软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(8):2557−2579 [doi: 10.13328/j.cnki.jos.006074]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

Storm 平台下的线程重分配与数据迁移节能策略*

蒲勇霖[1], 于 炯[1], 鲁 亮[2], 李梓杨[1], 卞 琛[3], 廖 彬[4]

[1](新疆大学 信息科学与工程学院,新疆 乌鲁木齐 830046)

[2](中国民航大学 计算机科学与技术学院,天津 300300)

[3](广东金融学院 互联网金融与信息工程学院,广东 广州 510521)

[4](新疆财经大学 统计与数据科学学院,新疆 乌鲁木齐 830012)

通讯作者: 蒲勇霖, E-mail: puyonglin1991@foxmail.com

摘 要: 作为流式大数据计算的主要平台之一,Storm 在设计过程中由于缺乏节能的考虑,导致其存在高能耗与低效率的问题.传统的节能策略并未考虑 Storm 的性能约束,可能会对集群的实时性造成影响.针对这一问题,设计了资源约束模型、最优线程重分配模型以及数据迁移模型.进一步提出了 Storm 平台下的线程重分配与数据迁移节能策略(energy-efficient strategy based on executor reallocation and data migration in Storm,简称 ERDM),包括资源约束算法与数据迁移算法.其中,资源约束算法根据集群各工作节点 CPU、内存与网络带宽的资源占用率,判断集群是否允许数据的迁移.数据迁移算法根据资源约束模型与最优线程重分配模型,设计了数据迁移的最优化方法.此外,ERDM 通过分配线程减少了节点间的通信开销,并根据大数据流式计算的性能与能效评估 ERDM.实验结果表明,与现有研究相比,ERDM 能够有效降低节点间通信开销与能耗,并提高集群的性能.

关键词: 大数据;流式计算;实时性;资源约束;数据迁移;能耗

中图法分类号: TP311

中文引用格式: 蒲勇霖,于炯,鲁亮,李梓杨,卞琛,廖彬.Storm 平台下的线程重分配与数据迁移节能策略.软件学报,2021,32(8):2557−2579. http://www.jos.org.cn/1000-9825/6074.htm

英文引用格式: Pu YL, Yu J, Lu L, Li ZY, Bian C, Liao B. Energy-efficient strategy based on executor reallocation and data migration in Storm. Ruan Jian Xue Bao/Journal of Software, 2021,32(8):2557−2579 (in Chinese). http://www.jos.org.cn/1000-9825/6074.htm

Energy-efficient Strategy Based on Executor Reallocation and Data Migration in Storm

PU Yong-Lin[1], YU Jiong[1], LU Liang[2], LI Zi-Yang[1], BIAN Chen[3], LIAO Bin[4]

[1](College of Information Science and Engineering, Xinjiang University, Urumqi 830046, China)

[2](College of Computer Science and Technology, Civil Aviation University of China, Tianjin 300300, China)

[3](College of Internet Finance and Information Engineering, Guangdong University of Finance, Guangzhou 510521, China)

[4](College of Statistics and Data Science, Xinjiang University of Finance and Economics, Urumqi 830012, China)

Abstract: As one of the most popular platforms in big data stream computing, Storm is suffering from the problem of high energy consumption and low energy efficiency due to the lack of consideration for energy saving strategy in the design process. Without taking the performance constraint of Storm into consideration, the traditional energy-efficient strategies may affect the real-time performance of cluster. Aiming at this issue, models of the resource constraint, the optimal executor reallocation, and the data migration are set up, and

* 基金项目: 国家自然科学基金(61862060, 61462079, 61562086, 61562078); 新疆维吾尔自治区研究生科研创新项目(XJ2019G038); 新疆大学博士生科技创新项目(XJUBSCX-201902)

Foundation item: National Natural Science Foundation of China (61862060, 61462079, 61562086, 61562078); Research Innovation Project of Graduate Student in Xinjiang Uygur Autonomous Region (XJ2019G038); Doctoral Innovation Program of Xinjiang University (XJUBSCX-201902)

收稿时间: 2019-11-04; 修改时间: 2020-01-30, 2020-03-21; 采用时间: 2020-04-30

the energy-efficient strategy based on executor reallocation and data migration in Storm (ERDM) is further proposed, while ERDM is composed of resource constraint algorithm and data migration algorithm. The resource constraint algorithm estimates whether the cluster is appropriate for data migration according to the utilization of CPU, memory, and network bandwidth in each work node. The data migration algorithm designs optimal method to migrate data according to the resource constraint model and the optimal executor reallocation model. Moreover, the ERDM allocates the executors so as to reduce communication cost between nodes. The ERDM is evaluated by measuring the cluster performance as well as energy consumption efficiency in big data stream computing environment. The experimental results show that the proposed strategy can reduce communication cost and energy consequence efficiently while the cluster performance is improved compared with existing researches.

Key words: big data; stream computing; real-time; resource constraint; data migration; energy consumption

目前,随着互联网的高速发展与平民化,智能医疗、智能汽车、智能家居以及智能工业等物联网场景[1]下产生的数据量日益增多,并与互联网共同成为了各行各业大数据的主要来源.但是随着大数据的飞速发展,大规模的数据中心在全球范围内广泛的部署,使其高能耗、高污染的问题日渐突出[2].据 2014 年数据中心能耗现状白皮书指出:全球数据中心 2013 年的耗电量为 8 102.5 亿 kWh,其中 IT 设备与软件的能耗占数据中心总能耗的 45.5%[3].而到 2015 年,Gartner 统计全球大型数据中心的电费支出超过 1 262 亿美元[4].我国数据中心电力的消耗同样惊人,截止到 2011 年底,我国数据中心的总量达到 43 万个,其中数据中心的耗电量,占据全国电力总消耗的 1.5%,并且所占比例仍在逐年上升[5].综上所述,解决大数据处理的高能耗问题已经刻不容缓.

希捷(Seagate)公司与 IDC 联合发布的《数据时代 2025》白皮书中预测:2025 年,全球数据量将达到 163ZB,比 2016 年创造出的数据量增加 10 倍.其中,超过 25%的数据将成为实时数据,而物联网实时数据占比将达到实时数据的 95%[6].针对大数据处理的高性能集群一般分为批量计算框架与流计算框架两类,其中,批量计算框架由于存在先存储后计算的特性,无法满足实时数据的处理需求;而流计算框架由于其强大的实时性,为实时大数据分析提供了良好的平台层解决方案.但是流式计算在高速处理实时数据的同时伴随着高能耗的问题[7],已经给产业界带来了巨大的能耗开销.特别在 2017 年后,针对大数据流式处理节能计算的研究[8]已经逐渐增多,其研究价值已被广大的科研人员认可.因此,大数据流式处理节能计算不仅减少了能源消耗保护环境,而且具有广阔的研究价值与应用前景.

目前,主流 IT 企业(如华为、百度以及小米等)针对大数据流式处理的业务主要以 Apache Storm 框架[9]为主.虽然主流 IT 企业的部分流式处理业务已被迁移至 Flink[10]、Spark Streaming[11]和 Heron[12]等框架,但其核心的流式处理业务还是基于 Storm 完成的.这是由于与目前主流的 Flink 与后起之秀 Heron 相比,Storm 具有更成熟的平台架构和更广泛的产业基础;与属于微批的框架 Spark Streaming 相比,Storm 具有更好地实时性;与不开源的 Puma[13]以及社区冷淡的 S4[14]相比,Storm 具有更广阔的发展前景.此外,Storm 为适应业界的需求而不断更新其版本,展现出强大的生命力.Storm 是一个主从式架构、开源、横向扩展性良好且容错能力强的分布式实时处理平台,其编程模型简单,支持包含 Java 在内的多种编程语言,且数据处理高效.目前,Storm 已经广泛运用到银行金融[15]、临床医疗[16]、社交网络[17]等行业进行实时大数据分析,并广泛运用到机器学习算法、分布式远程调用等领域进行理论研究[18].Storm 因其广泛的业界认可度,而被誉为“实时处理领域的 Hadoop”.

在 Storm 集群中,通常使用有向无环图(directed acyclic graph,简称 DAG)表示一个流式作业(拓扑)内数据的相互关联性,其中,DAG 的顶点表示工作线程及数据的处理,DAG 的边表示数据的相互依存性及顶点间的通信.Storm 集群采用轮询调度策略(round-robin,简称 RR),并将 DAG 中的任务平均分配到各工作节点之中.然而,Storm 的发展也面临着一定的挑战.首先,Storm 通过 RR 将任务均匀分配到各节点中,但是不同的任务对于节点计算资源的需求有所差异,如果节点的计算资源无法满足任务的需求,则会导致节点资源溢出与集群计算延迟升高的问题,进而影响集群的性能,并产生高能耗的问题.因此,通过研究任务调度优化策略从而最大化利用 Storm 集群的实时计算能力,是目前亟待解决的问题.其次大多数任务调度优化策略主要通过迁移计算负载来提高集群性能,但是无法对降低流式处理的高能耗带来帮助.因此,本文通过降低集群节点间的通信开销,在减少集群计算延迟的基础上,有效节约了能耗.

本文针对上述问题,其主要工作如下.

(1) 通过研究 Storm 集群的拓扑(topology)结构,建立 DAG、线程内的数据分配与路径开销这 3 个基本模型,从逻辑上将 Storm 集群的拓扑运行情况与数据分配策略表示出来,为寻找最优的数据迁移方式创造了条件,并为节能策略的提出奠定了理论基础.

(2) 根据 3 个基本逻辑模型以及集群内数据的传输及处理情况,建立了资源约束模型,通过 3 个条件证明了资源约束模型的必要性,并进一步建立最优线程重分配模型,其中,线程的最优分配由资源约束模型、通信成本、RR 与 CPU 优先级决定.在满足资源约束的条件下,实现了数据的迁移.

(3) 通过对集群内的数据进行分析,根据资源约束模型与最优线程重分配模型,提出了 Storm 平台下的线程重分配与数据迁移节能策略(energy-efficient strategy based on executor reallocation and data migration in Storm,简称 ERDM),该策略包括资源约束算法与数据迁移算法,其中,资源约束算法根据节点资源约束判断工作节点是否允许数据迁移;数据迁移算法根据资源约束模型和最优线程重新分配模型,确定了集群中数据的迁移情况.此外,实验通过 4 组基准测试[19],从不同角度验证了算法的有效性.

本文第 1 节针对目前国内外节能计算的相关研究进行总结与分析.第 2 节对 Storm 平台进行建模并给出相关定义.第 3 节详细介绍 ERDM 的算法并建立能耗模型.第 4 节进行实验对比并对实验结果进行分析.第 5 节对本文进行总结并对下一步工作进行展望.

1 相关工作

传统的大数据平台一直专注于延迟、容错性以及弹性计算等方面,但是随着 IT 行业能耗的不断增加,高能耗以及散热问题已经开始制约大数据平台性能的进一步发展.因此,大数据平台的发展目标已经逐步转移到功耗与能效方面.目前,用于大数据流处理平台的节能策略主要集中在硬件[20]与软件[21]两个方面.

