一种基于 Storm 平台的 ETL 方案实现

梁奎奎

(浙江工业大学计算机科学与技术学院 杭州 310023)

摘 要 随着互联网在各个领域的不断发展,数据开始呈现结构多样化与体积海量化。面对海量数据的冲击,如何提高 ETL 的效率至关重要。针对“信息孤岛”中数据来源及格式皆不统一、数据采集实时性差的问题,提出垂直切分 ETL 工作流和水平切分待处理数据集,建立一种基于 Storm 平台的流式 ETL 处理方案。同时,针对 Storm 在进行任务分配时对工作节点 CPU 负载不敏感的缺点,通过定时任务记录工作节点的 CPU 负载信息,对 Storm 调度器的 slot 分配方式进行优化,使得 Storm 集群的负载更加均衡。实验结果证明该方案可有效提高 ETL 的处理效率,同时针对 slot 分配优化可有效地提高系统稳定性与处理效率。

关键词 ETL,垂直切分,水平切分,Storm,负载优化

中图法分类号 TP399 **文献标识码** A

Implementation of ETL Scheme Based on Storm Platform

LIANG Kui-kui

(College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract With the continuous development of the Internet in various fields, data begin to show the characteristics of structural diversity and volumetric quantification. In the face of the impact of massive data, how to improve the efficiency of ETL is crucial. In view of the problem of inconsistent data source and format and poor real-time data collection in “information island”, this paper proposed a vertical segmentation ETL workflow and horizontal segmentation pending data set, and established a flow-based ETL processing scheme based on Storm platform. At the same time, for the shortcomings of Storm, which is insensitive to the CPU load of the working node during task assignment, the CPU load information of the working node is recorded by the timing task to optimize the slot allocation mode of the Storm scheduler, so that the load of the Storm cluster is more balanced. The experimental results show that the scheme can effectively improve the processing efficiency of ETL, and the system stability and processing efficiency for slot allocation optimization.

Keywords ETL, Vertical segmentation, Horizontal segmentation, Storm, Load optimization

1 引言

随着现代科技的快速发展以及互联网的愈加普及,网络中每时每刻都有大量的数据产生。面对海量数据的冲击,数据仓库[1]领域面临着新的问题和挑战。数据仓库的构建过程中 ETL 是决定数据仓库性能优劣的主要因素。如何提高 ETL 操作效率,成为了当下的研究热点。

针对 ETL 优化问题,国内外学者进行了大量的研究。Ali 等[2]基于 ETL 工作流逻辑模型,提出了 ETL 优化的理论框架,但并未给出实际可操作的方案。谢婷婷等[3]基于 CWM 建模优化了对元数据的提取,但是面对海量数据存在一定局限性。文献[4-6]提出基于分片思想通过划分数据与多主机协作提高 ETL 效率,但是这种方式实现难度较大,并且复用性不高。与此同时,为了更好地应对大数据挑战,分布式并行 ETL 技术成为提高 ETL 性能的重要手段。Thomsen 等[7]设计的 ETLMR 方式,基于 MapReduce 模型,提供了一种能支持星型模型、雪花型模型以及渐变维度处理的处理框架,基本实现了大数据量的并行 ETL 处理,但是该框架针对维度处理的实现方式较为复杂,且不易与其他分布式框架集成。文献[8-9]提出基于 Hadoop 平台实现 ETL 处理,实践表明 Hadoop 在处理海量数据时具有很大的性能优势。但是随着 ETL 任务剧增,这种 ETL 操作方案会产生大量的磁盘 I/O,从而造成较大的处理延迟。对此,文献[10-11]提出基于 Spark 框架处理大数据平台的 ETL 问题,其充分利用了内存计算的优势,大大减少了数据处理过程中的 I/O 消耗,从而加快了整个处理流程。上述优化方式大都集中在数据采集完毕之后,通过多节点的计算能力提高 ETL 的批量处理能力,却无法在数据采集的过程中同时进行数据处理。因此,处理结果存在一定滞后性。

