计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190300070

基于 Spark Streaming 的流式并行文本校对

杨宗霖[1] 李天瑞[1,2] 刘胜久[1] 殷成凤[1] 贾 真[1] 珠 杰[3]
1 西南交通大学信息科学与技术学院 成都 611756
2 西南交通大学人工智能研究院 成都 611756
3 西藏大学计算机科学系 拉萨 850000
(zlyang_swjtu@163.com)

摘 要 互联网的高速发展催生了海量网络文本,这对传统的串行文本校对算法提出了新的性能挑战。尽管近年来文本自动校对任务受到了较多关注,但相关研究工作多集中于串行算法,鲜有涉及校对的并行化。文中首先对串行校对算法进行泛化,给出一种串行校对的通用框架,然后针对串行校对算法处理大规模文本存在的耗时长的不足,提出3种通用的文本校对并行化方法:1)基于多线程的线程并行校对,它基于线程池的方式实现段落和校对功能的同时并行;2)基于 Spark MapReduce 的批处理并行校对,它通过 RDD 并行计算的方式实现段落的并行校对;3)基于 Spark Streaming 流式计算框架的流式并行校对,它通过将文本流的实时计算转为一系列小规模的基于时间分片的批处理作业,有效避免了固定开销,显著缩短了校对时延。由于流式计算兼有低时延和高吞吐的优势,文中最后选用流式校对来构建并行校对系统。性能对比实验表明,线程并行适合校对小规模文本,批处理并行适合大规模文本的离线校对,流式并行校对有效减少了约 110s 的固定时延,相比批处理校对,采用 Streaming 计算框架的流式校对取得了极大的性能提升。

关键词:自动校对;流式计算;并行计算;多线程;Spark
中图法分类号 TP391

Streaming Parallel Text Proofreading Based on Spark Streaming

YANG Zong-lin[1],LI Tian-rui[1,2],LIU Sheng-jiu[1],YIN Cheng-feng[1],JIA Zhen[1] and ZHU Jie[3]
1 School of Information Science and Technology,Southwest Jiaotong University,Chengdu 611756,China
2 Institute of Artificial Intelligence,Southwest Jiaotong University,Chengdu 611756,China
3 Department of Computer Science,Tibetan University,Lasa 850000,China

Abstract The rapid development of the Internet has prompted the generation of massive amounts of network text,which poses new performance challenges for traditional serial text proofreading algorithms. Although the text automatic proofreading task has received more and more attention in recent years,the related research work mostly focuses on serial algorithms,and rarely involves the parallelization of proofreading. Firstly,the serial proofreading algorithm is generalized,and a general framework of serial proofreading is given. Then,in view of the shortcomings of serial proofreading for processing large-scale texts,three general text proofreading parallelization methods are proposed:1)a parallel proofreading method based on multi-threading,which implements simultaneous parallelism of paragraph and proofreading functions based on the thread pool;2)a batch processing parallel proofreading method based on Spark MapReduce,which implements paragraph parallel proofreading by means of RDD parallel computing;3)a Spark Streaming-based parallel proofreading approach,which converts the real-time calculation of text streams into a series of small-scale time fragmentation based batch jobs,making it can effectively avoid fixed overhead and significantly reduce proofreading delay. Because the streaming computing has the advantages of low delay and high throughput,the paper finally chooses the streaming computing-based method to build the parallel proofreading system. Performance comparison experiments demonstrate that thread parallelism is suitable for proofreading small-scale text,batch processing is suitable for off-line proofreading of large-scale text,and streaming parallel proofreading effectively reduces the fixed delay of about 110 seconds. Compared with

到稿日期:2019-03-16 返修日期:2019-05-07 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61573292);四川省科技服务业示范项目(2016GFW0167)
This work was supported by the National Natural Science Foundation of China (61573292) and Science and Technology Service Industry Demonstration Project of Sichuan Province(2016GFW0167).
通信作者:李天瑞(trli@swjtu.edu.cn)

batch proofreading, the streaming proofreading using a real-time computing framework has achieved a great performance improvement.

