计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(4):76−83 [doi: 10.15888/j.cnki.csa.007345]

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

面向股票交易分析场景的流式大数据系统测试框架①

史凌云[1], 郑莹莹[2,3], 谭　励[1], 许利杰[2,4], 王　伟[2,3,4], 魏　峻[2,3,4]

[1](北京工商大学 计算机与信息工程学院, 北京 100048)
[2](中国科学院 软件研究所, 北京 100190)
[3](中国科学院大学, 北京 100049)
[4](计算机科学国家重点实验室, 北京 100190)
通讯作者: 谭　励, E-mail: tanli@th.btbu.edu.cn

摘　要: 分布式集群环境使得数据实时计算更为复杂, 流式大数据处理系统的正确性难以保障. 现有的大数据基准测试框架可以测试流式大数据处理系统的性能表现, 但是普遍存在应用场景设计简单、评价指标不充分等不足. 针对这一挑战, 本文构造了一个面向股票交易场景的流式大数据基准测试框架, 通过生成股票高频交易数据, 测试系统在高流速场景下的延迟、吞吐量、GC 时间、CPU 资源等的性能表现. 同时, 通过横向测试验证流式大数据系统的扩展性. 本文以 Apache Spark Streaming 为待测系统进行测试, 实验结果表明, 高流速场景下出现延迟增加、GC 时间提高等性能下降问题, 原因是系统输入速率的提高及并行度的增加.

关键词: 流式大数据处理; 系统性能; 基准测试; Apache Spark Streaming

引用格式: 史凌云,郑莹莹,谭励,许利杰,王伟,魏峻.面向股票交易分析场景的流式大数据系统测试框架.计算机系统应用,2020,29(4):76–83. http://www.c-s-a.org.cn/1003-3254/7345.html

System Test Framework of Stream Data for Stock Trading Analysis Scenario

SHI Ling-Yun[1], ZHENG Ying-Ying[2,3], TAN Li[1], XU Li-Jie[2,4], WANG Wei[2,3,4], WEI Jun[2,3,4]

[1](School of Computer and Information Engineering, Beijing Technology and Business University, Beijing 100048, China)
[2](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[3](University of Chinese Academy of Sciences, Beijing 100049, China)
[4](State Key Laboratory of Computer Science, Beijing 100190, China)

Abstract: Distributed cluster environment makes real-time data computation more complex, and the correctness of stream large data processing system is difficult to guarantee. The existing large data benchmarking framework can test the performance of stream large data processing system, but there are many shortcomings such as simple application scenario design and inadequate evaluation index. To address this challenge, this study constructs a stream large data benchmarking framework for stock trading scenarios, generates high-frequency stock trading data through a flow-based data generator, and tests the performance of the system in high-speed scenarios in terms of delay, throughput, GC time, CPU resources, and so on. At the same time, the scalability of large data streaming system is verified by horizontal test. In this study, Apache Spark Streaming is used as the test system to test. The experimental results show that the performance degradation problems such as delay increase and GC time increase occur in high-speed scenarios because of the increase of input rate and parallelism of the system.

Key words: stream data processing; system performance; benchmark; Apache Spark Streaming

① 基金项目: 北京市自然科学基金 (4172013); 北京市自然科学基金-海淀原始创新联合基金 (L182007); 国家自然科学基金 (61802377, 61702020) 及其配套项目 (PXM2018_014213_000033); 国家重点研发计划 (2016YFD0401104)
Foundation item: Natural Science Foundation of Beijing Municipality (4172013); Joint Fund of Natural Science Foundation of Beijing Municipality and Haidian Original Innovation Fund (L182007); National Natural Science Foundation of China (61802377, 61702020) and Its Supporting Projects (PXM2018_014213_000033); National Key Research and Development Program of China (2016YFD0401104)
收稿时间: 2019-09-04; 修改时间: 2019-09-23; 采用时间: 2019-10-11; csa 在线出版时间: 2020-04-05

引言

随着信息时代的到来, 互联网、物联网、云计算等技术的飞速发展和广泛应用, 数据在各个行业不断地产生、积累并爆发式增长, 已经成为一种重要的生产因素, 并渗透到每一个行业和业务职能领域[1]. 大数据被喻为“未来的新石油”[2], 已经成为社会各界关注的热点, 甚至成为各界争夺的焦点, 大数据时代已经到来. 相较于传统的数据, 大数据具有数据规模大、数据类型多、数据处理速度快、数据价值密度低等特征, 这些特征对大数据处理和应用提出了更高的要求和更大的挑战.