针对上述对 ETL 优化方案的讨论,本文首先结合 ETL 工作流模型与 Storm 流式处理模型[12],通过对业务流程的垂直切分和对处理数据的水平切分,提出并实现了基于 Storm 平台的 ETL 处理方案,大大提高了 ETL 操作效率;接着针对 Storm 默认调度器对节点负载不敏感的缺点,提出基于 CPU 负载的优化分配方案,实现对 Storm 集群的资源进行更加均匀的分配,有效提高了系统稳定性与处理效率。

梁奎奎(1993—),男,硕士生,主要研究方向为大数据应用,E-mail:1353701931@qq.com。

2　相关概念

ETL 是构建数据仓库的重要工具,流程包括数据抽取、数据转换、数据清洗、数据装载[13]。ETL 工作流模型将上述步骤统称为 activity,即活动。面对海量数据与实时的处理需求,ETL 性能问题至关重要,尤其是其中的转换(T)过程是一个计算密集型过程,其处理效率影响着整个 ETL 的性能。

Storm 是一个分布式、可靠、高效的数据流处理系统[14-15]。在 Storm 集群中,控制节点 nimbus 通过调度器 scheduler 在集群范围内分发代码,并分配工作节点 supervisor 的槽位 slot,启动 worker 进程,实现运行计算拓扑 topology。计算拓扑 topology 由 spout 和 bolt 组成[16-17]。spout 负责接收数据。bolt 负责处理输入的数据并且输出新的数据。

3　基于 Storm 的流式 ETL

通过前文可以知道,Storm 是一个高性能的流处理框架,但如何实现基于 Storm 平台的 ETL 方案,仍需解决以下问题:1)ETL 工作流是一个串行化的过程,如何分解 ETL 工作流,使得 activity 可以并行化执行;2)ETL 的 4 个流程处理逻辑各不相同,如何提高不同 activity 的并行度,提高整体处理效率。

因此,本文结合 Storm 计算拓扑模型,通过对 ETL 工作流的垂直切分,解决 ETL 并行化的问题,通过对处理数据的水平切分,解决 ETL 并行度的问题,从而实现基于 Storm 平台的高效 ETL 处理方案。

3.1　ETL 工作流垂直划分

传统的 ETL 操作为串行执行,通过活动的前后衔接完成所有操作。任一时刻 ETL 线程中只有一个活动在执行,而其余的活动处于阻塞状态。为了提高 ETL 工作流在数据处理上的效率,本文提出基于功能的 ETL 工作流垂直划分,以抽取(E)、转换(T)和加载(L)为划分边界,将 ETL 工作流分解为功能独立的 activity,并结合 Storm 计算拓扑封将分解后功能独立的 activity 装成 spout 与 blot,通过 spout 与 blot 的组合实现流式的 ETL 处理。ETL 工作流垂直划分流程如图 1 所示。

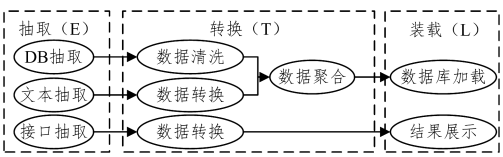

图 1　ETL 工作流垂直切分流程

通过 Storm 提供的流式处理功能,使得 ETL 分解后的 activity 间实现相互独立,此时 ETL 操作不再是串行执行而是并行执行。一个 activity 并不会阻塞另一个 activity 的执行,即数据抽取的运行阻塞并不会影响数据转换与数据加载的顺利运行,反之亦然。不同 activity 之间通过数据通行完成交互。通过一串 activity 的通信实现 ETL 的流处理。

3.2　处理数据的水平划分

传统 ETL 操作为单线程执行,为提高 ETL 的操作效率,本文针对不同的活动使用不同的水平切分方法,以提高 ETL 并行度。

针对 ETL 中数据抽取(E)操作,本文将操作输入流 v 水平分割成相等大小的 n 个水平分片$\{v_1, v_2, \cdots, v_n\}$,使后一个活动的执行不须等前一活动的所有数据完全处理结束,只须前一活动的部分数据分片到达即可开始执行,从而减少各个阶段的空闲等待时间,提高处理效率。

