计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(2):250−254 [doi: 10.15888/j.cnki.csa.007763]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Storm 平台的多任务分组调度策略与实现①

王中华, 柴小丽

(中国电子科技集团公司第三十二研究所, 上海 201808)
通讯作者: 王中华, E-mail: wzh201434@163.com

摘　要: 随着大数据与人工智能技术的飞速发展, 高性能, 实时性的流式计算系统逐渐取代传统基于数据仓库的批量计算系统. Apache storm 作为一款开源, 高容错, 实时处理的分布式大数据流式计算平台, 支持任务平均分配策略, 单机任务指定策略等多种任务分配方案. 当任务拓扑结构中存在多个任务时, 且集群中只有某些机器支持某一任务执行时, 传统的任务调度方法只能实现将单一的任务分配给单一指定的机器, 使得整个集群的资源没有充分的利用. 通过调整任务调度策略, 获得满足条件的机器队列, 查看机器队列中可用工作节点, 将指定任务均匀分配给可用工作节点, 其他任务仍通过默认策略分配给集群中的剩余机器, 实现多任务的分组调度策略.

关键词: Apache storm 平台; 分布式; 流式计算; 拓扑; 多任务分组调度策略

引用格式: 王中华,柴小丽.基于 Storm 平台的多任务分组调度策略与实现.计算机系统应用,2021,30(2):250–254. http://www.c-s-a.org.cn/1003-3254/7763.html

Multi-Task Group Scheduling Strategy and Implementation Based on Storm Platform

WANG Zhong-Hua, CHAI Xiao-Li

(The 32nd Research Institute of China Electronics Technology Group Corporation, Shanghai 201808, China)

Abstract: As big data and artificial intelligence technologies are booming, high-performance, real-time streaming computing systems are gradually replacing traditional batch computing systems based on data warehouses. As an open-source distributed big-data streaming computing platform that is highly fault-tolerant and can realize real-time processing, Apache storm supports a variety of task distribution schemes such as average task distribution strategy and single-machine task assignment strategy. When there are multiple tasks in the task topology and only certain machines in the cluster support the execution of a certain task, the traditional task scheduling method can only allocate a single task to a single designated machine, failing to make best use of resources in the entire cluster. By the adjustment to the task scheduling strategy, the eligible machine queue is obtained. Then, the assigned tasks are evenly distributed to available work nodes in the machine queue, and other tasks are distributed to the remaining machines in the cluster through the default strategy. In this way, multi-task group scheduling strategy can be achieved.

Key words: Apache storm platform; distributed; stream computing; topology; multi-task group scheduling strategy

随着信息科学和云计算技术的飞速发展, 智能设备的持续普及, 存储设备性能的提升和网络带宽的增长为大数据的存储和流通提供了物质基础, 云计算技术通过将分散的数据集中在数据中心, 从而可以更为集中有效地处理和分析大数据信息. 云计算技术为海量数据存储和分散的用户访问提供了必要的空间和途径[1]. 大数据主要有 4 种计算模式, 分别是批量计算, 流式计算, 图计算和交互计算. 其中, 适用于大数据分析

① 收稿时间: 2020-06-09; 修改时间: 2020-07-10; 采用时间: 2020-07-14; csa 在线出版时间: 2021-01-27

的计算模式主要是批量计算和流式计算这两种, 并且由于批量计算和流式计算针对的数据流类型不同, 所适用的大数据应用场景也不一样[2].

批量计算会将先将数据信息统一收集起来, 然后把大量的数据信息存储到本地或云端地数据库中, 最后对数据进行批量的处理. 由此可见, 批处理适用的数据一般是静态数据, 即保存在本地或云端数据库中的信息, 任务可一次性完成. 因此, 批量计算一般应用在实时性要求不高, 离线计算的场景下, 进行数据分析或实现离线报表等[3–5].

数据流是一组有序的, 有起点和终点的字节的数据序列, 一般包含输入流和输出流, 在实时通信领域中, 数据的价值与时间成反比, 即处理数据的时间越短, 数据的价值越大. 因此, 必须对实时的数据信息给出毫秒级的响应. 流式计算就是应用在实时场景下, 或对时效性要求比较高的场景, 如实时推荐, 业务监控[6–9].