目前对大数据处理的形式主要包括批式处理和流式处理[3]. 其中, 批式处理是指对静态有界数据的处理, 对计算的实时性要求不高且应用场景广泛, 具有代表性的批量大数据处理系统有 Apache Hadoop[4]和 Apache Spark[5]等. 但是, 随着社交网络、电子商务等技术的飞速发展和应用, 越来越多的应用场景要求从海量的数据中及时获取价值, 并以很低的延迟来分析实时数据. 例如, 以阿里巴巴为代表的电商平台基于流式大数据处理系统实时统计和分析用户行为, 更新商品搜索引擎. 因此, 针对流式大数据的实时处理越来越流行, 应用场景也越来越重要. 流式大数据处理系统的地位日渐凸显, 在业界也已经有了非常广泛的应用, 常见的有 Apache Strom[6]、Apache Flink[7]、Apache Spark Streaming[8]等.

流式数据本身的实时性、难重复以及动态变化等特性, 以及流式数据计算所需的数据无限性、计算有界性、计算实时性等特征, 对流式大数据处理系统的性能和可靠性提出了更高的要求. 流式大数据处理系统需要提供低延迟的数据处理, 同时保证计算的正确性. 然而, 随着集群规模的扩大, 系统发生故障的概率也会增大, 无法预知的错误可能会随时出现在任意一个节点. 一旦大数据处理系统出现问题, 可能会产生不可挽回的损失. 因此, 针对流式大数据处理系统及其基准测试框架的研究已经成为了一个热点问题.

现今已有多种流式大数据基准测试框架, 如 Yahoo! Streaming Benchmark[9]和 HiBench[10]等. 但 Yahoo! Streaming Benchmark 应用场景单一, 覆盖程度较低; HiBench 仅能够支持简单流式数据, 本质仍是一个批式大数据基准测试框架. 因此, 现有流式大数据处理系统基准测试框架仍存在不足, 比如应用场景的设计较为简单, 评价指标选取上较为单一, 集中在吞吐量和延迟等.

针对现有流式大数据处理系统基准测试框架的不足和面临的挑战, 本文设计并实现了一个股票交易场景下的流式大数据处理系统基准测试框架. 该框架包括流式数据生成方法、应用场景构造和评价指标 3 个部分, 以 Socket 作为流式数据源, 选取真实的股票交易数据构造了三个数据流, 具体应用覆盖 GroupBy 操作和 Join 操作, 并选取延迟、吞吐量、GC 时间和 CPU 利用率作为评价指标, 构建了一个实时计算与结构化数据相结合的场景. 此外, 本文还针对数据输入速率和执行器内核数量设计了两个实验, 在 Apache Spark Streaming 中对该框架进行实际的集群测试, 对测试结果进行分析并得出结论, 分析系统性能表现, 同时发现大数据处理系统存在的问题, 并分析瓶颈所在, 从而尽大可能地减少实际运行过程中可能出现的故障. 本文的主要贡献如下:

(1) 总结了现有流式大数据处理系统基准测试框架特点及其不足.

(2) 提出了一种流式数据生成方法, 并使用多种测试指标进行结果评测.

(3) 设计并实现了一个基于股票交易场景的流式大数据处理系统基准测试框架.

(4) 应用流式大数据处理系统基准测试框架对 Apache Spark Streaming 进行性能测试, 发现并分析系统的不足.

1 相关工作

不同于批式大数据处理, 流式大数据处理起步较晚, 目前在业内并没有统一的基准测试标准. 现有的流式大数据处理系统基准测试框架的相关研究如下.

Hesse 和 Lorenz[11]从体系结构等方面对比了 Apache Storm、Flink、Spark Streaming 和 Samza platforms. Gradvohl 等人[12]从系统容错方面分析对比了 Google Millwheel[13]、Yahoo Apache S4[14]、Spark Streaming 和 Storm. 然而, 这两篇文献仅限于概念性讨论, 没有实验性的定量性能评估. Nabi 等人[15]提出了一个基准测试, 测量 Apache Spark 和 Apache Storm 的延迟和吞吐量, 为流处理平台的实验比较创造了第一步. Dayarathna 和 Suzumura[16]使用基准测试比较了 3 个流

处理系统的吞吐量、CPU 和内存消耗以及网络使用情况.

此外, 部分流式大数据处理系统的开发厂商选择了自己认为有代表性、能验证系统功能性的应用场景进行测试. 如 Yahoo!开发的分布式流计算平台 S4 选取了广告点击率计算 (Click-Through Rate) 进行性能验证, 以测试 S4 处理流式数据的极限[2]. Apache Storm 选取一个简单的应用, 统计了不同应用下参与的用户数, 并测试其在故障下的表现[17]. Apache Spark Streaming 仅对 Grep、WordCount、TopKCount 这 3 个常见应用的吞吐量和故障恢复能力进行测试. 应用场景简单以及流式计算特征的覆盖率低使得这些测试框架无法全面的剖析流式大数据处理系统所面临的性能及可靠性问题.