硬件的节能策略主要体现在替换高能耗的电子元件[22]与对集群电源电压进行缩放管理[23],以达到节能的效果.该方法节能效果显著且操作简单,但其价格高昂不适合部署于大规模的集群当中.Wang 等人[24]使用了动态电压频率缩放技术(dynamic voltage frequency scaling,简称 DVFS),通过动态管理集群节点 CPU 的电压,以实现节能的目的.Pietri 等人[25]通过将流式处理平台的部分 CPU 替换成 GPU,使得 CPU 与 GPU 进行混合,从而减少了集群处理图数据的能耗.实验结果表明,在节约 9.69%能耗的前提下,减少了 8.63%访问时间.文献[26−28]通过替换高能耗的电子元件,从而提高了集群的能效,以达到节能的目的.软件的节能策略主要体现在建立能耗模型[29]以及通过资源调度[30]提高集群的能效,以达到节能的效果.Cordeschi 等人[31]从虚拟化数据中心(virtualized networked data center,简称 VNetDC)的角度出发,提出一种在 SaaS 模型下,针对实时处理应用的最小化能耗调度策略.该研究针对流式大数据传输不稳定、不可控以及实时数据量大等特性,在不影响响应时间约束条件的前提下,计算了最小化网络传输的总能耗.Cheng 等人[8]从流计算平台的本质出发,提出一种基于 Spark Streaming 自适应调度作业的节能策略.该策略通过在集群中构建一个实时能耗分析模型,并对数据流信息进行实时的捕捉分析,根据分析结果对数据进行预处理,以此提高了集群性能并减少了部分时间开销,达到了节能的效果.文献[32]提出一种作用于 Spark Streaming 的能耗分析基准测试方法,该方法通过使用机器学习算法,查找集群内数据流的大小与通信开销的平衡.实验结果表明,当集群内数据流的大小与通信开销达到平衡时,集群执行任务的功耗最小.Maroulis 等人[33]根据分析 Spark Streaming 在执行任务时性能与能耗的权衡,提出一种基于调度工作负载的高效节能策略.该策略通过建立时间序列预测模型来捕获任务的执行时间与能耗,并通过使用 DVFS 技术来将集群的能耗降至最低.Veiga 等人[34]设计了一种作用于 Flink 的能耗评估工具,该工具通过分析集群执行任务的工作负载,以找到不同条件下的集群能耗,为后期设计基于 Flink 的节能策略奠定了基础.文献[35]提出一种可同时兼顾低延迟与低能耗的弹性数据流处理策略(keep calm and react with foresight: strategies for low-latency and energy-efficient elastic data stream processing,简称 LEEDSP),该策略通过使用 DVFS 技术,建立了一种弹性自适应性的能耗感知模型.该模型通过合理分配集群资源,在提高集群的吞吐量的同时,减少任务执行的延迟,以此节约了集群的能耗.

文献[7]提出一种流式大数据处理环境下的实时资源调度节能策略(re-stream),该策略通过构建 CPU 利用率、响应时间以及能耗间的数学关系,以此获得了满足高能效与低响应时间的条件,从而实现了节能的目的.然而,该节能策略仍存在以下两点值得讨论:(1) 该策略仅考虑集群 CPU 的能耗问题,但对于集群其他电子元件的能耗并未做叙述;(2) 该策略仅使用自己定义的拓扑,并非公认的测试数据集.此外,除了集群自身算法,并未作其他对比实验,因此节能策略缺乏一定的通用性.

文献[36]针对 Storm 在遭遇资源瓶颈与网络报错时,缺乏合理应对手段,提出了基于数据恢复的节能策略.该策略通过监控集群拓扑内任务的执行吞吐量,判断任务的实际运行情况,确定是否终止集群内的任务,并根据数据恢复模型还原集群内的数据.该策略不仅有效降低了集群因资源瓶颈与网络报错而带来的额外资源与能耗的开销,而且提高了集群任务处理的性能.此外,该策略具有两个明显的优点:(1) 从内存重新恢复读取数据,有效地避免了从磁盘读取数据资源与能耗的峰值问题;(2) 该策略可以与其他节能策略进行融合,达到双重节能的效果.但也存在集群未发生资源瓶颈与网络报错,导致节能策略失效的问题.

文献[37]根据 Storm 平台在进行数据处理时存在高能耗的问题,提出了工作节点内存电压调控节能策略(energy-efficient strategy for work node by DRAM voltage regulation in Storm,简称 WNDVR-Storm).该策略通过对集群内的数据流设置阈值,从而对集群工作节点数据处理能力进行判别,动态调节工作节点的内存电压,以达到节能的目的.该节能策略不仅有效降低了集群的能耗,而且在一定程度上对集群的负载均衡进行了优化,但是还存在以下两点不足:(1) 动态调节工作节点的内存电压存在一定的偶然性,且实现难度较高;(2) 若集群规模较大且工作节点过多,存在节能算法失效的可能.

与已有成果相比,本文的不同之处在于.

(1) 文献[8,31−35]均是从集群整体的特性进行分析,并未细化各部件对集群的影响,如网络带宽、CPU 等部件对集群的影响.本文从网络带宽、CPU 以及内存这 3 个方面进行分析建模,确定因数据迁移而对集群各部件造成的影响,从而确保在不同场景下均能使节能策略顺利执行.

(2) 文献[7]通过对集群进行任务迁移调度而达到节能的效果,但并未考虑因任务迁移调度而带来各工作节点计算资源不足的问题,存在资源溢出的风险.本文通过建立资源约束模型,预防了集群数据处理的资源溢出问题.

(3) 文献[37]通过对工作节点的内存电压进行动态调节而达到了节能的效果,但是动态调节工作节点的内存电压存在较大误差,且会对集群性能造成一定的影响.本文由于是软件方面的节能策略,因此不存在动态调压的问题.此外,本文通过数据迁移算法减少了节点间的通信开销,在降低集群计算延迟的前提下节约了能耗.

(4) 实验选取 Intel 公司 Zhang 等人[19]发布在 GitHub 上的 Storm-benchmark-master 基准测试,而非已有文献中作者自己定义的拓扑结构,因此更具有通用性.此外,将 ERDM 与大数据流式计算框架 Storm 的任务迁移策略(task migration strategy in big data stream computing with Storm,简称 TMSH-Storm)[18]、LEEDSP[35]以及 WNDVR-Storm[37]进行对比,验证了策略的有效性.

2 问题建模与分析

为了确定 Storm 集群默认调度策略的能耗问题,建立了 DAG、线程内数据分配与路径开销这 3 个基本模型,并进一步设计了资源约束模型、最优线程重分配模型与数据迁移模型,为节能策略的设计与实现提供了理论依据.

2.1 基础逻辑模型

在流式处理中,通常用数据流图处理多个连续并行的任务,将其表示为 DAG.则在 Storm 集群中的流式作业可用定义 1 表示.

定义 1(DAG). 在 Storm 集群中,每个流应用程序的逻辑通常由 DAG[7]描述,而 DAG 由顶点与边构成.则令 $DAG=(C(G),B(G))$,其中,$C(G)=\{c_1,c_2,\ldots,c_n\}$表示由 n 个组件(component)构成的集合,包括数据源编程单元(spout)

与数据处理编程单元(bolt)两类;$B(G)=\{b_{1,2},b_{1,3},\ldots,b_{n-1,n}\}$为有向边的集合,表示各组件间的数据传输链路.如果$\exists b_{i,j}\in B(G)$且 $i\neq j$,则 $c_i,c_j\in C(G)$表示数据从 c_i 发出由 c_j 接收.则组件与有向边的对应关系可通过图 1 表示.

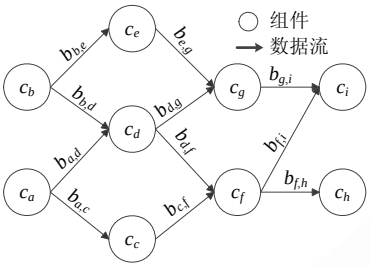


Fig.1 A logical DAG of a stream application

图 1 流应用的逻辑 DAG

为提高 Storm 集群的执行效率,需要满足同一时刻执行多个组件,即$\forall c_j\in C(G),E(C)=\{e_{j1},e_{j2},\ldots,e_{jn}\}$.其中,每个元素 e_{ji} 为一个线程(executor),且 e_{ji} 表示组件 c_j 运行第 i 个线程.图 2 为 Storm 集群数据处理及传输示意图,由 11 个线程与 20 条有向边组成.其中,$\{e_a,e_b,\ldots,e_i\}$为线程集合,且线程 e_a 与 e_b 通过拓扑链路发送数据,而后续线程接收上游线程发送的数据.以此类推,完成整个拓扑.

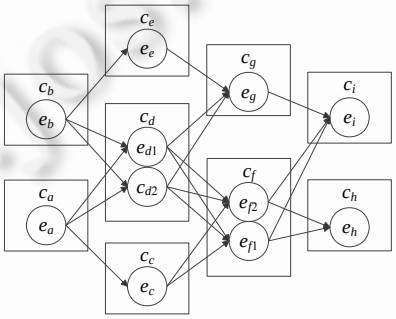


Fig.2 Data processing and transmission in Storm cluster

图 2 Storm 集群数据处理及传输

此外,令线程 e_a 为线程$\{e_c,e_{d1},e_{d2}\}$的父线程,则线程$\{e_c,e_{d1},e_{d2}\}$是线程 e_a 的子线程.线程 e_b 为线程$\{e_{d1},e_{d2},e_e\}$的父线程,则线程$\{e_{d1},e_{d2},e_e\}$是线程 e_b 的子线程.以此类推,完成父线程与子线程间的对应关系.

定义 2(线程内的数据分配). 令 $N(C)=\{n_1,n_2,\ldots,n_m\}$为集群工作节点集合,且每个工作节点内存在多个线程,由定义 1 可知,工作节点的数据均匀分配到集群的线程上.记工作节点分配给线程 e_{ji} 的数据为 d_{ji}(若线程的并行度为 1,则该线程上的数据为 d_j),则集合 $D_n=\{d_{j1},d_{j2},\ldots,d_{jn}\}$表示工作节点分配到线程上的数据集合.图 3 为线程内数据的分配示意图,由 3 个节点与 11 个线程组成.其中,$N=\{n_1,n_2,n_3\}$表示工作节点的集合,而线程内的数据为

$$D_{n_1}=\{d_a,d_{d1},d_{f1}\},D_{n_2}=\{d_b,d_{d2},d_{f2},d_h\},D_{n_3}=\{d_c,d_e,d_g,d_i\}.$$

此外,为消除节点内部进程间通信开销,图 3 为各工作节点仅分配一个工作进程(worker),因此,拓扑中的通信开销可分为两类:一类为类似于数据 d_{d1} 与数据 d_{f1} 之间的节点内部线程间通信;一类为类似于数据 d_{d2} 与数据 d_{f1} 之间的节点间通信.无论集群拓扑内如何传输数据,凡存在直接对应关系,都符合上述传输方式.

定义 3(路径开销). 令集合 $B(p(e_{ji},e_{mn}))$存在一条子路径 $p(e_{ji},e_{mn})$,表示从顶点 e_{ji} 开始到顶点 e_{mn} 结束.则需要满足的条件为:如果$\exists k$,则 $b_{j,k}\in p(e_{ji},e_{mn}),b_{k,i}\in p(e_{ji},e_{mn})$.由此,对于$\forall b_{j,i}\in p(e_{ji},e_{mn})$都存在.此外,对于$\forall b_{k,l}\in p(e_{ji},e_{mn})$,如果 $k\neq j$,则$\exists m$ 与 $b_{m,k}\in p(e_{ji},e_{mn})$;如果 $i\neq j$,则$\exists m$ 与 $b_{l,m}\in p(e_{ji},e_{mn})$.

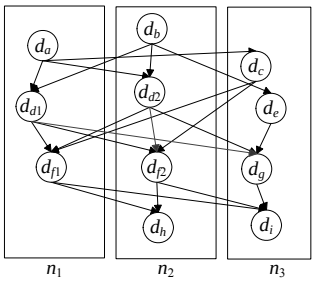


Fig.3 Allocation of data in executor

图 3 线程内的数据分配

路径开销 $l_p(e_{ji},e_{mn})$表示从顶点 e_{ji} 到顶点 e_{mn} 内所有线程与有向边的开销之和,则

$$l_p(e_{ji},e_{mn})=\sum_{e_{ji}\in E(p(e_{ji},e_{mn}))}e_{ji}+\sum_{b_{j,i}\in B(p(e_{ji},e_{mn}))}b_{j,i} \tag{1}$$

令路径的计算开销 $W^{\text{executor}}_{\text{computation cost}}$ 为线程数据的处理时间,路径的传输开销 $W^{\text{edge}}_{\text{communication cost}}$ 为线程间数据的传输时间,则路径的总开销 W_{cost} 为

$$W_{\text{cost}}=W^{\text{executor}}_{\text{computation cost}}+W^{\text{edge}}_{\text{communication cost}} \tag{2}$$

其中,路径的传输开销 $W^{\text{edge}}_{\text{communication cost}}$ 由节点内部线程间通信开销、节点内部进程间通信开销以及节点间通信开销这 3 部分组成.由于每个工作节点仅存在一个工作进程,则节点内部进程间通信开销为 0.故

$$W_{\text{cost}}=W^{\text{executor}}_{\text{computation cost}}+W^{\text{edge}}_{\text{executor cost}}+W^{\text{edge}}_{\text{node cost}} \tag{3}$$

其中,$W^{\text{edge}}_{\text{node cost}}$ 表示节点间通信开销,$W^{\text{edge}}_{\text{executor cost}}$ 表示节点内部线程间通信开销.此外,如果线程 e_{ji} 和 e_{mn} 位于相同的工作节点,则线程间的通信开销可忽略不计.因此,

$$W_{\text{cost}}=W^{\text{executor}}_{\text{computation cost}}+W^{\text{edge}}_{\text{node cost}} \tag{4}$$

令整个拓扑存在 n 条路径,则拓扑执行关键路径 $l(G_p)$为

$$l(G_p)=\max\{l_{p_1}(e_{ji},e_{mn}),l_{p_2}(e_{ji},e_{mn}),...,l_{p_n}(e_{ji},e_{mn})\} \tag{5}$$

此外,根据工作节点是否位于关键路径上,将工作节点分为关键节点与非关键节点;根据线程是否位于关键节点,将线程分为关键节点上的线程与非关键节点上线程,简称关键线程与非关键线程.

