 - 2393.

[15] Liao X K, Pang Z B, Wang K F, et al. High performance interconnect network for Tianhe System [J]. Journal of Computer Science and Technology, 2015, 30(2):259 - 272.