流数据处理中负载突发感知的弹性资源分配

李丽娜[1] 魏晓辉[1),2)] 李 翔[3)] 王兴旺[1)]

[1)](吉林大学计算机科学与技术学院 长春 130012)

[2)](吉林大学符号计算与知识工程教育部重点实验室 长春 130012)

[3)](吉林大学高性能计算中心 长春 130012)

摘 要 在分布式并行数据流处理中,面向实时变化且具有突发性的流数据负载,固定的资源分配将造成资源浪费或服务质量降低,因此,可伸缩的弹性资源分配是一个亟待解决的关键问题.然而,由于资源分配延迟和负载预测模型存在误差,已有的弹性资源分配策略无法准确地提供与突发负载相匹配的资源,且存在不必要的资源调整反复,增加了系统开销.该文主要解决弹性资源分配的调整延迟和调整颠簸问题.针对上述问题,主要的挑战在于突发负载的准确预测和节点间的协作.为此,该文提出了一个上、下游节点协同的弹性资源分配策略,最优化数据质量和资源使用率,兼顾考虑调整代价.在该策略中,基于数据负载关联模型和双向的控制机制,下游节点能够实时感知和预测上游节点产生的突发负载和负载的变化趋势,预先调整资源并避免调整颠簸;同时,上游节点能够基于反馈机制,动态调节数据处理速率以抑制下游节点的负载波动,降低其资源调整的可能性.实验结果表明,当负载变化较大时,该策略平均减少数据丢失达 85%,并显著降低了系统资源调整开销,同时,提高了资源使用率.

关键词 流数据;流数据处理;突发感知;资源分配;弹性调整

中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2018.02193

Burstiness-Aware Elastic Resource Allocation in Stream Data Processing

LI Li-Na[1)] WEI Xiao-Hui[1),2)] LI Xiang[3)] WANG Xing-Wang[1)]

[1)](*College of Computer Science & Technology, Jilin University, Changchun* 130012)

[2)](*Symbol Computation and Knowledge Engineer of Ministry of Education, Jilin University, Changchun* 130012)

[3)](*High Performance Computing Center, Jilin University, Changchun* 130012)

Abstract In distributed parallel data stream processing, facing the real-time-changing and bursting stream data, fixed resource allocation will cause waste of resources or reduce the quality of service. To achieve desirable performances without resource wastes, the scalable and elastic resource allocation is a critical problem to be solved, which allows applications to change dynamically their resource configuration in response to data load fluctuations at run-time. However, most elastic resource allocation policies only adjust their resources when the performance of nodes does not match the data load. The adjustment delay caused by the immediate resource allocation, and the unpredictable data load reduce the performance of the elastic allocation policy. In some work, the data load prediction is introduced into the elastic resource allocation, but in the data prediction model, the future data is predicted based on the historical data, which is not applicable to the bursty stream data. Moreover, the unnecessary resource adjustment bump increases the system overhead. This paper focuses on the adjustment delay and the adjustment jump in the elastic resource allocation. For the above problems, the main challenge lies in the prediction of burst

收稿日期:2016-07-18;在线出版日期:2017-05-16.本课题得到国家自然科学基金(61170004)和吉林省科技厅项目(20140204013GX)资助.**李丽娜**,女,1978 年生,博士研究生,副教授,中国计算机学会(CCF)会员,主要研究方向为流数据处理、无线传感网.E-mail:linal11@mails.jlu.edu.cn.**魏晓辉**,男,1972 年生,博士,教授,中国计算机学会(CCF)杰出会员,主要研究领域为分布式系统等.**李 翔**,男,1988 年生,硕士,中国计算机学会(CCF)会员,主要研究方向为流数据处理、分布式系统.**王兴旺**(通信作者),男,1990 年生,博士研究生,中国计算机学会(CCF)会员,主要研究方向为云计算、移动计算.E-mail:wangxw2109@mails.jlu.edu.cn.

load and the cooperation among nodes. Therefore, this paper firstly constructs a data load correlation model to predict the data load of nodes accurately. This model considers the correlation between nodes of the stream application, that is the node will experience the sudden load change, once its upstream nodes have carried out resource adjustments, then uses the queuing theory to predict the data load of the node in the next window, according to the states of its neighborhood upstream nodes in the current window, which includes the data load, the buffer occupy, the processing ability and the consumption ratio. Furthermore, a bi-directional control mechanism is designed to support the cooperative resource allocation between the upstream and downstream nodes, in which the feed forward control transmits the information of upstream nodes to downstream nodes, and the feedback control transmits the information inversely and selects the appropriate upstream nodes to participate in the resource allocation of the downstream nodes. Based on the data load correlation model and the bi-directional control mechanism, this paper proposes an upstream-downstream cooperative and elastic resource allocation strategy, optimizing for data quality and resource utilization rate quality of service (QoS) requirements and considering the adjustment cost. In this strategy, the downstream node can timely detect the sudden load and load variation trend produced by the upstream nodes, and thus the resource adjustment will be done in advance, and the adjustment bump can also be avoided. At the same time, according to the feedback mechanism, the upstream nodes can dynamically adjust the data processing rate to suppress the load fluctuation of the downstream node, aimed at reducing the possibility of resource adjustment. Experimental results show that, the strategy is effective, compared with two benchmark strategies used in similar scenarios. When the load change is large, the strategy can reduce the average data loss by 85%, and significantly reduce the overhead of system resource adjustment, and improves the resource utilization.

Keywords stream data; stream data processing; burstiness-aware; resource allocation; elastic adjustment

1 引 言

随着社交媒体、智能交通等互联网和物联网应用[1-3]的发展,流式大数据呈现急剧增长的态势,引发了关于流式计算的新一轮研究热潮.由于流数据负载具有时变和突发[4]的特点,要求系统能够动态地适应数据流的变化,并保证不丢或少丢数据.然而,现有的分布式流处理系统以静态资源配置居多,例如雅虎的 S4[5] 和 Twitter 的 Storm①,无法根据负载的变化动态地增加或减少资源,既影响了服务质量,又造成了资源浪费.因此,可伸缩性资源调整(也称"弹性资源分配",包括扩展调整和收缩调整)成为亟待解决的问题.

分布式并行流处理系统基于数据并行模型,每个操作对应多个并行单元,分别处理输入负载的不同部分.并行单元的实现主要有三种方式:单节点上的多线程并行、集群中的多虚拟机和多节点并行.无论采用哪种方式,上游操作的资源调整都将影响与其关联的下游操作.因此,可伸缩的分布式并行流处理系统将面临着两个挑战:(1)快速地应对关联调整.减少下游关联操作的调整延迟,使其尽量与上游操作实现同步调整;(2)有效地避免调整颠簸.对于短时间的突发数据负载,在不影响操作性能的前提下,需要尽量避免资源的颠簸调整.

目前,大部分的弹性资源分配策略主要以"延时调整"策略为主,即当操作性能与输入负载出现不匹配时才进行资源调整,因此,不可避免地面临着"调整滞后"问题和"调整颠簸"问题.为此,文献[6-8]进行了关于"预先调整"策略的研究工作,即在负载波动尚未到达操作之前,通过对输入负载的预测启动资源调整,但相关工作较少考虑对调整颠簸的控制.本文基于上、下游操作的数据负载关联性建立负载预测模型,并设计了双向控制机制支持操作间的

① http://storm.apache.org

负载信息传递和资源协同分配.

本文的主要贡献包括以下三个方面：(1) 定义了数据负载关联模型. 作为预先调整的实现前提，该模型根据邻居上游操作在第 n 个窗口的状态，预测操作在后续第 $n+1$ 窗口的输入负载；(2) 设计了双向的控制机制. 该机制由反馈控制和前馈控制组成，位于相邻操作之间，前者触发上游操作的反馈调整，后者支持下游操作的预先调整；(3) 提出了协同的资源分配策略. 在该策略中，预先感知突发性的负载变化，并及时地调整至最优的资源分配. 同时，根据对输入负载变化趋势的预测和反馈调整机制，避免不必要的调整颠簸.

本文第 2 节是相关工作介绍；第 3 节描述弹性资源调整中存在的阶跃现象和抖动现象；在第 4 节中，定义操作间的数据负载关联模型；第 5 节描述双向的控制机制；第 6 节提出协同的弹性资源分配策略；第 7 节是实验结果和性能分析；第 8 节是结论和展望.

2　相关工作

目前，国际上针对流式大数据的研究不断扩展和深化，流数据负载的实时预测和弹性资源分配方面的研究日益得到人们的关注. 在国内，以中国科学院院士李国杰为首的科研人员也认识到大数据的重要性[9-10]，关于流式计算的研究工作逐渐展开并取得了一些研究成果[11-14].

在弹性资源分配的研究中，大部分工作提出“延时调整”的反应策略，资源调整在波动的输入负载到达后进行，导致资源与负载出现滞后匹配. 在文献[15-20]中，使用基于 CPU 利用率阈值的规则增加或减少计算资源. 此外，一些工作也考虑吞吐量和延迟等性能量度. 文献[16]兼顾考虑了操作的吞吐量，并以此触发调整. 文献[21]监测拥塞索引和当前并行度对应的吞吐量，并基于历史参考值来避免调整反复. 文献[19]通过委托机制提高吞吐量，同时，采用后压机制限制端到端延迟上限. 文献[20]建立延迟模型估计操作移动的延迟，并研究如何在状态迁移期间最小化延迟突发. 文献[22]采用排队论模型化操作的资源分配与处理延迟之间的关系，并以 Jackson 网络理论估计应用的延迟. 文献[23]提出一个基于排队论的延迟模型，由服务时间和到达间隔时间增强延迟限制. 文献[24]则由节点级负载和全局负载触发资源调整. 文献[25]基于输入队列的占用确定最优的任务调整数量. 与上述工作不同，本文考虑资源使用率和资源调整代价，并以输入缓存作为监控对象，基于稳定排队论增减资源. 同时，在上述策略中，操作调整是独立的，并未考虑上、下游操作间的协作，且除了文献[21]之外对调整颠簸问题也鲜少提及. 因此，本文主要解决操作间协作调整和调整颠簸的避免问题.