针对 ETL 转换(T)操作中的非聚合操作(如数据的过滤、清洗以及格式转换等),由于单条的数据运行结果并不影响其他数据的运行结果,因此可通过对数据集的进一步分散以增加操作并行度,提高执行效率,在 Storm 中可以通过调整 blot 的运行实例调节并行度。

针对 ETL 转换(T)操作中的聚合操作(如数据的连接、聚合、排序等),由于聚合操作需要等待所有结果集收集完毕才开始执行,因此本文对聚合操作采用节点内预聚合的方式减少节点间通信量,以进一步提高数据执行效率。划分方式如图 2 所示。

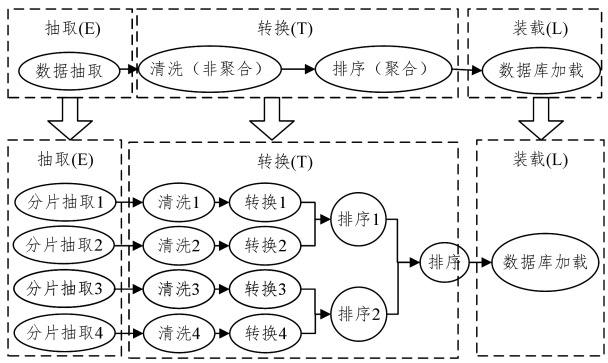

图 2　ETL 工作流水平切分流程

3.3　Topology 运行模型

在完成垂直切分、水平切分后,ETL 任务被拆解成一个个小模块,通过对每一个小模块的拓扑封装,就可以得到 Storm 平台上对应的计算拓扑 topology。当计算拓扑 topology 提交至 Storm 集群之后,控制节点通过调度器 scheduler 分发槽位 slot,并在其上运行 worker 进程实现流式计算。topology 运行模型如图 3 所示:节点集合 N 由 n_1, n_2, n_3, n_4 组成,s_i 与 b_i 分别表示 topology 中不同的运行单元,即 spout 和 blot。w_i 表示运行载体 worker,向量 l_{ij} 表示 w_i 与 w_j 之间的数据通信。以任务 1 为例,起始任务 s_1 代表一个 ETL 工作流的起始,通常的活动为数据抽取。其下游任务分别为 b_1,b_{13},b'_{13}。b_1 和 b_{13} 表示数据处理活动。b'_{13} 代表 ETL 工作流的结束,通常结合装载模块,对计算结果进行展示或持久化存储。