Keywords Automatic correction, Streaming computing, Parallel computing, Multi-threading, Spark

1 引言

随着信息技术与互联网的高速发展，网络文本数量呈爆炸式增长。然而，海量网络文本存在质量良莠不齐的特点，这对传统的以人工为主的校对提出了严峻挑战。为了减少人们的校对工作量，与自动校对相关的研究得到了人们的重点关注。例如，文献[1-2]构建了基于分类器的自动校对系统来解决英文文本校对问题；文献[3-5]将文本校对视作转换错误文本为正确文本的单语翻译任务，将基于短语的统计机器翻译(Phrase-based Statistical Machine Translation，PBMT)方法引入自动校对领域；文献[6-8]则充分利用机器翻译领域最近几年所取得的长足进展，构建了若干基于神经机器翻译(Neural Machine Translation，NMT)模型的英文文本校对系统；在中文校对领域主要有 3 类方法，即基于规则[9-10]、基于语言模型[11-14]和基于知识库[15]。

尽管研究人员在校对领域做了大量探索，但已有成果大都局限于串行校对，鲜有涉及文本校对的并行化。鉴于串行自动校对系统在处理大文本时存在耗时长的问题，本文结合多核计算和目前流行的内存分布式计算框架 Spark[16]，提出了 3 种通用的文本校对并行化方法：1)基于多线程的线程并行校对；2)基于 Spark MapReduce 的批处理并行校对；3)基于 Spark Streaming 流式计算框架[17]的流式并行校对。

线程并行校对用于加速小规模文本的校对过程，它基于线程池的方式实现段落和校对功能的同时并行。批处理并行校对面向大规模文本离线校对的场景，拥有集群计算所赋予的强大算力。流式并行校对则致力于缩短校对时延，提高系统吞吐量，它通过实时计算的方式有效解决了批处理作业启动固定开销大的问题。相比线程并行，流式并行具有处理大文本的能力；而相比批处理并行，流式校对又具有低时延的优势。因此，本文在设计开发文本校对系统时选用流式计算来加速文本校对。

2 背景

2.1 相关工作

2.1.1 英文文本的自动校对

英文自动校对系统致力于纠正文本中的非词错误(拼写错误)、变形错误(时态和单复数等)、真词错误(词语搭配)和词序乱序错误(针对非原生语言使用者)等。CoNLL-2014 语法错误纠正(Grammatical Error Correction，GEC)共享任务[18]的成功完成，使得自动校对任务得到了更多关注，一系列基于分类器和基于机器翻译的方法相继得到应用。2016 年 Rozovskaya 等[3]提出的一种管道架构，实现了分类器[2]和统计机器翻译(Statistical Machine Translation，SMT)系统的串联。同年，Junczys-Dowmunt 等[4]使用词级别的 SMT 模型和重打分机制取得了当时最先进的语法错误纠正性能。在此基础上，2017 年，Chollampatt 等[5]加入了自适应神经网络联合模型和基于字符级别 SMT 系统的拼写错误纠正器，在 CoNLL 2014 基准测试集[18]上取得了更好的性能。

然而，基于 SMT 的 GEC 系统受到泛化能力的限制，不能有效地访问更广泛的源端和目标端上下文，于是研究人员将 NMT 方法应用到文本自动校对任务中，提出了一系列基于循环神经网络(Recurrent Neural Network，RNN)的序列到序列(seq2seq)模型。2016 年，Yuan 等[6]首次将基于 RNN 的 NMT 模型应用到 GEC 任务中，他们基于 CLC 语料[19]构建了词级别的模型，并使用传统的后处理方法来解决目标端输出未知词的问题。针对词级别的模型受到有限容量词汇表的限制，不能很好地处理未登录词的问题，Xie 等[7]使用字符级别的编解码网络实现了开放词汇表。然而，完全基于字符的模型不能有效利用词级别的信息，于是 Ji 等[8]于 2017 年提出了词字符混合校对模型，他们拓展了 Luong 等[20]的词字符混合机器翻译模型并加入了嵌套注意力层。