与“延时调整”策略的调整时机不同，“预先调整”策略基于负载预测提前进行资源调整，能够减少调整延迟引起的数据丢失. 文献[6]以满足输入缓存的限制为目标，采用时间序列分析方法预测操作的输入数据，并基于排队论计算操作的并行度. 文献[7]采用高斯方法预测引擎(操作)在未来若干窗口的负载和处理延迟，通过最小化代价函数确定增加或减少的资源. 在文献[8]中，面向单节点多核系统，采用控制论中的“模型预测控制”预测未来的系统行为，并最小化调整中的延迟突发. 文献[6]中的预测模型是基于历史数据的，存在一定程度的预测误差，本文提出了基于当前数据的预测模型，对预测精度进行了改进. 此外，本文将在后续工作中考虑对首节点应用高斯预测方法.

与单机或集群环境不同，云中的弹性资源分配需要同时考虑资源代价和用户的特性. 文献[26]考虑资源性能的波动性，采用启发式的资源调整方法，以最小的资源代价维持应用的吞吐量. 文献[27]提出一种预测流负载和性能的方法，自适应地计划资源分配，在满足吞吐量的同时限制资源调整代价. 针对 MapReduce 流式框架的弹性研究也有一些成果，例如，文献[25]实现了任务级(非节点级)的伸缩调整，并给出最优的任务调整数量；文献[28]提出一种整合的流式 MapReduce 体系结构，并采用一致性哈希方法支持有状态操作的弹性调整. 此外，在弹性资源分配中，也涉及有状态操作的迁移、负载均衡和容错等方面的问题. 文献[15,19]通过操作迁移的方式分配资源，文献[21]涉及有状态操作的状态迁移. 文献[29]通过有状态操作的状态分片支持操作并行，并采用轻量级的事务迁移协议平衡节点间的负载，同时支持透明容错. 本文暂未考虑有状态操作的迁移问题. 文献[28]通过局部感知数据和状态副本提供有效的负载均衡和低开销的容错. 文献[18]和文献[25]也分别实现了操作间和任务间的负载均衡. 特别地，文献[18]从减少操作间数据分发开销的角度，实现了操作分组并行化. 文献[17,24]整合了弹性调整和容错.

3 问题描述

在分布式并行数据流处理中，应用通常描述为有向无环图 DAG(Directed Acyclic Graph)的形式. 在本文中，DAG 的定义如下：DAG=$\{\{DS_i\},\{U_i\},\{O_i\},\{S_i\}\}$，其中，$DS_i$表示数据源，$U_i$表示用户，$O_i$是操作，$S_i$是操作间的数据流. 此外，每个操作包括若干个并行单元 PU_i，且并行单元间共享操作的输入缓存 IB_i，如图 1 所示. DAG 具有多种分类形式：单源或多源、单用户或多用户、线性或非线性. 以图 1 为例，整个 DAG 是多源多用户的非线性拓扑结构，而由 DS_1、U_1、$O_1\sim O_5$ 和 $S_1\sim S_5$ 组成的 DAG 则属于单源单用户的线性结构.

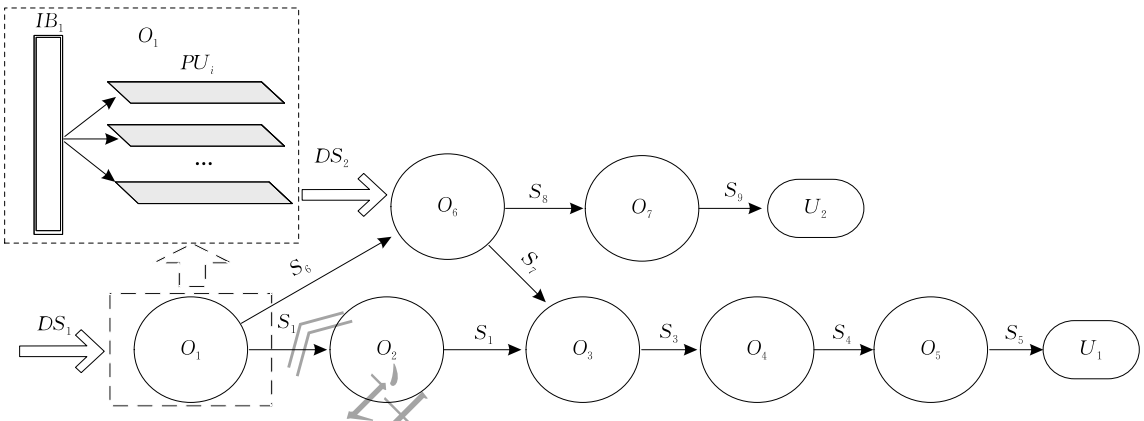


图 1 应用的 DAG

3.1 阶跃现象

在 DAG 中，操作的资源扩展调整将导致其邻居下游操作的输入负载突然增加，一旦负载超过了处理能力，下游操作也需要进行扩展调整，成为新的调整操作. 此时，下游操作称为操作的扩展关联操作. 由于调整延迟是不可避免的，在处理能力与输入负载尚未匹配之前，如果下游操作发生缓存溢出，出现数据丢失，此种现象被称为操作的阶跃上升现象(简称阶跃现象)，如图 2(a)所示，其中，左侧的细线纵坐标轴 $I(t)$和右侧的粗线纵坐标轴 $P(t)$分别表示操作在 t 时刻的输入速率和处理速率.

与扩展调整相反，操作的资源收缩调整将导致其下游操作的输入速率急剧减小. 当下游操作的输入缓存占用较少且资源使用率较低时，也需要进行收缩调整，此时，下游操作称为操作的收缩关联操作. 在下游操作调整期间，其处理能力大于突发减少的输入负载，存在资源使用浪费的现象，称之为操作的阶跃下降现象(简称阶跃现象)，如图 2(c)所示.

扩展关联操作和收缩关联操作统称为关联操作. 对于关联操作，如果能够在输入数据到达前完成与之匹配的弹性资源分配，则将避免阶跃现象，从而减少数据丢失或提高资源使用率，理想的情况如图 2(b)和(d)所示.

3.2 抖动现象

在本文中，假设数据流离散为连续的固定时间 Δt 的窗口，并以窗口作为基本的处理单元. 如果突发数据的持续时间很短，例如 Δt，则可能出现调整颠簸. 调整颠簸将导致资源分配出现短暂的反复，增加了不必要的调整开销，称此现象为操作的抖动现象，如图 2(e)所示.

当调整颠簸对操作的性能没有影响或者影响较小时，则可以考虑取消调整以减少调整代价，期待的结果如图 2(f)所示.

针对上述阶跃现象和抖动现象，本文解决的弹性资源分配问题的目标是：最小化数据丢失量和最大化资源使用率，并在满足上述两个目标的前提下，最小化调整代价. 目标函数 Φ 的定义如下：

$$\Phi=\min(AC(o_i)\,|\,(RS(o_i),DL(o_i)))\wedge \max(RS(o_i)\,|\,DL(o_i))\wedge\min(DL(o_i)) \quad (1)$$

在函数 Φ 中，$DL(o_i)$表示 O_i在调整期间的数据丢失量. $RS(o_i)$是资源使用率，即实际使用资源与分配资源的比率，最优的情况是 $RS(o_i)=1$. $AC(o_i)$标识 O_i的调整代价，本文假设扩展或收缩资源调整的代价基本相同，因此，总的调整代价由调整次数衡量，调整次数越多，调整代价越高. 后续工作将建立调整代价模型，考虑不同因素影响下的调整代价.

为了实现上述目标，首先需要构建操作的数据负载预测模型，并以此为基础进行预先的弹性资源分配，减少或避免数据丢失；同时，为了提高资源使用率和减少调整代价，需要支持操作间协同调整的

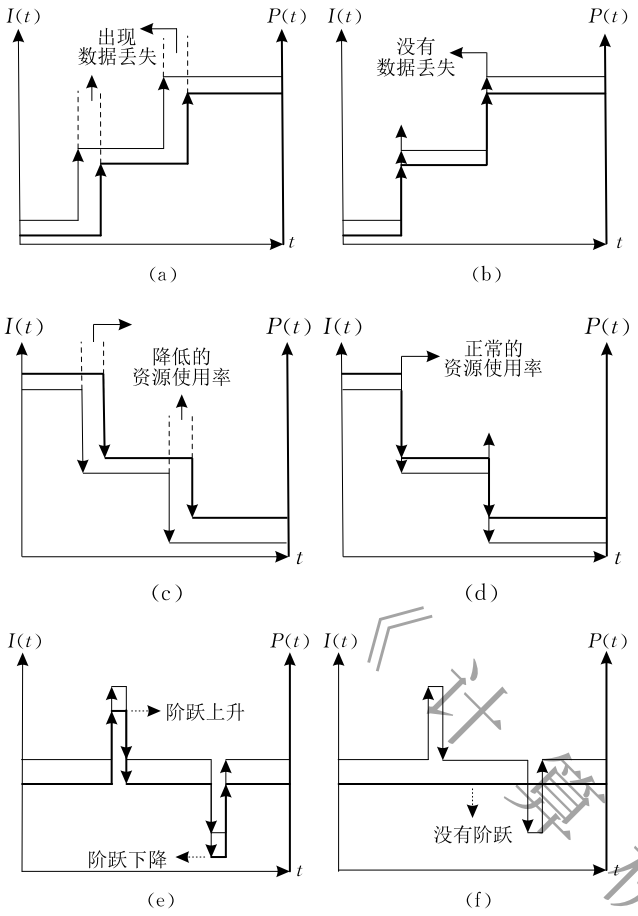


图 2　关联操作的状态对比图

机制和有效的弹性资源分配策略. 在表 1 中，给出了本文涉及的模型、机制和算法中的主要符号和参数说明.