Hadoop 是一个由 Apache 基金会开发的分布式系统基础架构, 实现了一个分布式文件系统 HDFS, 提供高吞吐量来访问应用程序的数据, 适合有着超大数据集的应用程序, HDFS 为海量数据提供存储, MapReduce 为海量数据提供计算[10]. Apache 开发 Flink 开源流处理框架, 核心是用 Java 和 Scala 编写的分布式数据流引擎, Flink 通过支持数据并行和流水线方式, 可执行任意流数据程序[11], Flink 的流水线运行方式支持系统执行批处理和流处理程序, 同时支持迭代算法的执行. Twitter 开发了 Storm 框架提供分布式的, 高容错的实时计算系统, 支持多语言: Java, Python, Ruby 等[12], 可实现亚秒级的低延迟. Storm 提供较高的可靠性, 所有信息都可保证至少处理一次, 确保不会丢失信息.

由于在项目研发过程中, 需要 8 台机器对视频流执行解码任务, 另外 8 台执行目标识别的推理任务, 并且由于两组机器都缺乏硬件条件完成其他任务. 传统的平均分配和单机任务指定策略不能满足项目需求, 本文通过设计新调度算法来实现对两组机器的任务分配.

1 Storm 模型架构

如图 1 所示, Storm 集群采用主从式架构, 集群中的节点主要分为以下 4 类[13]:

主节点 (Master node): 通过运行 Storm nimbus 命令启动 Storm 的主节点, nimbus 是 Storm 系统的主控节点, 主要用于向集群中提交作业, 通过读取 ZooKeeper 的工作信息分配集群的任务, 以及监控整个集群状态 (有进程级的也有线程级别的).

工作节点 (Worker node): 通过运行 Storm supervisor 命令启动 Storm 的工作节点.通过设定的端口与 ZooKeeper 进行信息交互, 读取主控节点分配的任务信息, 下载作业副本, 管理属于自己的 Worker 进程, 如启动, 暂停或撤销任务的工作进程及其线程. 一个工作进程中可运行多个线程, 每个线程中又可运行多个任务 (task).

控制台节点 (Web console node): 通过运行 Storm ui 命令启动用户界面服务节点, 默认的服务端口号为 8080, 可在 storm.yaml 中设定 ui.port 进行修改. 可以在 ui 界面上查看已提交的作业状态, 包括集群的整体状态, 已使用的节点数, 每个节点的运行情况和作业的执行状态, 支持手动停止正在执行的作业.

协调节点: 通过运行 ZooKeeper server start 命令启动 ZooKeeper 进程的节点, 实现 numbus 和 supervisor 之间的协调管理, 包含分布式状态维护和分布式配置管理等.

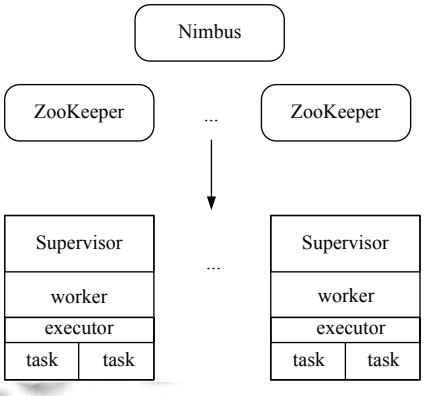


图 1 Storm 模型架构

2 ZooKeeper

Storm 使用 ZooKeeper 来保证集群的一致性. Storm 的所有的状态信息都是保存在 ZooKeeper 里面, nimbus 通过在 ZooKeeper 上面写状态信息来分配任务[14].

Supervisor, task 通过从 ZooKeeper 中读状态来领取任务, 同时 supervisor, task 也会定期发送心跳信息到 ZooKeeper, 使得 nimbus 可以监控整个 Storm 集群的状态, 从而可以重启一些停止的 task.

在每台机器上设置 myid 文件来分配机器在集群中的 id, 如图 2 所示, 51 表示式第几台服务器, 10.0.0.1 表示服务器的 IP 地址, 2888:3888 表示服务器中与集群中 leader 交换信息的端口.

