文章编号：2096-1472(2019)-12-05-06
DOI:10.19644/j.cnki.issn2096-1472.2019.12.002

分布式流处理系统的容错性能基准测试

蒋 程，王晓桐，张 蓉

(华东师范大学数据科学与工程学院，上海 200062)

摘 要：随着对数据处理的实时性要求越来越高，分布式流处理系统应运而生。但是在分布式的集群规模下，各种软硬件原因导致的故障很难避免的。现有的相关基准测试主要关注于分布式流处理系统的处理性能，很少对该类系统处理故障的容错性能进行评测，以至于关键应用在系统选型的时候特别艰难。针对分布式流处理系统的容错性能，本文设计并实现了一套灵活的基准测试框架。最后，本文在开源数据流处理系统Apache Storm和Apache Flink进行了容错性能的基准测试，验证定义的测试基准的正确性和有效性，实验结果也表明Flink的容错性能相对较好。

关键词：分布式系统；流处理；容错性能；基准测试

中图分类号：TP302.8 **文献标识码**：A

Benchmarking for Fault-tolerant Performance in Distributed Stream Processing Systems

JIANG Cheng,WANG Xiaotong,ZHANG Rong

(*School of Data Science and Engineering,East China Normal University,Shanghai* 200062,*China*)

Abstract: With the increasing real-time requirements for data processing,distributed stream processing systems have emerged.However,under the distributed cluster scale,failures caused by various hardware and software problems are inevitable.The existing related benchmarking mainly focus on the performance of the distributed stream processing system during failure-free time,while rarely evaluating the fault-tolerant performance of the system for handling faults.As a result,it is particularly difficult to select a system for mission-critical applications.This paper designs and implements a flexible benchmarking framework tailored for fault-tolerant performance.Finally,benchmarking the fault-tolerant performance of Apache Storm and Apache Flink verifies the correctness and effectiveness of the benchmark defined in this paper. Experimental results show that fault-tolerant performance of Flink outperforms that of Storm.

Keywords:distributed system;stream processing;fault-tolerant performance;benchmarking

1 引言(Introduction)

分布式流计算系统[1](Distributed Stream Processing Systems，DSPS)是对大规模流数据进行实时处理的系统，主流的开源系统有：Apache Flink[2]、Apache Storm[3]、Apache Spark Streaming[4]等。流计算的常见应用场景有：电商的商品推荐，IoT设备的监控预警，银行的金融欺诈检测等。流计算应用具有以下特征：①高性能：系统需要对流入的数据进行实时处理，延迟一般在毫秒级别。而且由于数据的不断流入，计算需要支持很高的吞吐(例如，推特每天处理5亿推文，Facebook有14.5亿活跃用户)；②容错性：因为流数据的无限性，系统的运行需要支持7×24小时服务。在大规模分布式计算中像节点故障，网络错误等故障经常发生。流处理系统需要使用容错机制来应对故障的发生。特别是对于金融领域的关键应用而言，快速的故障恢复和计算结果的正确性保证尤其重要，否则将会导致严重的财力损失。本文提出了一套针对分布式流处理系统容错性能的基准测试框架。

2 相关工作(Related work)

Linear Road Benchmark[5]最早提出了评测流数据管理系统(Stream Data Management Systems，SDMS)[6]的处理能力。它模拟了高速公路管理系统，该系统能通过实时收集

基金项目：科技部重大专项(2018YFB1003402)，国家自然科学基金资助项目(61432006)。

处理公路上汽车传来的位置数据，提供收费、事故检测和警报等功能。它用于评测Aurora[7]和关系型数据库管理系统能满足该公路系统的最大处理吞吐量。StreamBench[8]针对DSPS设计了七个微型(micro)工作负载和四种工作负载套件，设计了两种真实的应用场景：实时网站日志处理和网络流量监控。它测试了Storm和Spark Streaming的处理性能、稳定性和容错性能。但在容错性能的评测方面，StreamBench仅仅比较有无故障发生时的吞吐和延迟。Yahoo! Streaming Benchmark[9]模拟了广告分析应用，测试了Flink、Storm和Spark Streaming的延迟和吞吐。RIoTBench[10]设计了IoT应用场景下的相关负载，含有27种IoT相关的微型负载和四种由微型负载合成的应用负载，使用真实的IoT数据集评测了Storm的处理性能。Jeyhun等人[11]提出了针对含有连接和窗口等复杂操作的延迟计算方法，定义了系统的最大可持续吞吐量。它模拟了在线视频游戏应用的相关工作负载，评测了Flink、Storm和Spark Streaming三个系统的处理性能。Steffen等人[12]通过广告分析、公路管理和出租车业务查询这三种工作负载的测试，对比分析了Flink、Storm和Spark Streaming的性能瓶颈。它提出当前的分布式流处理系统存在没有充分利用硬件资源的问题，并提出了优化的设计方案。