表 1　主要符号和参数列表

符号	描述
Δt	固定时间的时间窗口，数据流的基本处理单元
$Io_i(n)$	操作 O_i 在窗口 n 的平均输入速率
$Oo_i(n)$	操作 O_i 在窗口 n 的平均输出速率
P_{unit}	单个并行单元的处理速率
$PNo_i(n)$	操作 O_i 在窗口 n 的并行单元数量
$Po_i(n)$	操作 O_i 在窗口 n 的平均处理速率，$Po_i(n)=P_{unit}\times PNo_i(n)$
BSo_i	操作 O_i 的输入缓存大小
$BOo_i(n)$	操作 O_i 在窗口 n 的输入缓存占用
$RBOo_i(n)$	操作 O_i 在窗口 n 的缓存状态，$RBOo_i(n)=BOo_i(n)/BSo_i$

4　数据负载关联模型

根据稳定排队论，在窗口 n 中，O_i 的平均输出速率（以下简称输出速率）$Oo_i(n)$ 与平均输入速率（以下简称输入速率）$Io_i(n)$ 和处理速率 $Po_i(n)$ 相关. 当输入速率大于或者等于处理速率时，输出速率由处理速率决定；否则，输出速率由输入速率和输入缓存占用共同决定，但不超过处理速率. 如果 $F(t)$ 表示 O_i 的输入数据函数，则在时间 t，O_i 的输入速率 $Io_i(t)=F'(t)$. 当窗口 Δt 足够小时，$Io_i(t)$ 近似等于 $Io_i(n)$. $Bo_i(n-1)$ 表示 O_i 在窗口 $n-1$ 中的输入缓存占用 $BOo_i(n)$ 的平均变化速率，$Bo_i(n-1)=BOo_i(n)/\Delta t$，则 $Oo_i(n)$ 能够通过传递函数 f 近似计算获得.

$$f(Io_i(n),Po_i(n))=\begin{cases}CR\times(\varpi_1\times e\times Po_i(n)+\varpi_2\times(Io_i(n)+Bo_i(n-1))), & Io_i(n)<e\times Po_i(n)\\ CR\times e\times Po_i(n), & Io_i(n)\geq e\times Po_i(n)\end{cases}\quad(2)$$

在函数 f 中，CR 是 O_i 的消费比，表示输出数据量与有效输入数据量（即处理数据量）的比值；e 是效率系数，表示处理能力的可用性，e 属于 $(0,1]$；ϖ_1 和 ϖ_2 是关联系数，分别表示输出速率与处理速率、输入速率和输入缓存的关系，取值为 0 或 1，当 $\varpi_1=0$，$\varpi_2=1$ 时，表示输出速率由输入速率和输入缓存变化速率决定；反之，则由处理速率决定.

4.1　数据负载关联函数

在 DAG 中，数据自上游流向下游，上游操作的输出负载将在后续的窗口到达下游操作，并且与下游操作的输入负载之间具有关联性. 首先，考虑邻居操作之间的数据关联性. 如果已知 O_i 的上游操作 Uo_i 在窗口 n 的输入速率，则通过式(3)可以预测 O_i 在窗口 $n+1$ 的输入速率 $Io_i(n+1)$. 其中，b 是 Uo_i 与 O_i 间的带宽影响系数，即 O_i 的输入速率与 Uo_i 的输出速率的比值，且每对 (Uo_i,O_i) 的 b 值可以不同. 当 $b=1$ 时，表示带宽够用，对 O_i 的输入速率没有影响；b 值越小，则带宽越不充足，对 O_i 的输入速率影响越大.

$$Io_i(n+1)=\sum b\times f(I_{Uo_i}(n),P_{Uo_i}(n))\quad(3)$$

依此类推，对于任意具有上、下游关系的操作 O_j 和 O_i，如果已知操作 O_j 在窗口 n 的输入速率 $Io_j(n)$ 和 O_i 的所有上游操作 Uo_i 的处理速率 $P_{Uo_i}(n)$，那么，如式(4)所示，能够递归地预测 O_i 在窗口 $n+k$ 的输入速率 $Io_i(n+k)$，k 是从 O_j 到 O_i 的路径长度.

$$Io_i(n+k)=\begin{cases}\sum b\times f(I_{Uo_i}(n),P_{Uo_i}(n)), & k=1\\ \sum b\times f(I_{Uo_i}(n+(k-1)),\\ \quad P_{Uo_i}(n+(k-1))), & k>1\end{cases}\quad(4)$$

以图 1 中的线性拓扑 DAG 为例，假定操作 O_1 在窗口 Δt 的输入速率为 20，且每个操作的处理速率等于其输入速率，设置 3 种 CR 配置（标识在 O_i 之上），则可由式(4)计算出操作 O_i 在不同的后续窗口 $n\Delta t$ 的输入速率，计算结果如图 3 所示.

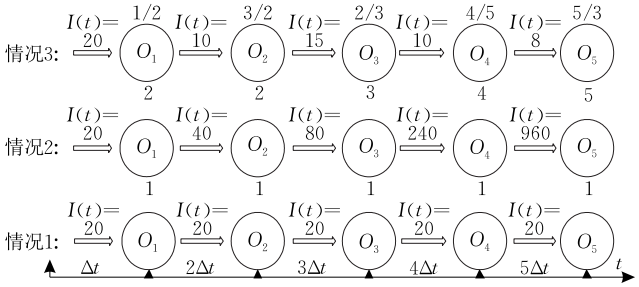


图 3 数据负载关联示例图

5 双向控制机制

双向控制机制借鉴了控制论的思想,用于上、下游操作间的信息传递,支持协同的弹性资源分配策略.该机制由反馈控制和前馈控制组成,分别位于操作与其邻居上、下游操作之间,线性结构如图 4 所示.

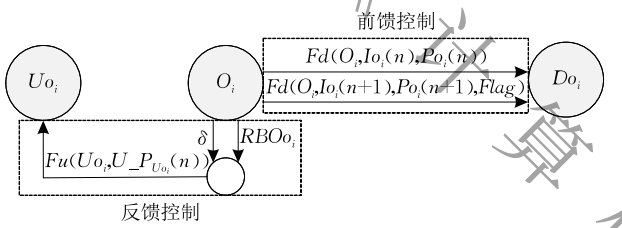


图 4 双向的线性控制图

5.1 反馈控制

由式(2)和(3)可知:具有非空输入缓存的上游操作的处理速率越高,则到达下游操作的数据负载越多,下游操作越有可能出现阶跃上升现象,从而进行资源调整.如果能够适当地控制下游操作的输入缓存的上升趋势,则资源调整将被避免或者推迟.基于上述分析,采用反馈控制支持上游操作对其邻居下游操作提供调整抑制.具体来说,上游操作在状态允许的情况下,根据下游操作发送的反馈调整信息调整处理速率,暂存下游操作的输入负载,减少下游操作的调整次数.

5.1.1 反馈调整原则

在窗口 n 中,Uo_i 的状态可以分为以下 3 种:扩展调整或反馈调整、收缩调整和无调整.当 Uo_i 处于第一种状态时,Uo_i 的处理能力不能满足自身的数据处理需求,需要增加资源或者保持不变,无法协助 O_i 进行反馈调整;在第二种状态下,Uo_i 的资源会减少,相当于协助 O_i 进行了非直接的反馈调整;只有在"无调整"的状态下,Uo_i 存在反馈调整的可能性,但需要进一步判断调整的可行性,即确定反馈调整是否影响其性能.以 Uo_i 的输入缓存占用作为衡量其性能的指标,τ 表示输入缓存的调整上限百分比阈值,则调整原则定义为:在反馈调整之后,Uo_i 在窗口 n 的输入缓存占用保持在 τ 或者以下,并且在窗口 $n+1$ 的输入缓存没有溢出.

5.1.2 反馈调整参数

基于反馈调整原则,确定反馈调整参数如下:O_i 在窗口 $n+1$ 的上限输入速率 $UI_{O_i}(n+1)$,Uo_i 在窗口 n 的处理速率调整上限 $UP_{Uo_i}(n)$ 和输出速率调整上限 $UO_{Uo_i}(n)$.

其中,$UI_{O_i}(n+1)$ 是确保 O_i 的输入缓存占用恰好处于警戒阈值 δ(在第 6 节)的输入速率,用于限定 Uo_i 的上限输出速率,通过式(5)计算.

$$UI_{O_i}(n+1)=Po_i(n)\times\Delta t+BSo_i\times\delta-BOo_i(n) \tag{5}$$

在式(5)中,BSo_i 是 O_i 的输入缓存大小.$BOo_i(n)$ 表示 O_i 在窗口 n 的输入缓存占用.