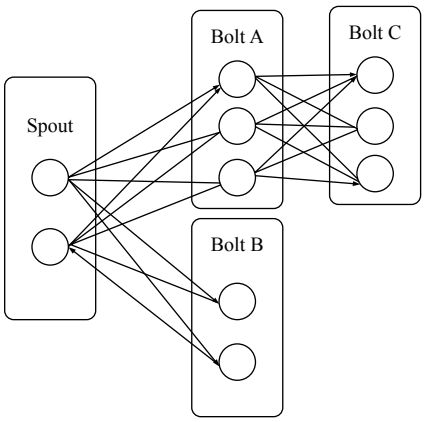


图 2 拓扑结构

3 Topology 结构

Topology 是 Storm 中运行的一个实时应用程序, 因为各个组件间的消息流动形成逻辑上的一个拓扑结构[15], 如图 2 所示. 主要由以下几部分组成:

Spout: 在一个 topology 中产生源数据流的组件. 通常情况下 spout 会从外部数据源中读取数据, 然后转换为 topology 内部的源数据. Spout 是一个主动的角色, 其接口中有个 nextTuple() 函数, Storm 框架会不停地调用此函数, 源源不断地发送数据. Spout 另一个重要的方法时 ack 和 fail, Storm 监控到 tuple 从 spout 发送到 toplogy 成功完成或失败时调用 ack 和 fail, 保证数据的可靠性.

Bolt: 在一个 topology 中接受数据然后执行处理的组件. Bolt 可以执行过滤, 函数操作, 合并, 写数据库等任何操作. Bolt 是一个被动的角色, 其接口中有个 execute (Tuple input) 函数, 在接受到消息后会调用此函数, 用户可以在其中执行自己想要的操作[16].

Tuple: Storm spout, bolt 组件消息传递的基本单元 (数据模型), Tuple 是包含名称的列表, Storm 支持所有原生类型, 字节数组为 Tuple 字段传递, 如果要传递自定义对象, 需要实现接口 serializer[17].

Stream: 源源不断传递的 Tuple 就组成了 stream.

4 任务调度

4.1 传统任务调度

Storm 集群默认的调度器是 EventScheduler[18–21], 采用轮询策略来搜索集群中所有拓扑结构的工作节点, 将资源较为均匀的分配给任务进程. 具体分配流程如下:

先由 nimbus 来计算拓扑的工作量, 及计算多少个 task, task 的数目是指 spout 和 bolt 的并发度的分别的和 nimbus 会把计算好的工作分配给 supervisor 去做, 工作分配的单位是 task, 即把计算好的一堆 task 分配给 supervisor 去做, 即将 task-id 映射到 supervisor-id+port 上去, 具体分配算法如算法 1.

算法 1. 传统任务调度算法

1) 从 ZooKeeper 上获得已有的 assignment(新提交的 topology 为空).
2) 查找所有可用的 slot, slot 就是可用的 worker, 在所有 supervisor 上配置的多个 worker 的端口.
3) 将任务均匀地分配给可用的 worker, supervisor 会根据 nimbus 分配给他的任务信息来让自己的 worker 做具体的工作, worker 会到 ZooKeeper 上去查找给他分配了哪些 task, 并且根据这些 task-id 来找到相应的 spout/bolt, 它还需要计算出这些 spout/bolt 会给哪些 task 发送消息, 然后建立与这些 task 的连接, 然后在需要发消息的时候就可以给相应的 task 发消息.

4.2 多任务分组调度策略

而在当前项目应用过程中, 需要将视频解码和目标识别任务分别运行在两组机器上, 并且由于任务的硬件需求, 解码的任务不能在处理目标识别的机器上运行.

为了实现多任务分组调度, 实现了算法 2.

算法 2. 多任务分组调度算法

1) 从 ZooKeeper 上获得已有的 assignment(新提交的 topology 为空).
2) 在配置集群时为每台机器设置 supervisor 名称, 如下图所示, 通过 supervisor.scheduler.meta 设置节点名称.
3) 采用循环的方式, 通过判断 meta. get(“name”)==”supervisor51”或 meta.get(“name”)==”supervisor52”等得到匹配的 supervisor 列表.
4) 通过 componentToExecutors.get("decode") 获得解码任务的线程数.
5) 通过 getAvailableSlots 函数提取上述指定 supervisor 的所有可用节点.
6) 构建 map<WorkSlot, List<ExecutorDetails>>将可用节点与线程情况相匹配.
7) 通过 cluster.assign 函数将匹配情况提交给集群, 集群将按照对应关系分配线程, 其余任务采用平均分配, 由于已经将解码机器组的可用节点全部占满, 剩余的推理任务将自动均匀地分配到推理机器组.

