DOI:10.16836/j.cnki.jcuit.2014.01.001
第29卷第1期 成都信息工程学院学报 Vol.29 No.1
2014年2月 JOURNAL OF CHENGDU UNIVERSITY OF INFORMATION TECHNOLOGY Feb. 2014

文章编号:1671-1742(2014)01-0001-12

大数据技术中计算与数据的协作机制

王 鹏[1], 黄 焱[2,3], 刘 峰[1], 安俊秀[1]

(1.成都信息工程学院并行计算实验室,四川 成都 610225;2.中国科学院成都计算机应用研究所,四川 成都 610041;3.中国科学院大学,北京 100049)

摘要:大数据系统也被称为面向数据的高性能计算系统,与传统高性能计算系统相似,其计算和数据存储通常也是基于机群实现的分布式系统。以计算与数据的协作机制为主线分析对比了面向计算的高性能计算和面向数据的高性能计算,指出正是计算与数据的协作机制决定着大数据系统的基本结构和性能。分布式文件系统与计算通过协助机制的融合是大数据系统实现自动并行化的基础。与面向计算的高性能计算系统不同,大数据系统以切分数据并将计算向数据迁移作为协作机制的主要原则,实现对海量数据的自动并行批处理。元数据映射方法、哈希映射方法及流式拓朴方法是实现计算和数据协作的基本方法,特别是利用流式拓朴方法可以实现实时大数据处理。

关 键 词:面向数据;协作机制;面向计算;高性能计算;大数据

中图分类号:TP316 文献标志码:A

0 引言

信息技术的发展越来越清晰地呈现出两大主题——计算和数据,伴随这两大主题,信息技术领域出现了大数据这个技术概念,大数据技术的出现与云计算技术的出现几乎是同时的,早在1960年John McCarthy预言:“今后计算机将会作为公共设施提供给公众”[1] 这一云计算核心理念,1984年SUN公司董事会主席John Gage提出“网络就是计算机”[2] 这一具有云计算特征的论点,到网络的繁盛期2006年Google公司CEO Eric Schmidt提出云计算概念,再到2008年云计算概念全面进入中国,网络技术在云计算发展历程背后发挥了重要的推动作用。网络技术的发展促使服务向云端集中,并使数据量出现爆发式增长,因此面向数据成为云计算技术的重要特征之一,在一段时间里云计算和大数据两个概念甚至被当成同一个概念使用,一些大数据系统如Hadoop也被称为云计算系统,概念的模糊使不少人产生困惑。严格来讲,大数据技术是指针对海量数据的存储、分析和发布技术,而云计算是对资源和服务网络化提供方式的一种描述,两个概念之间的区别是很明显的。大数据系统是一种计算和数据都密集的分布式系统,计算需要为数据分析服务,也可以称之为面向数据的高性能计算,计算和数据的协作机制是研究这类系统的一个视角。

李国杰院士认为:“信息系统需要从数据围绕着处理器转改为处理能力围绕着数据转,将计算用于数据,而不是将数据用于计算”[3]。海量数据本身很难直接使用,只有通过处理的数据才能真正成为有用的数据,因此计算和数据两大主题可以进一步明确为数据和针对数据的计算,计算可以使海量数据成为有用的信息,进而处理成为知识。在大数据系统中存储不是一个独立存在的系统,特别是在集群条件下,计算和存储都是分布式的,如何让计算“找”到自己需要处理的数据是大数据系统需要具有的核心功能。面向数据要求计算是面向数据的,那么数据的存储方式将会深刻地影响计算实现的方式。这种在分布式系统中实现计算和数据有效融合从而提高数据处理能力,简化分布式程序设计难度,降低系统网络通讯压力从而使系统能有效地面对大数据处理的机制称为计算和数据的协作机制,在这种协作机制中计算如何找到数据并启动分布式处理任务的问题是需要重点研究的课题,在文中这一问题被称为计算和数据的位置一致性问题。

1 从面向计算到面向数据

1.1 计算机技术向大数据的演进历程

大数据系统架构的基本设计思想就是面向数据,面向数据可以更准确地称为“面向数据的计算”,要求系统的

收稿日期:2014-01-09
基金项目:国家自然科学基金资助项目(60702075);国家社会科学基金资助项目(12XSH019);中国博士后科学基金资助项目(20090451420);广东省科技厅高新技术产业化科技攻关资助项目(2011B010200007);四川省青年科学基金资助项目(09ZQ026-068)

设计和架构是围绕数据为核心展开的，而计算与数据的有效协作是面向数据的核心要求。

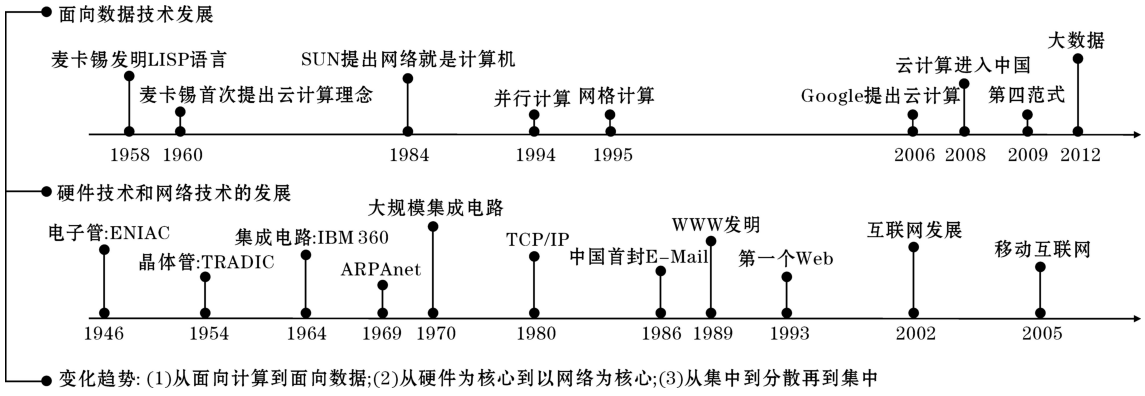

图 1 计算技术向大数据的演进

回顾计算机技术的发展历程，可以清晰地看到计算机技术从面向计算逐步转变到面向数据的过程。这一过程的描述如图 1 所示，该图从硬件、网络和大数据的演进过程等方面以时间为顺序进行了纵向和横向的对比。