$UP_{Uo_i}(n)$ 是保证 Uo_i 满足调整原则的最大调整处理速率.采用式(6)描述调整原则,当 H 取真值时,$UP_{Uo_i}(n)$ 是两个不等式均取等号时的最小处理速率调整数值.

$$H=(BOT_{Uo_i}(n)\leq 1)\wedge(BO_{Uo_i}(n+1)\leq BS_{Uo_i}) \tag{6}$$

在式(6)中,$BOT_{Uo_i}(n)$ 表示输入缓存占用与调整上限阈值占用之间的比值,由式(7)计算.$BO_{Uo_i}(n+1)$ 是 Uo_i 在窗口 $n+1$ 的输入缓存占用.

$$BOT_{Uo_i}(n)=\frac{(P_{Uo_i}(n)-UP_{Uo_i}(n))\times\Delta t+BO_{Uo_i}(n)}{BS_{Uo_i}\times\tau} \tag{7}$$

$UO_{Uo_i}(n)$ 是 Uo_i 能够调整的最大输出速率,$UO_{Uo_i}(n)=(PU_{o_i}(n)-UP_{Uo_i}(n))\times\Delta t$.

5.1.3 反馈调整策略

在反馈控制中,反馈调整策略用于判断 Uo_i 进行反馈调整的可行性和确定反馈调整信息.该策略的基本思想是:首先,确定所有 Uo_i 的状态,如果不存在处于"无调整"状态的 Uo_i,则取消反馈调整;否则,计算 $I_{O_i}(n+1)$,如果 $I_{O_i}(n+1)$ 小于或等于 $UI_{O_i}(n+1)$,则取消反馈调整,否则,计算两者的差值 $NDO_{Uo_i}(n)$,即 Uo_i 调整的输出速率下限.然后,分别计算每个处于"无调整"状态 Uo_i 的处理速率调整上限 $UP_{Uo_i}(n)$ 和输出速率调整上限 $UO_{Uo_i}(n)$,并降序排列 $UO_{Uo_i}(n)$,同时,计算所有 $UO_{Uo_i}(n)$ 的和值 $SUO_{Uo_i}(n)$.最后,比较 $NDO_{Uo_i}(n)$ 与 $SUO_{Uo_i}(n)$,如果前者大于后者,则取消反馈调整;否则,降序选择使 $SUO_{Uo_i}(n)$ 大于或等于 $NDO_{Uo_i}(n)$ 的最小 Uo_i 集合进行反馈调整,并

分别设置 Uo_i 的反馈调整信息：$Fu(Uo_i, UP_{Uo_i}(n))$.

算法 1 实现了反馈调整策略，分别定义数组 U_Array 和 U_N_Array 存放所有 Uo_i 和处于“无调整”状态的 Uo_i，$U\_P\_O\_List$ 列表用于存放 $UP_{Uo_i}(n)$ 和 $UO_{Uo_i}(n)$ 信息，FU_List 是反调整信息列表，U_Set 表示最小的反调整集合.

算法 1. Feedback Adjustment Algorithm(FAA).

输入：U_Array 和 $UI_{O_i}(n+1)$

输出：FU_List

1. FOR (Uo_i) in U_Array
2. IF ($Uo_i.Flag==0$)
3. Uo_i 添加到 U_N_Array
4. IF ($U\_N\_Array.length==0$)
5. RETURN;
6. ELSE
7. 计算 $I_{O_i}(n+1)$;
8. IF ($I_{O_i}(n+1)\leq UI_{O_i}(n+1)$)
9. RETURN;
10. ELSE
11. $NDO_{Uo_i}(n)=I_{O_i}(n+1)-UI_{O_i}(n+1)$;
12. FOR (Uo_i) in U_N_Array
13. 计算$UP_{Uo_i}(n)$和$UO_{Uo_i}(n)$并添加到$U_P_O_List$;
14. 按照$UO_{Uo_i}(n)$降序排列$U_P_O_List$;
15. 计算 $SUO_{Uo_i}(n)$;
16. IF ($NDO_{Uo_i}(n)>SUO_{Uo_i}(n)$)
17. RETURN;
18. ELSE
19. WHILE ($NDO_{Uo_i}(n)\leq SUO_{Uo_i}(n)$) $\wedge$ $U_P_O_List\neq$null
20. 从$U_P_O_List$ 的列表头取出Uo_i并添加到U_Set;
21. $Fu(Uo_i, UP_{Uo_i}(n))$添加到 FU_List;
22. RETURN FU_List;

在 FAA 中，寻找 m 个无调整状态的 Uo_i 和计算每个 Uo_i 的参数分别都需要 $O(m)$ 时间，排序 $U\_P\_O\_List$ 需要 $O(m\log(m))$ 时间；找到 m' 个进行反馈调整的 Uo_i 的时间是 $O(m')$，$m'<m$，因此，算法 1 的时间复杂度是 $O(m\log(m))$.

5.2　前馈控制

不同于反馈控制，前馈控制的控制信息由 Uo_i 传递到 O_i，支持 O_i 的预先资源分配. 控制信息分为两类. 第一类控制信息表示为三元组：$Fd(O_i, Io_i(n), Po_i(n))$，用于预测 O_i 在窗口 $n+1$ 的输入速率，支持操作的基本调整(在第 6.2 节)；第二类控制信息是四元组的形式：$Fd(O_i, Io_i(n+1), Po_i(n+1), Flag)$，辅助进行操作的协同判定(在第 6.3 节). 其中，O_i 在窗口 $n+1$ 的处理速率 $Po_i(n+1)$ 由式(8)计算. $Flag$ 是调整的标志，取值为 1、−1 或 0，分别对应扩展调整、收缩调整和无调整的情况.

$$Po_i(n+1)=Po_i(n)+CPo_i(n+1) \tag{8}$$

其中，$CPo_i(n+1)$ 表示 O_i 在窗口 $n+1$ 的调整处理速率.

6　协同的资源分配策略

针对流数据处理，以数据负载关联模型和双向控制机制为基础，提出了一个实时的协同弹性资源分配策略，其核心思想是：通过上、下游操作间的协作，在突发数据到达下游操作之前预先启动与之匹配的最优调整并减少不必要的调整颠簸. 在策略中，设置了 4 个阈值：β、γ、δ 和 α，分别用于触发不同类型的操作调整，其中，β 是输入缓存的上限百分比阈值，γ 是输入缓存的下限百分比阈值，δ 是输入缓存的警戒百分比阈值，α 是资源使用率阈值，$\alpha=(PNo_i-1)/PNo_i$. 策略的实现过程如下：根据缓存状态函数和输入缓存大小计算 O_i 的输入缓存占用，同时，观察 O_i 的资源使用情况，当 O_i 的输入缓存占用处于 δ 和 β 之间时，触发反馈调整；超过 β 或者低于 γ 且资源使用率大于 α 时，触发扩展调整或收缩调整，通过基本调整获得最优的资源调整数量，再由协同判定决定是否执行基本调整的结果，避免调整颠簸.

6.1　缓存状态函数

本文将缓存状态定义为 O_i 的输入缓存占用与缓存大小的比值，即 $RBOo_i=BOo_i/BSo_i$. 根据排队论，如果已知 O_i 在窗口 $n-1$ 的缓存占用 $BOo_i(n-1)$、在窗口 n 的输入速率 $Io_i(n)$ 和处理速率 $Po_i(n)$，则 O_i 在窗口 n 的状态 $RBOo_i(n)$ 由式(9)计算.

$$RBOo_i(n)=\frac{((Io_i(n)-Po_i(n))\times\Delta t+BOo_i(n-1))}{BSo_i} \tag{9}$$

6.2　基本调整

在基本调整中，采用最优的方法确定资源分配的数量，即以最合适的处理速率匹配窗口 $n+1$ 的输入速率，操作的调整处理速率 $CPo_i(n+1)$ 由式(10)计算.

$$CPo_i(n+1)=\frac{BOo_i(n+1)-BSo_i(n)\times\delta}{\Delta t} \tag{10}$$

在式(10)中，$CPo_i(n+1)$ 为正值代表增加处理速率，负值则表示处理速率减少.

具体来说，基本调整分为两步进行：首先对 O_i 的参数 e 进行调整，如果 e 的变化能够满足 $CPo_i(n+1)$，

则调整结束;否则,进行增加或减少并行单元数量的调整,调整原则是:对于扩展调整,使用最少数量的并行单元确保变化的处理速率大于或等于$CPo_i(n+1)$;收缩调整则保证小于或等于$CPo_i(n+1)$即可.使用式(11)计算并行单元的调整数量$CPNo_i(n+1)$,其中,P_{unit}表示单个并行单元的处理速率.

$$CPNo_i(n+1)=\begin{cases}\left\lceil\dfrac{CPo_i(n+1)}{P_{unit}}\right\rceil, & CPo_i(n+1)>0\\ \left\lfloor\dfrac{CPo_i(n+1)}{P_{unit}}\right\rfloor, & CPo_i(n+1)<0\end{cases} \tag{11}$$

6.3 协同判定

在基本调整之后,结合Uo_i的前馈信息,需要对O_i的资源调整决定进行判定,防止出现抖动现象.协同判定的观测参数是O_i的输入数据趋势,判定过程如下:首先,基于前馈信息中的$I_{Uo_i}(n+1)$和$P_{Uo_i}(n+1)$,计算O_i在窗口$n+2$的输入速率$Io_i(n+2)$;然后,确定输入数据的趋势ITo_i,表示为$Io_i(n+2)$与$Io_i(n+1)$比值,即$ITo_i=Io_i(n+2)/Io_i(n+1)$,当$ITo_i$大于或等于1时,表示增加趋势,否则,输入数据处于减少趋势;最后,判断"执行条件"是否满足,满足则执行基本调整的结果,否则,取消结果.

基本调整的"执行条件"包括子条件1和子条件2,两者满足其一,则表示"执行条件"满足.其中,子条件1用于确保操作的调整趋势与输入趋势保持一致,避免发生抖动现象;子条件2则在上述两种趋势不一致的情况下,保证缓存不溢出.