现有的针对多种流式大数据处理系统的基准测试框架, 其测试系统大多以 Apache Storm、Apache Spark 和 Apace Flink 为主. 例如 Lopez 等人[18]提出了一个针对 Apache Storm、Apache Spark 和 Apace Flink 的基准测试框架, 测试了 3 个系统在节点故障情况下的吞吐量. Karimov 等人[19]提出了一个分布式流处理引擎基准测试框架, 对 Apache Storm、Apache Spark 和 Apache Flink 的性能进行评估, 并定义和测试流式大数据处理系统的可持续性能. 由 Yahoo!的一个团队设计并实现的 Yahoo! Streaming Benchmark 通过 Kafka 和 Redis 进行数据检索和存储, 对 Apache Storm、Apache Spark 和 Apace Flink 进行实验, 测量了延迟和吞吐量[9]. Perera 等人使用 Yahoo! Streaming Benchmark 和 Karamel[20]在云环境中提供 Apache Spark 和 Apache Flink 的可复制批处理和流基准[21]. 但 Yahoo! Streaming Benchmark 在应用场景上覆盖度较低, 且只支持一个工作负载.

综上所述, 现有的流式大数据处理系统基准测试框架还存在各种不足, 应用用场景设计较为简单, 评价指标选取上较为单一, 集中在吞吐量和延迟. 针对现有的流式大数据处理系统基准测试框架存在的不足, 本文构造了股票高频交易场景, 并选择延迟、吞吐量、GC 时间和 CPU 利用率作为评价指标.

2 流式大数据系统基准测试框架设计与实现

本文基于流式大数据及其特征, 设计并实现了一个流式大数据处理系统基准测试框架, 包括流式数据生成方法、应用场景构造和评价指标.

通过流式数据生成方法, 生成符合流式特征的股票交易数据; 通过应用场景构造, 构建一个股票交易场景用于系统测试; 通过明确测试评价指标, 收集并分析指标数据, 从而分析系统性能并得出测试结论. 基准测试系统架构图如图 1 所示.

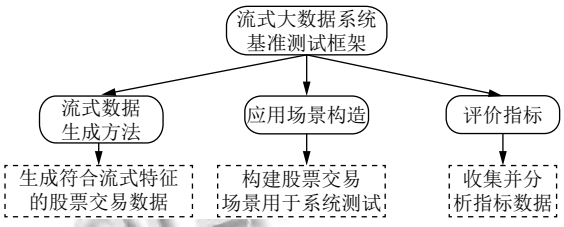

图 1 基准测试系统架构图

2.1 流式数据生成方法

一般的批式大数据可以预先产生和存储, 使用方便. 而由于流式计算的有界性和实时性等特点, 数据生成需提供实时、高速的流式数据, 从而通过测试发现系统的瓶颈, 保证基准测试的有效性, 这些都对流式大数据的生成方式提出了更为严格的要求.

Apache Spark Streaming 提供了两种数据源, 基础数据源和高级数据源. 基础数据源是 Streaming API 中直接提供的数据源, 如 socket 套接字、文件系统等. 高级数据源是通过第三方类提供支持, 如 Kafka、Flume、Kinesis、Twitter 等. 由于使用第三方工具生成数据本身会对性能有所影响[22], 因此本文选用基础的 Socket 传输方式作为 Apache Spark Streaming 的数据源. Socket 传输方式可通过程序控制向指定端口发送数据, 并可以通过调节参数改变数据的传输速度, 以生成不同流速的流式大数据.

针对股票高频交易这一应用场景, 本文选择了现实的股票交易数据作为数据源, 主要涉及的数据类型为数值型和字符型, 方便进行流式 SQL 的计算.

2.2 应用场景构造

流式计算的应用场景有很多, 其中比较典型的是在金融银行业的应用. 在金融银行领域的日常运营过程中往往会产生大量的实时数据, 需要对这些海量数据进行实时分析处理以获得其内在价值, 从而帮助金融银行进行分析决策[23]. 其中本文选取的股票的高频交易就是流式处理系统在金融银行业的一个应用.

(1) 数据流构造

实验构建了一个股票交易数据分析的场景, 在满

足实际生产生活要求的同时, 尽可能多地覆盖流式计算特征. 股票交易数据分析场景涉及 3 个数据流: 股票变化流、用户交易流和用户持仓流, 如表 1 所示.