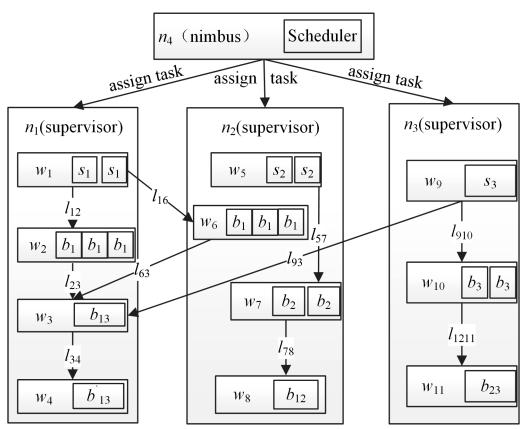

图 3　topology 运行模型图

4 Storm 的调度器优化

4.1 EventScheduler 调度器

在计算拓扑 topology 提交至 Storm 集群后，控制节点通过默认调度器 EventScheduler 按 supervisor-id 对当前集群可用的 slot 资源进行分组排序，并根据排序结果将系统中的 slot 资源均匀分配给 topology 运行 worker 进程[18]。Storm 所采用的这种分配算法虽然使集群中的工作节点拥有几乎相等数目的 worker，但是在一些特定情况下依然有一定的局限性。例如图 4 所示情况：集群由 3 台工作节点 $n1$，$n2$，$n3$ 组成，每台节点配置 3 个槽位，EventScheduler 调度器对可用 slot 进行排序，结果为{[$n1$ $s1$] [$n2$ $s1$] [$n3$ $s1$] [$n1$ $s2$] [$n2$ $s2$] [$n3$ $s2$] [$n1$ $s3$] [$n2$ $s3$] [$n3$ $s3$]}。此时假设提交需要 4 个 slot 的 topology，即 $T1$，根据分配算法，它会占用{[$s1$ $s1$] [$n2$ $s1$] [$n3$ $s1$] [$n1$ $s2$]}这 4 个 slot。接着继续提交下一个 topology，即 $T2$，假设 $T2$ 需要两个 slot 资源，EventScheduler 调度器依然会先对可用 slots 进行排序，排序结果为{[$n1$ $s3$] [$n2$ $s2$] [$n3$ $s2$] [$n2$ $s3$] [$n3$ $s3$]}。根据分配算法，它会占用{[$n1$ $s3$] [$n2$ $s2$]}这两个 slot。此时，节点 $n1$ 的 slot 资源已耗尽，可节点 $n3$ 的 slot 资源却只用了 1/3。

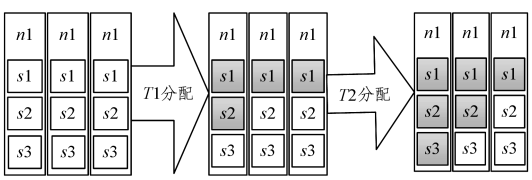

图 4 slot 默认分配图

同时 Storm 以 slot 为分配单位，并没有考虑不同活动的差异，对于数据读取与数据装载操作，大部分执行时间都在等待 I/O 操作完成，对 CPU 消耗很少。但是对于数据清洗与数据转换操作，其对 CPU 的使用率将大大上升。若分配大量数据清洗与数据转换任务到同一工作节点，集群也会发生负载倾斜现象。因此 Storm 的自带调度器并无法很好地满足我们的需求，实现集群资源的合理分配。本文通过 Storm 提供的 IScheduler 接口实现了基于工作节点 CPU 负载的负载调度器 LoadScheduler，使 Storm 集群资源得到更加合理的分配。

4.2 负载调度器 LoadScheduler

负载调度器是在默认调度器的基础上，通过获取节点的 CPU 负载，改变 EventScheduler 对可用 slot 采用的 supervisor-id 分组排序算法，采用 CPU 负载升序排序算法，使得集群资源更合理的分配。负载调度器调度流程如图 5 所示。

步骤 1 从 cluster 对象中获取需要调度的任务，以 topologyDetails 类型表示

步骤 2 判断该任务是否需要通过该调度器进行调度，若已完成调度，则直接跳过。若还未调度，则执行下一步。

步骤 3 获取当前集群可用的 slot 资源，并转换成形式为〈$node$，$slot$〉的可用槽位集合 aS，如式(1)所示。并取得该任务需要使用的 slot 数目 uS。

$$aS=[\langle n1,s1\rangle,\langle n2,s1\rangle,\cdots,\langle nm,sn\rangle] \tag{1}$$

步骤 4 计算该任务执行的总任务数量 tN，如式(2)所示。它由 spout 数目、blot 数目、并发任务数确定，同时取得任务的总线程集合 aE_1，如式(3)所示。

$$tN=spoutNum*parallelism+blotNum*parallelism \tag{2}$$

$$aE_1=[t_1,t_2,\cdots,t_{tN}] \tag{3}$$

步骤 5 通过式(4)与式(5)计算任务集合分配后的总线程集合 aE_2。

$$eS=tN/uS \tag{4}$$

$$aE_2=[\langle t_1,t_{eS}\rangle,\langle t_{eS+1},t_{2eS}\rangle,\cdots,\langle t_{tN-eS+1},t_{tN}\rangle] \tag{5}$$