以图 4 为例,定义一条拓扑执行关键路径为 $e_a\rightarrow e_{d1}\rightarrow e_{f2}\rightarrow e_h$,则工作节点 n_1 与 n_2 为关键节点,工作节点 n_3 为非关键节点.

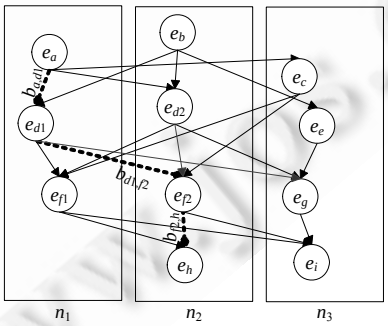


Fig.4 Topology execution of critical path data transmission and processing

图 4 拓扑执行关键路径的数据传输及处理

2.2 资源约束模型

定义 4(资源约束). 为满足 Storm 集群进行数据迁移时各工作节点的资源需求,需设置工作节点计算资源集

合为 $R_N=\{r_{n_1},r_{n_2},...,r_{n_m}\}$,则工作节点 CPU、内存以及网络带宽这 3 类计算资源占用的极限为 $R_N=(R_N^C,R_N^M,R_N^B)$.其中, R_N^C 表示工作节点 CPU 资源占用率的极限为 $R_N^C=\{r_{n_1}^C,r_{n_2}^C,...,r_{n_m}^C\}$, R_N^M 表示工作节点内存资源占用率的极限为 $R_N^M=\{r_{n_1}^M,r_{n_2}^M,...,r_{n_m}^M\}$, R_N^B 表示工作节点网络带宽资源占用率的极限为 $R_N^B=\{r_{n_1}^B,r_{n_2}^B,...,r_{n_m}^B\}$.若线程 e_{ji} 所在工作节点的 CPU 资源占用率为 $o_{n_i}^C$ (单位%),内存资源占用率为 $o_{n_i}^M$ (单位%),网络带宽资源占用率为 $o_{n_i}^B$ (单位%),由于 Storm 集群拓扑一旦提交数据将源源不断产生,且持续运行下去,因此为确保集群的高效运行,且工作节点的资源不会溢出,这 3 类资源需要满足如下条件:

$$\sum_{\forall r_{n_i}\in R_N} o_{n_i}^C \leqslant r_{n_i}^C \tag{6}$$

$$\sum_{\forall r_{n_i}\in R_N} o_{n_i}^M \leqslant r_{n_i}^M \tag{7}$$

$$\sum_{\forall r_{n_i}\in R_N} o_{n_i}^B \leqslant r_{n_i}^B \tag{8}$$

为保证集群拓扑能够正常运行,则集群工作节点各类计算资源需要满足资源约束.本文将满足工作节点 CPU 的正常计算称为符合 CPU 资源临界原则,将满足工作节点内存的正常计算称为符合内存资源临界原则,将满足工作节点网络带宽的正常传输称为符合网络带宽资源临界原则.此外,具体结果在第 4.2 节体现.

定理 1. 当集群准备进行数据迁移时,判断被选中节点资源是否满足 CPU 资源临界原则、内存资源临界原则以及网络带宽资源临界原则:若满足,则允许节点迁入数据.即,数据迁入原则 tr 需要满足如下条件:

$$r_{n_i}^{C'}+\sum_{\forall r_{n_i}\in R_N} o_{n_i}^C \leqslant r_{n_i}^C \tag{9}$$

$$r_{n_i}^{M'}+\sum_{\forall r_{n_i}\in R_N} o_{n_i}^M \leqslant r_{n_i}^M \tag{10}$$

$$r_{n_i}^{B'}+\sum_{\forall r_{n_i}\in R_N} o_{n_i}^B \leqslant r_{n_i}^B \tag{11}$$

其中, $r_{n_i}^{C'}$ 表示工作节点迁入数据后增加的 CPU 资源占用率, $r_{n_i}^{M'}$ 表示工作节点迁入数据后增加的内存资源占用率, $r_{n_i}^{B'}$ 表示工作节点迁入数据后增加的网络带宽资源占用率.

证明:根据定义 4 可知,当节点迁入数据后,该工作节点 CPU 的计算资源占用率小于极限值时,工作节点的 CPU 可以正常计算.则称满足 CPU 资源临界原则,即

$$r_{n_i}^{C'}+o_{n_i}^C \leqslant r_{n_i}^C \tag{12}$$

由于当流式处理集群执行任务时,拓扑一旦提交将持续运行下去,即

$$r_{n_i}^{C'}+\sum_{\forall r_{n_i}\in R_N} o_{n_i}^C \leqslant r_{n_i}^C;$$

同理可得,满足内存资源临界原则,即

$$r_{n_i}^{M'}+\sum_{\forall r_{n_i}\in R_N} o_{n_i}^M \leqslant r_{n_i}^M;$$

同理可得,满足网络带宽资源临界原则,即

$$r_{n_i}^{B'}+\sum_{\forall r_{n_i}\in R_N} o_{n_i}^B \leqslant r_{n_i}^B.$$

仅符合以上 3 条原则,允许节点迁入数据,即得到定理 1. □

2.3 最优线程重分配模型

根据第 2.1 节与第 2.2 节建立最优线程重分配模型,该模型通过定义 3 与定义 4 确定非关线程的分配情况,并生成新的拓扑路径,为建立数据迁移模型做铺垫.

根据定义 3 可知,集群内的工作节点包括关键节点与非关键节点两类,集群内的线程包括关键线程与非关键线程两类,而集群拓扑内的通信开销由节点间通信开销、节点内部进程间通信开销与节点内部线程间的通信

开销这 3 部分组成.

定义 5(最优线程重分配). 现对非关键线程进行重分配,首先需要考虑集群内工作节点的资源占用率,在满足资源约束的条件下,为减少节点间的通信开销,需要将非关键线程重新分配到运行其父线程的关键节点上.此外,在进行非关键线程重新分配时,除了需要满足资源约束模型,还需要防止非关键线程分配出现扎堆现象.其原因为线程在进行重分配时,一般倾向于往通信开销较小的节点分配.由于上游非关键线程已经进行重分配,则下游非关键线程优先考虑分配到上游非关键线程所在的关键节点上.为避免上述情况,故非关键线程重分配需要加入 RR 与 CPU 优先级(工作节点中 CPU 的利用率最低)两个限制条件.

如果一个非关键子线程仅存在一个关键父线程,则将非关键子线程重新分配到运行其关键父线程的关键节点上;如果一个非关键子线程存在两个或多个关键父线程,则为防止扎堆现象的出现,首先需要考虑 RR,在满足 RR 的条件下,优先将非关键子线程重新分配到 CPU 利用率最低的关键父线程所在的关键节点上;如果运行父线程的关键节点的资源利用率达到极限,则同样为防止扎堆现象的出现,在满足 RR 的条件下,优先将非关键子线程重新分配到 CPU 利用率最低的关键节点上.

以图 4 为例,n_1 与 n_2 为关键节点,n_3 为非关键节点,且 n_1 存在 3 个线程,n_2 存在 4 个线程,则非关键子线程 e_c 被分配到 n_1,而非关键子线程 e_i 被分配到 n_1,即

$$e_c \xrightarrow{\text{communication cost between nodes}} n_1 \tag{13}$$

$$e_i \xrightarrow[\text{RR}]{\text{communication cost between nodes}} n_1 \tag{14}$$

如果 n_1 与 n_2 内的线程数量相等,且 n_1 的 CPU 占用率高于 n_2,则非关键子线程 e_i 被分配到 n_2,即

$$e_i \xrightarrow[\text{CPU priority}]{\text{RR}} n_2 \tag{15}$$

如果 n_1 的资源占用率已达到极限,则非关键子线程 e_c 在满足 RR 与 CPU 优先级的前提下分配到关键节点 n_i,即

$$e_c \xrightarrow[\text{CPU priority}]{\text{RR}} n_i \tag{16}$$

此外,选择 CPU 优先级为评判指标,是由于 CPU 的利用率对集群的性能影响最大.

2.4 数据迁移模型

根据第 2.3 节可知,集群已生成新的拓扑路径.现通过最优线程重分配模型建立数据迁移模型,将原非关键线程上的数据迁移到对应的关键线程中.

若父线程 e_{ab} 传输给非关键子线程 e_{ji} 数据的大小为 d_{ji},在完成非关键子线程重分配后,父线程对子线程的数据分配发生改变,类似数据流 d_{ji} 从 e_{ji} 迁移到 e'_{ji},即

$$e_{ji} \xrightarrow{da_{ji}} \sum_{f(n_j)\to f(n_i)} e'_{ji} \tag{17}$$

根据定义 4 可知,数据迁入存在资源约束的问题,需要满足数据迁入原则 tr,即

$$r_{n_{i_{\text{input}}}} + o_{n_i} \leqslant r_{n_i} \tag{18}$$

其中,$r_{n_{i_{\text{input}}}}$ 表示数据迁入后工作节点增加的资源占用率,o_{n_i} 表示原工作节点的资源占用率,r_{n_i} 表示工作节点资源占用率的极限,则式(17)符合数据迁入原则 tr.此外,具体结果在第 4.2 节体现.

定理 2. 根据定义 3 可知,原集群的路径成本为 W_{cost},数据迁移完成后集群的路径成本为 W'_{cost},则

$$W'_{\text{cost}} < W_{\text{cost}} \tag{19}$$

证明:根据定义 3 可知,集群的路径成本由节点间通信开销、节点内部进程间通信开销、节点内部线程间的通信开销与线程的计算开销这 4 部分组成.其中,节点间通信开销为 $W^{\text{edge}}_{\text{node cost}}$;节点内部线程间的通信开销为 $W^{\text{edge}}_{\text{executor cost}}$;线程的计算开销为 $W^{\text{executor}}_{\text{computation cost}}$;由于每个节点仅分配一个进程,则节点内部进程间的通信开销为 0.令节点间存在 n 条路径,节点内部线程间存在 m 条路径.集群拓扑数据迁移完成后,共有 s 条节点间路径发生改变,其中有 d 条节点间路径变为节点内部线程间路径,有 c 条节点间路径变为新的节点间路径.则当前节点间的

通信成本为 $W'^{\text{edge}}_{\text{node cost}}$,即

$$W'^{\text{edge}}_{\text{node cost}}=W^{\text{edge}}_{\text{node cost}}-\frac{sW^{\text{edge}}_{\text{node cost}}}{n}+\frac{cW^{\text{edge}}_{\text{node cost}}}{n}=\left(1-\frac{s}{n}+\frac{c}{n}\right)W^{\text{edge}}_{\text{node cost}}=\left(\frac{n-d}{n}\right)W^{\text{edge}}_{\text{node cost}} \tag{20}$$

当前节点内部线程间的通信成本为 $W'^{\text{edge}}_{\text{executor cost}}$,即

$$W'^{\text{edge}}_{\text{executor cost}}=W^{\text{edge}}_{\text{executor cost}}+\frac{dW^{\text{edge}}_{\text{executor cost}}}{m}=\left(\frac{m+d}{m}\right)W^{\text{edge}}_{\text{executor cost}} \tag{21}$$

由于数据迁移完成后,集群内的线程不发生改变,则当前线程的计算开销为 $W'^{\text{executor}}_{\text{computation cost}}$,即

$$W'^{\text{executor}}_{\text{computation cost}}=W^{\text{executor}}_{\text{computation cost}} \tag{22}$$

因此,数据迁移完成后集群的路径成本 W'_{cost} 为

$$W'_{\text{cost}}=W'^{\text{executor}}_{\text{computation cost}}+\left(\frac{n-d}{n}\right)W^{\text{edge}}_{\text{node cost}}+\left(\frac{m+d}{m}\right)W^{\text{edge}}_{\text{executor cost}} \tag{23}$$

此外,由于相同工作节点内线程间的通信开销可忽略不计,即

$$W'^{\text{edge}}_{\text{executor cost}}=0 \tag{24}$$

则存在:

$$\left(\frac{n-d}{n}\right)W^{\text{edge}}_{\text{node cost}}<W^{\text{edge}}_{\text{node cost}} \tag{25}$$

即获得式(19). □

此外,根据定理 2 可知,集群内节点间的通信开销降低.由于集群内所有的线程都是通过路径相互关联的,而非关键线程被重新分配,则在拓扑结构上,位于非关键线程下游所有线程(包括位于关键路径上的线程)的数据都将提前到达.因此集群内所有路径的计算延迟降低,进而达到提高集群性能的目的.