现有的DSPS基准测试相关信息统计如表1所示。目前大家关注的还是在系统无错情况下DSPS的处理性能，而没有对容错机制和性能影响做深度调查和研究。StreamBench虽然涉及了容错度量，但它只是通过性能指标的变化很粗略地估计故障对性能带来的影响。本文第一次以评测DSPS的容错机制作为研究对象，定义和实现了考察这些容错机制关键因素的benchmark和测试框架，并定义了评测故障恢复机制优劣的性能指标。该测试框架可以有效地评测不同的容错技术给系统性能带来的影响，为不同应用场景下流处理系统的选择提供依据和参考。

表1 相关的基准测试比较

Tab.1 Comparison of relevant benchmarks

基准测试	测试平台	工作负载	度量指标
Linear Road Benchmark	Aurora,STREAM	高速公路管理	吞吐
StreamBench	Storm,Spark	合成负载	延迟，吞吐
Yahoo!Streaming Benchmark	Storm,Spark,Flink	广告点击流分析	延迟，吞吐
RIoTBench	Storm	物联网相关合成负载	延迟，吞吐，资源利用率，抖动率
Benchmarking Distributed Stream Processing Engines	Storm,Spark,Flink	在线视频游戏监控	延迟，吞吐
Analyzing Efficient Stream Processing on Modern Hardware	Storm,Spark,Flink	广告点击流分析，高速公路管理，出租车业务查询	延迟，吞吐，资源利用率

3 容错机制(Fault-tolerant mechanism)

本章节将介绍本文评测的两个最典型分布式流处理系统的容错机制。

3.1 Apache Flink

Flink根据分布式快照算法Chandy-Lamport Algorithm[13]设计出了分布式轻量级异步快照机制。Flink会定期地发送一个栅栏标记到输入的数据流中，从而把源头的数据流按段切割成版本递增的快照。当接收到所有输入流中的栅栏标记后，算子会对当前版本的计算状态进行快照操作，把状态持久化存储到HDFS[14]等可靠的分布式存储系统。一旦所有的算子都确认完成了快照操作，Flink会记录当前版本的全局一致快照已经完成。在恢复期间，Flink首先会重新部署整个计算拓扑；接着每个算子从分布式存储系统中加载各自最近版本的检查点快照；然后根据快照的版本，数据源需重发从最近检查点时刻到故障发生时刻的数据，从而保证了Exactly-Once消息处理语义[15]。

3.2 Apache Storm

Storm的容错机制由消息管理机制和快照机制共同完成，保证了At-Least-Once消息处理语义。消息管理机制指Storm会追踪每条流入系统的数据，为其后续生成的子消息维护一个“消息树”。当且仅当子消息都被成功处理，消息树才会被判定为成功处理。否则，系统会重发对应的输入源消息。快照机制指Storm会将算子的计算状态进行持久化保存。与Flink的容错机制类似，Storm会定期向数据流中插入“快照事务”的消息；算子接收到快照事务消息后触发准备操作与提交操作：收到“准备”事务消息时，算子将当前版本的状态临时持久化；收到“提交”事务消息时，算子将当前版本的状态持久化，并删除临时状态。在恢复期间，算子的状态会根据故障发生时的快照事务状态做出相应的恢复。如果快照事务处于“正在准备”状态，由于部分算子并没有临时持久化准备阶段的状态，则所有算子回滚至最近稳定的快照版本；如果快照事务处于“正在提交”状态，由于所有算子都已经临时持久化准备阶段的状态，则所有算子继续原来的计算任务。故障导致未完全处理的消息会因为消息超时或者算子主动发送失败消息而标记成失败状态，由消息管理机制负责重发。

4 基准测试框架(Benchmarking framework)

本章节介绍本文的基准评测设计，主要按包含的三个部分：容错相关的度量定义，速率可控的数据实时生成，特征可控的负载设计。评测框架如图1所示，包括如下四个部分：①数据生成器负责按给定速率实时生成流数据。该框架中的数据生成主要指控制数据流的产生流量和数据分布特征，从而改变DSPS的计算节点处理量。数据集包括两类：一是下载的公共数据集，二是合成数据集。②消息队列Kafka负责输入