步骤 6 遍历可用槽位集合 aS，根据工作节点 id、可用槽位数目以及具体的槽位定义自定义实体类 comparableSlots。该类实现了 Comparable 接口，排序方式按照可用槽位数目降序排列。当节点可用槽位数目相同时，按节点 CPU 负载大小升序排列。

步骤 7 将 comparableSlots 集合放到一个优先级队列 slotQueue 中。

步骤 8 循环从总线程集合 aE_2 中获取第一个需要分配的 $task$ 集合并从优先级队列中顺序获取可用 comparableSlots，将任务分配到 comparableSlots 中的工作节点与槽位中。结果以〈[$task_i$，$task_j$]→[$node$，$slot$]〉形式保存到集合 assigntment 中，assigntment 如式(6)所示。

$$assignment=[\{\langle t_1,t_{eS}\rangle\rightarrow\langle n1,s1\rangle\},\cdots,\{\langle t_{tN-eS+1},t_{tN}\rangle\rightarrow\langle ni,sj\rangle\}] \tag{6}$$

步骤 9 若 comparableSlots 中 slot 仍有剩余，则将 comparableSlots 放回队列 slotQueue 中，重新排序。

步骤 10 调用 cluster 的 assign 方法，根据结果集 assignment 进行任务分配。

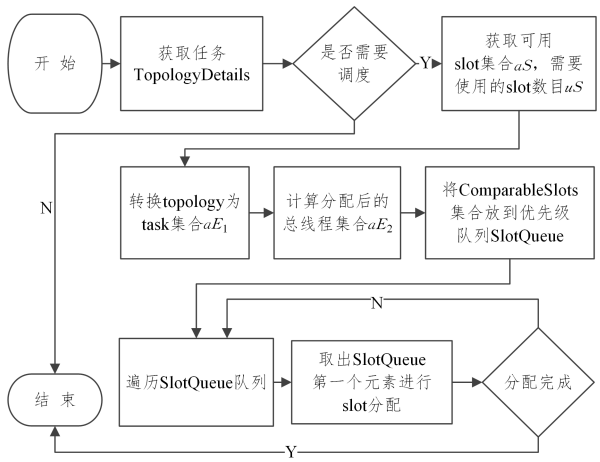

图 5 LoadScheduler 调度流程图

需要指出，由于 Storm 系统中并没有提供获取节点 CPU 负载的接口，因此笔者通过自定义定时任务，读取/proc/stat 文件获取工作节点 CPU 负载信息，并将结果记录至 mysql 数据库，方便负载调度器 LoadScheduler 读取。同时为方便定义集群稳定性，本文使用变量 ε 表示集群稳定度：

$$\varepsilon=\sqrt{\frac{1}{|N|}\sum_{x=1}^{|N|}(\overline{C}-c_x)^2} \tag{7}$$

其中，$\overline{C}$ 表示集群平均 CPU 负载；c_x 表示工作节点 n_x 的 CPU 负载；x 表示工作节点变量；$|N|$ 表示工作节点数量。

5 实验及评估

5.1 实验环境

实验环境物理主机采用 intel(r) core(tm) i7-6700HQ

CPU @2.60 Hz 2.60 Hz 处理器，8 GB 内存，Windows10 操作系统。采用虚拟机 VMware Workstation 12.5.2 搭建 4 台主机的分布式实验环境，4 台虚拟主机均配置 1 核 CPU、1GB 内存和 10G 硬盘，并安装 centos6.4，java1.7，storm0.9.2，zookeeper3.4 等软件。4 台主机分配如下：主机 m 运行 nimbus 进程作为控制节点，主机 $s1$，$s2$，$s3$ 运行 supervisor 进程作为工作节点，由于资源限制 zookeeper 服务采用单节点形式运行于控制节点。每台工作主机配置 4 个 slot 端口，提供 topology 运行。