对于子条件1,采用ATo_i标识操作O_i的调整趋势,$ATo_i=1$表示扩展调整;$ATo_i=-1$对应收缩调整.同时,To_i代表O_i的调整趋势与输入趋势之间的关系,当两种趋势皆为增加或减少时,To_i的值为0;前者增加、后者减少,To_i的值为1;反之,$To_i=-1$.上述关系形式化表示为式(12).

$$To_i=\begin{cases}0, & (ATo_i>0\wedge ITo_i\geq 1)\vee(ATo_i<0\wedge ITo_i\leq 1)\\ 1, & (ATo_i>0)\wedge(ITo_i<1)\\ -1, & (ATo_i<0)\wedge(ITo_i>1)\end{cases} \tag{12}$$

基于上述定义,协同判定形式化表示为式(13).其中,$CJo_i(n+1)$是协同判定的标识符,取值为1或-1,分别表示执行基本调整的结果和取消调整.

$$CJo_i(n+1)=\begin{cases}1, & (To_i==0)\vee(To_i==1\wedge BOo_i(n+1)\geq BSo_i)\\ -1, & (To_i==-1)\vee(To_i==1\wedge BOo_i(n+1)<BSo_i)\end{cases} \tag{13}$$

6.4 协同的资源分配算法

本文采用集中式算法实现了协同的弹性资源分配策略.该算法周期地收集每个操作在窗口n的状态信息,用于预测其邻居下游操作在窗口$n+1$的调整情况.对于首操作,借鉴文献[25]中的思想,使用卡尔曼滤波器[30]作为其数据预测模型.为了描述算法,进行了一些符号定义.其中,操作的状态信息存储在操作列表o_List中,包括输入速率、缓存占用和处理速率等信息,$I[o_i][k]$则记录窗口$n+k$的输入速率,k的取值为0、1或2.类似地,f_List是操作的前馈控制信息列表,下标1和2分别表示两种前馈信息,p_List和a_List分别是操作的并行度列表和调整的并行度列表.对于算法中的其他符号,其含义与本文之前部分中的定义保持一致.

算法2. Cooperative Resource Allocation Algorithm(CRAA).

输入:o_List和p_List
输出:FU_List和a_List

```
1.  FOR (O_i) in o_List
2.  获取 O_i 的当前状态并更新 o_List;
3.  添加 Fd(O_i, I[o_i][0], Po_i(n)) 到 f_List[Do_i][1];
4.  FOR (O_i) in o_List
5.   IF (i==1) THEN
6.    采用卡尔曼滤波器预测 I[o_i][1];
7.   ELSE
8.    使用式(3)计算 I[o_i][1];
9.  FOR (O_i) in o_List
10.  使用式(9)计算 RBOo_i(n+1);
11.  IF (i!=1∧RBOo_i(n+1)<β∧RBOo_i(n+1)>δ)
     THEN
12.   使用式(5)计算 UI_{O_i}(n+1);
13.   获取 O_i 的邻居上游操作列表 U_Array;
14.   根据算法1获取 FU_List;
15.  ELSE
16.   IF (RBOo_i(n+1)>β∨I[o_i][1]/(α×Po_i(n))<
        α∧RBOo_i(n+1)<γ) THEN
17.    使用式(10)~(13)分别计算 CPo_i(n+1),
         CPNo_i(n+1), To_i 和 CJo_i(n+1);
18.    IF (CJo_i(n+1)==1) THEN
19.     添加 CPN[o_i] 到 a_List;
20.     更新 p_List;
21.     使用式(8)计算 Po_i(n+1);
22.     添加 Fd(o_i, Io_i(n+1), Po_i(n+1), Flag) 到
          f_List[Do_i][2];
23. RETURN FU_List 和 a_List;
```

在CRAA中,获取和更新每个操作的状态需要进行n次收集完成;判断每个操作是否进行反馈调整

或者基本调整，仍然需要 n 次；单个操作的反馈调整需要 $O(m\log(m))$ 的时间，则反馈调整的时间复杂度为 $O(Km\log(m)+n-K)$，K 表示参与反馈调整的操作数量；所有操作的反馈调整或基本调整统一地执行. 因此，算法 2 的时间复杂度是 $O(Km\log(m))$.

7　实验结果与性能分析

在实验部分，本文使用 Java 语言实现了 CRAA，并进行了大量的模拟实验，主要从 4 个方面评价 CRAA 的性能：模型正确性、阈值影响、功能性和通用性. 同时，选择基本的阈值算法（Threshold Algorithm，TA）和卡尔曼滤波预测算法[24]（Kalman Prediction Algorithm，KPA）作为基准比对算法. 在实验中，考虑线性结构的 DAG，该 DAG 由 5 个操作组成，对应图 1 中的 $O_1\sim O_5$. 为了观察操作的缓存大小和处理能力对算法性能是否产生影响，对操作进行了不同的参数设置，如表 2 所示.

表 2　操作参数配置表

O_i	BS/元组	P_{unit}/(元组/s)	CR
O_1	50	500	1
O_2	500	400	1
O_3	1000	300	1
O_4	2000	200	1
O_5	5000	100	1

7.1　实验设置

为了研究算法在不同输入负载下的性能，选择了 3 种具有代表性的负载：正弦分布的平滑负载（简称正弦负载）、阶段变化的突变负载（简称阶段负载）和任意变化的随机负载（简称随机负载），并将负载量化为元组数量. 实验在吉林大学高性能中心的 6 个节点上运行，节点之间通过千兆以太网互连. 每个节点的配置如下：Intel(R) Xeon(R) E5-2670 2.60 GHz/16 核 CPU，32 GB 内存，Redhat Linux version 2.6.32-358.el6.x86_64 操作系统，分配 1 个节点作为 Zookeeper 服务器节点，管理集群中的资源并运行 CRAA 算法和比对算法，其余 5 个节点作为操作的执行节点，节点之间以 Socket 机制传输数据. 操作的并行单元以线程的形式运行，每个节点上配置 1 个线程池并部署 1 个操作，初始时，每个操作分配 1 个线程. 对于所有测试，算法至少运行 10 min，实验结果取运行期间的平均值.

7.2　模型正确性测试

在数据负载模型中，由于操作的消费比是固定的，输入负载的准确性（即模型的正确性）主要由处理速率和输出带宽决定. 对于处理速率，当并行单元数量增加时，可能存在非线性增长的情况，造成实际数据输出率与计算输出率不一致，因此，需要验证并行单元的可扩展性；在网络带宽方面，一旦输出速率增加，网卡的数据发送速率和网络的传输速率可能无法与之匹配，导致数据无法在处理时间间隔内到达操作，因此，需要测试网络传输的时间能否满足数据输出的要求.

在处理速率测试中，选择操作 O_1 作为测试对象，输入负载设置为 50 000 个元组，以统计单词计数为应用，并行单元数量从 1 扩展到 20. 如图 5 所示，执行时间随着并行单元数量的增加近似线性地增长. 由此可知，对于内存中数据的计算，数据负载模型中的处理速率误差可以忽略不计. 后续工作将在资源分配计算中考虑处理速率误差.

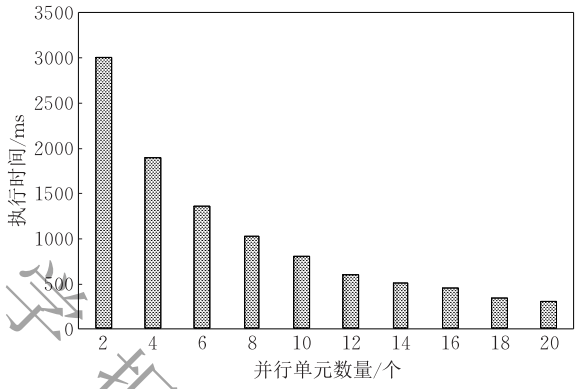


图 5　处理速率的扩展性测试图

在带宽测试中，选择操作 O_1 和 O_2 作为测试对象，O_1 发送数据，O_2 接收数据，O_1 的并行单元数量从 1 增加到 100，输出数据从 1 k 增加到 10 M. 在图 6 中，1 k、10 k 和 100 k 数据的传输时间对应左侧坐标轴，单位为 ms；1 M 和 10 M 数据对应右侧坐标轴，单位是 s. 从图 6 可以看出：当数据从 1 k 增加到 1 M（1 个并行单元除外）时，传输时间都小于 2000 ms，且并行单

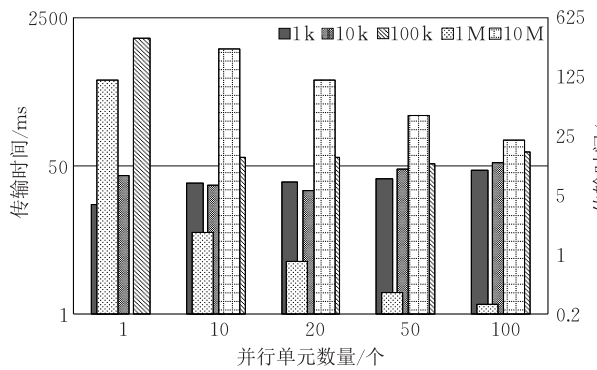
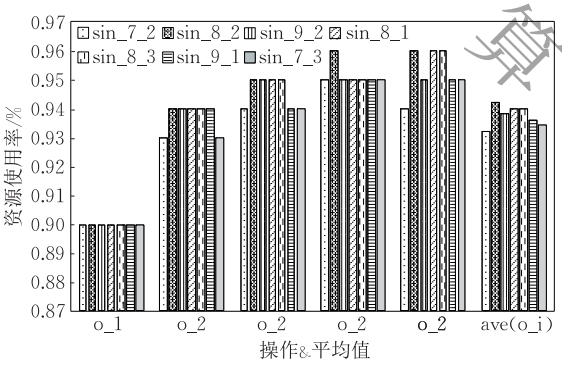


图 6　网络带宽的影响测试图

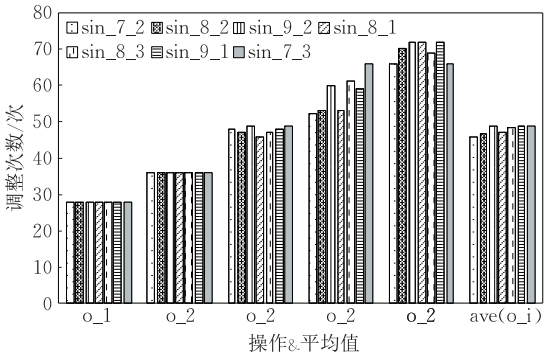
元数量对传输时间影响较小，但当数据量达到 100 k 以上时，单个并行单元的传输时间较长，尤其是增大到 10 M 时，数据传输时间均增加到 20 s 以上. 此时，传输时间超出设定的处理时间间隔 5 s，将导致数据不能完整到达操作，造成资源浪费，但不影响模型的正确性.