5 实验环境与结果

5.1 实验环境

实验共使用 16 台服务器, 且均使用 Linux 系统, 其中 8 台服务器作为视频解码组, 搭载 Arm v8 多核处理器和中科睿芯解码卡; 另外 8 台服务器作为推理组, 搭载 Arm v8 多核处理器和寒武纪 MUL100 加速卡.

Storm 集群由 1 个 nimbus 和 16 个 supervisor 节点组成 (为了实现资源充分使用, 其中一个服务器既作为

nimbus 用来分发任务和监控集群状态, 也用来处理任务).

拓扑结构:

Spout: 用于读取视频文件作为输入, 组件命名为 filename-reader, 共 8 个线程;

Bolt1: 用于实现视频解码任务, 组件命名为 decode, 共 23 个线程;

Bolt2: 用于实现推理任务 (目标识别), 组件命名为 inference, 共 32 个线程.

5.2 实验结果

(1) 任务分配情况

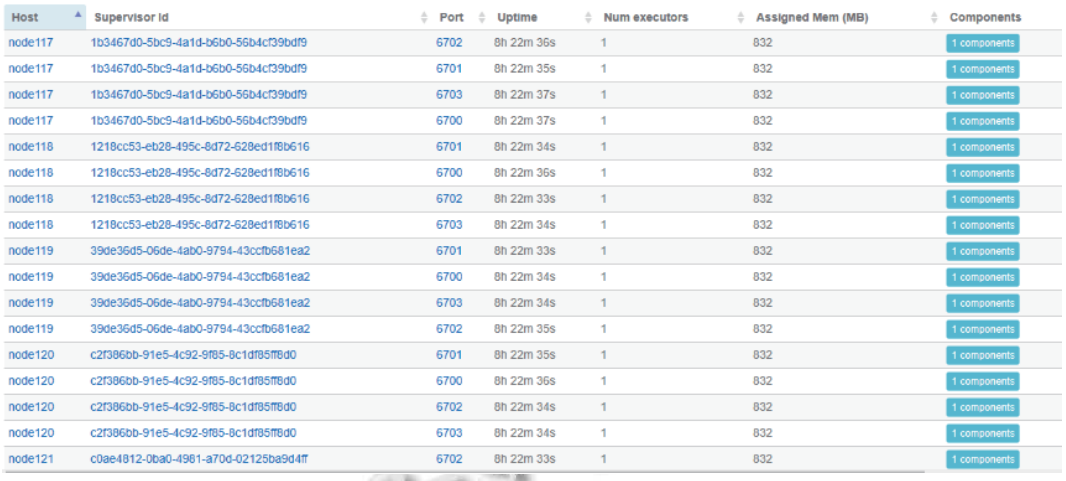
由表 1 可知, 视频解码组同时执行文件读取和视频解码任务, 推理组全部执行推理任务 (目标识别), 具体 ui 结果如图 3 所示. 其中, 在视频解码组中预留一个节点, 用于预防解码任务时可能出现的节点阻塞.

(2) 作业执行情况

由图 4 知整体拓扑的数据处理情况, 10 分钟时处理了 48 685 条数据, 3 小时处理了 872 481 条数据. 整个拓扑运行 24 小时并未发生中断, 可见调度算法的稳定性.