2.1.2 中文文本的自动校对

早期，中文领域多用规则和统计学习方法来解决校对任务。2002 年，Xu 等[9]使用基于动态规划的最长公共子序列长度求解算法，通过比对考生输入的有序字符集和标准答案来解决计算机考试文字录入的自动阅卷问题。2003 年，Gong 等[10]基于两阶段检查方法来检测中文文本的语法错误：第一遍全文扫描时，他们基于从错误语料中总结出来的搭配错误规则，通过模式匹配的方法来检测搭配错误；第二遍全文扫描时，他们基于句型成分分析方法检查与句子成分相关的错误。同年，Chen 等[11]结合二元词性模型、单字词共现概率极低的性质和相邻词的互信息对所构建的候选集进行排序，用第一候选字词纠正文本错误。

2016 年，Liu 等[12]基于 N-Gram 模型计算邻接二元概率和三元概率并加权求和，利用多特征融合模型的结果对混淆集进行排序，最后再通过规则标记错误真词的查错状态；他们[13]还提出了一种模糊分词算法，旨在解决“非多字词错误”[21]。2017 年，Zhang 等[15]通过词语搭配知识库和证据理论进行中文文本语义错误的侦测。同年，Zhang 等[14]将校对转换成混淆集排歧的任务，他们通过正向最大策略进行组块，通过拼音相似和笔画相似来获取当前组块的相似词列表，并借助字典树[22]加快词库的模糊搜索，最后基于二元模型进行打分排序并校对。

2.2 串行文本校对和问题定义

Zhang 等[14]指出文本校对应涉及词语的校对、标点符号的校对[23]、拼音的校对和数字的校对(相关系统框图见首页二维码)。考虑到文本由段落构成，本文泛化串行校对算法，给出了一种串行校对的通用框架。通用的串行校对过程(流程图见首页二维码)的描述如下：串行校对系统顺序地处理文本的各个段落，然后对每个段落依次进行词语校对、标点符号校对、拼音校对和数字校对，各校对功能均返回错误列表，将它们合并后即得到被检测文本整体的错误列表。其中，各校

对功能算法的实现不做限制。

将串行校对框架并行化，是本文要解决的问题。为此，本文提出了3种通用的文本校对并行化方法(见第3节)，其中各校对功能算法的实现不做限制。

本文各校对功能(词语审校、标点符号审校、数字审校等)算法的具体实现请参考文献[14]。其中，词语校对方法的阐述请见2.1.2节，标点符号、数字和拼音的校对均基于规则，通过错误模板对待检测文本进行模式匹配，然后对匹配处进行相应的修改。

3 并行文本校对

基于2.2节所描述的文本校对串行处理过程，本节首先阐述串行算法的两个并行点(见3.1节)，其次介绍文本校对的线程并行方法(见3.2节)，接着说明批处理校对的并行策略(见3.3节)，然后基于Spark Streaming重构批处理校对提出流式并行文本校对(见3.4节)，最后介绍流式校对的性能调优(见3.5节)。

3.1 段落并行和校对功能并行

由于各个段落的校对独立进行，且各校对功能间也无依赖关系，因此结合2.2节中的串行处理过程易知存在两个并行点：段落并行和校对功能并行。段落并行即数据并行，段落之间并行执行，但各段落还是顺序地执行词语、标点符号、数字和拼音校对，最后合并各个段落各校对功能的输出，并将合并的结果作为被检测文本的整体错误列表。校对功能并行即方法并行，同时对文档进行各种校对，但各个校对过程还是顺序地处理各个段落，最后将各校对功能返回的错误列表合并为被检测文本的整体错误列表。

本文提出的3种文本校对通用并行化方法基于不同的并行算法：1)线程并行校对同时进行段落并行和校对功能并行；2)批处理并行和流式校对均采用段落并行。

如2.2节所述，各校对功能返回的均为错误列表，且段落并行和校对功能并行均将各错误列表合并的结果作为被检测文本整体的校对结果，因此本文的段落并行和校对功能并行相比串行校对算法不会遗漏文本或重复校对，即并行算法与串行算法的结果一致。

此外，现有的校对方法均基于局部信息，例如基于分类器的GEC方法基于短语上下文信息、基于NMT的GEC方法基于句子上下文信息，因此针对2.2节所述的串行校对框架进行段落并行不会丢失上下文信息。