3 节能策略分析

基于以上理论分析,提出 Storm 平台下的线程重分配与数据迁移节能策略,该节能策略包括资源约束算法与数据迁移算法,在减少通信开销的前提下,提高了集群性能,并节约了集群的总能耗.图 5 为节能策略流程图.

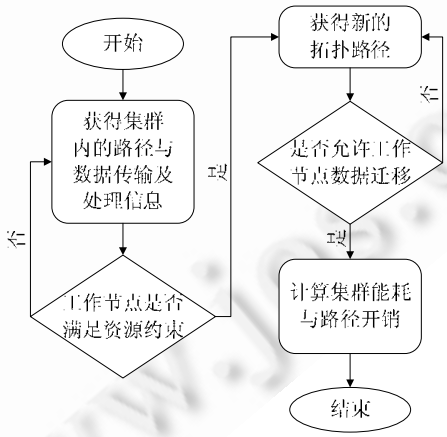


Fig.5 Flowchart of energy-efficient strategy

图 5 节能策略流程图

该策略主要分为以下 5 个步骤.

步骤 1:通过负载监控器获得原系统拓扑路径以及数据传输与处理的基本信息.

步骤 2:根据集群内数据的传输与处理,确定工作节点的资源约束.

步骤 3:根据资源约束、通信成本、RR 与 CPU 优先级,确定非关键线程的分配情况.

步骤 4:根据资源约束模型与最优线程重分配模型,确定集群内数据的迁移情况.

步骤 5:根据节能策略计算集群的路径成本与总能耗.

3.1 资源约束算法

为确定集群内工作节点的资源约束,采用资源约束算法,其中需要考虑工作节点 CPU、内存与网络带宽的资源占用率.该算法保证集群关键节点在满足资源约束的条件下进行数据迁移.此外,当关键节点的一类资源占用率达到极限时,则将该节点设置为极限节点,表示无法再将数据迁入该节点,因此需要重新选择关键节点.具体的算法描述在算法 1 中体现.

算法 1. 资源约束算法.

输入:关键节点的极限资源 $r_{n_i} \leftarrow (r_{n_i}^C, r_{n_i}^M, r_{n_i}^B)$,关键节点的初始资源 $o_{n_i} \leftarrow (o_{n_i}^C, o_{n_i}^M, o_{n_i}^B)$,关键节点迁入数据后增加的资源 $r'_{n_i} \leftarrow (r_{n_i}^{C'}, r_{n_i}^{M'}, r_{n_i}^{B'})$.

输出:允许关键节点迁入数据表示 tr.

初始化: $N''(C) \leftarrow \{n_1'', n_2'', ..., n_m''\}$. /*极限节点集合*/

1. **if** $o_{n_i}^C \geqslant r_{n_i}^C$ or $o_{n_i}^M \geqslant r_{n_i}^M$ or $o_{n_i}^B \geqslant r_{n_i}^B$ **then**
2. $N''(C) \leftarrow n_i$;
3. **else**
4. **while** n_i=tr **do**
5. **if** $r_{n_i}^{C'} + o_{n_i}^C \leqslant r_{n_i}^C$ **then**
6. $n_i.put(r_{n_i}^{C'}$,"CPU") ;
7. **else**
8. $n_i \neq$tr;
9. **end if**
10. **if** $r_{n_i}^{M'} + o_{n_i}^M \leqslant r_{n_i}^M$ **then**
11. $n_i.put(r_{n_i}^{M'}$,"DRAM") ;
12. **else**
13. $n_i \neq$tr;
14. **end if**
15. **if** $r_{n_i}^{B'} + o_{n_i}^B \leqslant r_{n_i}^B$ **then**
16. $n_i.put(r_{n_i}^{B'}$,"Network Bandwidth") ;
17. **else**
18. $n_i \neq$tr;
19. **end if**
20. **end while**
21. **end if**

算法 1 的输入参数为关键节点的极限资源、关键节点的初始资源与关键节点迁入数据后增加的资源;输出参数为允许关键节点迁入数据;初始化为极限节点集合.算法的第 1 行、第 2 行表示对关键节点 n_i 是否为极限节点进行判断:若为极限节点,则该节点不能被迁入数据;否则,需要判断工作节点数据迁入是否满足 3 条原则.算法的第 5 行~第 9 行表示对关键节点是否满足 CPU 资源临界原则进行判断:若满足,则判断之后的两条原则;否则,节点数据迁入不满足 tr.算法的第 10 行~第 14 行表示在满足 CPU 资源临界原则后,对关键节点是否满足内存资源临界原则进行判断:若满足内存资源临界原则,进入下一环节;否则,节点数据迁入不满足 tr.算法的第

15 行~第 19 行表示在满足之前的两条原则后,对关键节点是否满足网络带宽资源临界原则进行判断:若满足,则被选节点允许数据迁入;否则,节点数据迁入不满足 tr.

Storm 框架节能策略首先需要考虑算法对集群性能的影响,原集群内拓扑处理任务为轮询调度算法,其时间复杂度为 O(n).算法 1 首先需要对关键节点是否为极限节点进行判断,其时间复杂度为 O(1);其次,算法 1 的本质为依次判断数据迁入节点是否满足 3 条原则,其时间复杂度为 3O(1);最后,由于需要遍历整个集群满足 3 条原则的关键节点,类似于轮询调度算法,因此时间复杂度为 O(n).则算法 1 的时间复杂度 $T(A)$为

$$T(A)=O(1)+3O(1)+O(n)=O(n) \tag{26}$$

3.2 数据迁移算法

数据迁移算法主要包括两部分组成:其一为数据的迁移需要满足资源约束条件;其二为接收数据的节点需要满足最优线程重分配.该调度算法可以根据重写 Storm 平台的 IScheduler 接口[9]来实现.具体的算法描述在算法 2 中体现.

算法 2. 数据迁移算法.

输入:重分配后的线程集合 $E'(C)$;关键节点 CPU 的优先级 $n_{i_{\text{pr\_cpu}}}$.

输出:生成一条新的拓扑路径用于数据的传输与处理.

1. **if** 满足算法 1 允许数据迁入关键节点 **then**
2. **while** $E'(C)=e'_{ji}$ **do**
3. **if** 一个非关键线程存在一个父线程 **then**
4. $e'_{ab}\leftarrow e_{ab}$;
5. **end if**
6. **if** 一个非关键线程存在两个及以上父线程 **then**
7. 非关键线程的重分配需要考虑 RR;
8. 非关键线程的重分配需要考虑 $n_{i_{\text{pr\_cpu}}}$;
9. $e'_{ji}\leftarrow e_{ji}$;
10. **end if**
11. **if** 运行父线程的关键节点资源达到极限 **then**
12. 非关键线程的重分配需要考虑 RR;
13. 非关键线程的重分配需要考虑 $n_{i_{\text{pr\_cpu}}}$;
14. $e'_{mn}\leftarrow e_{mn}$;
15. **end if**
16. *master.remappingState*(Zookeeper);
17. 非关键节点的数据通过式(17)迁出;
18. 删除集群内非关键节点;
19. **end while**
20. *Zookeeper.update*(“configuration file”);
21. **end if**

算法 2 的输入参数为重分配后的线程集合与关键节点 CPU 的优先级;输出参数为生成一条新的拓扑路径用于数据的传输与处理.算法的第 1 行表示判断关键节点是否满足资源约束条件;算法的第 3 行~第 10 行表示需要减少集群内节点间的通信开销,并预防非关键线程重分配出现扎堆现象;算法的第 11 行~第 15 行表示避免关键节点出现资源溢出现象,并预防非关键线程重分配出现扎堆现象;算法的第 16 行表示重新计算 Zookeeper 内状态信息到节点的映射关系,为保证数据迁移后数据处理的一致性做铺垫;算法的第 19 行表示更新 Zookeeper 的配置文件,防止因拓扑的记忆功能而影响到集群数据的传输与处理.

为保证数据迁移后数据处理的一致性与正确性,在将数据从非关键线程内迁出时,第一,需要选择合适的关键节点并建立关键线程;第二,通过非关键线程的父线程将待处理的数据复制两份分别发送至两个子线程中,并由原非关键线程继续处理数据,实时产生计算结果并发送至新增的关键线程;第三,将集群拓扑内的状态信息存储至HDFS;第四,修改Zookeeper内状态信息到节点的映射关系,同时,由新增的关键线程以异步的方式从HDFS中拉取对应的状态信息,并执行状态的合并;第五,通过非关键线程的父线程将待处理的数据发送至新增的关键线程中,由新增的关键线程执行数据处理并向下游线程输出计算结果,以此来保证数据迁移后数据处理的一致性与正确性.

为确定数据迁移算法对时间开销带来的影响,算法2首先判断了关键节点是否满足算法1,其时间复杂度为O(1);其次,算法 2 通过遍历整个集群查找合适的关键节点来解决非关键子线程的重分配问题,其时间复杂度为O(n);此外,算法 2 在遍历整个集群过程中,通过不同的限制条件确定最合适的关键节点来进行非关键子线程的重分配,其时间复杂度为 3O(1);最后,算法 2 在非关键子线程分配完成后,将数据迁入相对应的关键线程,并更新Zookeeper 的配置文件,其时间复杂度为 O(1).则算法 2 的时间复杂度 $T(B)$为

$$T(B)=O(n)+3O(1)+O(1)=O(n) \tag{27}$$

此外,ERDM 由算法 1 与算法 2 组成,则 ERDM 的时间复杂度 $T(C)$为

$$T(C)=O(n)+O(n)=O(n) \tag{28}$$

3.3 节能模型

在 Storm 集群中,能耗一般分为基础能耗与动态能耗两种.其中,基础能耗为物理机的待机能耗,一般来说,同一类型物理机的待机能耗是一个固定常量;动态能耗是任务执行时集群产生的能耗,通常根据任务、功率与时间的不同,产生的动态能耗不同,因此动态能耗是一个变量.令单位时间 $t(s)$内基础能耗为 E_t^{base},动态能耗为 E_t^{dynamic},则集群的总能耗 E_t 为

$$E_t = E_t^{\text{base}} + E_t^{\text{dynamic}} \tag{29}$$

由于 E_t^{base} 是一个固定常量,在这里不做考虑.而 E_t^{dynamic} 作为一个变量,因此引入经典的物理公式表述,即 E_t^{dynamic} 单位为(J),可通过对单位时间 t 内的集群功率进行积分,其计算公式为

$$E_t^{\text{dynamic}} = \int_t^{t+\Delta t} P^{\text{dynamic}} \mathrm{d}t \tag{30}$$

其中,P^{dynamic} 单位为(W),表示时间段$[t,t+\Delta t]$内计算节点的功率.令原集群的动态能耗为 $E_t^{\text{original dynamic}}$,执行节能策略后集群的动态能耗为 $E_t^{\text{ERDM dynamic}}$,则执行节能策略后节约的能耗 $E_t^{\text{save dynamic}}$ 为

$$E_t^{\text{save dynamic}} = E_t^{\text{original dynamic}} - E_t^{\text{ERDM dynamic}} \tag{31}$$

将式(31)带入式(30),则集群节约的能耗可通过式(32)表示.

$$E_t^{\text{save dynamic}} = W_{\text{cost}} MP^{\text{dynamic}} - W'_{\text{cost}} MP^{\text{dynamic}} \tag{32}$$

其中,M 单位为(tuple),表示单位时间 t 内集群的数据量.现将式(4)和式(23)带入式(32),化简获得式(33):

$$\begin{aligned} E_t^{\text{save dynamic}} &= (W_{\text{computation cost}}^{\text{executor}} + W_{\text{node cost}}^{\text{edge}}) MP^{\text{dynamic}} - \left(W_{\text{computation cost}}^{\text{executor}} + \left(\frac{n-d}{n} \right) W_{\text{node cost}}^{\text{edge}} \right) MP^{\text{dynamic}} \\ &= W_{\text{node cost}}^{\text{edge}} MP^{\text{dynamic}} - \left(\frac{n-d}{n} \right) W_{\text{node cost}}^{\text{edge}} MP^{\text{dynamic}} \\ &= \frac{d}{n} W_{\text{node cost}}^{\text{edge}} MP^{\text{dynamic}} \end{aligned} \tag{33}$$

其中,式(33)中的相关参数与式(4)与式(23)相同.式(33)表示集群拓扑执行节能策略后节约的能耗.