表 1 数据流设计

名称	属性
STOCK	szcode、eventTime、lastPrice
TRANSACTION	szcode、userID、eventTime、Turnover、Price
POSITION	userID、szcode、lastPrice、openInterest

STOCK 是股票变化流, 用于描述不同时间点股票的价格情况. 其中, szcode 代表股票编号, eventTime 代表当前时间, lastPrice 代表当前价格.

TRANSACTION 是股票交易流, 用于描述不同时间点股票交易情况. 其中, szcode 代表交易的股票编号, userID 代表用户编号, eventTime 代表交易时间, Turnover 代表成交量, Price 代表成交价格.

POSITION 是用户持仓流, 用于描述不同时间点用户股票持仓情况. 其中, userID 代表用户编号, szcode 代表该用户持有的股票编号, lastPrice 代表上次成交价格, openInterest 代表持仓量.

(2) 具体应用构造

本文主要实现了对 GroupBy 和 Join 应用的覆盖, 设计如下:

1) GroupBy

GroupBy 是 Apache Spark 中基本、常见的 API, 相当于 SQL 查询中的 groupby() 函数. 实验中实时获取用户交易流的数据, 并按照股票编号这一字段进行聚合, 计算每只股票在一段时间内的总成交额.

```
# SQL Query (GroupBy)
#实时计算 n 分钟内每只股票的成交额.
SELECT szcode, SUM(Price*Turnover)
FROM TRANSACTION [Range n, Slide s]
GROUP BY szcode
```

2) Join

实验中对股票变化流和用户持仓流进行 Join 操作, 按照股票编号进行连接, 实现对各个用户持仓股票市值的实时计算.

```
# SQL Query (Join)
#每个用户所持股票的市值 (用户持有量*当前价格)
SELECT c.userID, SUM(lastPrice* openInterest)
FROM POSITION[Range n, Slide s] as p,
STOCK[Range n, Slide s] as s,
ON p.szcode = s.szcode
GROUP BY p.userID
```

Join 可以建立不同数据流之间的连接, 是大数据计算中的高级特性, 复杂且代价大, 但大多数场景都需要进行复杂的 Join 操作.

Spark Streaming 会将逐条采集的数据按照事先设置好的批处理间隔汇总成一批数据进行处理, Join 操作是在每一个批数据上进行的, 因此可通过对批处理间隔的合理设置避免 Join 操作造成的运算复杂度较高.

2.3 评价指标

针对流式大数据处理系统基准测试中评价指标单一的问题, 本文通过延迟、吞吐量、GC 时间等个方面对流式大数据处理系统进行评价.

延迟 (Latency) 是流式大数据处理中的一项常见且重要的指标, 表示在处理过程中由于网络或者计算产生的时间差. 一般可以将延迟分为系统延迟和事件延迟[11]. 本文将延迟定义为数据从源段到输出端所经历的非计算时间.

吞吐量 (Throughput) 是系统在单位时间内的数据处理量. 本文通过计算单位时间内数据源端输出的数据总量作为系统的吞吐量.

GC 时间 (GC time) 是系统执行过程中垃圾回收机制的执行时间. 垃圾回收即遍历应用程序在 Heap 上动态分配的所有对象, 识别那些已经死亡即不再被引用的对象, 将该对象占用的内存空间回收. 垃圾回收的开销是流式大数据处理系统内存管理需要考虑的因素之一, 本文通过 GC 执行时间衡量.

CPU 资源 (CPU resources) 即系统运行时的 CPU 使用率.

Apache Spark Streaming 提供了 Web UI 界面, 在任务执行过程中, 可以实时查询任务运行情况, 便于测试指标的查看和收集. 本文借助 Apache Spark Streaming 提供的 API 接口, 实时收集运行时的延迟、吞吐量、GC 时间等评价指标数据, 并以此进行实验分析.

3 实验验证

本节对基于 Apache Spark Streaming 实现的股票交易场景下的应用进行了以下两组测试: (1) 测试数据输入速率对系统性能的影响; (2) 测试执行器内核数量

对系统性能及扩展性的影响. 通过对测试结果的分析, 总结了系统的在不同的测试参数下的性能表现.

3.1 实验环境

为了模拟流式大数据处理系统在现实中的应用, 提高大数据处理的效率, 实验搭建了集群, 部署了分布式计算环境用于 Apache Spark Streaming 测试. 测试集群由五台机器组成, 包括 1 台 Master 节点和 4 台 Slave 节点, 共 20 cores, 集群架构如图 2 所示, 各个节点的配置信息如表 2 所示.