3.2 线程并行文本校对

线程并行校对同时进行段落并行和校对功能并行(见图1)，它基于可变容量线程池，工作线程可被重复利用。线程并行的具体步骤如下：1)遍历文本和校对功能，生成$M*N$(M为段落数，N为校对功能数目)个包含待校对段落、校对功能ID和运行方法的校对任务，并将它们提交到线程池中；2)线程池中的工作线程并发地执行队列中的任务，工作线程根据校对功能ID对目标段落进行不同的校对；3)基于Java的可重入锁ReentrantLock，各线程校对完毕后互斥地将段落错误列表添加到文档错误列表中；4)平缓地关闭线程池，待队列中的任务均被执行完毕后才返回文档错误列表。

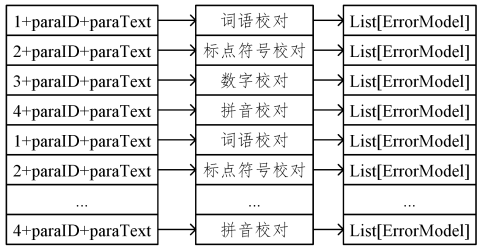

图1 段落和校对功能同时并行

Fig. 1 Paragraph and proofreading functions parallelization

3.3 批处理并行文本校对

线程并行适合小规模文本的校对，而批处理并行则适合大规模文本的离线校对。批处理的并行方案为段落并行(见图2)。

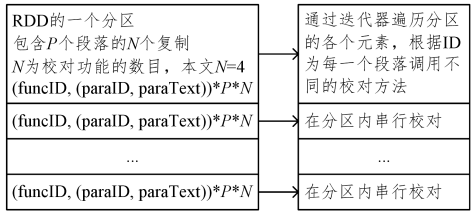

图2 段落(数据)并行

Fig. 2 Paragraph (data) parallel

具体的并行策略为：1)将待校对文本从HDFS读入RDD，读入时指定RDD的分区数，得到的初始RDD的元素与段落一一对应；2)通过flatMap算子将文本的各个段落复制$N-1$份(N为校对功能的数目)，并在每一个段落前面加上校对功能ID；3)由于Spark以分区为并行的粒度，因此本文使用mapPartitions算子替代map，通过调用mapPartitions算子获取各分区的迭代器，然后通过迭代器遍历分区的各个元素，再根据ID为每一个段落调用不同的校对方法；4)通过collect算子拉取结果RDD的数据，并将其返回错误列表。

批处理并行尽管能够利用集群算力来加速大文本的校对过程，但存在作业启动固定开销大的弊端，这在小文本的情况下尤为严重。基于批处理的文本校对在每次作业启动时都存在若干固定开销：1)上传待校对文本到HDFS；2)Driver向集群资源管理器申请Executors作为临时资源池；3)将jar包序列化分发到各个节点，该操作耗时约30 s；4)一些校对类还要加载词库和规则库，加载操作耗时50 s左右。

在实际使用中发现，由于存在上述若干固定重复操作，批处理作业启动耗时过长。为了缩短校对时延，提高系统吞吐量，本文基于Spark Streaming流式计算框架重构并行批处理校对，提出基于Spark Streaming的流式并行文本校对。

3.4 流式并行文本校对

文本校对使用的是文件输入源，针对文件输入源的特点，本文提出了如图3所示的流式校对方法。由于本质上Streaming是将流计算转化成一系列小规模的基于流的批处理作业，因此批处理部分的逻辑与3.3节所述批处理文本校对的并行策略基本一致，皆为段落并行。

流式校对方法的具体内容为：1)将待校对文本上传到HDFS上的临时目录中；2)将文本从临时目录移动到监控目录中，以避免上传过程产生的临时文件被Streaming误以为

是要处理的文件；3）Streaming 会按照固定的批时间间隔，把每个时间分片内新进入监控目录的文本合并成一个 RDD 进行批处理校对；4）每个文本的每一行都已事先存有对应文件的路径，根据路径将结果回写到用户目录。