从图 1 可以看到在计算机技术的早期由于硬件设备体积庞大，价格昂贵，这一阶段数据的产生还是“个别”人的工作，这个时期的数据生产者主要是科学家或军事部门，他们更关注计算机的计算能力，计算能力的高低决定了研究能力和一个国家军事能力的高低，相对而言由于这时数据量很小，数据在整个计算系统中的重要性并不突出。这时网络还没有出现，推动计算技术发展的主要动力是硬件的发展，这个时期是硬件的高速变革时期，硬件从电子管迅速发展到大规模集成电路。1969 年 ARPANET 网络的出现改变了整个计算机技术的发展历史，网络逐步成为推动技术发展的一个重要力量，1989 年 Tim Berners-Lee 发明的万维网改变了信息的交流方式，特别是高速移动通信网络技术的发展和成熟使现在数据的生产成为全球人的共同活动，人们生产数据不再是在固定时间和固定地点进行，人们随时随地都在产生数据，微博、博客、社交网、视频共享网站、即时通讯等媒介随时都在生产着数据并被融入全球网络中。

从云计算之父 John McCarthy 提出云计算的概念到大数据之父 Gray 提出科学研究的第四范式，时间已经跨越半个世纪。以硬件为核心的时代也是面向计算的时代，那时数据的构成非常简单，数据之间基本没有关联性，物理学家只处理物理实验数据，生物学家只处理生物学数据，计算和数据之间的对应关系非常简单和直接。到以网络为核心的时代，数据的构成变得非常复杂，数据来源多样化，不同数据之间存在大量的隐含关联性，这时计算所面对的数据变得非常复杂，如社会感知、微关系等应用将数据和复杂的人类社会运行相关联，由于人人都是数据的生产者，人们之间的社会关系和结构就被隐含到所产生的数据中。数据的产生目前呈现出大众化、自动化、连续化、复杂化的趋势，大数据概念正是在这样的背景下出现，这一时期的典型特征就是计算必须面向数据，数据是架构整个系统的核心要素，这就使计算和存储的协作机制研究成为需要重点关注的核心技术，计算能有效找到自己需要处理的数据可以使系统能更高效地完成海量数据的处理和分析。

1.2 第四范式——大数据时代的科学研究方法

信息技术领域提出面向数据的概念同时也开始深刻地改变科学研究的模式，2007 年著名的数据库专家 Gray 提出科学研究的第四范式。他认为利用海量的数据已可以为科学研究和知识发现提供除经验，理论，计算外的第四种重要方法。科学研究的 4 个范式的发展历程也同样反映了从面向计算走向面向数据的过程。

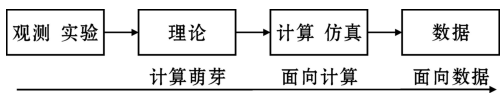

图 2 科学研究 4 个范式的发展历程

如图 2 所示，人类早期知识的发现主要依赖于经验、观察和实验，需要的计算和产生的数据都很少，人类在这一时期对于宇宙的认识都是这样形成的，就像伽利略为了证明自由落体定理，是通过在比萨斜塔扔下两个大小不一的小球一样，人类在那个时代知识的获取方式是原始而朴素的。当人类知识积累在一定程度后，知识逐渐形成了理论体系，如牛顿力学体系，Maxwell 的电磁场理论，人类可以利用这些理论体系去预测自然并获取新知识，这时对计算和数据的需求已经在萌生，人类已可以依赖这些理论发现新的行星，如海王星、冥王星的发现不是通过观测而是通过计算

得到。计算机的出现为人类发现新的知识提供了重要的工具,这个时代正好对应于面向计算的时代,这时可以在某些具有完善理论体系领域利用计算机仿真计算来进行研究,这时计算机的作用主要是计算,例如人类利用仿真计算可以实现模拟核爆这样的复杂计算。现在人类在一年内所产生的数据可能已经超过人类过去几千年产生的数据的总和,即使是复杂度为 $O(n)$的数据处理方法在面对庞大的 n 时都显得力不从心,人类逐步进入大数据的时代,第四范式说明可以利用海量数据加上高速计算发现新的知识,计算和数据的关系在大数据时代变得十分紧密,也使计算和数据的协作问题面临巨大的技术挑战。

2　机群系统中计算和数据的协作机制分析

回顾高性能计算的发展过程可以将高性能计算分为面向计算的高性能计算和面向数据的高性能计算。传统的高性能计算通常指面向计算的高性能计算,这种系统以实现高速的计算能力为目标;而面向数据的高性能计算以实现对海量数据的存储和处理为目标。由于他们都需要快速的计算能力所以这两类系统常常以机群方式实现强大的计算。在机群系统中实施计算都存在计算如何获得数据的问题,在面向计算系统中这一问题并不突出,在面向数据时代计算和数据的协作机制就成为必须考虑的问题,通常这种机制的实现与系统的架构有紧密的关系,系统的基础架构决定了系统计算和数据的基本协作模式,下面以常见的分布式机群系统为例对计算和数据的协作机制进行分析对比。

2.1　面向计算的高性能计算

面向计算的高性能计算系统出现在以硬件为核心的时代,从 Cray C-90 为代表的并行向量处理机[4] 发展到 IBM R50 为代表的对称多处理器机(SMP)[5] 最终到工作站集群(COW)及 Beowulf 机群结构,这一过程对应的正是CPU 等硬件技术的高速发展,可以采用便宜的工作站甚至通用的 PC 机来架构高性能系统,完成面向计算的高性能计算任务。

基于消息传递机制的并行计算技术 MPI(Message-Passing Interface)帮助工作站机群和 Beowulf 机群实现强大的计算能力,提供了灵活的编程机制。MPI 将大量的节点通过消息传递机制连接起来,从而使节点的计算能力聚集成为强大的高性能计算,主要面向计算密集的任务。MPI 提供 API 接口,通过 MPI_Send()和 MPI_Recv()等消息通讯函数实现计算过程中数据的交换。高性能计算是一种较为典型的面向计算的系统,通常处理的是计算密集的工作,因此在基于 MPI 的分布式系统中并没有与之匹配的文件系统支持,计算在发起前通过 NFS 等网络文件系统从集中的存储系统中读出数据并用于计算。基于 MPI 的分布式系统的典型系统结构如图 3 所示。