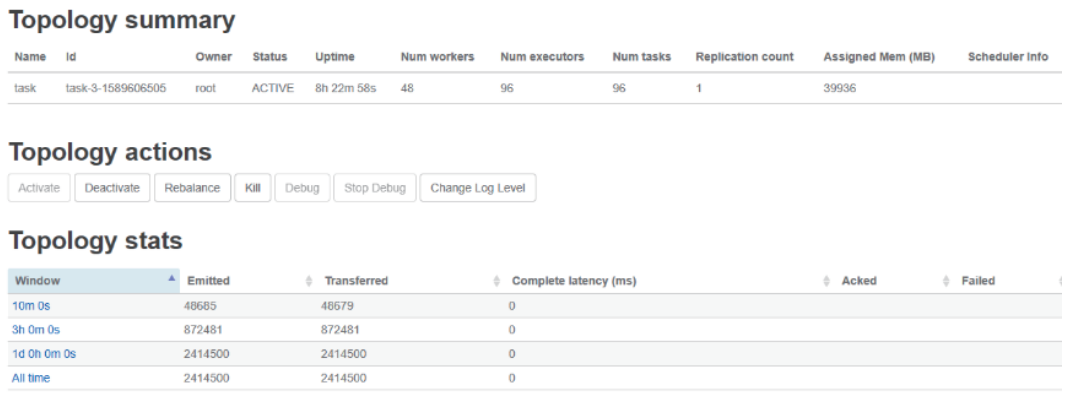
表 1 集群任务分配情况

机器名称	可用节点	已用节点	组件名称
node50	4	4	filename-reader decode
node51	4	4	fielname-reader decode
node52	4	4	fielname-reader decode
node53	4	4	decode
node54	4	4	decode
node55	4	4	decode
node56	4	4	decode
node57	4	3	decode
node117	4	4	inference
node118	4	4	inference
node119	4	4	inference
node120	4	4	inference
node121	4	4	inference
node122	4	4	inference
node123	4	4	inference
node124	4	4	inference



Host	Supervisor Id	Port	Uptime	Num executors	Assigned Mem (MB)	Components
node117	1b3467d0-5bc9-4a1d-b6b0-56b4cf39bdf9	6702	8h 22m 36s	1	832	1 components
node117	1b3467d0-5bc9-4a1d-b6b0-56b4cf39bdf9	6701	8h 22m 35s	1	832	1 components
node117	1b3467d0-5bc9-4a1d-b6b0-56b4cf39bdf9	6703	8h 22m 37s	1	832	1 components
node117	1b3467d0-5bc9-4a1d-b6b0-56b4cf39bdf9	6700	8h 22m 37s	1	832	1 components
node118	1218cc53-eb28-495c-8d72-628ed1f8b616	6701	8h 22m 34s	1	832	1 components
node118	1218cc53-eb28-495c-8d72-628ed1f8b616	6700	8h 22m 36s	1	832	1 components
node118	1218cc53-eb28-495c-8d72-628ed1f8b616	6702	8h 22m 33s	1	832	1 components
node118	1218cc53-eb28-495c-8d72-628ed1f8b616	6703	8h 22m 34s	1	832	1 components
node119	39de36d5-06de-4ab0-9794-43ccfb681ea2	6701	8h 22m 33s	1	832	1 components
node119	39de36d5-06de-4ab0-9794-43ccfb681ea2	6700	8h 22m 34s	1	832	1 components
node119	39de36d5-06de-4ab0-9794-43ccfb681ea2	6703	8h 22m 34s	1	832	1 components
node119	39de36d5-06de-4ab0-9794-43ccfb681ea2	6702	8h 22m 35s	1	832	1 components
node120	c2f386bb-91e5-4c92-9f85-8c1df85ff8d0	6701	8h 22m 35s	1	832	1 components
node120	c2f386bb-91e5-4c92-9f85-8c1df85ff8d0	6700	8h 22m 36s	1	832	1 components
node120	c2f386bb-91e5-4c92-9f85-8c1df85ff8d0	6702	8h 22m 34s	1	832	1 components
node120	c2f386bb-91e5-4c92-9f85-8c1df85ff8d0	6703	8h 22m 34s	1	832	1 components
node121	c0ae4812-0ba0-4961-a70d-02125ba9d4ff	6702	8h 22m 33s	1	832	1 components

图 3 多任务分组调度结果



Topology summary

Name	Id	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)	Scheduler Info
task	task-3-1589606505	root	ACTIVE	8h 22m 58s	48	96	96	1	39936	

Topology actions

Activate | Deactivate | Rebalance | Kill | Debug | Stop Debug | Change Log Level

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	48685	48679	0		
3h 0m 0s	872481	872481	0		
1d 0h 0m 0s	2414500	2414500	0		
All time	2414500	2414500	0		

图 4 作业执行情况

6 结束语

本文研究了 Storm 环境下, 多任务在两组机器上分别运行并存在信息交互的情况, 提出了多任务分组调度策略, 该机制可以将存在不同需求的两个任务分别分配到对应的机器群组中, 以达到运行和资源分配最优情况. 实验证明, 该调度机制可实现视频解码和推理任务的分组运行, 并且通过持续运行拓扑 24 小时, 验证了该调度机制的稳定性. 后续的工作将继续完善该调度机制, 当存在 3 个或更多任务需要指定机器群组资源运行时, 能够实现多个指定任务分配到指定机器群组中, 以实现资源的正确分配与最优分配.