3.4 算法的部署与实现

一个完整的 Storm 集群由主控节点、工作节点与关联节点这 3 类节点组成,其中,主控节点上运行 Nimbus 后台服务,是 Storm 集群的中心,负责接受用户提交的拓扑并为工作节点分配任务;工作节点上运行 Supervisor

后台服务,负责监听主控节点分配的数据并开启工作进程及工作线程;关联节点上运行 Zookeeper 后台服务,负责主控节点和工作节点间所有的关联协调,存储整个集群的状态信息与数据分配信息.为部署与实现 Storm 平台下的线程重分配与数据迁移节能策略,需要重写 Storm 平台 org.apache.storm.scheduler.IScheduler 接口[9]中的 schedule 方法,其原型为 public void schedule(topologies topologies, cluster cluster).本文在 Storm 集群原有框架的基础上新增了 6 个模块,如图 6 所示.

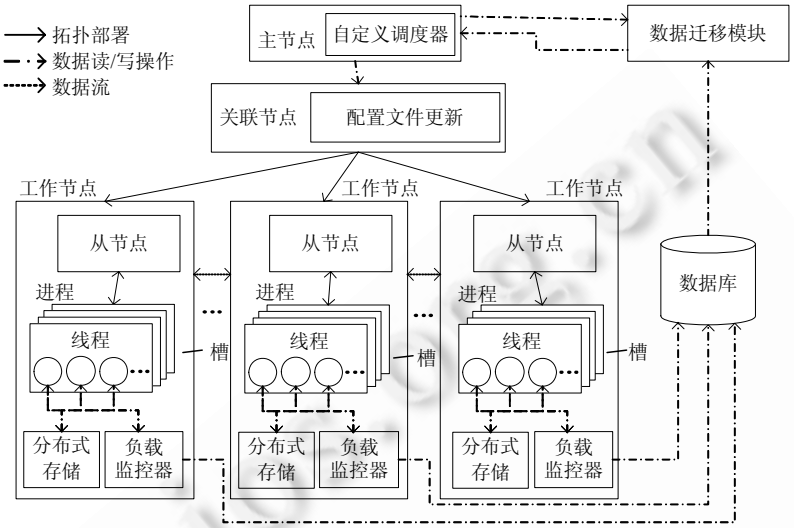


Fig.6 Improved architecture of Storm

图 6 改进后的 Storm 框架

新增模块的功能介绍如下.

(1) 负载监控器:在一定的时间窗口内,收集各线程 CPU、内存和网络带宽的资源占用信息以及各线程间数据流的大小.由于每个工作节点仅分配一个进程,因此可用线程监测的方法对任务运行时的各类信息进行采样与分析.各线程的 CPU 资源占用大小,可通过 Java API 函数中 ThreadMXBean 类的 getThreadCpuTime(long id)方法获得其 CPU 的占用时间,并与其所处工作节点的 CPU 主频相乘获得;各线程的内存资源占用大小,可通过 jmap -heap 指令进行检测;各线程的网络带宽占用信息,可通过实际测得的线程间数据流传输速率与实验中设置的元组大小进行累加,并由简单估算求得;线程间传输的数据流大小可使用计数器变量统计各线程接收到的上游线程发送的元组数量,并与时间窗口容量相除获得数据流传输速率.最后,CPU 的占用率与数据传输速率通过 nmon[38]软件获取,并由 Excel 表格导出.具体实现需添加在拓扑组件中各 Spout 的 *open*(·)和 *nextTuple*(·)方法以及各 Bolt 的 *prepare*(·)和 *execute*(·)方法中.

(2) 数据库:存储主控节点传来的数据分配信息和负载监控器传来的各类资源占用信息以及集群拓扑信息,并实时更新.

(3) 配置文件更新:针对算法 2 造成的集群路径的改变,实时更新 Zookeeper 的配置文件,防止因 Zookeeper 的记忆功能而出现数据传输错误的问题.

(4) 数据迁移模块:部署算法 1 与算法 2,负责读取数据库中的各类信息,并执行数据迁移算法.该模块为本文算法部署的核心环节,亦是集群执行节能策略的基础.

(5) 分布式存储:存储集群拓扑内的状态信息,并在算法 2 执行过程中,以异步的方式从 HDFS 中拉取对应的状态信息,保证了数据迁移后数据处理的一致性和正确性.

(6) 自定义调度器:覆盖 Nimbus 节点默认的调度策略,读取数据迁移模块内的决策并执行.此外,代码编译

完成后,将其打 jar 包至主控节点 Nimbus 的 STORM_HOME/lib 目录下,并在/conf/storm.yaml 中配置好相关参数后运行.

此外,本文的负载监控器与工作节点上运行的任务分别位于同一台物理节点的两个不同进程中,由于两个进程之间的内存区域互相隔离,负载监控进程并不会使用工作进程的内存区域,故对节点的内存资源开销没有影响;负载监控器与节点的工作进程位于相同的工作节点,则不存在网络带宽的开销;负载监控器收集信息所占用的 CPU 资源非常少,因此对节点 CPU 资源开销的影响可忽略不计;负载监控器收集信息会上传至数据库,且负载监控器与数据库位于不同的工作节点,故存在一定网络带宽的开销,该网络带宽的开销与时间窗口设置的大小相关,具体结果在第 4.1 节体现.

本文提出了算法 1 与算法 2,设计并实现了 Storm 平台下的线程重分配与数据迁移节能策略,减少了集群内的通信开销,提高了集群性能,并节约了能耗.

4 实验结果及数据分析

为评估集群执行 ERDM 的性能与能耗,本节首先讨论了集群的实验环境与参数集,然后对实验结果进行讨论与分析.

4.1 实验环境

为验证 ERDM 的有效性,实验搭建的 Storm 集群包括 19 台 PC 机,其中 1 台 PC 机上运行 1 个 Nimbus 进程、1 个 UI 进程与数据库.16 台 PC 机上运行 Supervisor 进程,3 台 PC 机上运行 Zookeeper 进程.此外,每台 PC 机的网卡统一为 100Mb/s LAN,且内存统一为 8GB.根据不同节点的运行状况,具体环境配置见表 1 和表 2.

Table 1 Hardware configuration of Storm cluster

表 1 Storm 集群的硬件配置

节点	CPU	内存	网络带宽
Nimbus ZooKeeper 1(leader)	Intel core i7 4790 3.6 GHz Quad Core	8GB DDR3 1 066MHz	100Mb/s LAN
Supervisor 1~16	Intel core i7 4790 3.6 GHz Quad Core	8GB DDR3 1 066MHz	100Mb/s LAN
ZooKeeper 2, 3 (follower)	Intel core i7 4790 3.6 GHz Quad Core	8GB DDR3 1 066MHz	100Mb/s LAN

Table 2 Software configuration of Storm cluster

表 2 Storm 集群的软件配置

参数	数值
OS	CentOS 6.8
Storm	1.0.3
Hadoop	2.7.4
JDK	1.8.0_121
Zookeeper	3.4.6
Python	2.6.6
MySQL	5.7.18
VMware	12.1.1

此外,为在各类不同资源开销下验证 ERDM 的有效性,实验选取 Intel 公司 Zhang 等人[19]发布在 GitHub 上的 4 组基准测试,分别为 CPU 敏感型(CPU-sensitive)的 WordCount、网络带宽敏感型(network-sensitive)的 Sol、内存敏感型(memory-sensitive)的 RollingSort 以及 Storm 在真实场景下的应用 RollingCount.为消除节点内部进程间的通信开销,执行各基准测试需满足工作节点与工作进程的数量保持一致(即一个工作节点内仅分配一个工作进程),其余参数保留其默认值,具体的参数配置见表 3.

表 3 中的 *component.xxx_num* 为基准测试的组件并行度,Sol 中的 *topology.level* 为拓扑的层次,需要与 *component.xxx_num* 结合在一起分析.由于拓扑存在 Spout 与 Bolt 两种组件,且 1 个 Spout 对应 2 个 Bolt,因此拓扑的层次设置为 3.其中,1 个 Spout 组件运行着 50 个实例,2 个 Bolt 组件运行着 100 个实例.*topology.works* 统一

设置为 16,表示各基准测试运行时,一个工作节点内仅分配一个工作进程;*topology.acker.executors* 统一设置为 16,表示保证集群内数据流的可靠传输;此外,为防止数据传输因超时而发生重传,需要通过多次实验验证结果,实验结果为 *topology.max.spout.pending* 统一设置为 200;最后,统一设置每个 *message.size* 等于一个 tuple 的大小.

Table 3 Configuration of benchmarks

表 3 基准测试参数配置

基准侧试	参数	数值
WordCount	*component.spout_num*	50
	component.split_bolt_num	100
	component.count_bolt_num	100
	topology.works	16
	topology.acker.executors	16
	topology.max.spout.pending	200
RollingSort	*component.spout_num*	50
	*component.sort*_bolt_*num*	100
	emit.frequency	10
	chunk.size	2 000 000
	message.size	100 000
Sol	*topology.level*	3
	message.size	2 000
	component.spout_num	50
	component.bolt_num	100
RollingCount	*component.spout_num*	50
	component.split_bolt_num	100
	component.rolling_count_bolt_num	100
	window.length	150
	emit.frequency	30

为了验证 ERDM 的效果,本文还与 TMSH-Storm[18]、LEEDSP[35]和 WNDVR-Storm[37]进行了对比实验.其中,

- TMSH-Storm 的核心思想是:对集群的任务调度进行优化,继而达到提高集群性能的目的.
- LEEDSP 的核心思想是:弹性调节集群节点的资源,并通过 DVFS 技术动态调节节点 CPU 的电压,以此达到节能的效果.且该策略为流式处理节能策略的主要代表,适用于大多数流式处理平台(如 Storm、Flink[10]以及 Spark Streaming[11]等).
- WNDVR-Storm 的核心思想为:通过动态调节工作节点的内存电压而达到节能的效果,且 WNDVR-Storm 由非关键路径内存电压调节(DRAM voltage regulation on non-critical path,简称 DVRNP)与关键路径内存电压调节(DRAM voltage regulation on critical path,简称 DVRCP)两种算法组成.

此外,为保证在同等条件下验证本文策略的效果,TMSH-Storm、LEEDSP 以及 WNDVR-Storm 的相关参数与 ERDM 保持一致.

为选择合适的时间窗口用于监控集群内各节点资源的负载信息,本文以 WordCount 为例,在系统默认调度策略下,根据额外增加的网络开销与系统延迟为条件选择合适的时间窗口取值,具体的结果见表 4.