数据和结果的存储。Kafka作为本文的消息传输组件，不仅负责实时传输生成的数据至DSPS中进行消费，还负责存储计算产生的结果消息。③DSPS负责运行拓扑任务。DSPS根据负载配置，如算子并行度、状态大小等参数，在集群上生成并运行分布式测试工作负载。④度量收集器负责收集并统计度量指标。度量收集器具有获取集群的资源利用情况、获取DSPS的实时吞吐和获取延迟信息并进行统计等功能。

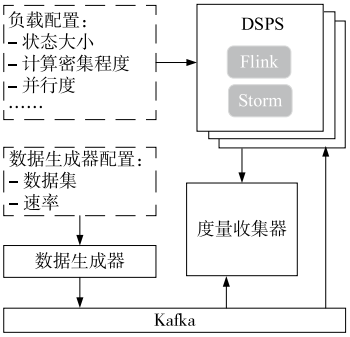


图1 基准测试框架架构

Fig.1 Benchmarking framework architecture

4.1 度量定义

根据流计算和容错机制的特性，我们设计三个相关度量：延迟、资源利用率、故障恢复时间。

延迟：本文采用的是事务时间的延迟。数据产生的时候，该数据会存储产生时刻的时间戳，这个时间戳称为生产时间。输入DSPS系统的数据称为原始数据。经过DSPS运算，原始数据可能产生多个子数据，子数据的生产时间按照原始数据的生产时间不变。输出时间定义为子数据经过DSPS计算处理后时间，但是不包含结果传输时间。这为了防止由结果存储组件的不合理配置，性能瓶颈等原因可能造成因传输导致延迟增大的问题。一条数据的生产时间和输出时间差称为这条数据的事务时间延迟，如图2所示。

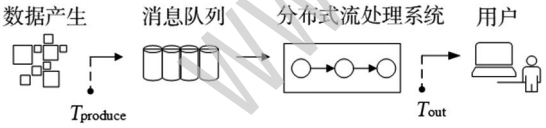


图2 事务时间延迟

Fig.2 Event-time latency

每个元组延迟计算公式：

$$T_{tuple}=T_{out}-T_{produce} \tag{1}$$

T_{out}指结果算子接收到该元组的时间，$T_{produce}$指数据生成器生成该元组的时间。如果一条元组经过计算产生多个子元组，那么子元组的$T_{produce}$跟产生该元组的原始元组相同。本文指的延迟是所有元组延迟的平均值。

资源利用率：容错机制对状态的存储，传输等操作会给系统带来额外的资源使用。本文关注于节点在任务运行时间段内的平均CPU使用率。

故障恢复时间：本文通过软件的方法实现在节点中随机终止DSPS的运算进程从而达到模拟故障的效果。故障发生时间定义为故障脚本的启动时间。故障恢复时间可从宏观和微观的角度进行定义。宏观的角度指：从故障发生到系统的吞吐恢复到正常数值(无故障情况下)的时间；微观的角度指：故障恢复时间可具体划分为重载时间和重播时间。重载时间指从故障发生后到算子经过重新部署并且从存储系统中重载快照数据所花费的时间，重播时间指数据源重播到故障前消费的数据所花费的时间。从故障发生的时刻到系统恢复故障前状态的时刻，这一时间段称为故障恢复时间，如图3所示。

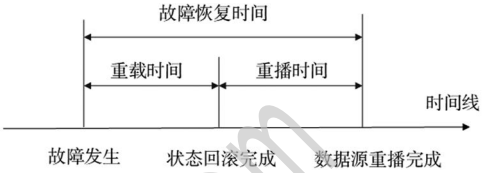


图3 微观的故障恢复时间

Fig.3 Microscopic failure recovery time

根据上述定义，在Flink中，故障恢复时间从微观角度按故障发生的时间到数据源重新处理到故障前的数据时间计算；而在Storm中，由于快照机制和消息重播机制分离，故障恢复时间只能从宏观角度按故障发生的时间到数据源的吞吐恢复稳定的时间计算。