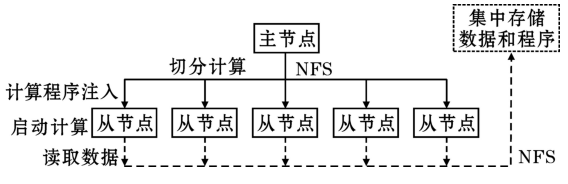

图 3　MPI 的典型系统架构

从图 3 知,典型的利用 MPI 实现的分布式计算系统在发起计算时,首先将计算程序由主节点通过 NFS 等网络共享文件系统分发到各子节点内存启动计算,由于没有分布式文件系统的支持,MPI 一般不能直接从节点存储设备上读取数据,计算程序在子节点发起后只有通过网络共享文件读取需要处理的数据来进行计算,在这里数据和计算程序一般都是被集中存储在阵列等专门的存储系统中。这一过程并没有计算寻找数据的过程,计算程序只是按设计要求先被分发给所有参与计算的节点。在进行 MPI 并行程序设计时,程序设计者需要事先将计算任务本身在程序中进行划分,计算程序被分配到节点后根据判断条件启动相应的计算工作,计算中需要进行节点间的数据交换时通过 MPI 提供的消息传递机制进行数据交换。由于 CPU 的运行速度远远大于网络数据传输的速度,通常希望不同节点间的任务关联性越小越好,在 MPI 的编程实践中就是"用计算换数据通讯"的原则,使系统尽可能少的进行数据交换。MPI 的消息传递机制为计算的并行化提供了灵活的方法,但目前对于任意问题的自动并行化并没有非常有效的方法,因此计算的切分工作往往需要编程人员自己根据经验来完成,所以这种灵活性是以增加编程的难度为代价的。

基于 MPI 的高性能计算是一种典型的面向计算的分布式系统,这种典型的面向计算的系统往往要求节点的计算能力越强越好,从而降低系统的数据通讯代价。MPI 的基本工作过程可以总结为:切分计算,注入程序,启

动计算,读取数据。MPI 虽然是典型的面向计算的分布式系统,但它也有类似于后来 Google 系统中的 MapReduce 能力,如 MPI 提供 MPI-Reduce()函数实现 Reduce 功能[6],只是没有像 GFS 这样的分布式文件系统的支持,MPI 的 Reduce 能力是相对有限而低效的,并不能实现计算在数据存储位置发起的功能。

通常将 MPI 这样以切分计算实现分布式计算的系统称为面向计算的高性能计算系统。这种系统计算和存储的协作是通过数据向计算的迁移实现,也就是说系统先定位计算节点再将数据从集中存储设备通过网络读入计算程序所在的节点,在数据量不大时这种方法是可行的,但对于海量数据读取这种方式会很低效。

2.2 面向数据的高性能计算

进入网络高速发展的时期,数据的产生成为了全民无时无刻不在进行的日常行为,数据量呈现出爆炸式增长,大数据时代到来,数据的作用被提到很高的地位,人们对数据带来的知识发现表现出强烈的信心。长期以来数据挖掘技术的应用一直都处于不温不火的状态,大数据时代的到来也使这一技术迅速地被再次重视起来,基于海量数据的挖掘被很快应用于网页数据分析、客户分析、行为分析、社会分析[7],现在可以经常看到被准确推送到自己电脑上的产品介绍和新闻报道就是基于这类面向数据的数据挖掘技术。基于数据切分实现分布式计算的方法被称为数据并行(data parallel)方法,但在面向计算时代真正的问题在于计算和数据之间只是简单的协作关系,数据和计算事实上并没有很好的融合,计算只是简单地读取其需要处理的数据而已,系统并没有太多地考虑数据的存储方式,网络带宽的利用率等问题。

通过数据切分实现计算的分布化是面向数据技术的一个重要特征,2003 年 Google 逐步公开了它的系统结构,Google 的 GFS 文件系统实现了在文件系统上就对数据进行了切分,这一点对利用 MapReduce 实现对数据的自动分布式计算非常重要,文件系统自身就对文件施行了自动的切分完全改变了分布式计算的性质,MPI、网格计算都没有相匹配的文件系统支持,从本质上看数据都是集中存储的,网格计算虽然有数据切分的功能,但只是在集中存储前提下的切分。具有数据切分功能的文件系统是面向数据的分布式系统的基本要求。

2004 年 Jeffrey Dean 和 Sanjay Ghemawat 描述了 Google 系统的 MapReduce 框架[8],与 MPI 不同这种框架通常不是拆分计算来实现分布式处理,而是通过拆分数据来实现对大数据的分布式处理,MapReduce 框架中分布式文件系统是整个框架的基础,如图 4 所示。这一框架下的文件系统一般将数据分为 64MB 的块进行分布式存放,需要对数据进行处理时将计算在各个块所在的节点直接发起,避免了从网络上读取数据所耗费的大量时间,实现计算主动"寻找"数据的功能,大大简化了分布式处理程序设计的难度。在这里数据块被文件系统预先切分是 MapReduce 能自动实现分布式计算的重要前提,系统通过主节点的元数据维护各数据块在系统中存储的节点位置,从而使计算能有效地找到所需要处理的数据。MapReduce 这种大块化的数据拆分策略非常适合对大数据的处理,过小的数据分块会使这一框架在进行数据处理时的效率下降。这一框架在获得良好的大数据并行处理能力的时候也有其应用的局限,MapReduce 框架在对同类型大数据块进行同类型的计算处理时具有非常好的自动分布式处理能力,但在数据较小、数据类型复杂、数据处理方式多变的应用场景却效率相对低下。为了实现 Google 系统良好的计算和数据的协作机制 GFS 和 MapReduce 是密不可分的,没有 GFS 支持单独的采用 MapReduce 是没有太大价值的。

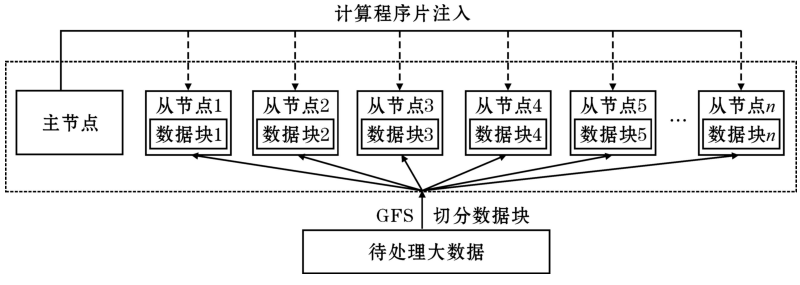

图 4 基于数据切分的分布式系统结构

MapReduce 框架使计算在机群节点中能准确找到所处理的数据所在节点位置的前提是所处理的数据具有相同的数据类型和处理模式,从而可以通过数据的拆分实现计算向数据的迁移,事实上这类面向数据系统的负载均衡在其对数据进行分块时就完成了,系统各节点的处理压力与该节点上的数据块的具体情况相对应,因此