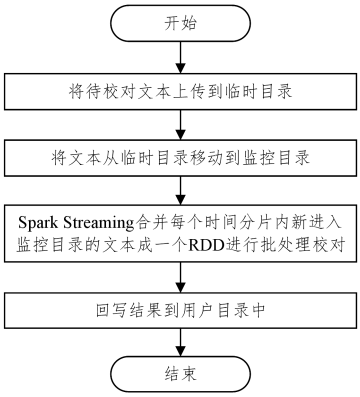

图 3 流式校对

Fig. 3 Streaming proofreading

由于流应用是一个长期运行的进程，因此基于流计算的并行校对方法可以很好地解决固定开销大的问题。另外，由于是在批时间间隔内上传文本到 HDFS，该过程存在的固定开销问题也得到了解决。因此，采用 Spark Streaming 可较好地解决作业启动固定开销大的问题，显著缩短了并行校对的时延。

3.5 流式校对调优

鉴于基于流式计算的并行校对具有低时延和高吞吐的优势，同时还拥有集群计算所赋予的强大算力，本文基于流式校对构建自动校对系统，因此性能优化部分只涉及流式校对的调优。因为 Spark Streaming 是将流计算转化成一系列小规模的基于流的批处理作业，所以流式校对的调优涉及两个方面：1）针对批处理部分的调优；2）针对流应用的调优。为了充分利用计算资源，流式校对部署模式选择 client，即 Driver 进程会在 Master 节点启动，同时校对集群为 Spark standalone 集群，即使用 Spark 自带的集群资源管理器。

3.5.1 批处理部分的调优策略

（1）硬件供给

由于校对集群为专用集群，本文配置并行校对使用计算集群全部 CPU 核心和 2/3 的内存空间。因为校对逻辑存在 collect 算子，这会拉取 RDD 的全部数据到 Driver 上进行处理，为避免内存溢出，本文设置 Driver 的内存为 8000 MB。

（2）内存管理

从 1.6 版本开始，Spark 采用统一内存管理策略，JVM 堆减去保留区域余下的为 Spark 和用户使用内存区域。由于文本校对涉及大规模词库的加载，通过 Web UI 执行器界面观察到作业垃圾回收（Garbage Collection，GC）的时间占比较大，这意味着用户内存区域不够，同时由于校对逻辑不涉及 RDD 持久化和 Shuffle 操作，不存在因 Spark 内存区域过小而溢写磁盘的问题，因此本文设置 Spark 内存区域的占比为 20%。

（3）分区数设置

分区数是 Spark 应用并行的程度。文本校对通过加载 HDFS 上的文本生成 RDD，默认分区数为文件 Block 的数目，不能充分利用集群计算资源。官方建议将分区数设置为集群 CPU 核心总数的 2～3 倍，但在实际使用中发现，若继续增加分区数，文本校对的速度能得到进一步的提升。本文通过分区数调优实验（见 4.2 节），设置分区数为集群 CPU 核心总数的 8 倍。

（4）数据倾斜调优

根据作业运行时 Web UI 显示的信息，可以发现各任务执行的耗时不一致，这会导致各节点不能同时结束任务，而理想情况是各个节点几乎同时结束计算。

造成任务耗时不一致的根本原因是 RDD 分区数据倾斜：由于并行校对是按照段落将文本读入 RDD，即段落与 RDD 的元素一一对应，而文本段落的长度往往很不均匀（例如标题与正文）。本文通过增大分区数使得各个节点更趋近于同时结束计算，因为各分区数据量的减少，使得数据倾斜得到了相对改善。

3.5.2 流应用的调优策略

（1）减少批处理时间

由于 Spark Streaming 各时间分片均为批处理作业，因此调优分区数可以有效缩短批处理时间，提高系统的吞吐量。同时，基于批处理的调优实验结果依然适用，于是按 4.2 节所述方法设置分区数。

（2）设置正确的批处理间隔

正确的批处理间隔应该稍大于批处理时间，这使得流应用可以平稳地处理每个时间分片接收到的数据。若批处理间隔小于批处理时间，则意味着源产生的数据超过了实时校对系统的消费能力，会造成作业的堆积；若批处理间隔设置得过大，又会导致集群资源闲置。为保证校对的实时性，本文设置批处理间隔为 10 s，尽管文本校对的耗时可能大于 10 s，但校对系统的输入文件流密度较为不稳定，即大多数时间片并没有文本进入监控目录，因此这仅会造成一些空作业的堆积。