Table 4 Choose of time windows

表 4 时间窗口的选择

时间窗口取值(s)	额外增加的网络开销(%)	额外增加的系统延迟(ms)
10	17.4	7.5
20	10.3	13.3
30	6.1	27.1
40	4.6	51.8
50	3.2	94.7

根据表 4 可知,当时间窗口为 10s 与 20s 时,集群内额外增加的网络开销较大.其原因为:时间窗口取值较低而导致集群内数据库的读写过于频繁,读写数据库产生的网络开销相对较大,从而影响集群拓扑内任务的正常执行,造成无法触发 ERDM 的问题.当时间窗口为 40s 和 50s 时,集群内额外增加的系统延迟较高,已影响到集群

拓扑内的任务正常执行.其原因为:集群内数据库的读写过于缓慢而延后了 ERDM 的触发时机,进而影响集群拓扑内的任务执行效率,造成影响集群性能的问题.综上所述,时间窗口的取值设置为 30s,在该时间窗口下能够较好满足实验的执行.此外,同理可得 Sol、RollingSort 以及 RollingCount 的时间窗口取值.

4.2 数据迁移有效性测试

为便于观察集群内数据迁移完成后,关键节点的资源占用情况,首先需要确定集群内的节点类型,即关键节点与非关键节点.根据表 1 可知,共存在 16 个工作节点.为查找集群内关键节点与非关键节点的分布情况,则通过 Storm UI 检测集群执行 4 个基准测试后的实验结果,具体实验结果如图 7 所示.

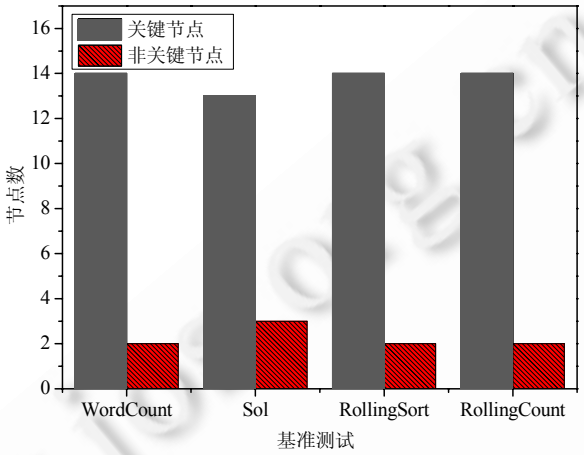
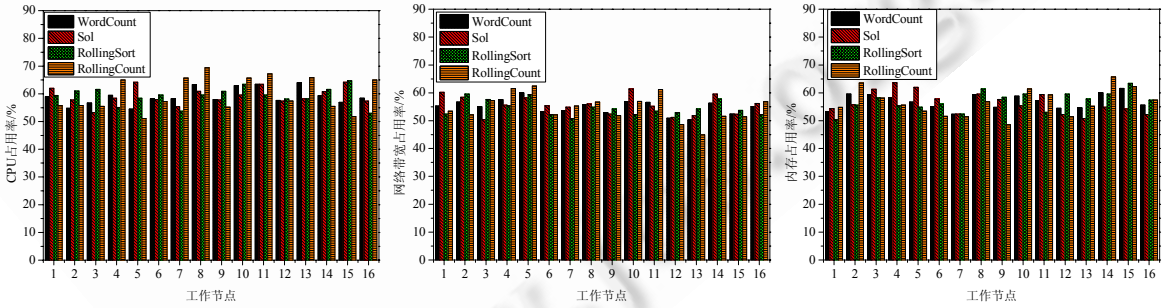


Fig.7 Node distribution under different benchmarks

图 7 不同基准测试下的节点分布情况

由图 7 可以看出:集群执行不同的基准测试关键节点与非关键节点分布并不相同,WordCount 下集群存在 2 个非关键节点与 14 个关键节点,Sol 下集群存在 3 个非关键节点与 13 个关键节点,RollingSort 下集群存在 2 个非关键节点与 14 个关键节点,RollingCount 下集群存在 2 个非关键节点与 14 个关键节点.此外,为对比数据迁移完成后,各工作节点资源占用率的变化,需要对原集群各工作节点的资源占用率进行检测,具体结果如图 8 所示.



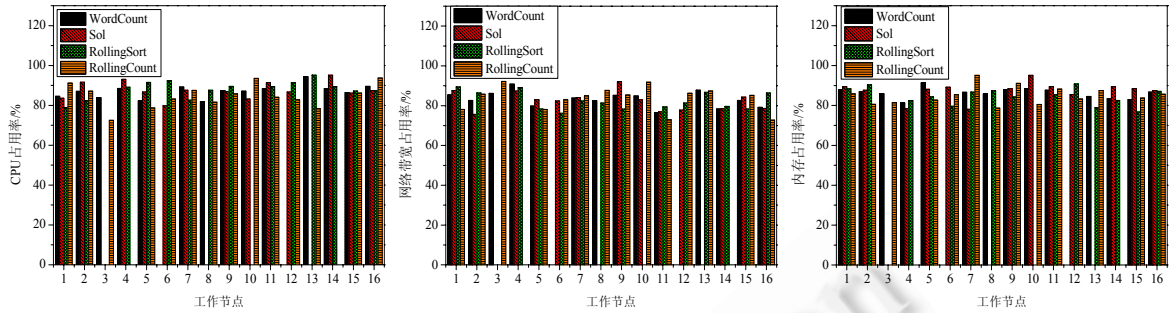
(a) CPU 占用率 (b) 网络带宽占用率 (c) 内存占用率

Fig.8 Resources utilization of 16 work nodes in the original cluster

图 8 原集群 16 个工作节点的资源占用率

图 8 为原集群运行 4 个基准后,16 个工作节点(关键节点和非关键节点)的平均资源占用率.如图 8 所示,原集群 16 个工作节点 3 类资源的平均占用率主要集中在 50%~70%,这为集群执行数据迁移奠定了基础.数据迁移完成后,各基准测试根据节点分布对 16 个工作节点资源占用率的平均值进行观测,验证资源约束算法的实际效

果.具体结果如图 9 所示.



(a) CPU 占用率 (b) 网络带宽占用率 (c) 内存占用率

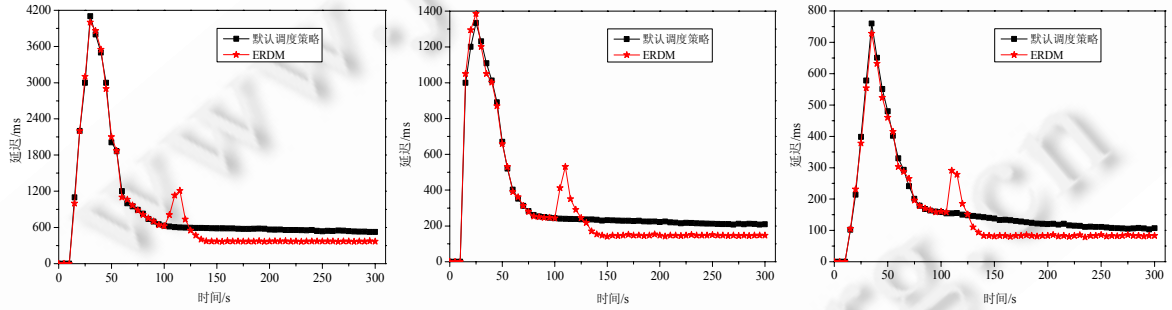
Fig.9 Resources utilization of 16 work nodes after the cluster data migration

图 9 集群数据迁移后 16 个工作节点的资源占用率

如图 9 所示,数据迁移完成后,集群 16 个工作节点 3 类资源的平均占用率主要集中在 80%~100%.图 9 中缺失的部分为非关键节点,表示非关键线程重分配后,非关键节点已不存在线程与数据,因此予以删除.此外,非关键节点在集群拓扑中的分布并不相同,表示运行不同基准测试下的拓扑并不相同.

为确定集群在执行算法时,算法的响应时间对集群性能的影响,现通过测试一个元组从 Spout 出发到最终被集群拓扑处理完成后所产生的时长,以此反映集群数据的处理效率.

图 10 统计了 WordCount、Sol 与 RollingSort 这 3 种基准测试在系统默认的调度策略与 ERDM 下的延迟.



(a) WordCount (b) Sol (c) RollingSort

Fig.10 Comparison of system latency under different benchmarks

图 10 在不同的基准测试下系统延迟的比较

如图 10 所示,3 种基准测试下集群执行 ERDM 的平均延迟为 801.33ms,279.28ms,131.02ms,与 Storm 默认的调度策略相比,平均降低了 6.3%,8.7%,10.4%.这是由于节点间的通信开销降低而导致集群路径的延迟下降.在 105s 前存在一个峰值,且 Storm 默认的调度策略与 ERDM 的延迟基本相同,表示集群在提交各拓扑时的部署过程,并且 105s 前集群统一遵循 Storm 默认的调度策略.在 105s 时,集群触发数据迁移算法,数据的传输延迟出现峰值.这是由于数据迁移算法根据集群 CPU、网络带宽与内存的负载以及线程间数据流的大小情况,对所有包含线程的工作节点资源进行重新分配,相当于对集群的任务进行初始化分配,此时集群的开销较大,因此导致数据流因无法被及时处理而使延迟急剧上升.根据图 10 可知,3 种基准测试下集群触发数据迁移算法后,其平均延迟高于 Storm 默认的调度策略的时长为[105s,135s],共耗时约 30s;但是由于时间间隔相对较短,故对整个集群拓扑数据处理性能造成的影响可忽略不计.此外,3 种基准测试的延迟并不相同,这是由于不同的基准测试,组件中包含的线程数量并不相同,但对实验的结果不会造成影响.综上所述,数据迁移算法在执行过程中并不会对集

群数据传输与处理的实时性造成影响.

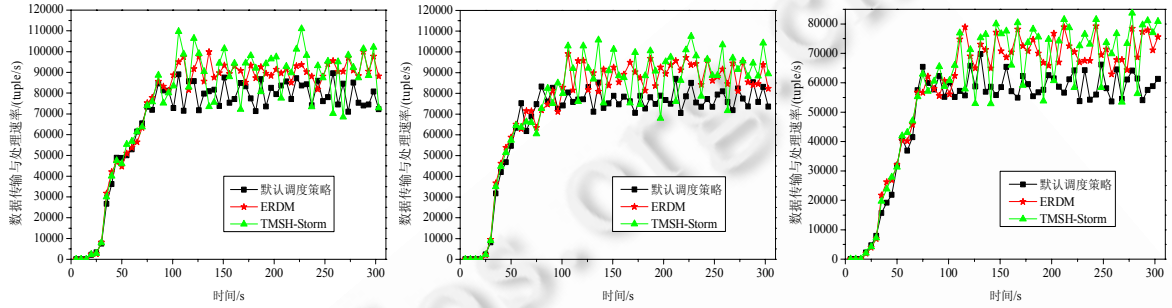
此外,可通过使用布隆过滤器对集群拓扑内的数据进行预处理,删除数据集内的重复数据,导致集群拓扑单位时间内处理及传输的数据量减少,从而降低了在执行 ERDM 过程中集群拓扑数据传输延迟过长的问题.

4.3 节能策略的实验结果与分析

ERDM 的评估标准主要体现在集群性能与能耗两个指标.

(1) 集群性能

集群执行 ERDM 后的性能由集群节点间的通信开销判断,集群性能可通过单位时间内数据的传输与处理速率决定.具体结果如图 11 所示.此外,集群性能也可由 Storm UI(Storm 平台提供)进行计算.引入 TMSH-Storm 与 ERDM 作对比,以验证 ERDM 的实际效果.