MapReduce 框架下某一节点处理能力低下可能会造成系统的整体等待形成数据处理的瓶颈。在 MapReduce 框架下节点服务器主要是完成基本的计算和存储功能,因此可以采用廉价的服务器作为节点,这一变化改变了人们对传统服务器的看法。2005 年 Apache 基金会以 Google 的系统为模板启动了 Hadoop 项目,Hadoop 完整地实现了上面描述的面向数据切分的分布式计算系统,对应的文件系统为 HDFS[9],Hadoop 成为了面向数据系统的一个被广泛接纳的标准系统。类似的如 HPCC(High Performance Computing Cluster)系统则不是通过基于数据块的数据分割而是通过基于记录的数据分割来实现对数据的分布式计算,但进行数据分割的方法都是一样的。

同时数据分析技术是面向数据的高性能计算的研究热点。对类似于 Web 海量数据的分析需要对大量的新增数据进行分析,由于 MapReduce 框架无法对以往的局部,中间计算结果进行存储,MapReduce 框架只能对新增数据后的数据集全部进行重新计算,以获得新的索引结果,这样的计算方法所需要的计算资源和耗费的计算时间会随着数据量的增加线性增加。Percolator 是一种全新的架构,可以很好地用于增量数据的处理分析,已在 Google 索引中得到应用,大大提升 Google 索引更新速度[10],但与 MapReduce 等非增量系统不再兼容,并且编程人员需要根据特定应用开发动态增量的算法,使算法和代码复杂度大大增加。Incoop[11] 提出增量 Hadoop 文件系统(Inc-HDFS),HDFS 按照固定的块大小进文件划分,而 Inc-HDFS 则根据内容进行文件划分,当文件的内容发生变化时,只有少量的文件块发生变化,大大减少了 Map 操作量。

迭代操作是 PageRank、K-means 等 Web 数据分析的核心操作,MapReduce 作为一种通用的并行计算框架,其下一步迭代必须等待上一步迭代完成并把输出写入文件系统才能进行,如果有终止条件检查也必须等待其完成。同时,上一步迭代输出的数据写入文件系统后马上又由下一步迭代读入,导致了明显的网络带宽,I/O、CPU 时间的浪费。iHadoop 在分析了迭代过程存在的执行相关,数据相关,控制相关之后对潜在的可并行性进行挖掘,提出了异步迭代方式,比 Hadoop 实现的 MapReduce 执行时间平均减少了 25%[12]。Twister 对 MapReduce 的任务复用、数据缓存、迭代结束条件判断等进行调整以适合迭代计算,但其容错机制还很欠缺[13]。

Pregel 是 Google 提出专用于解决分布式大规模图计算的计算模型[14],适合计算 FaceBook 等社交关系图分析,其将处理对象看成是连通图,而 MapReduce 将处理对象看成是 Key-Value 对;Pregel 将计算细化到顶点,而 MapReduce 将计算进行批量化,按任务进行循环迭代控制[15]。

在分布式文件系统条件下数据的切分使对文件的管理变复杂化,因此此类集群系统下文件系统的管理和数据分析是需要进行重点关注的研究领域之一,面向数据的高性能计算系统就是大数据系统。

2.3　两种高性能计算系统的分析对比

表 1　两种高性能计算系统的对比

	面向计算的高性能计算系统	面向数据的高性能计算系统
系统典型架构	机群架构	机群架构
分布式计算的实现方法	计算拆分	数据拆分,批处理
典型的存储方式	集中存储	分布式存储
计算与数据的位置一致性关系	数据向计算迁移	计算向数据迁移
系统物理位置模式	集中	集中
节点性能要求	高	中
计算与数据协作机制	计算直接读取数据	一致性哈希,主节点元数据
并行程序开发难度	难	易
计算灵活性	高	低
应用场景	计算密集	数据密集
典型系统	MPI,高性能计算	Hadoop,Dyname,Cassandra,Google,HPCC

从面向计算发展到面向数据,分布式系统的主要特征发生了变化,表 1 对面向计算的高性能计算系统和面向数据的高性能计算系统进行了对比和分析。面向数据的高性能计算系统往往有对应的分布式文件系统的支持,从文件存储开始就实现数据块的划分,为数据分析时实现自动的分布式计算提供了可能,计算和数据的协作机制在面向数据的系统中成为了核心问题,其重要性凸现出来。

由于面向计算的高性能计算系统具有灵活和功能强大的计算能力,能完成大多数问题的计算任务,而面向数据的高性能计算系统虽然能较好地解决海量数据的自动分布式处理问题,但目前其仍是一种功能受限的分布式

计算系统，并不能灵活地适应大多数的计算任务，因此现在已有一些研究工作在探讨将面向计算的高性能计算系统与面向数据高性能计算系统进行结合，希望能在计算的灵活性和对海量数据的处理上获得良好的性能。文献[16]初步探讨了 MPI 和 Hadoop 结合问题，Amazon EC2 也发布了面向高性能计算的解决方案 CCI(Cluster Compute Instances)，文献[17]利用标准测试程序对比了在 Amazon EC2 CCI 上实现的云计算模式的高性能计算和在本机群上实现的高性能计算之间的性能。文献[18]讨论了将 MPI 应用于处理数据密集问题的可能性，将 MPI 的消息传递机制和 Hadoop RPC 进行对比，Hadoop RPC 使 Hadoop 具有消息传递机制，这使其分布式编程能力变得更加灵活，但目前来说与 MPI 相比还有一定的差距。文献[19]探讨了采用 MPI 的机制实现 MapReduce 的可能性。可以看到目前技术的发展正在使面向计算和面向数据的系统之间的界限越来越不明确，很难准确地说某一个系统一定是面向计算的还是面向数据的系统，数据以及面向数据的计算在大数据时代到来时已紧密地结合在一起。

3 实现计算和数据协作机制的方法

面向数据的系统通常是分布式系统，往往是计算向数据迁移从而降低数据在系统中传输的通信代价，实现计算寻找数据，定位计算的前提是定位数据，而且数据存储和切分的方式又会影响计算(数据分析)的处理效率和模式，因此实现计算和数据的有效协作首先需要研究数据在分布式文件系统中的存储方法，同时由于在分布式系统中需要解决数据的备份、冗余、节点失效处理等问题，这给研究计算和数据的协作机制提出巨大的挑战。由于计算和数据的位置一致性是协作机制的核心研究内容，下面主要从解决计算和数据位置一致性的角度进行讨论。