7.3 阈值影响测试

在 CRAA 中，操作的资源分配调整由扩展阈值 β 和收缩阈值 γ 触发，不同阈值的设置可能影响算法性能. 为此，针对 3 种典型的输入负载（详见第 7.5 节），选择 7 组阈值组合 (β,γ) 进行性能测试，其中，以 $\beta=80\%$，$\gamma=20\%$ 为基准，β 的取值集合为 $\{70,80,90\}$，γ 属于 $\{10,20,30\}$. 测试结果表明：不同的阈值选择没有影响数据丢失量，并且，资源使用率和调整次数也基本持平，其中，正弦负载的结果如图 7 所示. 由于 TA 在 $(80\%,20\%)$ 时的性能较好，为了保持一致性，在后续的实验中，CRAA、KPA 和 TA 的阈值 β 和 γ 均分别设置为 80% 和 20%.



(a) 不同阈值的资源使用率对比图



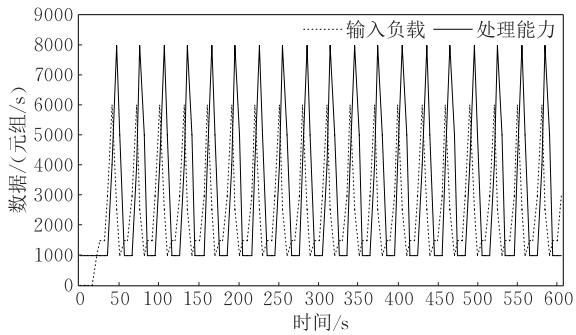
(b) 不同阈值的调整次数对比图

图 7 不同阈值的性能对比图

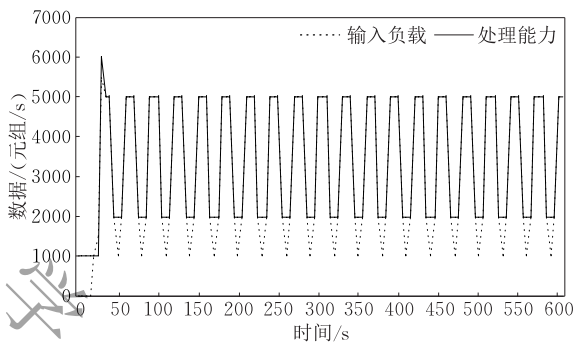
7.4 功能性测试

7.4.1 阶跃现象测试

在阶跃现象测试中，输入负载设置为阶段负载，变化周期为 5 s，即从 1000（元组/s）开始，逐渐地达到 2000（元组/s）、5000（元组/s）和 6000（元组/s），然后，再依次降到 5000（元组/s）、2000（元组/s）和 1000（元组/s），此后，重复以上过程. 以操作 O_4 为例，在执行 TA 和 CRAA 之后，如图 8 中的（a）和（b）所示，O_4 的处理能力都随着输入负载的变化而增加或减少，但是，TA 的调整变化滞后于负载变化，而 CRAA 的处理能力曲线基本与输入负载曲线重合，且未受到操作的输入缓存大小和处理能力的影响.



(a) O_4 在 TA 中的状态图

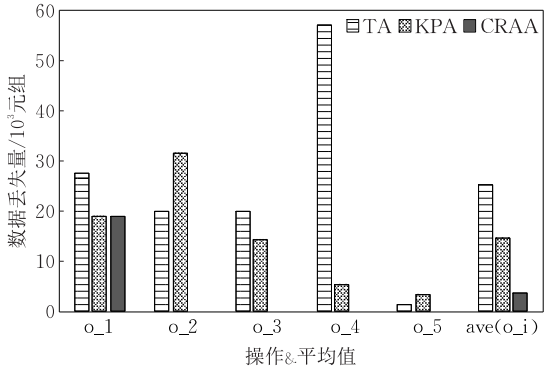


(b) O_4 在 CRAA 中的状态图

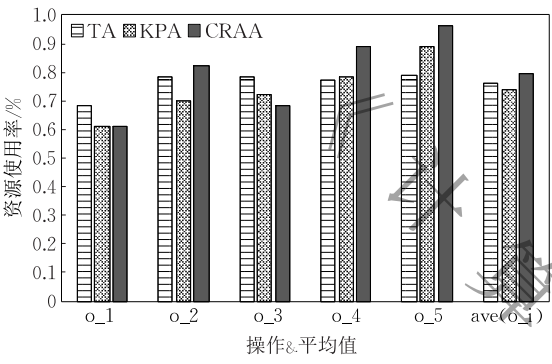
图 8 阶跃测试的操作状态图

在性能对比测试中，由于 CRAA 进行了最优的预先资源分配，且操作的处理能力与输入负载实现了无延迟的匹配，因此，CRAA 产生的 $DL(o_i)$ 明显少于 TA 和 KPA，类似地，KPA 也有效地减少了数据丢失量，如图 9(a) 所示. 在 CRAA 中，所有操作的平均 $RS(o_i)$ 也是高于 TA 和 KPA 的数值，如图 9(b) 所示. 但是，由于减少了调整次数，O_3 出现了低的 $RS(o_3)$，同样地，KPA 的资源使用率也略低于 TA. 在图 9(c) 中，通过观察发现，由于 O_3 和 O_4 都进行了调整避免，分别减少了 19 次调整，因此，CRAA 的 $AC(o_i)$ 明显都小于 TA 中的数值；相反，没有调整避免的 O_1、O_2 和 O_5 的 $AC(o_i)$ 与 TA 的数值接近，且在 O_1、O_2 和 O_4 上与 KPA 基本持平. 特别地，O_5 的 $AC(o_5)$ 略微高于 TA 的数值，原因是：O_5 的输入负载波动较频繁，为了避免数据丢失，增加了调整次数. 从上述分析可知，当输入负载突变时，CRAA 仍然能够有效地避免阶跃现象，减少数据丢失，同时，

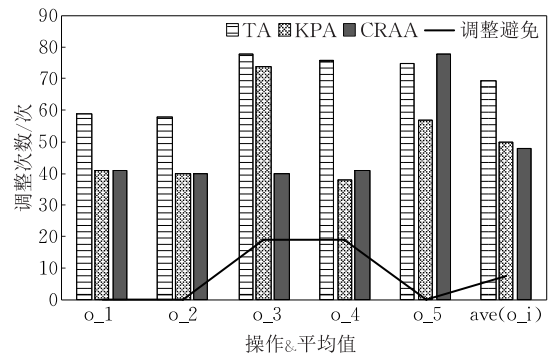
具有较高的资源使用率和较低的调整代价. KPA 的性能也整体上优于 TA.



(a) 阶跃测试的数据丢失量对比图



(b) 阶跃测试的资源使用率对比图

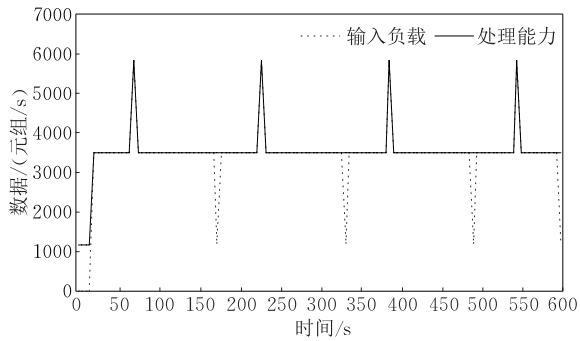


(c) 阶跃测试的调整次数对比图

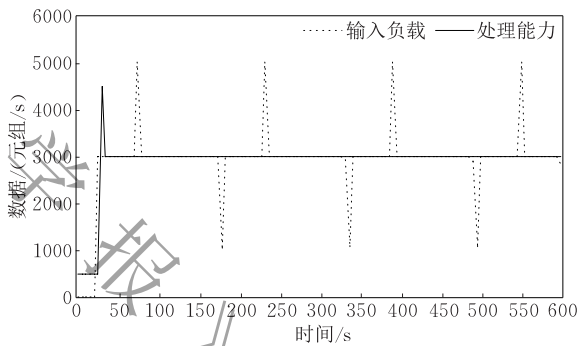
图 9　阶跃测试的性能对比图

7.4.2　抖动现象测试

在抖动现象测试中，仍然选择阶段负载作为输入负载，且设置变化的持续时间小于变化的间隔时间. 具体来说，负载以 3000(元组/s)开始并持续 50 s，接下来，突增到 6000(元组/s)，保持 5 s 后降到 3000(元组/s)，100 s 后继续降到 1000(元组/s)，5 s 后再次回到 3000(元组/s)，然后，重复以上过程. 如图 10 所示，CRAA 能够有效地避免 O_4 和 O_5 的调整颠簸，尤其在收缩调整中表现得更加明显. 在图 11(c)中，由于 CRAA 能够避免不必要的调整，CRAA 的 $AC(o_i)$ 普遍小于或等于 TA 和 KPA(O_2 除外)的数值. 其中，O_5 的表现最为突出，原因是其输入缓存足够大且输入负载与处理能力匹配度高，避免了所有的调整颠簸；对于 O_2，虽然 KPA 的调整次数少于 CRAA，但其数据丢失量明显增加，甚至超过了 TA，如图 11(a)所示. 结合图 10 和图 11 的对比数据，进一步得出结论：当突发负载持续时间较短时，CRAA 确实能够很好地避免抖动现象，减少调整次数，同时，其 $DL(o_i)$ 和 $RS(o_i)$ 的性能保持优于 TA 和 KPA.