(a) WordCount (b) Sol (c) RollingSort

Fig.11 Comparison of data processing and transmission rate under different benchmarks

图 11 在不同的基准测试下比较数据传输与处理速率

如图 11 所示,集群在执行 ERDM 后,各基准测试(WordCount、SOL、RollingSort)在单位时间内的数据传输与处理速率得到了改善.执行 ERDM 后,集群中数据传输与处理速率的平均值为 77 572(tuple/s),76 471(tuple/s)和 59 763(tuple/s),与 Storm 默认的调度策略相比提高了 13.7%,13.3%和 18.2%.其原因为:与 Storm 默认的调度策略相比,由于执行 ERDM 减少了节点间的通信开销,降低了路径的计算延迟,从而导致集群数据传输与处理的总时间减少.因此,集群中数据传输与处理的速率得到了改善,单位时间内使集群增加了数据流的大小.集群执行 TMSH-Storm 后,数据传输与处理速率的平均值为 80 170(tuple/s),81 639(tuple/s)和 62 647(tuple/s),与 ERDM 相比提高了 3.3%,4.9%和 5.3%.但是执行 TMSH-Storm 时,集群数据传输与处理速率的波动较大.这是由于 TMSH- Storm 在任务迁移过程中并未考虑线程迁移出现扎堆现象,导致任务并未按照原定计划进行迁移,从而增加了额外的节点间通信开销,故对集群数据传输与处理的速率造成了一定的影响.此外,从优化的角度来看,拓扑内线程的总数为 326 个、286 个和 318 个.执行 ERDM 后,集群重新分配非关键线程的个数为 17 个、21 个和 15 个,其中,从节点间的数据传输改变为线程之间数据传输的线程个数为 14 个、18 个和 12 个,且平均完成一次线程之间数据传输的改变可降低节点间的通信成本为 0.8%,1.1%和 1.5%.因此,集群平均节约的通信成本为 11.2%,19.8%和 18%.图 12 显示了在实际应用场景下集群拓扑内数据传输与处理速率.

如图 12 所示,在执行 ERDM 运行 RollingCount 后,集群中数据传输与处理速率的平均值为 57 530(tuple/s),与 Storm 默认的调度策略相比提高了 12.5%.在执行 TMSH-Storm 运行 RollingCount 后,由于集群数据传输与处理速率波动较大的原因,其平均值为 59 800(tuple/s),相比于 Storm 默认的调度策略提高了 15.8%.两种策略的差距较小,但是 ERDM 的稳定性更佳.此外,拓扑内线程的总数为 266 个,执行 ERDM 后,集群重新分配非关键线程的个数为 14 个,其中,从节点间的数据传输改变为线程之间数据传输的线程个数为 11 个,且平均完成一次线程之间数据传输的改变可降低节点间的通信成本为 1.3%,则集群平均节约的通信成本为 14.3%.因此,相比于 Storm 默认的调度策略,本文提出的 ERDM 具有更好的集群性能.

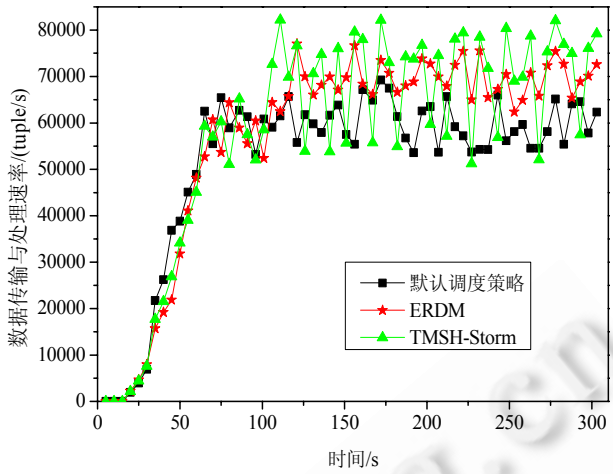


Fig.12 Comparison of data processing and transmission rate under the RollingCount

图 12 在 RollingCount 下比较数据传输与处理速率

(2) 集群能耗

集群能耗反映了集群中数据传输与处理总的能量消耗.本文通过集群拓扑中数据传输与处理的功率与总成本开销相乘,以计算集群能耗.具体节约的能耗可通过式(33)计算.此外,引入 WNDVR-Storm、LEEDSP 与 ERDM 作对比,以验证 ERDM 的实际效果.具体的实验结果如图 13 所示.

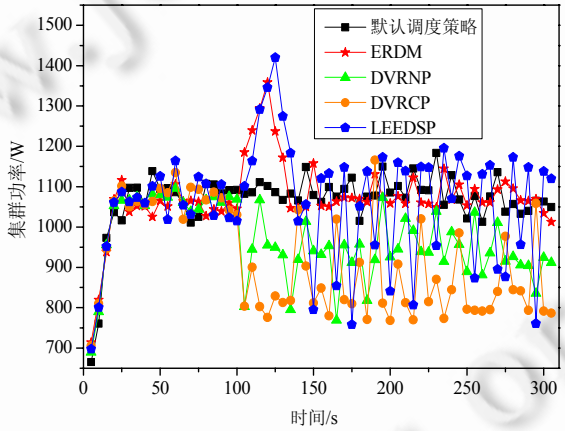


Fig.13 Comparison of power on RollingCount among different strategies

图 13 RollingCount 在不同策略下的功率对比

如图 13 所示,在执行 ERDM 运行 RollingCount 后,集群的平均功率为 1 065.37W,执行 Storm 默认调度策略的平均功率为 1 063.15W,执行 DVRNP 与 DVRCP 的平均功率为 1 045.32W 与 1023.17W,执行 LEEDSP 的平均功率为 1 073.71W.相比于 Storm 默认的调度策略,在 105s 前,两种算法的功率基本相等,其原因为数据迁移算法尚未触发.但在[105,115s]内执行 ERDM,功率急剧升高.其原因为:在[105s,115s]内集群拓扑执行数据迁移算法,工作节点的资源占用率消耗巨大,导致集群功率急速上升.而 115s 后,数据迁移算法执行完毕,集群功率逐渐降低,并于 130s 后趋于稳定,因此不会对单位时间内的集群功率造成影响.与 WNDVR-Storm 相比,执行 ERDM 的平均功率高于 WNDVR-Storm,但是执行 WNDVR-Storm 的功率波动较大,非常不稳定,且策略实现较为困难,不适合在大规模集群中使用.与 LEEDSP 相比,执行 ERDM 的平均功率略低于 LEEDSP.其原因为:LEEDSP 并未考虑除 CPU 之外部件(如内存与网络带宽等)的功率问题,然而 Storm 集群拓扑在执行任务时,内存与网络带宽的

功率相对较高是不容忽视的.此外,使用DVFS技术调节节点CPU电压存在较大的偶然性,非常不稳定,且技术实现较为困难,同样不适合部署在大规模地集群当中.为计算集群的能耗,需要对集群内的功率进行积分,具体结果如图 14 所示.

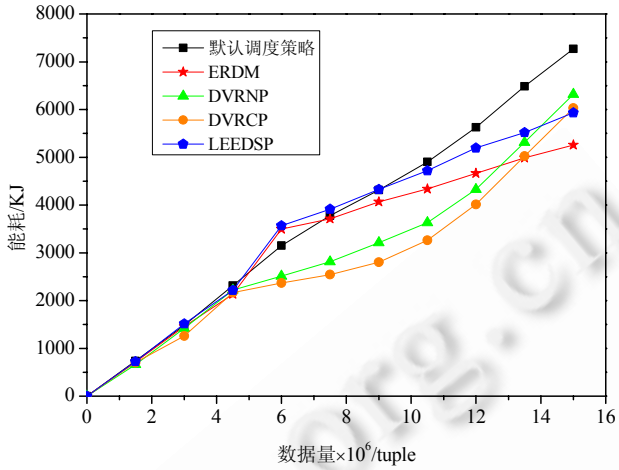


Fig.14 Comparison of energy consumption on RollingCount among different strategies

图 14 RollingCount 在不同策略下的能耗对比

图 14 为 RollingCount 在不同策略下集群处理相同数据量的能耗,其中,执行 ERDM 集群的能耗为 5 286.8KJ,执行 Storm 默认调度策略的能耗为 7 270.3KJ,执行 DVRNP 与 DVRCP 的能耗为 6 317.3KJ 与 6 031.5KJ,执行 LEEDSP 的能耗为 5 934.7KJ.与 Storm 默认调度策略相比,执行 ERDM 集群能耗节约了 27.3%.但是在[4300000tuple,6000000tuple]内,集群执行 ERDM 的能耗高于 Storm 默认的调度策略.其原因为在[4300000tuple,6000000tuple]内集群拓扑执行数据迁移算法,导致集群能耗升高.而在[9000000tuple,15000000tuple]执行 ERDM 的能耗远低于 Storm 默认的调度策略,其原因为数据迁移算法执行完毕,由于路径的计算延迟减少导致集群拓扑内的数据传输与处理速率高于 Storm 默认的调度策略,因此相同数据量下能耗远低于 Storm 默认的调度策略.与 WNDVR-Storm 相比,执行 ERDM 集群的能耗略低于 WNDVR-Storm,但是执行 WNDVR-Storm 的能耗上升幅度不断变化且逐渐升高.这是由于随着集群数据量的不断增大,数据量始终超过额定阈值,导致 WNDVR-Storm 基本失效.与 LEEDSP 相比,执行 ERDM 集群的能耗始终低于 LEEDSP.这是由于 LEEDSP 并未考虑内存与网络带宽等部件的能耗,且执行 LEEDSP 内资源调度算法的能耗高于执行 ERDM 内数据迁移算法的能耗.此外,随着节点数的增加,集群内存与网络带宽的能耗比重会不断增大,从而始终影响 LEEDSP 的节能效果.为量化集群内数据的传输与处理时间与能耗的关系,需要集群在相同数据量下对 ERDM 与 Storm 默认调度策略进行对比,具体的实验结果如图 15 所示.

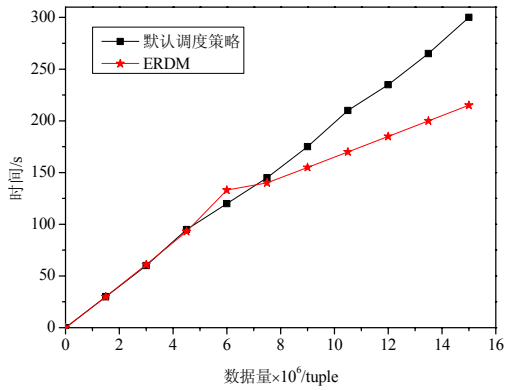


Fig.15 Comparison of data processing and transmission time under the RollingCount

图 15 在 RollingCount 下比较数据传输与处理时间

图 15 为 RollingCount 在相同数据量下两种策略的时间对比,其中,15 000 000 tuple 下执行 ERDM 集群的时间为 216s,而执行 Storm 默认调度策略的时间

为 300s.与 Storm 默认调度策略相比,相同数据量下执行 ERDM 集群拓扑中数据的传输与处理时间提高了 28%.因此可以确定:在相同条件下,集群数据传输与处理的时间每提高 1%,则集群的能耗降低 1%.综上所述,相比于 Storm 默认的调度策略,本文提出的 ERDM 具有更好的节能效果.

5 总结与展望

高能耗问题,是限制大数据流式处理平台发展的主要障碍之一.Storm 是大数据流式处理中最具代表性的平台之一,但是在最初的设计中并未考虑能耗问题,从而导致目前高能耗问题始终制约其发展.针对这一问题,本文通过研究 Storm 集群的拓扑结构,建立了资源约束模型与最优线程重分配模型,并进一步提出了 Storm 平台下的线程重分配与数据迁移节能策略.该策略由资源约束算法与数据迁移算法组成,使集群在减少节点间通信成本的前提下,缩短了数据传输与处理的时间,并节约了能耗.最后,实验通过 4 组基准测试,从资源占用、性能与能耗的角度验证了策略的有效性.

下一步的研究工作主要包括以下 4 个方面:(1) 将 ERDM 进一步部署到更为复杂的商业应用领域,使其可以在更广阔的应用场景下使用;(2) 将布隆过滤器运用到集群,通过对集群拓扑内的数据进行预处理,删除数据集内的重复数据,使集群单位时间内处理及传输的数据量减少,从而降低了集群延迟,并节约了能耗;(3) 目前,Storm 集群内部的电子元件限制了性能与能效的发展,可通过替换高能效的电子元件,以提高集群的性能,并节约能耗;(4) 目前,集群拓扑内的进程与线程的数量需要用户手动设置,研究拓扑内组件并行度自适应调节的调度算法,由此提高了资源利用率,并节约了能耗.