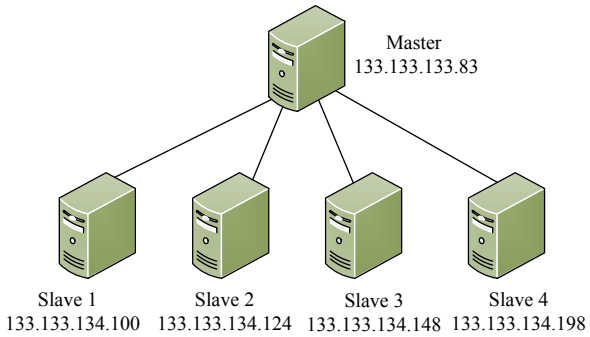

图 2 集群架构图

表 2 测试集群配置

参数	Master 节点配置	Slave 节点配置
处理器	4 Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83 GHz	4 Intel(R) Core(TM) i5-3470 CPU @ 3.20 GHz
内存	64 GB	64 GB
操作系统	CentOS(Linux)	Ubuntu(Linux)
Cores	4	4
Java	1.8.0	1.8.0
Scale	2.11.8	2.12.8
Hadoop	2.7.4	2.7.4
Spark	2.4.0	2.4.0

3.2 实验设计

实验采用控制变量法, 每次改变单一变量测试系统在不同情况下的性能表现, 每组进行五次测试, 记录平均值. 本文针对 Apache Spark Streaming 设计了两个实验, 从数据输入速率和任务并发度两个方面考虑.

(1) 通过控制 socket 数据输入端的线程休眠时间, 测试系统输入端的速率、执行时间、任务数、延迟和 GC 时间.

(2) 通过控制每个执行器的内核数, 改变任务并发度, 测试系统的扩展性, 以及吞吐量、延迟和 GC 时间的影响.

3.3 实验结果及结论

(1) 测试一: 数据输入速率对系统性能的影响

实验通过 Thread.sleep();函数控制 socket 端向指定端口发送数据的速率, 将线程的休眠时间分别设为 500 毫秒, 100 ms, 50 ms, 10 ms, 1 ms 以及 0, 即每间隔 500 ms, 100 ms, 50 ms, 10 ms, 1 ms 以及无间隔地发送数据. 同时, 将数据的生成时间固定为 5 min, 实现对输入端数据速率的控制. 测试一的实验结果如表 3 所示.

表 3 测试一实验结果

休眠时间 (ms)	输入端速率 (records/s)	执行时间 (min)	任务数	延迟 (ms)	GC 时间 (s)
500	3.97	8.0	6877	790	4
100	19.30	8.5	7563	821	6
50	39.26	8.5	7542	826	6
10	192.50	8.5	7565	823	6
1	1701.06	8.6	7611	855	7
0	46 071.31	9.6	7556	1060	9

通过实验数据分析, 可得结论如下:

1) 发现一: 当数据输入速率较高时, 系统延迟呈现较大的增长.

实验记录了不同速率下的系统总延迟, 如图 3 所示. 可以发现随着速率的增加, 系统延迟总体呈上升趋势的, 这是符合逻辑的. 但是在输入速率较低、相差不大的情况下, 系统延迟增加缓慢, 控制在一定范围之内, 而当输入速率较高时, 系统延迟会呈现一个较大的增长.

2) 发现二: 输入速率的提高会使系统资源利用率增加.

随着输入速率的提高, 实验从 GC 时间和 CPU 资源两个方面分析了系统资源利用率的变化.

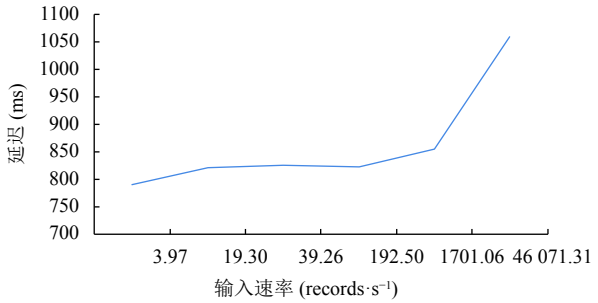

图 3 不同速率下延迟比较

Apache Spark 默认会将每个执行器的 60% 的内存空间用于缓存 RDD, 则在任务执行期间, 只有 40% 的

内存空间可以用来存放创建的对象. 如果创建的对象过大, 超过可用的内存空间, 就会触发 java JVM 的垃圾回收机制. 从图 4 中可以看到, 随着输入速率的提高, 系统的 GC 执行时间也会增加, 这是因为输入速率的提高, 任务数量及创建的对象也会增加, 从而使 GC 执行次数增加, 即系统运行时的内存占用量增加.