(a) O_4 在CRAA中的状态图

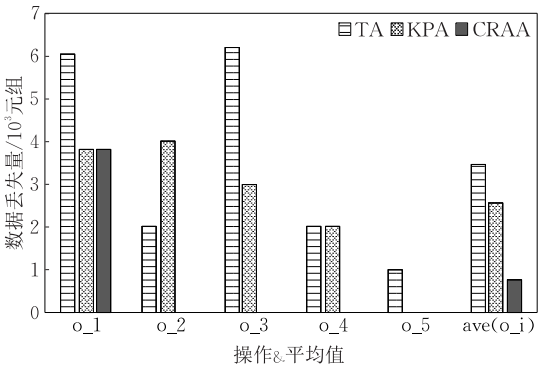


(b) O_5 在CRAA中的状态图

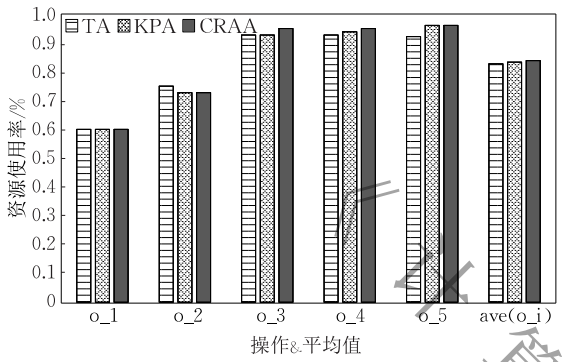
图 10　抖动测试的 CRAA 操作状态图

7.4.3　反馈调整测试

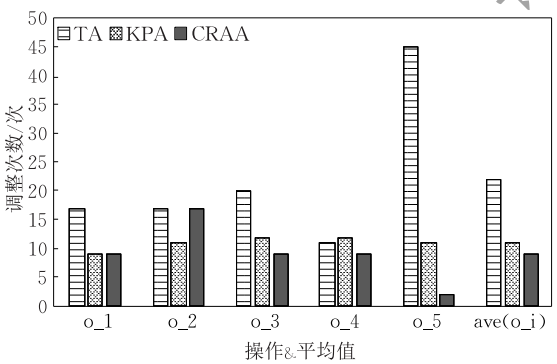
在测试中，设置所有操作的 BS 均为 10 000 元组，δ 取值为 70%，其他参数不变，且输入负载与抖动测试中的相同. 在图 12(c)中，反馈曲线的弯曲程度随着反馈调整次数的增加而增大，当 τ 取值为 70%和 80%时，反馈调整次数最多，此时，$AC(o_i)$ 相比其他情况下的数值小. 由此可以看出，反馈调整确实能够起到辅助减少调整次数的作用. 此外，当 $\tau=80\%$ 时，对比了 CRAA 与 TA 和 KPA 的其他性能. 如图 12(a)和图 12(b)所示，CRAA 的 $DL(o_i)$ 和 $RS(o_i)$ 仍然远优于 TA. 但是，为了减少数据丢失量，CRAA 通过增加调整次数适应输入负载的变化，导致 $AC(o_i)$ 略高于 TA 和 KPA 的数值.



(a) 抖动测试的数据丢失量对比图

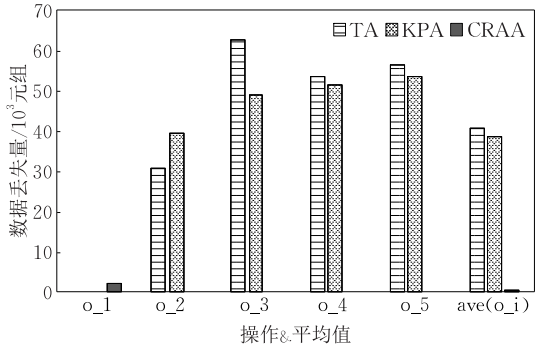


(b) 抖动测试的资源使用率对比图

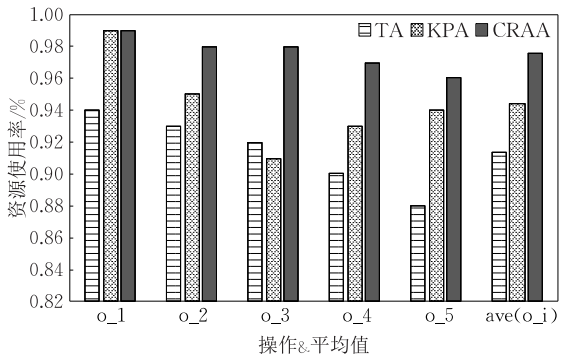


(c) 抖动测试的调整次数对比图

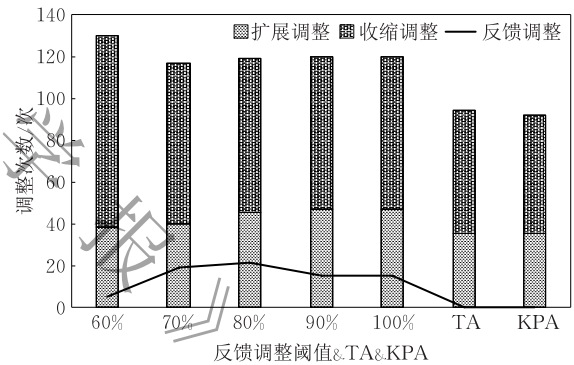
图 11 抖动测试的性能对比图



(a) 反馈调整测试的数据丢失量对比图



(b) 反馈调整测试的资源使用率对比图



(c) 反馈调整测试的调整次数对比图

图 12 反馈调整测试的性能对比图

7.5 通用性测试

采用 3 种典型的输入负载进行 CRAA 的通用性测试:(1) 正弦负载. 该负载的平均速率是 5000 (元组/s),调幅是 2500(元组/s),周期为 500 s;(2) 阶段负载. 其变化过程是:起始的输入速率是 5000 (元组/s),持续 250 s 后达到 10 000(元组/s),再经过 100 s 后降到 2000(元组/s),之后再次升到 5000 (元组/s),然后,重复上述变化;(3) 随机负载. 参考其它类型负载的变化范围,将随机负载的变化区间设置为[1000,50000](元组/s). 以阶段负载为例,由于所有操作都具有相似的负载变化趋势,因此,选择 O_4 作为观测对象. 在图 13 中,虚线代表 O_4 的输入负载,实线表示 O_4 的处理能力,两者的变化趋势保持一致(其他两种负载也存在类似的情况),且基本重合,意味着操作具有零数据丢失量和接近于 1 的资源使用率.

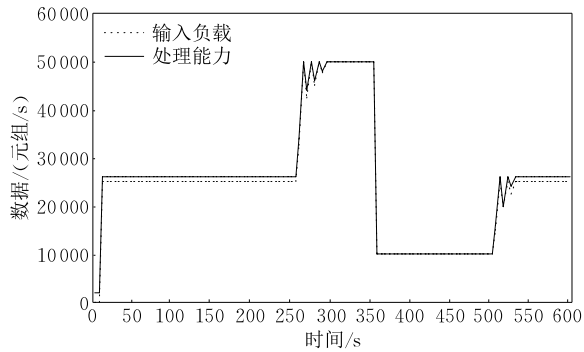
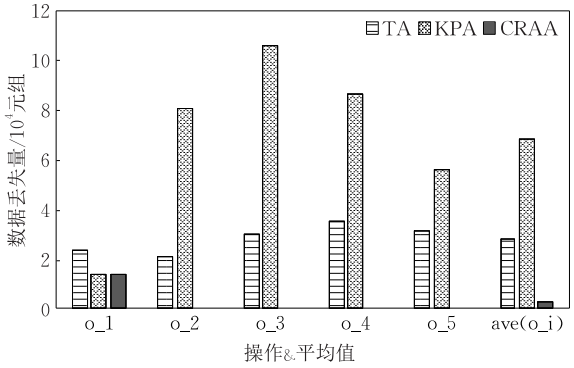
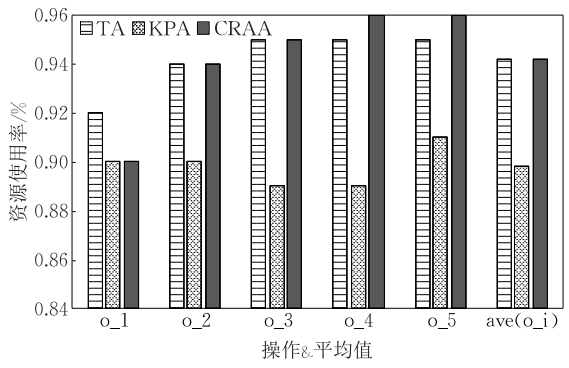