4.2 数据集与工作负载

本章节抽象出数据流的数据特征和有状态负载的特征，通过调控特征参数，可以模拟并控制系统工作负载。

4.2.1 输入数据流特征

数据流数据本身有三个可调控的特征参数。

输入速率：为了使系统运行在稳定的状态，本文控制一个稳定并且合适的数据生产速率，防止系统负载过高进入反压状态[16]。

数据倾斜度：数据倾斜是数据集中常见的特性。不均匀的数据分布将会导致大量数据集中在某些节点，造成节点的运算负荷不同。本文按Zipf定律生成数据倾斜的合成数据集。

输入数据大小：数据流具有无限性，但是根据实验需求，可根据输入的吞吐速率和运行时间修改原始数据集大小，计算公式如下：

$$L=P*T \tag{2}$$

其中，L是修改后的数据集总量，P是设定的输入吞吐速率(条/秒)，T是任务运行的时间(秒)。

本文内置数据集含有两种：第一种是从古登堡计划(Project Gutenberg)获取的英文小说集；第二种是根据数据倾斜程度生成的合成数据集。数据生成器根据配置的输入吞吐速率，实时从数据集中获取数据并输入到Kafka中，模拟生产环境中的实时数据生成。

4.2.2 工作负载设计

本文设计了两类工作负载：①计算简单、状态大小可调

控的Word Count负载[17]；②状态大小固定、计算密集程度可调控的圆周率计算负载。工作负载的特征如图4所示。通过分析有状态计算的特征，本文的工作负载设有两个可调控的特征参数。

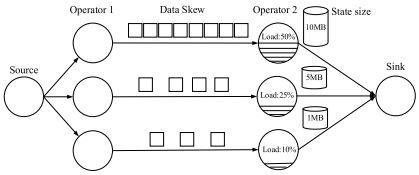


图4 工作负载

Fig.4 Workload

算子的状态大小：影响存储和备份即memory和磁盘I/O。算子分为有状态计算和无状态计算。无状态计算指不需要依赖历史数据进行计算的算子，如切分算子，只需对当前数据进行分词操作。有状态计算指当前计算需要根据历史数据进行计算，如窗口算子，计算需要对到达窗口内的所有数据或者计算的中间结果值等状态进行聚合计算或者更新。为了防止因为故障导致状态的丢失，保证恢复后计算的准确性，有状态的算子需要对状态进行持久化存储。本文使用全历史计算而不使用窗口算子，因为窗口算子的状态大小不可控。在DSPS中，连接操作需要使用窗口算子，本文也不使用。在窗口算子中，触发checkpoint操作的时间点在窗口内呈现无规则分布，这种现象导致每次实验中checkpoint保存的状态大小不一致，无法通过控制变量法研究状态大小和checkpoint间隔对系统带来的影响。本文在2号算子中根据配置参数进行自定义大小的字符串类型状态存储，保证每次存储的状态大小一致，从而研究不同状态大小和checkpoint间隔对系统的影响。

算子的计算密集程度：影响CPU。本文研究不同计算密集型的算子受checkpoint操作的影响。本文设计状态大小固定的圆周率计算算子，通过传入配置参数实现控制2号算子中格雷戈里-莱布尼茨级数的运算次数，以此来调控该算子的计算密集程度。格雷戈里-莱布尼茨级数的计算公式如下：

$$\frac{\pi}{4}=\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1} \tag{3}$$

本文通过对抽象出的两个特征的调控，可以模拟出其他工作负载的特征，比如含有窗口操作的负载需要对到达窗口内的所有数据进行保存，存储状态较大；含有连接操作的负载需要对多条输入流进行连接操作，连接算子的运算密集程度大，并且需要对多条流的数据都进行保存，存储状态较大。

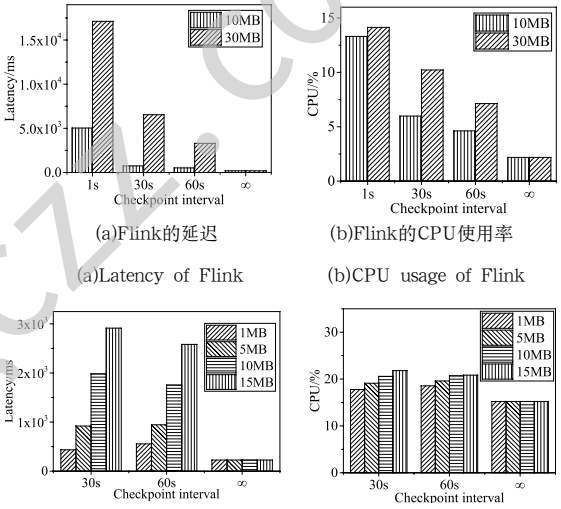
5 实验(Evaluation)