4 实验

4.1 实验说明

并行校对使用 7 台服务器构建 Spark 集群，其中 1 台作为主节点，余下 6 台作为计算节点。7 台同构服务器的硬件配置如表 1 所列。

表 1 集群机器的硬件配置

Table 1 Hardware configuration of cluster machines

硬件	具体配置
CPU 型号	Intel(R) Xeon(R) CPU E5506
CPU 主频	2.13 GHz
CPU 核数	4
内存容量	12 GB
硬盘容量	550 GB

实验所用机器的软件配置如表 2 所列。

表 2 集群的软件配置

Table 2 Software configuration of cluster machines

软件名称	软件版本
操作系统版本	CentOS 6.4
Java 版本	JDK 1.8.0_111
Scala 版本	Scala-2.11.8
Spark 版本	Spark-2.1.0

4.2 分区数调优

由于 Spark Streaming 本质上是一系列串行的小规模批处理作业，因此本文使用基于批处理的并行校对来选择最优分区数，然后将调优结果应用到流式校对中。

本实验所用测试文本为 Text_1_4－Text_4_4，它们是真实文本的不同截取，例如 Text_1_4 包含真实文本约 1/4 的文字，而 Text_4_4 的规模为 Text_1_4 的 4 倍。本文做分区数调优实验时集群规模尚未拓展，计算节点仅有 3 个，故本节实验所用的 Spark 集群均为 1 个 Master 和 3 个 Worker 节点。

本文先用 Text_3_4 对最佳分区数进行粗略搜索（即每种分区数设置实验只做一次），设置分区数以 12 为步长从 12 递增至 108，以确保分区数为计算集群 CPU 核心总数的整数倍。粗略搜索的实验结果如图 4 所示。

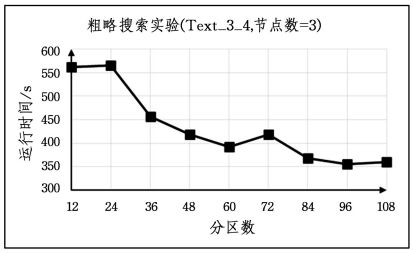

图 4 分区数粗略搜索实验

Fig. 4 Rough search experiment for partition number

从图 4 中可以发现，随着分区数的增大，校对耗时整体呈下降趋势，且在[84,108]区间取得较好结果，但存在些许局部波动。对此，后续又用不同规模的文本进行了更为精细的分区数调优实验。

进一步的分区数实验使用 Text_1_4－Text_4_4 共 4 个文本，它们的规模呈线性增长，设置分区数依然以 12 为步长从 60 增长至 132。由于任务调度存在一定的随机性，因此各测试文本的每种分区数设置实验均重复 3 次，然后取它们校对耗时的均值。精细调优的实验结果如图 5 所示。

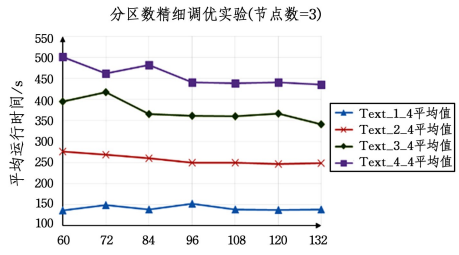

图 5 分区数精细调优实验结果

Fig. 5 Partition number fine tuning experiment results

由图 5 可知，当分区数为 96 时，大部分文本都取得了较好的性能表现。尽管官方推荐将分区数设置为集群 CPU 总核数的 2～3 倍，但由于文本校对中文本段落长度不均，导致各分区数据倾斜，因此继续增大分区数可以进一步缩短校对时间。然而，分区数也不能设置得过大，若段落数小于分区数，则会有一些分区为空（此时有数据的分区都只包含一个段落），从而导致操作系统做一些无意义的调度。综上，设置分区数为集群 CPU 总核数的 8 倍比较合适，于是将基于批处理的调优实验结果拓展到流式校对中，流式并行校对的分区数亦设置为集群 CPU 核心总数的 8 倍。