参考文献

1 蔡宇, 赵国锋, 郭航. 实时流处理系统 Storm 的调度优化综述. 计算机应用研究, 2018, 35(9): 2567–2573. [doi: 10.3969/j.issn.1001-3695.2018.09.002]

2 Kulkarni S, Bhagat N, Fu MS, *et al*. Twitter heron: Stream processing at scale. Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. Melbourne, Australia. 2015. 239–250.

3 张楠, 柴小丽, 谢彬, 等. Storm 流处理平台中负载均衡机制的实现. 计算机与现代化, 2017, (12): 65–70, 76. [doi: 10.3969/j.issn.1006-2475.2017.12.013]

4 鲁亮, 于炯, 卞琛, 等. Storm 环境下基于权重的任务调度算法. 计算机应用, 2018, 38(3): 699–706. [doi: 10.11772/j.issn.1001-9081.2017082125]

5 孙怀英, 虞慧群, 范贵生, 等. 大数据流计算环境下的低延时高可靠性的资源调度方法. 华东理工大学学报 (自然科学版), 2017, 43(6): 855–862.

6 张译天, 于炯, 鲁亮, 等. 大数据流式计算框架 Heron 环境下的流分类任务调度策略. 计算机应用, 2019, 39(4): 1106–1116. [doi: 10.11772/j.issn.1001-9081.2018081848]

7 Hsu CH, Slagter KD, Chen SC, *et al*. Optimizing energy consumption with task consolidation in clouds. Information Sciences, 2014, 258: 452–462. [doi: 10.1016/j.ins.2012.10.041]

8 Patan R, Babu MR. Re-Storm: Real-time energy efficient data analysis adapting Storm platform. Jurnal Teknologi, 2016, 78(10): 139–146.

9 Assunção MD, Calheiros RN, Bianchi S, *et al*. Big data computing and clouds: Trends and future directions. Journal of Parallel and Distributed Computing, 2013, 79–80: 3–15.

10 张维维, 魏海涛, 于俊清, 等. COStream: 一种面向数据流的编程语言和编译器实现. 计算机学报, 2013, 36(10): 1993–2006.

11 Andonov R, Poirriez V, Rajopadhye S. Unbounded knapsack problem: Dynamic programming revisited. European Journal of Operational Research, 2000, 123(2): 394–407. [doi: 10.1016/S0377-2217(99)00265-9]

12 马可, 李玲娟. 分布式实时流数据聚类算法及其基于 Storm 的实现. 南京邮电大学学报 (自然科学版), 2016, 36(2): 104–110.

13 廖彬, 于炯, 张陶, 等. 基于分布式文件系统 HDFS 的节能算法. 计算机学报, 2013, 36(5): 1047–1064.

14 Aniello L, Baldoni R, Querzoni L. Adaptive online scheduling in storm. Proceedings of the 7th ACM International Conference on Distributed Event-based Systems. New York, NY, USA. 2013. 207–218.

15 梁毅, 侯颖, 陈诚, 等. 面向大数据流式计算的任务管理技术综述. 计算机工程与科学, 2017, 39(2): 215–226. [doi: 10.3969/j.issn.1007-130X.2017.02.001]

16 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, 25(4): 839–862. [doi: 10.13328/j.cnki.jos.004558]

17 刘月超, 于炯, 鲁亮. Storm 环境下一种改进的任务调度策略. 新疆大学学报 (自然科学版), 2017, 34(1): 90–95.

18 熊安萍, 王贤稳, 邹洋. 基于 Storm 拓扑结构热边的调度算法. 计算机工程, 2017, 43(1): 37–42. [doi: 10.3969/j.issn.1000-3428.2017.01.007]

19 Mohamed SH, El-Gorashi TEH, Elmirghani JMH. On the energy efficiency of MapReduce shuffling operations in data centers. 2017 19th International Conference on Transparent Optical Networks. Girona, Spain. 2017. 1–5.

20 Peng BY, Hosseini M, Hong ZH, *et al*. R-Storm: Resource-aware scheduling in storm. Proceedings of the 16th Annual Middleware Conference. New York, NY, USA. 2015. 149–161.

21 杨秋吉, 于俊清, 莫斌生, 等. 面向 Storm 的数据流编程模型与编译优化方法研究. 计算机工程与科学, 2016, 38(12): 2409–2418. [doi: 10.3969/j.issn.1007-130X.2016.12.005]