5.1 实验环境

本文的实验在具有五个节点的集群上进行，节点的操作系统版本是CentOS v.6.5。测试平台为Apache Flink 1.7.0，Apache Storm 1.2.2。其中一个节点配置为24核Intel(R)Xeon(R)CPU E5-2620、频率2.40GHz、内存31GB，部署非计算组件，如HDFS、Zookeeper、Redis等服务。其余四个节点配置为8核Intel(R)Xeon(R)CPU E5606，频率2.13GHz，内存94GB，部署计算组件，如Flink中的Taskmanager，Storm中的Worker等计算进程。计算组件和非计算组件的分开部署能提高度量指标的准确性，如资源利用率。节点之间通过千兆以太网连接。默认的数据输入吞吐速率为5000条/秒；数据集使用真实的英文小说集。

5.2 无故障性能评测

系统在未发生故障的时候，容错机制对性能的影响源自周期性地进行的快照操作，对计算产生的中间状态进行持久化存储。Checkpoint操作的频率和持久化的状态大小均是影响系统性能的重要因素。本组实验研究不同状态大小和checkpoint间隔对延迟和CPU使用率的影响。



(a)Flink的延迟 (b)Flink的CPU使用率

(a)Latency of Flink (b)CPU usage of Flink

(c)Storm的延迟 (d)Storm的CPU使用率

(c)Latency of Storm (d)CPU usage of Storm

图5 性能测试

Fig.5 Performance test

从图5(a)和图5(b)中可以看出，在Flink中，当状态大小保持一致时，checkpoint间隔越短，计算的延迟越大，系统的CPU消耗越多。因为越频繁的checkpoint操作会导致系统花费更多的资源在状态处理上，使得正常的计算暂停的时间越多；当checkpoint间隔保持一致时，状态越大，计算的延迟越大，系统的CPU消耗越多。因为状态越大，每次对状态的持久化操作所需时间更久，对性能造成的影响更大。在Storm平台上1秒的checkpoint间隔过于频繁并且较大的状态会严重影响系统的性能，导致其无法正常运行。故checkpoint间隔始于30s。从图5(c)和图5(d)中可以看出，在Storm中，checkpoint间隔的影响和Flink稍有不同。Storm的容错机制对系统处理的影响主要有两种操作。第一种操作是对状态的存储，该操作造成的延迟受状态大小的影响；第二种是对checkpoint间隔时间内缓存的元组进行消息管理操作，该操作造成的延迟

受checkpoint间隔的影响。状态较小时(0—5MB)第一种操作的延迟影响比第二种操作小。状态较大时(5—15MB)第二种操作的延迟影响比第一种操作小。CPU使用率变化趋势也是类似的情况，但是平衡点在10MB左右。关于状态的影响，当checkpoint间隔保持一致时，状态与延迟和CPU使用率成线性关系。因为状态越大，状态持久化的操作所花费的时间越久，使得正常运算的延迟增大。

观察Flink和Storm该组实验，如状态大小为10MB，checkpoint间隔为30s时，Flink的延迟比Storm低，而且CPU使用率也更低。

Flink通过栅栏的对齐操作来保证Exactly-Once消息处理语义。本文通过生成合成数据来模拟数据倾斜程度的不同，图6反应不同栅栏到达时间对延迟的影响。本组实验的研究参数：checkpoint模式为NCP(不开启checkpoint)、CP+NA(开启30秒间隔的checkpoint，但是不开启对齐操作)和CP+A(开启30秒间隔的checkpoint，并且开启对齐操作)。

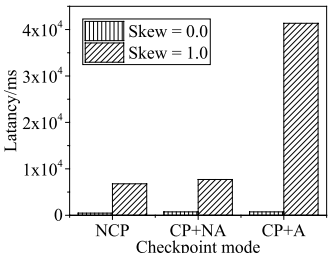
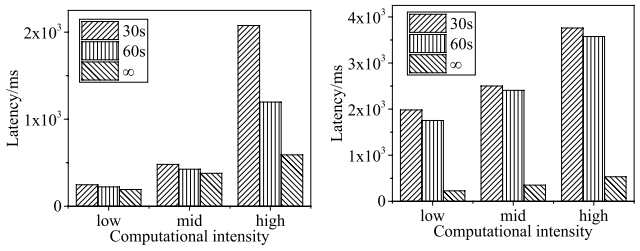