本文选择某大学 2014 年至 2018 年图书馆数据作为实验数据，其中借阅记录表 z_book_lend_sys 共 1915544 条记录、还书记录表 z_book_back_sys 共 1882836 条记录、进馆记录表 z_gctrl_ctrl_sys 共 3645518 条记录、打印机使用记录表 z_printcs_sys 共 1965175 条记录。总记录数 9409073 条。表结构如图 6 所示。

z_book_lend_sys	Id	account	.....	book_name	book_no
	1	200920201093		机器人工程	10851163
z_gctrl_ctrl_sys	Id	account	.....	enter_time	gender
	1	200920201093		2017/09/02 17:56:42	M
z_book_back_sys	Id	account	.....	academy	gender
	1	200920201093		法政学院	F
z_printcs_sys	Id	account	.....	print_time	location
	1	200920201093		2017/05/08 10:47:43	IC空间文印1

图 6　数据示例图

5.2　实验方案

本文实验数据集均放置于内存数据库 redis，模拟实际生产环境下消息队列的作用。

实验方案一通过本文流处理与串行处理统计历年每月各学院图书馆进馆人次、借阅人次、还书人次、打印人次。对于流处理，通过对数据集水平分片，spout 分批次从 redis 中读取 10000 条数据，之后依次经过异常值过滤、字段转换与结果统计、排序等操作，最后将结果保存至 mysql 数据库。对于串行处理，一次性读取所有数据后，对数据集同样依次进过异常值过滤、字段转换和结果统计、排序等操作。通过比较对不同数据集的两种实验方案所消耗的时间差异，得到实验结论。

实验方案二采用默认调度器的 ETL 操作（方案一）与基于负载调度器的 ETL 操作（方案二）作为实验对比方案，选取上述 4 组不同 topology 计算拓扑进行 ETL 操作。为增加实验效果，本文在数据清洗、转换的 blot 中增加了不同数量的空循环来增加 CPU 消耗。通过比较集群负载稳定度与数据处理效率得出实验结论。

同时，为排除实验的偶发性，所有实验均进行 5 次，以平均实验结果作为最终结果。

5.3　实验结果

实验 1　流式处理与串行处理的对比

由图 7 可知，面对不同的数据集合，基于流式处理的 ETL 操作相比于传统串行处理在处理性能上都大幅度的提升。实验结果证明：通过对 ETL 工作流垂直切分与处理数据水平切分，基于 Storm 平台的 ETL 流处理拥有更好的处理效率。

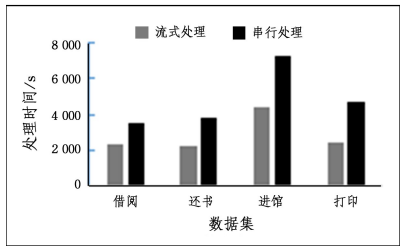

图 7　流式处理与串行处理的对比

实验 2　默认调度器与负载调度器对比

由图 8 可知，由于 ETL 活动差异，未优化的 Storm 集群中工作节点 CPU 负载差异较大，节点 $n1$ 的 CPU 负载为 0.59，节点 $s2$ 的 CPU 负载为 0.41，$s3$ 的 CPU 负载为 0.44，集群稳定度 ε 为 0.0787。与之相反，优化后的 Storm 集群运行期间，节点的 CPU 负载更为接近，节点 $n1$ 的 CPU 负载为 0.49，节点 $s2$ 的 CPU 负载为 0.46，$s3$ 的 CPU 负载为 0.50，集群稳定度 ε 为 0.0173。同时通过对 Storm 自带的 UI 界面观察，实验方案一每条消息的处理延时为 1.30 ms，实验方案二的处理延时为 1.22 ms，经过优化后处理效率提高了 6% 左右。实验结果可以证明，经过优化调度后集群状态相对更加稳定，且 ETL 的处理效率更高。