3.1 计算和数据位置一致性的映射模型

在分布式系统中计算和数据的位置一致性问题可以等效地理解为将计算和数据映射到同一个节点位置上，也就是说使计算在数据存储的位置发起。例如网格计算系统就是计算先于数据到达客户节点，数据根据客户端请求被映射到指定的客户端进行处理；Hadoop 系统是数据先于计算被存储于分布式系统的某一个节点，计算发起时通过元数据查询获得数据的存储位置，Map 任务被映射到相应的节点进行处理。因此可以把计算和数据的位置一致性问题抽象为如图 5 的映射模型。数据和计算的映射过程其实就是数据到节点的映射过程，计算程序片和数据片按照一定的映射规则定位到节点，将数据和计算注入节点，当集群节点发生失效时，数据片按规则进行数据迁移和备份，计算程序片则按照相应的规则重新映射到其对应的节点。

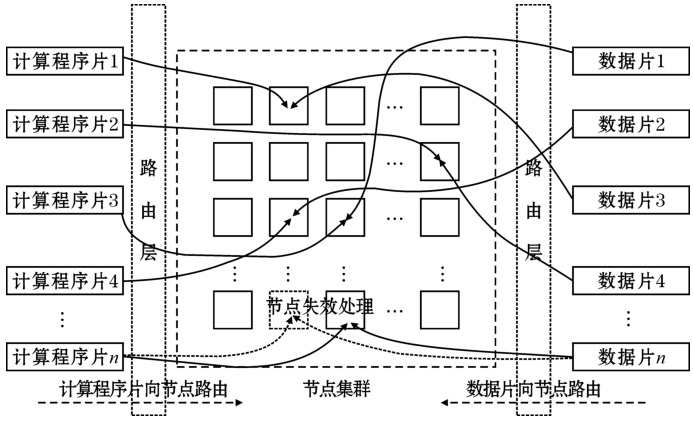

图 5 计算和数据位置一致性的映射模型

在这个模型中计算本身也被视为一种特殊数据，因为计算其实就是某种程序语言设计的可执行程序片，在被系统映射时可以与数据同等对待，而且计算程序中往往包含了其所要处理的数据的逻辑位置信息。分布式文件系统中定位数据块的算法其实就是起到了将数据映射到相应的节点上的功能。所以在前面讲到要实现计算和数据的位置一致性系统必须要有相应的分布式文件系统的支持。同时由于分布式系统存在数据冗余、计算迁移、存储迁移等问题，在具体实现时会与节点负载均衡调度算法、存储冗余技术(如副本策略、纠删码)[20]等技术相结合，实现一个计算和数据有效协作条件下的健壮稳定的高可用系统。相应典型的映射方法可以分为元数据映射

方法,哈希映射方法。

3.2　元数据映射方法

元数据映射方法是最容易想到的实现计算和存储位置一致性的方案,元数据方法通过在元数据库中保存数据块的存储位置,使计算按照元数据库中的位置被映射到指定的存储节点上。元数据方法实现数据和计算的定位非常类似于网络路由中的路由表,计算和数据通过查询路由表来保证计算和数据能被分配到同一个节点。采用元数据方法的分布式系统通常是主从结构,单点失效对系统的影响较为严重,GFS、HDFS 的结构就是采用元数据方法构建,在 Hadoop 中的 Namenode 就是负责存储元数据的管理节点,元数据方法系统在存储数据时的策略通常会根据各节点当前的存储负载来判断,为了避免主从结构对单节点失效的敏感,文献[21]通过元数据复制方法实现 Hadoop 系统的高可用性,Hadoop 也可以通过 Zookeeper 组件利用主备机方案来提升系统的可用性。

采用元数据方法可以较为容易地利用集群系统当前的工作状态作为依据实现分布式系统负载均衡,这时主节点会根据监控系统获得的数据利用一定的调度算法对数据的存储和计算的进行分配实现系统的负载均衡,并将相应的分配信息作为元数据进行保存。不少针对集群负载均衡的算法都可以用在元数据方法中作为主节点分配资源的依据,这类方法学者已进行了较为充分的研究。例如,文献[22]研究了由海量不同性能的 PC 构成的异构集群的负载均衡算法,使弱计算节点不再成为异构集群的性能瓶颈;文献[23]提出了一种基于分布式架构的动态自适应集群负载均衡算法,该算法具备在线负载预测机制,通过调整负载信息的采样方式,有效降低网络交换压力,提高算法响应速度;文献[24]提出了一种基于认知可信模型的动态优先级调度算法,确保任务在安全的环境中运行,提高了集群任务分配的成功率;文献[25]总结了 3 种分布式集群负载均衡算法:蜂群算法、随机抽样算法和动态聚集算法,比较分析了使用这 3 种算法时集群性能与节点数和节点性能的关系;文献[26]提出一种基于指数平滑预测的加权最小连接算法,根据系统的当前任务对集群的负载进行动态预测,提高集群的资源使用率;文献[27]提出了一种基于遗传算法的任务调度策略,将智能算法运用于集群调度。

元数据映射方法虽然在面对网络信息搜索这类大块数据应用时特别有效,也非常便于大量成熟的负载均衡算法的应用,但在面对有大量小文件的系统时由于元数据服务器需要维护大量的路由数据,查询的效率会变低。

3.3　哈希映射方法

哈希算法是一种从稀疏值范围到紧密值范围的映射方法,在存储和计算定位时可以被看作是一种路由算法,通过这种路由算法文件块能被唯一地定位到一个节点的位置。传统的哈希算法容错性和扩展性都不好,无法有效地适应面向数据系统节点的动态变化。1997 年 David Karger 提出了一致性哈希算法来定位数据[28],实现了机群系统在节点变化时的单调性,实现了较小的数据迁移代价。Amazon 的云存储系统 Dynamo 改进了基本的一致性哈希算法,引入虚拟节点,使系统具有更加均衡的存储定位能力。Facebook 开发的 Cassandra 系统也采用了一致性哈希算法的存储管理算法。一致性哈希算法及其改进算法已成为分布式存储领域的一个标准技术。使用一致性哈希算法的系统无需中心节点来维护元数据,解决了元数据服务器的单点失效和性能瓶颈问题,但对于系统的负载均衡和调度节点的有效性提出了更高的要求。