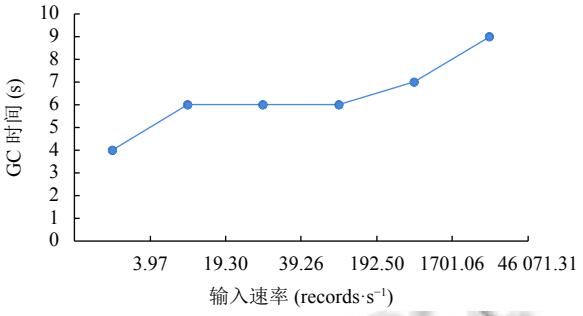

图 4 不同速率下 GC 时间比较

此外, 随着输入速率的提高, CPU 利用率呈增长趋势, CPU 负载逐步提升. 但即使在速度达到最大时, master 节点 CPU 利用率平均为 28.71%, 最大可达 48.47%, CPU 资源并未得到充分利用.

综上所述, 输入速率的提高会使系统资源利用率增加, 但在 Socket 输入最大值的情况下, 仍未实现系统资源的充分利用.

3) 发现三: 数据输入速率在一定阈值内, 系统性能相对稳定; 数据输入速率超过阈值时, 系统性能下降.

通过对表 3 实验结果中的数据输入速率与其他性能指标的关系进行分析, 可以发现当数据输入速率在 19.30 records/s 到 92.50 records/s 范围内时, 程序执行时间和 GC 时间分别稳定在 8.5 s 和 6 s, 任务数和延迟波动较小. 当数据输入速率提升到 1701.06 records/s 时, 系统在执行时间、任务数、延迟、GC 等性能指标的度量上略有增长. 然而, 当数据输入速率上升到 46 071.31 records/s 时, 系统的执行时间比数据输入速率为 1701.06 records/s 时增长了 1 min, 延迟增长了 205 ms, 执行的任务数却有所减少. 由此可以得出结论: 数据输入速率在一定阈值内时, 系统的整体性能相对稳定; 当数据输入速率超过这一阈值时, 系统性能会有所下降.

分析原因为随着输入速率的提高, 系统接收的数据越来越多, 系统的数据处理能力趋于饱和, 甚至可能会出现计算过程中一个批次花费的时间大于系统设置的批处理间隔, 这意味着数据接收速率大于数据处理速率, 数据处理能力降低, 系统性能也发生一定程度下降. 但 Spark Streaming 系统自身带有反压机制 (Back Pressure), 即使时间间隔内无法完全处理当前接收的数据, 也不会导致执行器内存泄漏.

此外, 从图 5 中可以看出, 任务数除了在速率从 3.97 records/s 上升到 19.3 records/s 时出现大幅度增长外, 之后不再随着数据输入速率的增加发生较大变化, 且在速率达到最大时出现下降, 可见系统的任务数随着数据输入速率的增加出现瓶颈.

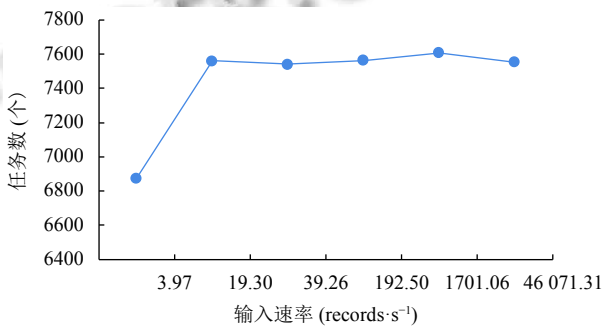

图 5 不同速率下任务数变化

(2) 测试二: 执行器内核数量对对系统性能及扩展性的影响

执行器 (executor) 是 Apache Spark 任务的执行单元, 运行在 worker 上, 是一组计算资源的集合. 执行器的内核 (core) 数量可理解为执行器的工作线程, 实验通过改变执行器的内核数控制系统的并发度.

实验中共系统设置了 4 个执行器, 将每个执行器的内核个数分别设置为 2、4、8 和 16, 测试了 2 min 内数据接收情况、系统总延迟以及 GC 时间. 测试二实验结果如表 4 所示.

表 4 测试二实验结果

每个执行器内核数	数据接收总量 (records)	延迟 (ms)	GC 时间 (s)
2	5790 152	2941	6.4
4	5696 935	1688	6.9
8	5638 742	1466	7.9
16	5513 483	1060	9.6

通过实验数据分析, 可得结论如下:

1) 发现一: 执行器的内核数对系统吞吐量影响不大.