4.3 线程并行与批处理并行的性能对比

线程并行校对基于单机四核 Linux 服务器，批处理并行校对所用集群规模与 4.1.1 节的描述一致。本实验通过人工生成不同规模的文本 C1－C10，它们是同一段落（该段落包含 153 个字）复制不同次数的结果，各文本段落数从 1 递增至 10。线程并行和批处理并行校对 C1－C10 的运行耗时对比如图 6 所示。

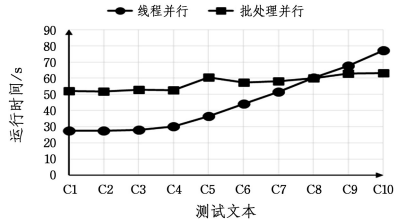

图 6 线程并行与批处理并行的性能对比

Fig. 6 Performance comparison between thread parallel and batch parallel based approach

由图 6 可知，C8 为线程与批处理并行的性能临界点：当文档规模小于 8 时，批处理并行校对由于存在序列化传输 jar 包的固定开销，性能弱于线程并行；当文本包含多于 8 个相同段落时，线程并行由于受到单机计算资源的限制，其性能被批处理并行反超。

4.4 批处理校对和流式校对的耗时对比

本实验所选取的真实文本经过匿名化处理为 T1－T4。匿名文本的规模如表 3 所列。批处理和流式校对处理这些文本的耗时如图 7 所示。

表 3 匿名文本的规模

Table 3 Size of anonymous text

文档名	字符数	段落数
T_1	526	20
T_2	975	54
T_3	37965	450
T_4	43242	786

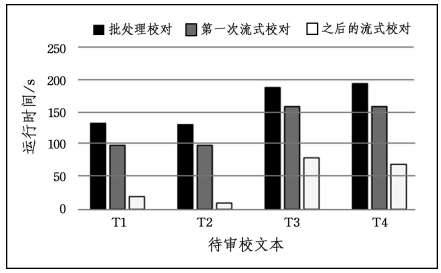

图 7 批处理和流式校对的耗时对比

Fig. 7 Time-consuming comparison experiment on batch processing and streaming proofreading

由图 7 可知，第一次流式校对耗时比批处理校对少了 30 s 左右，这是因为流应用在启动时就会将 jar 包序列化分发到各个节点，于是第一次流式校对时就不再需要做重复操作；另外，之后的流式校对耗时比第一次流式校对少了 80 s 左右，这是因为第一次流式校对过程会加载词库和规则库，加载耗时固定为 50 s 左右，同时加载过程中使用了许多辅助对象，这会导致任务运行时内存不足而频繁触发垃圾回收，观察 Web

UI 执行器页面可知作业 GC 的耗时约为 30 s。

根据实验结果和分析可知，基于 Spark Streaming 流计算框架的流式校对性能取得了极大的提升，相比离线批处理校对作业，实时校对系统有效降低了约 110 s 的固定时延。由于缩减的耗时为固定值，因此文本规模越小，性能提升的幅度就越大，并且随着时间的推移，累计节约的时间也呈线性增长。

结束语 本文针对所定义的串行校对框架提出了 3 种通用的文本校对并行化方法，同时通过实验分析了线程并行和批处理并行的不足之处，最后选用流式并行加速文本校对。未来工作可从如下方面进行进一步的研究：1）由于 Spark Streaming 更为适合处理稳定的输入流，而文本校对的文件流密度不够稳定，因此后续可探索其他的实时计算框架；2）按照段落切分文本会导致 RDD 分区数据倾斜，今后可考虑其他的数据切分方式。

参 考 文 献

[1] DAHLMEIER D，NG H T，NG E J F. NUS at the HOO 2012 Shared Task[C]// Proceedings of the Seventh Workshop on Building Educational Applications Using NLP. 2012：216-224.