一致性哈希算法的基本实现过程为:对 Key 值首先用 MD5 算法将其变换一个长度 32 位的 16 进制数值,再用这个数值对 232 取模,将其映射到由 232 个值构成的环状哈希空间,对节点也以相同的方法映射到环状哈希空间,最后 Key 值会在环状哈希空间中找到大于它的最小的节点值作为路由值。

基于一致性哈希的原理可以给出计算和存储的一致性哈希方法,从而使计算能在数据存储节点发起。对于多用户分布式存储系统来说:“用户名+逻辑存储位置”所构成的字符串在系统中是唯一确定的,如属于用户 wang,逻辑存储位置为/test/test1.txt 的文件所构成的字符串“wang/test/test1.txt”在系统中一定是唯一的,同时某一个计算任务需要对 test1.txt 这个文件进行操作和处理,则它一定会在程序中指定用户名和逻辑位置,因此存储和计算 test1.txt 都利用相同的一致性哈希算法就能保证计算被分配的节点和当时存储 test1.txt 文件时被分配的节点是同一个节点。

现在以下面这个应用场景为例,说明一致性哈希算法实现计算和存储位置一致性的方法:

(1)面向相对“小”数据进行处理,典型的文件大小为 100MB 之内,通常不涉及对文件的分块问题,这一点与 MapReduce 框架不同;

(2)待处理数据之间没有强的关联性,数据块之间的处理是独立的,数据处理是不需要进行数据块之间的消息通讯,保证节点间发起的计算是低偶合的计算任务;

(3)程序片的典型大小远小于需要处理的数据大小,计算程序片本质上也可以看作是一种特殊的数据,这一假设在大多数情况下成立;

(4)数据的存储先于计算发生。

根据一致性哈希算法的基本原理在面向数据的分布式系统中计算和存储位置一致性方法如图 6 所示,其主要步骤如下:

(1)将服务器节点以 IP 地址用为 Key 值,以一致性哈希方法映射到哈希环上;

(2)在数据存储时以(用户名+文件逻辑位置)作为唯一的 Key 值,映射到哈希环上,并顺时针找到离自己哈希值最近的节点作为实际数据存储的位置;

(3)在发起计算任务时提取计算任务所要操作的数据对应的(用户名+文件逻辑位置)值作为 Key 值,映射到哈希环上,并顺时针找到离自己哈希值最近的节点注入程序并发起计算的节点。由于相同用户的相同数据其(用户名+文件逻辑位置)在一致性哈希算法作用下一定会被分配到相同的节点,从而保证了计算所发起的节点刚好就是计算所需要处理的数据所在的节点。

在这种算法的支持下只要计算程序片需要处理的数据逻辑位置是确定的,系统就会将计算程序片路由到数据存储位置所在的节点,这时节点间的负载均衡性是由数据分布的均衡化来实现。

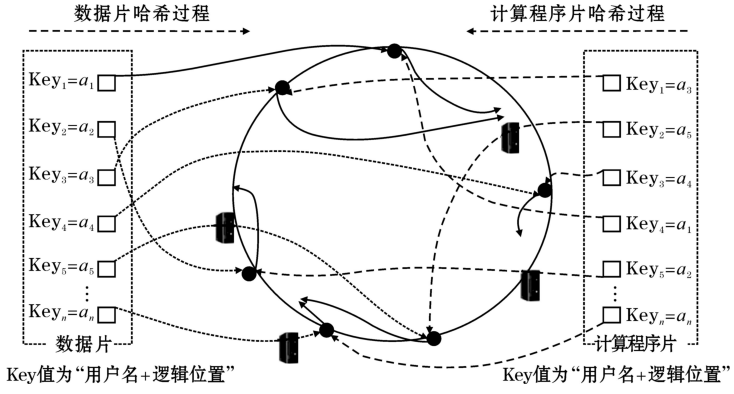

图 6 一致性哈希算法实现计算与数据的位置一致性

一致性哈希算法可以实现无中心节点的计算和数据定位,使计算可以唯一地找到其所要处理和分析数据,使计算能最大可能地在数据存储的位置发起,节约大量的网络资源,同时避免了系统单点失效造成的不良影响,利用一致性哈希方法在面对海量文件时系统不用维护一个庞大的元数据库用于保存文件的存储信息,计算寻找数据的速度非常直接,路由算法复杂度低。

4 计算和数据的流式拓朴协作机制

多数大数据系统(如 Hadoop, HPCC)的实现都是以非实时批处理方式进行的,在实时处理领域不能有效的发挥作用,实时大数据系统的出现填补了大数据系统在实时处理上的弱点,Storm 就是一种较为典型的实时大数据处理系统。

4.1 典型的流式分布式系统——**Storm** 系统

在高性能数据处理中流水线(pipelining)技术[29]是一项重要的并行技术,基本思想为:将一个任务 t 分成一系列有先后关系的子任务 t_1, t_2 …, t_m,在流水线模式中 t_i 任务的启动依赖于 t_{i-1} 任务的完成。对于数据具有强的先后相关性的数据分析任务十分适用。采用流式技术作为分布式系统计算和数据协作机制的框架,已越来越显示出其灵活性和生命力,与 Dynamo 和 MapReduce 等采用的技术形成鼎立的关系,微软发布 Dryad[30]就是将任务表示为一个有向无环图(Directed Acycline Graph, DAG)实现分布式任务设计,与其相似的开源实现 Storm

中采用的 Topology 也是这种模式, 本节以Storm 为例进行介绍。

Storm 是由 Twitter 推出的面向实时应用的流式分布式系统[31], 集群由一个主节点和多个工作节点组成, 主节点用于分配代码, 布置任务及故障检测。

如图 7 所示, Storm 要完成一个实时计算任务需要建立一个 Topology, Topology 对数据处理的逻辑计算规划, 在 Storm 系统中数据流的基本单位为元组 tuple, tuple 可以看作是一个被封装的数据结构, Storm 最高一级的执行单元就是 Topology, Topology 是由一个个计算节点构成的拓扑, 拓扑上的每一个节点完成一定的计算逻辑, 图中的箭头表示数据的流向。流水线技术也叫管道技术, 所以Storm 的设计者把数据流的生成器叫做 Spout, 把每一个处理位置叫做 Bolt。由于 Spout 是数据流的源头, Spout 读取数据并形成流传送给 Bolt, Bolt 可以接收任意多个输入流, 并对流中的数据进行特定的处理。相比在高性能计算领域传统的流水线并行化技术, Storm 采用 Topology 结构后使数据处理更为灵活功能更为强大。在 Storm 中主节点依据 Topology 的逻辑任务图分配 Bolt 任务, 最终的任务会被分配到相应的物理节点上。从 Storm 的架构上看, 在计算和数据协作机制的处理上 Storm 是由主节点依据 Topology 进行物理分配, 元组 tuple 数据流按 Topology 的描述逐步被相应 Bolt 节点上的计算程序所处理, 并由主节点将这一逻辑过程映射为物理节点的顺序。