图6 不同数据倾斜下延迟对比

Fig.6 Latency under different data skewness

从图6可以看出，在数据倾斜度较大时，对齐操作对延迟的影响很大。这是因为在数据倾斜程度均匀的时候，每个算子的多个输入通道中的栅栏到达时间相近，对齐操作导致的堵塞时间较少，所以延迟无明显增大。但是在数据倾斜程度较大的时候，因为一个算子含有多个输入通道时，数据量较少的低负载通道中的栅栏会先到达。这时对齐操作会堵塞已到达栅栏的通道，等到数据量较多的高负载通道中的栅栏。不同通道的栅栏到达时间相差越大将会导致该算子的同步堵塞操作时间越长，最终延迟会因此增大。



(a)Flink测试　　(b)Storm测试

(a)Test on Flink　　(b)Test on Storm

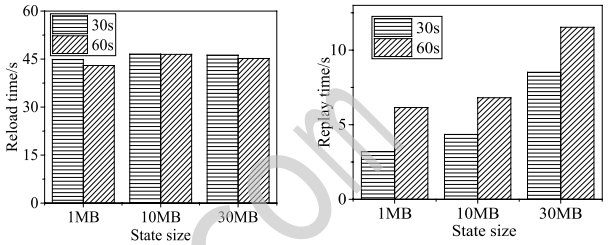
图7 不同计算密集程度下延迟对比

Fig.7 Latency under different computational intensities

图7展示负载计算密集程度受checkpoint操作的影响。越频繁的checkpoint操作会导致频繁的线程调度，切换等问题，负载计算密集程度越高受其干扰的影响越大，最终导致计算的延迟增大。

5.3 故障实验

本文模拟的故障实验是进程级别的故障。由于程序错误、计算资源限制等原因，某个计算进程出错的概率很大。本文在流计算任务稳定运行一段时间后，使用软件脚本随机终止某个节点上的某个计算进程，从而模拟计算进程故障。本组实验设置的固定条件与上组实验相同。



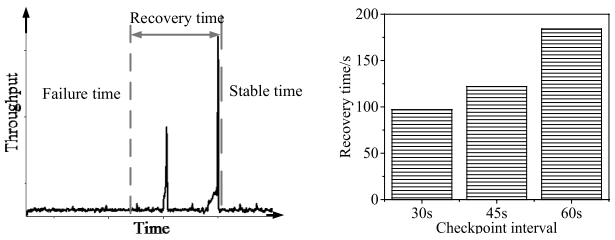
(a)重载时间　　(b)重播时间

(a)Reload time　　(b)Replay time

图8 Flink故障试验

Fig.8 Failure test on Flink

Flink的故障恢复时间可以根据恢复阶段来划分成重载时间和重播时间。重载时间指从故障发生的时间到任务重新部署完成的时间。重播时间指任务部署完成到数据源重新消费到故障发生前数据时间。从图8中可以看出，故障恢复时间中重载阶段的耗时占比较大。因为系统探测到TaskManager故障的时间跟配置参数心跳超时时间成正相关关系。在默认配置下，重载阶段花费约45秒左右，总恢复时间在60秒之内。从图8(b)中可以看出，状态的大小和重播时间成正相关关系。因为在Flink的恢复过程中，算子各自进行恢复操作，状态大导致算子的平均恢复时间大，数据源的重发速率受其影响。



(a)吞吐　　(b)恢复时间

(a)Throughput　　(b)Recovery time

图9 Storm故障试验

Fig.9 Failure test on Storm

Storm中的恢复时间采用从宏观的角度评测，根据吞吐随时间变化情况测量恢复时间，如图9(a)所示。由于Storm的整体处理性能较低，本组实验中数据输入吞吐速率降为500条/秒，从而保证Storm能正常故障恢复。Storm的故障恢复由消息管理机制与快照机制共同完成，二者相互独立，且前者对性能

的影响占主导因素。Storm恢复故障算子的时候不需要部署整个任务，只需重启故障的计算进程，这部分操作耗时约在10秒。但是消息重发阶段需要等待消息超时后由消息管理机制负责重发。本实验在保证实验正常运行的情况下，研究不同checkpoint间隔对恢复时间的影响。从图9(b)中可以看出，checkpoint间隔越大，恢复时间越长。因为checkpoint间隔影响了消息超时的时间，越长的checkpoint间隔导致失败的消息被判定超时并且重发的所需时间越久，所以恢复时间越久。