[2] ROZOVSKAYA A，CHANG K W，SAMMONS M，et al. The Illinois-Columbia System in the CoNLL-2014 Shared Task[C]// Proceedings of the Eighteenth Conference on Computational Natural Language Learning：Shared Task. 2014：34-42.

[3] ROZOVSKAYA A，ROTH D. Grammatical Error Correction：Machine Translation and Classifiers[C]// Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. 2016：2205-2215.

[4] JUNCZYS-DOWMUNT M，GRUNDKIEWICZ R. Phrase-based machine translation is state-of-the-art for automatic grammatical error correction[J]. arXiv：1605. 06353，2016.

[5] CHOLLAMPATT S，NG H T. Connecting the dots：Towards human-level grammatical error correction[C]// Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications. 2017：327-333.

[6] YUAN Z，BRISCOE T. Grammatical error correction using neural machine translation[C]// Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics：Human Language Technologies. 2016：380-386.

[7] XIE Z，AVATI A，ARIVAZHAGAN N，et al. Neural language correction with character-based attention[J]. arXiv：1603. 09727，2016.

[8] JI J，WANG Q，TOUTANOVA K，et al. A nested attention neural hybrid model for grammatical error correction[J]. arXiv：1707. 02026，2017.

[9] XU L C，SHI L. The Design and Application of A Dynamic Program Algorithm in Automatic Text Collationg[J]. Computer Science，2002，29(9)：149-150.

[10] GONG X J，LUO Z S，LUO W H. Automatically Detecting Syntactic Errors in Chinese Texts[J]. Computer Engineering and Applications，2003，39(8)：98-100.

[11] CHEN X R，QIN J，WANG W J，et al. Research and Implementation of Chinese Text Proofreading[J]. Computer Science，2003，30(11)：53-55.

[12] LIU L L，CAO C G. Chinese Real-word Error Automatic Proofreading Based on Combining of Local Context Features[J]. Computer Science，2016(12)：37-42.

[13] LIU L L，CAO C G. Study of Automatic Proofreading Method for Non-multi-character Word Error in Chinese Text[J]. Computer Science，2016，43(10)：200-205.

[14] ZHANG T. Design and Implementation of Chinese Text Automatic Proofreading System[D]. Chengdu：Southwest Jiaotong University，2017.

[15] ZHANG Y S，ZHENG J. Study of Semantic Error Detecting Method for Chinese Text[J]. Chinese Journal of Computers，2017(4)：911-924.

[16] ZAHARIA M，XIN R S，WENDELL P，et al. Apache spark：a unified engine for big data processing[J]. Communications of the ACM，2016，59(11)：56-65.

[17] 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. 2013：423-438.

[18] NG H T，WU S M，BRISCOE T，et al. The CoNLL-2014 shared task on grammatical error correction[C]// Proceedings of the Eighteenth Conference on Computational Natural Language Learning：Shared Task. 2014：1-14.

[19] NICHOLLS D. The Cambridge Learner Corpus：Error coding and analysis for lexicography and ELT[C]// Proceedings of the Corpus Linguistics 2003 conference. 2003，16：572-581.

[20] LUONG M T，MANNING C D. Achieving open vocabulary neural machine translation with hybrid word-character models[J]. arXiv：1604. 00788，2016.

[21] ZHANG Y S，CAO Y D，YU S W. A Hybrid Model of Combining Rule-based and Statistics-based Approaches for Automatic Detecting Errors in Chinese Text[J]. Journal of Chinese Information Processing，2006，20(4)：1-7，55.

[22] WANG Y. Match Algorithm of Approximate String with Wildcard based on Trie Data Structure[J]. Journal of Computer Applications，2004，24(10)：121-124.

[23] LI J L，YIN C F，JIA Z，et al. Attention-based bidirectional LSTM for Chinese punctuation prediction[C]// Proceedings of the 13th FLINS Conference on Data Science and Knowledge Engineering for Sensing Decision Support. 2018：708-714.

YANG Zong-lin，born in 1994. His main research interests include Chinese text proofreading and parallel computing，etc.

LI Tian-rui，born in 1969，Ph.D，professor，Ph.D supervisor，is an outstanding member of CCF. His main research interests include cloud computing，data mining and artificial intelligence ，etc.