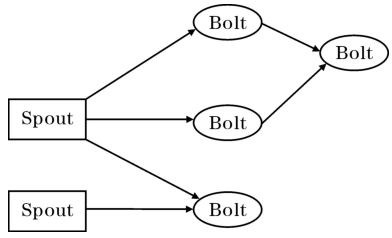

图 7　Storm 的 Topology 结构示意

4.2　计算和数据协作机制的流式拓扑映射模型

Storm 的系统结构提示利用类似 Topology 这样的逻辑结构可以灵活地实现非常复杂的分布式数据处理任务。图 8 将计算和数据协作机制的流式拓扑映射方法进行了抽象, 在这种方法中 Topology 相当于是对一个计算任务的逻辑规划, 并不直接对应于物理节点, 系统的主节点可能维护大量的这种 Topology 结构, 每一个 Topology 结构都相当于是处理某一个问题的逻辑规划。Topology 结构几乎可以描述大多数问题的处理方法。图中的操作相当于是Storm 系统的 Bolt, 数据发生器相当于是 Spout。系统主节点监控和管理着大量的处理节点, 对于每一个维护的 Topology 逻辑规划主节点都会依据一定的策略为其分配相应的物理节点以完成指定的计算任务。如图 8 中所示, 主节点为操作 1 分配物理节点 1, 为操作 2 分配物理节点 2, 为操作 3 分配物理节点 3, 为操作4 分配物理节点1, 这种分配完毕后 Topology 逻辑结构就被映射为集群中的物理结构, 并能实际地完成相应的计算任务。作为编程人员只需要定义问题的 Topology 逻辑, Topology 逻辑物理映射工作由主节点上的系统来维护, 程序设计人员不用担心节点的失效问题, 因为当某一操作对应的节点失效时, 主节点会将对应的操作重新映射给一个完好的物理节点, 从而保证整个 Topology 规划能顺利地执行。

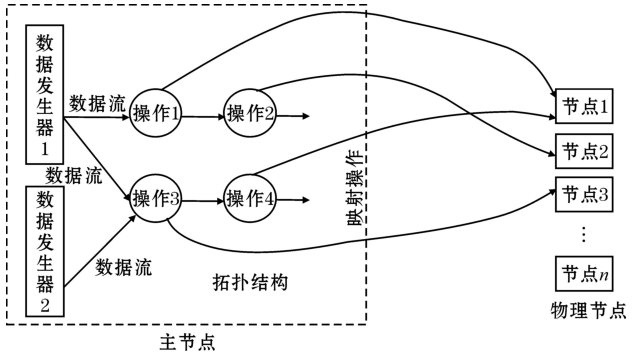

图 8　计算和数据协作机制的流式拓扑映射方法

下面举例说明 Topology 的映射过程, 定义操作 1 是对输入整型数据流的加 2 计算并输出, 操作 2 是对输入整型数据流的加 3 计算并输出, 操作 3 是对输入整型数据流的乘 2 计算并输出, 操作 4 是对输入整型数据流的乘 3 计算并输出, 数据发生器不断的产生整型数据。按照这一 Topology 的逻辑规划, 系统将操作 1 的计算程序注入物理节点 1, 操作 2 的计算程序注入物理节点 2, 操作 3 的计算程序注入物理节点 3, 操作 4 的计算程序注入物理节点 1, 并按 Topology 描述的流向建立节点 1 和节点 2 之间的流消息传递机制, 节点 3 和节点 1 之间的流消息传递机制。启动运算后如数据发生器 1 生成一个整型数据 5 后, 节点 1 对其加 2 后将结果 7 传送给节点 2, 节点 2 将其加 3 后输出结果 10, 同时根据 Topology 的描述数据发生器 1 的数据也会送给节点 3, 节点 3 对其乘 2 后将结果 10 传送给节点 1, 节点 1 将其乘 3 后输出最后结果为 30。数据发生器 2 产生的数据处理方法与此相似。

可以看出利用计算和数据协作机制的流式拓扑映射方法集群系统可以根据 Topology 的描述自动组合成不同的集群计算结构, 从而能灵活面对复杂问题的处理。在这里主节点起到计算和数据的路由工作, 计算和数据的

协作机制就是依据 Topology 的描述来跟踪定位的。

用 MPI 来形式化的模拟从 Topology 到物理的映射过程，节点间通过MPI_Send()函数将流数据元组注入指定节点，在该节点上发起相应的操作，并通过MPI-Recv()函数接收前端发来的数据，实现节点间的通讯，如图 9 所示。

图 9 用 MPI 模拟流式拓扑映射

主节点即节点 0 为数据发生器，发起两个数据流。在数据流 1 中，节点 0 将其产生的数据通过MPI_Send()函数发送到节点 1，节点 1 通过MPI_Recv()函数接收节点 0 发送来的数据，并发起加 2 的操作，将结果通过MPI_Send()函数发送到节点 2，节点2 通过MPI_Recv()函数接收节点1 发送来的数据，并发起加3 的操作。如果主节点不断产生数据并向子节点发送数据就形成了流式系统的模式，MPI 的灵活性在这里也体现得非常明显。

从以上分析可以看出流式拓朴协作机制由于是基于机群结构的因此可以实现海量数据的实时处理，避免了一些大数据系统只能对数据进行非实时批处理的问题，同时利用拓朴结构可以实现更为灵活的分布式计算规划，使大数据系统能灵活的设计数据分析算法。

5 结束语

从计算和数据的协作机制这一视角对大数据技术的发展历程，主要的系统架构分类，主流系统的实现机制进行了介绍、对比和分析。认为计算和数据的协作机制是实现大数据系统的关键核心技术，协作机制的实现与分布式系统的整体架构有紧密的联系，特别是分布式文件系统与计算的融合是解决协作机制的关键，单独地考虑存储和单独考虑计算在面向数据的分布式系统中都是不全面的，由于数据的分析处理也是非常重要的，所以大数据系统应该是一种计算和数据都密集的系统。

未来大数据时代的数据会朝着：数据量更大，数据的关联性更强，数据类型更多样化的方向发展。同时大数据技术未来会呈现出以下发展趋势和特点：(1)大数据系统中机群结构会成为主流架构；(2)对单个节点计算能力和稳定性的要求会下降；(3)传统的面向计算的高性能计算技术会重新被人们认识并与大数据技术相结合，通过大数据系统使阳春白雪式的高性能计算变的更为贴近人们的生活；(4)新的大数据实现技术会涌现，如实时大数据处理、大规模图处理技术；(5)数据挖掘算法特别是大数据架构下的数据挖掘算法会得到更多的关注。

致谢：感谢成都市科技局创新发展战略研究项目(11RKYB016ZF)对本文的资助

参考文献：

[1] Parkhill, D F. The challenge of the computer utility (Addison-Wesley Publishing Company Reading, 1966. 1966).