对比Flink和Storm的故障实验，即使在较高的输入吞吐和较大的状态下，Flink的恢复时间更低，并且保证的语义更强，总体性能优于Storm。

6 结论(Conclusion)

本文提出一种针对分布式流处理系统的容错性能评测框架，使用真实和模拟的数据集，定义了影响容错性能的负载特征以及容错评估指标，评测了Flink和Storm的容错性能。在非故障期间，对容错机制对系统的性能影响进行了评测；在故障发生后，对系统的恢复时间进行了评测。实验结果表明，Flink的容错机制不仅保证了更高级的处理语义，而且对系统的性能影响较小，故障恢复也更快速。未来，我们将在几方面开展工作：评测其他分布式流处理系统；增加输入流的相关特征控制，如动态变化的输入速率、动态变化的skew分布，更真实地模拟生产环境；添加复杂工作负载；加入恢复准确性等评测指标。

参考文献(References)

[1] Cherniack M,Balakrishnan H,Balazinska M,et al.Scalable Distributed Stream Processing[C].CIDR,2003,3:257−268.

[2] Carbone P,Katsifodimos A,Ewen S,et al.Apache flink:Stream and batch processing in a single engine[J].Bulletin of the IEEE Computer Society Technical Committee on Data Engineering,2015,36(4):28−38.

[3] Toshniwal A,Taneja S,Shukla A,et al.Storm@ twitter[C].Proceedings of the 2014 ACM SIGMOD international conference on Management of data.ACM,2014:147−156.

[4] Zaharia M,Das T,Li H,et al.Discretized streams:Fault−tolerant streaming computation at scale[C].Proceedings of the twenty−fourth ACM symposium on operating systems principles.ACM,2013:423−438.

[5] Arasu A,Cherniack M,Galvez E,et al.Linear road:a stream data management benchmark[C].Proceedings of the Thirtieth international conference on Very large data bases−Volume 30.VLDB Endowment,2004:480−491.

[6] 金澈清,钱卫宁,周傲英.流数据分析与管理综述[J].软件学报,2004(08):1172−1181.

[7] Abadi D J,Carney D,Çetintemel U,et al.Aurora:a new model and architecture for data stream management[J].the VLDB Journal,2003,12(2):120−139.

[8] Lu R,Wu G,Xie B,et al.Stream bench:Towards benchmarking modern distributed stream computing frameworks[C].2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing.IEEE,2014:69−78.

[9] Chintapalli S,Dagit D,Evans B,et al.Benchmarking streaming computation engines:Storm,flink and spark streaming[C].2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW).IEEE,2016:1789−1792.

[10] Shukla A,Chaturvedi S,Simmhan Y.Riotbench:a real−time iot benchmark for distributed stream processing platforms[J].arXiv preprint arXiv:1701.08530,2017.

[11] Karimov J,Rabl T,Katsifodimos A,et al.Benchmarking distributed stream processing engines[J].arXiv preprint arXiv:1802.08496,2018.

[12] Zeuch S,Monte B D,Karimov J,et al.Analyzing efficient stream processing on modern hardware[J].Proceedings of the VLDB Endowment,2019,12(5):516−530.

[13] Mattern F.Efficient algorithms for distributed snapshots and global virtual time approximation[J].Journal of parallel and distributed computing,1993,18(4):423−434.

[14] Shvachko K,Kuang H,Radia S,et al.The hadoop distributed file system[C].MSST,2010,10:1−10.

[15] Lopez M A,Lobato A G P,Duarte O C M B.A performance comparison of open−source stream processing platforms[C].2016 IEEE Global Communications Conference (GLOBECOM).IEEE,2016:1−6.

[16] 熊安萍,朱恒伟,罗宇豪.Storm流式计算框架反压机制研究[J].计算机工程与应用,2018,54(1):102−106.

[17] Ranger C,Raghuraman R,Penmetsa A,et al.Evaluating MapReduce for multi−core and multiprocessor systems[C].hpca.2007,7(3):19.

作者简介：

蒋 程(1995−)，男，硕士生.研究领域：数据流基准测试.

王晓桐(1994−)，女，博士生.研究领域：分布式数据流处理，数据流基准测试.

张 蓉(1978−)，女，博士，教授.研究领域：分布式数据管理.本文通讯作者.