在数据输入速率相同的情况下, 随着每个执行器内核个数的增加, 系统在 2 min 内接收的数据整体呈现减少的趋势, 但从图 6 中可以看到, 在考虑到网络波动

的情况下, 系统接收数据的能力相似. 因此, 系统并发度的提升对 Apache Spark Streaming 的吞吐量并没有太大影响.

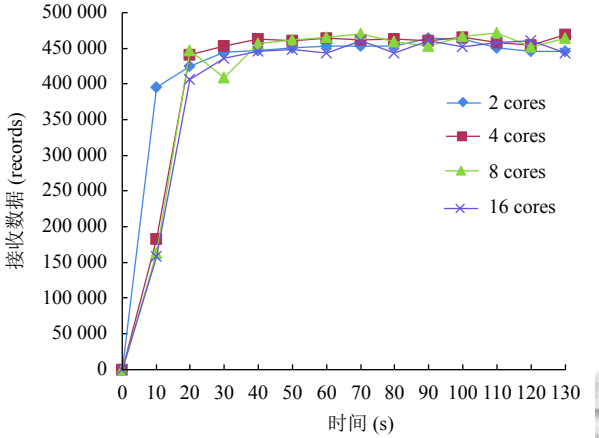

图 6 不同内核数下的数据接收情况

2) 发现二: 执行器内核数量的增加可降低系统延迟.

结合结论一分析, 在系统接收的数据量相差不大的情况下, 随着每个执行器内核个数的增加, 系统的延迟会大幅度降低, 内核数为 16 时的延迟只有内核数为 2 时的 1/3, 如图 7 所示.

分析原因为{任务执行的并发度 = 执行器的总数目 * 每个执行器的内核数}, 当每个执行器内核数量增加时, 任务并发度也会提高, 多任务的并发执行使得从而使系统延迟大幅度降低. 可见系统并行度的提高使得 Apache Spark Streaming 系统资源的利用率也随之提高, 系统在延迟上的扩展性良好.

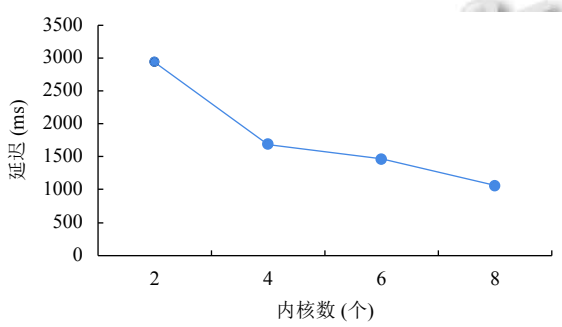

图 7 不同内核数下延迟比较

3) 发现三: 执行器内核数量的增加会造成系统资源利用率增加.

随着内核数量的增加, 任务并发度提高, 导致系统 GC 时间增加, 如图 8 所示.

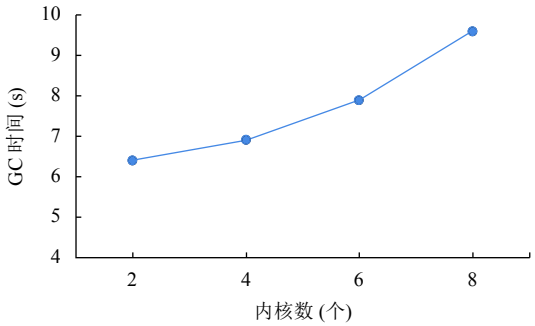

图 8 不同内核数下 GC 时间比较

分析原因为内核数量的增加提高了任务并发度, 使得大量的对象会被创建, 出发 Java 垃圾回收机制的次数也会增加, 从而使 GC 时间增加. 因此适当减少内核个数也是降低系统 GC 开销的一种方法.

此外, 系统并发度的提高也使得 CPU 利用率有所提高, 在 16 cores 时 CPU 利用率平均为 31.05%,最大可达 35.35%.

综上所述, 执行器内核数的增加会提高系统并发度从而增加系统资源利用率.

4 结束语

本文设计并实现了一个流式大数据处理系统基准测试框架, 以 Socket 作为流式数据生成, 构造股票高频交易场景, 并实现了 GroupBy 和 Join 两个典型应用, 将实时计算与结构化数据相结合. 测试框架搭建在分布式集群环境下, 选取了 Apache Spark Streaming 作为待测系统, 从数据输入速率和系统并行度两个方面设计实验, 得到延迟、吞吐量、GC 时间等测试指标, 以图表的形式进行分析, 发现流式大数据处理系统中出现的性能问题. 实验结果表明, 随着数据输入速率的提高, 系统性能保持相对稳定, 当输入速率达到一定阈值, 系统会出现性能下降, 资源利用率增加的现象; 系统并行度的增加对吞吐量的影响较小, 但系统延迟会大幅度降低, GC 时间有所增加, 提高了系统资源的利用率.