[2] http://en.wikipedia.org/wiki/John_Gage.

[3] Li, G., Scientific value of big data research[J]. Communications of the CCF, 2012, 8(9): 8—15.

[4] Batcher, K E. Design of a massively parallel processor[J]. Computers, IEEE Transactions on, 1980, 100(9): 836—840.

[5] Duncan, S H, Keefer, C D, McLaughlin T A. High performance I/O design in the AlphaServer 4100 symmetric multiprocessing system[J]. Digital Technical Journal, 1996, 8(4): 61—75.

[6] Wang, P. Cloud Computing: Key Technique and Application[M]. Posts &Telecom Press, 2010.

[7] Wang F Y, Carley K M, Zeng D. Social computing: From social informatics to social intelligence[J]. Intelligent Systems, IEEE, 2007, 22(2): 79—83.

[8] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the Acm, 2008, 51(1): 107—113.

[9] Borthakur D. HDFS architecture guide[EB/OL]. Hadoop Apache Project. http://hadoop. apache. org/common/docs/current/hdfs-design. pdf, 2008.

[10] Peng D, Dabek F. Large-scale incremental processing using distributed transactions and notifications[R]. Proc. Proceedings of the 9th USENIX conference on Operating systems design and implementation, 2010: 1—15.

[11] Bhatotia P, Wieder A, Rodrigues R. Incoop: MapReduce for incremental computations[R]. Proc. Proceedings of the 2nd ACM Symposium on Cloud Computing, 2011: 7.

[12] Elnikety E, Elsayed T, Ramadan H E. iHadoop: asynchronous iterations for MapReduce[R]. Proc. Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011: 81—90.

[13] Ekanayake J, Li H. Zhang B. Twister: a runtime for iterative mapreduce[R]. Proc. Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, 2010: 810—818.

[14] Malewicz G, Austern M H, Bik A J. Pregel: a system for large-scale graph processing[R]. Proc. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, 2010: 135—146.

[15] http://wuyanzan60688.blog.163.com.

[16] Guo, B, Wang P, Chen G. Cloud Computing Model Based on MPI[J]. Computer Engineering, 2009, 35(24).

[17] Zhai Y, Liu M, Zhai J. Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications[R]. Proc. State of the Practice Reports, 2011: 11.

[18] Lu X, Wang B, Zha L. Can mpi benefit hadoop and mapreduce applications[R]. Proc. Parallel Processing Workshops (ICPPW), 2011 40th International Conference on, 2011: 371—379.

[19] Hoefler T, Lumsdaine A, Dongarra J. Towards efficient mapreduce using mpi[R].: Recent Advances in Parallel Virtual Machine and Message Passing Interface(Springer, 2009), 2009: 240—249.

[20] Luo X, Shu J. Summary of Research for Erasure Code in Storage System[J]. Journal of Computer Research and Development, 2012, 49(1): 1—11.

[21] Wang F, Qiu J, Yang J, Dong B. Hadoop high availability through metadata replication[R]. Proc. Proceedings of the first international workshop on Cloud data management, 2009: 37—44.

[22] Bohn C A, Lamont G B. Load balancing for heterogeneous clusters of PCs, Future Generation Computer Systems[J]. 2002, 18(3): 389—400.

[23] Dong B, Li X, Wu Q. A dynamic and adaptive load balancing strategy for parallel file system with large-scale I/O servers[J]. Journal of Parallel and Distributed Computing, 2012, (1).

[24] Wang W, Zeng G, Tang D. Cloud-DLS: Dynamic trusted scheduling for Cloud computing[J]. Expert Systems with Applications, 2012, 39(3): 2321—2329.

[25] Randles M, Lamb D, Taleb-Bendiab A. A comparative study into distributed load balancing algorithms for cloud computing[R]. Proc. Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on, 2010: 551—556.

[26] Ren X, Lin R, Zou H. A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast[R]. Proc. Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on, 2011: 220—224.

[27] Ge Y, Wei G. Ga-based task scheduler for the cloud computing systems[R]. Proc. Web Information Systems and Mining (WISM), 2010 International Conference on, 2010: 181—186.

[28] Karger D. Sherman A. Berkheimer A. Web caching with consistent hashing, Computer Networks, 1999, 31 (11): 1203—1213.

[29] Shires D. Mohan R. Mark A. A Discussion of Optimization Strategies and Performance for Unstructured Computations in Parallel HPC Platforms', 2001.

[30] Isard M. Budiu M. Yu Y. Dryad: distributed data-parallel programs from sequential building blocks, ACM SIGOPS Operating Systems Review, 2007, 41, (3): 59—72.

[31] http://storm-project.net/.

Cooperation Mechanism of Computation and Data in Big Data Technology

WANG Peng[1], HUANG Yan[2,3], LIU Feng[1], AN Jun-xiu[1]

(1. Parallel Computing Lab, Chengdu University of Information Technology, Chengdu 610225, China; 2. Chengdu Institute of Computer Application, Chinese Academy of Sciences, Chengdu 610041, China; 3. University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Big data system is the data-oriented high performance system. Computation and storage in big data system are all disturbed achieved by cluster technology. Corporation mechanism between computation and data is the main topic in this paper. By contrasting computation-oriented and data-oriented high performance computing system, we find that corporation mechanism between computation and data determines big data system's architecture and performance. Integration between distributed file system and computation is the foundation of automatic parallelization in big data system. In big data system, data are divided and distributedly stored to achieve automatic parallelization. Metadata mapping, hash mapping and stream topology are often used in the corporation between computation and data, and stream topology is often used to achieve realtime big data analysis.

Key words: data-oriented; cooperation mechanism; computation-oriented; high performance computing; big data