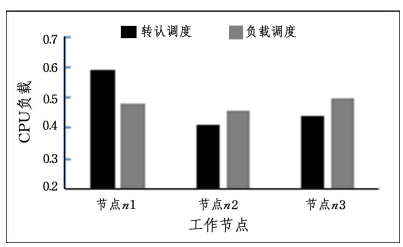

图 8　集群负载情况对比图

结束语　本文通过对 ETL 工作流垂直切分与对处理数据集水平切分，基于 Storm 平台提出流式 ETL 处理方案，有效提高了 ETL 处理效率。针对流处理环节，本文优化了 Storm 的调度模块，通过记录各工作节点的负载情况，实现对 slot 的分配优化。实验证明，优化后的集群处理效率更高，各节点的负载也更加均匀，集群也更加稳定。本文主要利用 CPU 负载作为节点负载，后续工作可以利用更多的指标来评判节点负载，进一步优化调度的性能。

参 考 文 献

[1]　徐俊刚，裴莹. 数据 ETL 研究综述[J]. 计算机科学，2011，38(4)：15-20.

[2]　ALI S M F，WREMBEL R. From conceptual design to performance optimization of ETL workflows：current state of research and open problems[J]. Vldb Journal，2017，26(6)：1-25.

[3]　谢婷婷，李伟华. 专用 ETL 模式设计与实现[J]. 计算机工程与应用，2010，46(35)：133-135.

[4]　CHEN G，AN B，LIU Y. A novel agent-based parallel ETL system for massive data[C]// Control and Decision Conference. IEEE，2016：3942-3948.

（下转第 240 页）

(5)第二次边界增强更注重的是一些边缘轮廓的优化和对比度增加,进一步让分割图像更接近真实值。图 10 展示了图像损失曲线在测试中的变化。

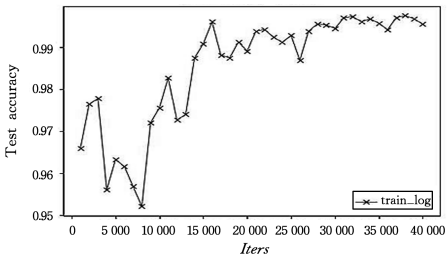

图 10 图像损失曲线

7 结果和评价

根据不同方法的实验,最终本文选用了基于 segnet 的超声图像分割,基于多次边界增强的自适应对比度增强算法(ACE)进行图像预处理,并对原始数据进行数据扩增,使得臂丛神经的超声图像精确度更高,图像损失越少,越可以更精准地被识别和分析。

结束语 本文实验通过对基于深度学习的臂丛神经超声图的对比度增强,得到了以下结论:1)利用基于 Segnet 模型的卷积神经网络,根据情况分类臂丛神经超声影像,可以更直观且简单地看到患者神经的特征;2)利用增强对比度算法处理医学超声图像,可以提高卷积神经网络对臂丛神经判断的准确度;3)其中基于自适应低通滤波的边界增强算法能高效准确地提升图像对比度,有效地提升医师识别的准确率和效率。

参 考 文 献

[1] 金烈烈. 臂丛神经的超声影像学研究[J]. 中华手外科杂志,2007,23(4):248-251.

[2] 吴道珠. 高频超声对臂丛神经显像和定位的价值[J]. 中华超声影像学杂志,2006,15(6):449-452.

[3] CASATI A,DANELLI G,BACIARELLO M,et al. A prospective randomized comparison between ultrasound and nerve stimulation guidance for multiple injection axillary brachial eels block[J]. Anesthesiology,2007,106:992-996.

[4] 劳杰,何继银,顾玉东,等. 创伤性臂丛神经损伤合并肩袖及肱二头肌长头腱损伤的超声诊断[J]. 中华手外科杂志,2006,22:l44-l45.

[5] 曾炜,王弘士,朱世亮,等. 颈部颈丛和臂丛神经鞘瘤的彩色多普勒超声定位诊断[J]. 中国超声医学杂志,2000,16:819-822.