未来将在以下几个方面进行深入研究. 一是可以将基准测试框架应用到不同的处理系统中, 分析系统之间的差异, 进行对比研究. 二是对流式大数据处理系统的基准测试不再仅限于对独立的系统, 而是与第三方工具结合, 如使用 Kafka、Flume 等高级数据源产生数据, 模拟现实环境下的应用.

参考文献

1 Manyika J, Chui M, Brown B, *et al*. Big data: The next frontier for innovation, competition, and productivity. http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation, 2011.

2 李国杰. 大数据研究的科学价值. 中国计算机学会通讯, 2012, 8(9): 8–15.

3 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.

4 Apache Hadoop v.3.2.0. http://hadoop.apache.org/. [2019-07-31].

5 Apache Spark v.2.4.0. https://spark.apache.org/. [2019-07-31].

6 Apache Storm v.2.0.0. http://storm.apache.org/. [2019-07-31].

7 Apache Flink v.1.7. https: //flink.apache.org/ . [2019-07-31].

8 Apache spark streaming guid. http://spark.apache.org/streaming/. [2019-03-31].

9 Chintapalli S, Dagit D, Evans B, *et al*. Benchmarking streaming computation engines: Storm, flink and spark streaming. Proceedings of 2016 IEEE International Parallel and Distributed Processing Symposium Workshops. Chicago, IL, USA. 2016. 1789–1792.

10 Huang SS, Huang J, Dai JQ, *et al*. The HiBench benchmark suite: Characterization of the mapreduce-based data analysis. In: Agrawal D, Candan KS, Li WS, eds. New Frontiers in Information and Software as Services. Berlin, Heidelberg: Springer, 2011. 209–228.

11 Hesse G, Lorenz M. Conceptual survey on data stream processing systems. Proceedings of 2015 IEEE 21st International Conference on Parallel and Distributed Systems. Melbourne, VIC, Australia. 2015. 797–802.

12 Gradvohl ALS, Senger H, Arantes L, *et al*. Comparing distributed online stream processing systems considering fault tolerance issues. Journal of Emerging Technologies in Web Intelligence, 2014, 6(2): 174–179.

13 Akidau T, Balikov A, Bekiroğlu K, *et al*. MillWheel: Fault-tolerant stream processing at internet scale. Proceedings of the VLDB Endowment, 2013, 6(11): 1033–1044. [doi: 10.14778/2536222.2536229]

14 Neumeyer L, Robbins B, Nair A, *et al*. S4: Distributed stream computing platform. Proceedings of 2010 IEEE International Conference on Data Mining Workshops. Sydney, NSW, Australia. 2010. 170–177.

15 Nabi Z, Bouillet E, Bainbridge A, *et al*. Of streams and storms. IBM White Paper, 2014: 1–31.

16 Dayarathna M, Suzumura T. A performance analysis of system S, S4, and Esper via two level benchmarking. In: Joshi K, Siegle M, Stoelinga M, *et al*., eds. Quantitative Evaluation of Systems. Berlin, Heidelberg: Springer, 2013. 225–240.

17 Toshniwal A, Taneja S, Shukla A, *et al*. Storm@twitter. Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. Snowbird, UT, USA. 2014. 147–156.

18 Lopez MA, Lobato AGP, Duarte OCMB. A performance comparison of open-source stream processing platforms. Proceedings of 2016 IEEE Global Communications Conference. Washington, DC, USA. 2016. 1–6.

19 Karimov J, Rabl T, Katsifodimos A, *et al*. Benchmarking distributed stream data processing systems. Proceedings of 2018 IEEE 34th International Conference on Data Engineering. Paris, France. 2018. 1507–1518.

20 Karamel, Orchestrating Chef Solo. http://www.karamel.io/. [2019-07-31].

21 Perera S, Perera A, Hakimzadeh K. Reproducible experiments for comparing apache flink and apache spark on public clouds. arXiv preprint arXiv: 1610.04493, 2016.

22 DataArtisans. Extending the Yahoo! Streaming Benchmark. https://www.ververica.com/blog/extending-the-yahoo-streaming-benchmark. (2016-02-02)[2019-07-31].

23 程学旗, 靳小龙, 王元卓, 等. 大数据系统和分析技术综述. 软件学报, 2014, 25(9): 1889–1908. [doi: 10.13328/j.cnki.jos.004674]