References:

[1] Sun DW, Zhang GY, Zheng WM. Big data stream computing: Technologies and instances. Ruan Jian Xue Bao/Journal of Software, 2014,25(4):839−862 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4558.html [doi: 10.13328/j.cnki.jos.004558]

[2] Guo BL, Yu J, Liao B, Yang DX, Lu L. A green framework for DBMS based on energy-aware query optimization and energy-efficient query processing. Journal of Network and Computer Applications, 2017,84:118−130. [doi: 10.1016/j.jnca.2017.02.015]

[3] Guo BL, Yu J, Yang DX, *et al*. Energy modeling and plan evaluation for queries in relational databases. Journal of Computer Research and Development, 2019,56(4):810−824 (in Chinese with English abstract).

[4] Zhao XG, Hu QP, Ding L, *et al*. Energy saving scheduling strategy based on model prediction control for data centers. Ruan Jian Xue Bao/Journal of Software, 2017,28(2):429−442 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5026. html [doi: 10.13328/j.cnki.jos.005026]

[5] Yang T, Wang M, Zhang YJ, *et al*. HDFS differential storage energy-saving optimal algorithm in cloud data center. Chinese Journal of Computers, 2019,42(4):47−61 (in Chinese with English abstract).

[6] Seagate. Data age 2025. 2017. https://www.seagate.com/files/www-content/our-story/trends/files/data-age-2025-white-paper-simplified-chinese.pdf

[7] Sun DW, Zhang GY, Yang SL, Zheng WM, Khan SU, Li KQ. Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. Information Sciences, 2015,319:92−112. [doi: 10.1016/j.ins.2015.03.027]

[8] Cheng D, Chen Y, Zhou X, Gmach D, Milojicic D. Adaptive scheduling of parallel jobs in spark streaming. In: Proc. of the IEEE Conf. on Computer Communications. Piscataway: IEEE, 2017. 1−9. [doi: 10.1109/INFOCOM.2017.8057206]

[9] Apache. Storm. 2017. http://storm.apache.org

[10] Li ZY, Yu J, Bian C, *et al*. Flow-network based auto rescale strategy for Flink. Journal on Communications, 2019,40(8):85−101 (in Chinese with English abstract).

[11] Ying CT, Yu J, Bian C, *et al*. Criticality checkpoint management strategy based on RDD characteristics in Spark. Journal of Computer Research and Development, 2017,54(12):2858−2872 (in Chinese with English abstract).

[12] Kulkarni S, Bhagat N, Fu M, Kedigehalli V, Kellogg C, Mittal S, Patel JM, Ramasamy K, Taneja S. Twitter heron: Stream processing at scale. In: Proc. of the ACM Conf. on Management of Data. New York: ACM, 2015. 239−250. [doi: 10.1145/2723372.2742788]

[13] Chen GJ, Wiener JL, Iyer S, Jaiswal A, Lei R, Simha N, Wang W, Wilfong K, Williamson T, Yilmaz S. Realtime data processing at Facebook. In: Proc. of the ACM Conf. on Management of Data. New York: ACM, 2016. 1087−1098. [doi: 10.1145/2882903.2904441]

[14] Xhafa F, Naranjo V, Caballe S. Processing and analytics of big data Streams with Yahoo!S4. In: Proc. of the IEEE Conf. on Advanced Information Networking & Applications. Piscataway: IEEE Computer Society, 2015. 263−270. [doi: 10.1109/AINA.2015.194]

[15] Tian XH, Han R, Wang L, Lu G, Zhan JF. Latency critical big data computing in finance. The Journal of Finance and Data Science, 2015,1(1):33−41. [doi: 10.1016/j.jfds.2015.07.002]

[16] Ta VD, Liu CM, Nkabinde GW. Big data stream computing in healthcare real-time analytics. In: Proc. of the IEEE Conf. on Cloud Computing and Big Data Analysis (ICCCBDA). Piscataway: IEEE, 2016. 37−42. [doi: 10.1109/ICCCBDA.2016.7529531]

[17] Batyuk A, Voityshyn V. Apache Storm based on topology for real-time processing of streaming data from social networks. In: Proc. of the IEEE Conf. on Data Stream Mining & Processing (DSMP). Piscataway: IEEE, 2016. 345−349. [doi: 10.1109/DSMP.2016.7583573]

[18] Lu L, Yu J, Bian C, *et al*. A task migration strategy in big data stream computing with Storm. Journal of Computer Research and Development, 2018,55(1):71−92 (in Chinese with English abstract).

[19] Zhang M. Intel-Hadoop/Storm-benchmark forked from manuzhang/storm-benchmark. 2015. https://github.com/intel-hadoop/storm-benchmark

[20] Shabestari F, Rahmani AM, Navimipour NJ, Jabbehdari S. A taxonomy of software-based and hardware-based approaches for energy efficiency management in the Hadoop. Journal of Network and Computer Applications, 2019,126:162−177. [doi: 10.1016/j.jnca.2018.11.007]

[21] Zhou S, Chelmis C, Prasanna VK. High-Throughput and energy-efficient graph processing on FPGA. In: Proc. of the IEEE Conf. on Field-Programmable Custom Computing Machines. Piscataway: IEEE, 2016. 103−110. [doi: 10.1109/FCCM.2016.35]

[22] Jin Z, Finkel H, Yoshii K, Cappello F. Evaluation of a floating-point intensive kernel on FPGA. In: Proc. of the European Conf. on Parallel Processing. Springer-Verlag, 2017. 664−675. [doi: 10.1007/978-3-319-75178-8_53]

[23] De MT, Mencagli G. Proactive elasticity and energy awareness in data stream processing. Journal of Systems and Software, 2017, 127:302−319. [doi: 10.1016/j.jss.2016.08.037]

[24] Wang ZW, Wang H, Zhao WQ, Cheng LL. Energy optimization of parallel programs in a heterogeneous system by combining processor core-shutdown and dynamic voltage scaling. Future Generation Computer Systems, 2019,92:198−209. [doi: 10.1016/j.future.2018.09.039]

[25] Pietri I, Zhuang S, Casas M, Moretó M, Sakellariou R. Evaluating scientific workflow execution on an asymmetric multicore processor. In: Proc. of the European Conf. on Parallel Processing. Springer-Verlag, 2017. 439−451. [doi: 10.1007/978-3-319-75178-8_36]

[26] Schneider FP, Wienke S, Müller MS. Operational concepts of GPU systems in HPC centers: TCO and productivity. In: Proc. of the European Conf. on Parallel Processing. Springer-Verlag, 2017. 452−464. [doi: 10.1007/978-3-319-75178-8_37]

[27] Campeanu G, Saadatmand M. Run-Time component allocation in CPU-GPU embedded systems. In: Proc. of the ACM Conf. on Applied Computing. New York: ACM, 2017. 1259−1265. [doi: 10.1145/3019612.3019785]

[28] Lang J, Rünger G. An execution time and energy model for an energy-aware execution of a conjugate gradient method with CPU/GPU collaboration. Journal of Parallel and Distributed Computing, 2014,74(9):2884−2897. [doi: 10.1016/j.jpdc.2014.06.001]

[29] Liao B, Yu J, Zhang T, Guo BL, Sun H, Ying CT. Energy-Efficient algorithms for distributed storage system based on block storage structure reconfiguration. Journal of Network and Computer Applications, 2015,48:71−86. [doi: 10.1016/j.jnca.2014.10.008]

[30] Liao B, Zhang T, Yu J, *et al*. Energy consumption modeling and optimization analysis for MapReduce. Journal of Computer Research and Development, 2016,53(9):2107−2131 (in Chinese with English abstract).

[31] Cordeschi N, Shojafar M, Amendola D, Baccarelli E. Energy-saving QoS resource management of virtualized networked data centers for big data stream computing. In: Proc. of the Emerging Research in Cloud Distributed Computing Systems. Hershey: IGI Global, 2015. 1−31. [doi: 10.4018/978-1-4666-8213-9.ch004]

[32] Dayarathna M, Li Y, Wen Y, Fan R. Energy consumption analysis of data stream processing: A benchmarking approach. Software: Practice and Experience, 2017,47(10):1443−1462. [doi: 10.1002/spe.2458]

[33] Maroulis S, Zacheilas N, Kalogeraki V. Express: Energy efficient scheduling of mixed stream and batch processing workloads. In: Proc. of the IEEE Conf. on Autonomic Computing (ICAC). Piscataway: IEEE, 2017. 27−32. [doi: 10.1109/ICAC.2017.43]

[34] Veiga J, Enes J, Expósito RR, Touriño J. BDEv 3.0: Energy efficiency and microarchitectural characterization of big data processing frameworks. Future Generation Computer Systems, 2018,86:565−581. [doi: 10.1016/j.future.2018.04.030]

[35] De MT, Mencagli G. Keep calm and react with foresight: Strategies for low-latency and energy-efficient elastic data stream processing. Journal of Systems and Software, 2016,51(8):1−12. [doi: 10.1145/3016078.2851148]

[36] Pu YL, Yu J, Lu L, *et al*. Energy-efficient strategy based on data recovery in storm. Journal of Computer Research and Development, 2021,58(3):479−496 (in Chinese with English abstract).

[37] Pu YL, Yu J, Lu L, *et al*. Energy-efficient strategy for work node by DRAM voltage regulation in Storm. Journal on Communications, 2018,39(10):101−121 (in Chinese with English abstract).

[38] Griffiths N. nmon analyser—A free tool to produce AIX performance reports. 2020. https://developer.ibm.com/articles/au-nmon_analyser/

附中文参考文献:

[1] 孙大为,张广艳,郑纬民.大数据流式计算:关键技术及系统实例.软件学报,2014,(4):839−862. http://www.jos.org.cn/1000-9825/4558.html [doi: 10.13328/j.cnki.jos.004558]

[3] 国冰磊,于炯,杨德先,等.面向关系数据库查询的能耗建模及计划评价.计算机研究与发展,2019,56(4):810−82.

[4] 赵小刚,胡启平,丁玲,等.基于模型预测控制的数据中心节能调度算法.软件学报,2017,28(2):429−442. http://www.jos.org.cn/1000-9825/5026.html [doi: 10.13328/j.cnki.jos.005026]

[5] 杨挺,王萌,张亚健,等.云计算数据中心 HDFS 差异性存储节能优化算法.计算机学报,2019,42(4):47−61.

[10] 李梓杨,于炯,卞琛,等.基于流网络的 Flink 平台弹性资源调度策略.通信学报,2019(8):85−101.

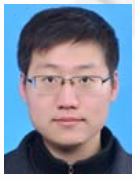
[11] 英昌甜,于炯,卞琛,等.基于 RDD 关键度的 Spark 检查点管理策略.计算机研究与发展,2017,54(12):2858−2872.

[18] 鲁亮,于炯,卞琛,等.大数据流式计算框架 Storm 的任务迁移策略.计算机研究与发展,2018,55(1):71−92.

[30] 廖彬,张陶,于炯,等.MapReduce 能耗建模及优化分析.计算机研究与发展,2016,53(9):2107−2131.

[36] 蒲勇霖,于炯,鲁亮,等.基于 Storm 平台的数据恢复节能策略.计算机研究与发展,2021,58(3):479−496.

[37] 蒲勇霖,于炯,鲁亮,等.Storm 平台下工作节点的内存电压调控节能策略.通信学报,2018,39(10):101−121.

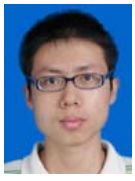


蒲勇霖(1991－),男,博士,CCF 学生会员,主要研究领域为流式计算,绿色计算.

李梓杨(1993－),男,博士,CCF 学生会员,主要研究领域为流式计算,内存计算.

于炯(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网格计算,绿色计算,分布式计算.

卞琛(1981－),男,博士,副教授,CCF 专业会员,主要研究领域为分布式计算,并行计算.



鲁亮(1990－),男,博士,讲师,CCF 专业会员,主要研究领域为分布式计算,机器学习,运筹与优化.

廖彬(1975－),男,博士,副教授,CCF 专业会员,主要研究领域为分布式计算,数据库理论与技术.