[6] 李丽玮. 基于自适应对比度增强和深度 CNN 的脂肪肝 B 超影像诊断[J]. 计算机应用,2018(S2).

[7] BENGIO Y S. Learning Deep Architectures for AI[J]. Machine Learning,2009,2(1):1-127.

[8] 王振,高茂庭. 基于卷积神经网络的图像识别算法设计与实现[D]. 上海:上海海事大学,2015.

[9] 柴睿,林岳,杨桐. 基于 SegNet 模型的臂丛神经超声图像分割[J]. 计算机工程,2017.

[10] 李彦冬,郝宗波,雷航. 卷积神经网络研究综述[J]. 计算机应用,2016,36(9):2508-2515,2565.

[11] 朱其刚,朱志强. 基于自适应邻域灰度直方图均衡的超声内窥镜图像增强[J]. 山东科技大学学报(自然科学版),2004,23(3).

[12] 张志龙,李吉成,沈振康. 一种保持图像细节的直方图均衡新算法[J]. 计算机工程与科学,2006,28(5):36239.

[13] 上官伟,李金. 综合运用灰度变换方法改善超声医学图像质量[J]. 应用科技,2005,32(11):40243.

[14] 刘丽娜. 超声定量技术诊断弥漫性脂肪肝的研究[D]. 广州:南方医科大学,2014.

[15] 康伟. 脂肪肝医学超声图像定量分析的研究[D]. 大连:大连理工大学,2007.

(上接第 211 页)

[5] 赵俊,夏小玲. 公共数据中心的 ETL 系统设计与实现[J]. 计算机应用与软件,2011,28(10):167-169.

[6] 宋杰,郝文宁,陈刚,等. 基于 mapreduce 的分布式 etl 体系结构研究[J]. 计算机科学,2013,40(6):152-154.

[7] LIU X,THOMSEN C,PEDERSEN T B. ETLMR:A Highly Scalable Dimensional ETL Framework Based on MapReduce[C]// International Conference on Data Warehousing & Knowledge Discovery. Springer,Berlin,Heidelberg,2011.

[8] 汪保友,钱晶,袁时金. 基于 Hadoop 的电信大数据采集方案研究与实现[J]. 电信科学,2017(1):135-142.

[9] 孙莉,何刚,李继云. 基于 Hadoop 平台的事实并行处理算法[J]. 计算机工程,2014,40(3):59-62,81.

[10] 丁祥武,解书亮,李继云. 基于 Spark 的并行 ETL[J]. 计算机工程与设计,2017,38(9):2580-2585.

[11] 曲朝阳,陈贺新,胡可为,等. 基于 Spark 的电力调度数据整合模型[J]. 计算机工程与应用,2017,53(19):65-70.

[12] LEI C,RUNDENSTEINER E A,GUTTMAN J D. Robust distributed stream processing[J]. Computer Science Faculty Publications,2012:817-828.

[13] 李庆阳,彭宏. 面向数据质量的 ETL 框架的设计与实现[J]. 计算机工程与设计,2010,31(9):2057-2060.

[14] 季一木,张永潘,郎贤波,等. 面向流数据的决策树分类算法并行化[J]. 计算机研究与发展,2017,54(9):1945-1957.

[15] YUAN S G,ZHU Y L,ZHOU G L,et al. Research on Dynamic Scheduling of Grid Monitoring Data Processing Tasks Based Storm[J]. Applied Mechanics & Materials,2014:1051-1055.

[16] 单莘,祝智岗,张龙,等. 基于流处理技术的云计算平台监控方案的设计与实现[J]. 计算机应用与软件,2016,33(4):88-90.

[17] 王春凯,孟小峰. 分布式数据流关系查询技术研究[J]. 计算机学报,2016(1):80-96.

[18] 蒋溢,罗宇豪,朱恒伟. Storm 集群下一种基于 Topology 的任务调度策略[J]. 计算机工程与应用,2018,54(7):84-88.