图 13 阶段负载的直观图

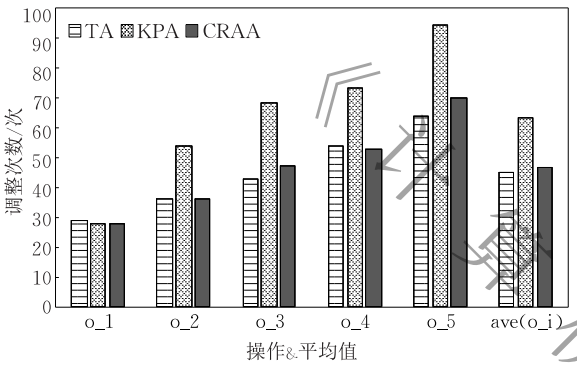
针对不同的输入负载，分别进行了 CRAA、KPA 和 TA 的性能比较，测试结果如图 14～图 16 所示，通过观察可以得出结论：无论负载是平滑变化的、突发变化的，甚至无规律变化的，CRAA 都能够获得远优于 TA 和 KPA 的 $DL(o_i)$性能，同时，保持 $RS(o_i)$和 $AC(o_i)$与 TA 的数值持平或略微之，并且



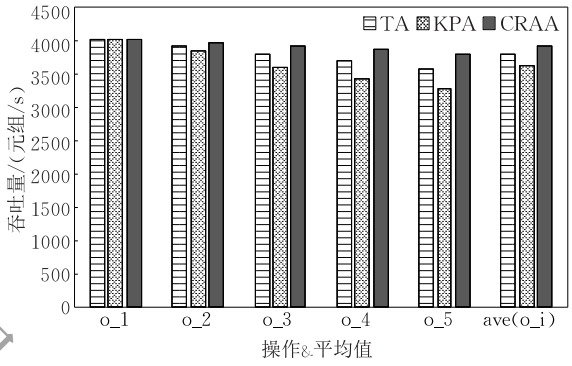
(a) 正弦负载的数据丢失量对比图



(b) 正弦负载的资源使用率对比图

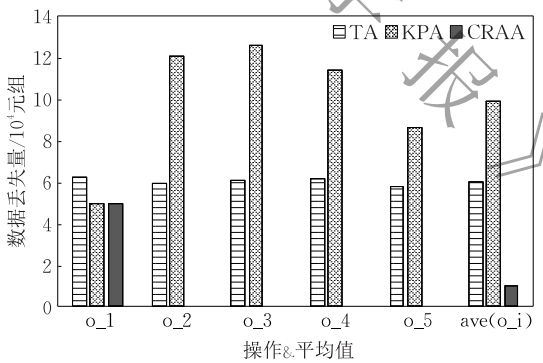


(c) 正弦负载的调整次数对比图

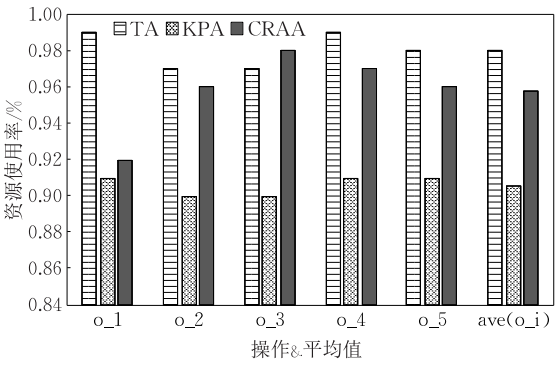


(d) 正弦负载的吞吐量对比图

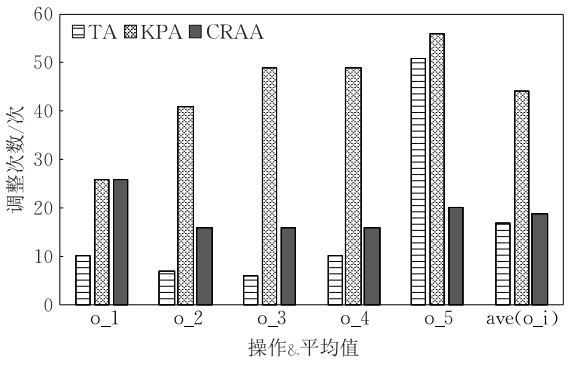
图 14　正弦负载的性能对比图



(a) 阶段负载的数据丢失量对比图

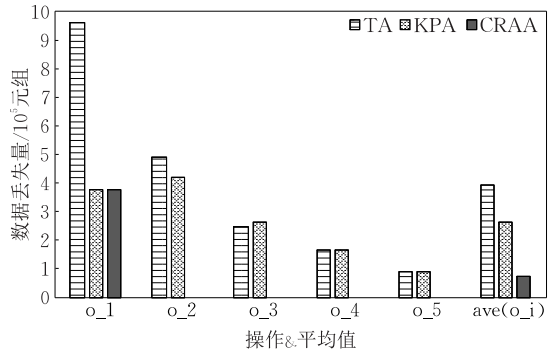


(b) 阶段负载的资源使用率对比图

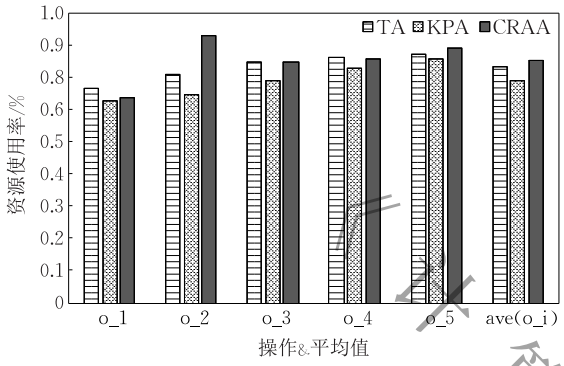


(c) 阶段负载的调整次数对比图

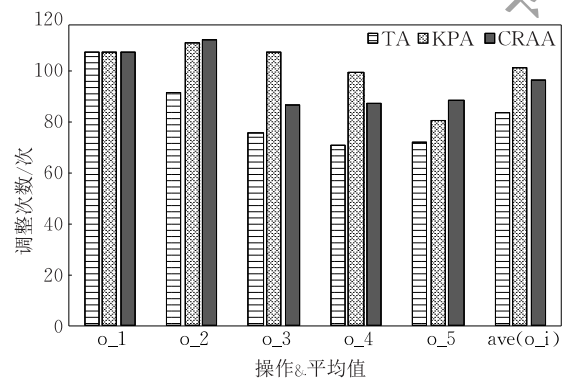
图 15　阶段负载的性能对比图



(a) 随机负载的数据丢失量对比图



(b) 随机负载的资源使用率对比图



(c) 随机负载的调整次数对比图

图 16 随机负载的性能对比图

与操作的配置无关. 诸如,在图 14(c)中,为了避免数据丢失,CRAA 的调整次数 $AC(o_i)$略微高于 TA 的值,类似的情况也出现在图 15 和图 16 中. 此外,对于 3 种负载,CRAA 的吞吐量略高,TA 次之,KPA 差点,以正弦负载为例,对比情况如图 14(d)所示. 与 CRAA 不同,KPA 的 $DL(o_i)$性能只在随机负载时表现较好,其他两种负载下的数据丢失量明显高于 TA,且资源使用率和调整次数也都比 TA 要差,原因在于:KPA 的数据预测存在误差,且误差的大小与输入负载的变化情况相关.

上述实验结果表明,针对不同变化类型的数据负载,CRAA 能够比 TA 和 KPA 更加有效地进行资源的分配调整,保证操作的处理能力与输入负载实现无延迟匹配,从而获得稳定且更好的性能,包括:几乎不存在数据丢失(首操作除外),同时,保持较高的资源使用率、较低的调整代价和略优的操作吞吐量. 此外,对于首操作,由于采用了卡尔曼滤波预测方法,与 TA 相比,有效地减少了首操作的数据丢失量,提高了应用的整体性能. 相比之下,KPA 的性能较不稳定,其性能表现与输入负载密切相关,部分情况下甚至差于 TA,通用性不强.

8 结论和展望

在数据流处理中,弹性资源分配的挑战在于:针对阶跃现象和抖动现象,保证高的应用性能和低的系统开销需求. 本文提出了一个上、下游操作间协同的弹性资源分配策略,其创新性主要表现在:(1)基于操作间的数据负载关联模型,准确地预测操作的输入负载,提高了资源分配的准确性;(2)通过双向控制机制,一方面,下游操作能够感知突发负载和负载的变化趋势,进行预先调整并避免了调整颠簸;另一方面,上游操作通过反馈调整实现对下游操作输入负载的控制,减少了不必要的资源分配调整,降低了调整代价;(3)以输入缓存占用和资源使用率作为触发调整的观测对象,通过最优的方法进行了资源分配,有效地避免了数据丢失,同时提高了资源使用率.

与基准的阈值策略和卡尔曼预测策略的对比实验表明,该策略能够确保在弹性资源调整中减少或避免阶跃现象和抖动现象,具有高的应用性能和低的系统开销. 在后续的工作中,主要从以下 3 个方面展开研究:(1)应用复杂的控制论模型,实现智能化的预先调整;(2)考虑处理速率和带宽的影响因素,进一步增强负载预测模型的适应性;(3)建立资源调整代价模型,考虑应用性能和调整代价的折衷.

参 考 文 献

[1] Chen H, Chiang R H L, Storey V C. Business intelligence and analytics: From big data to big impact. MIS Quarterly, 2012, 36(4): 1165-1188

[2] Demirkan H, Delen D. Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data incloud. Decision Support Systems, 2013, 55(1): 412-421

[3] Ottenwälder B, Koldehofe B, Rothermel K, et al. MCEP: A mobility-aware complex event processing system. ACM Transactions on Internet Technology, 2014, 14(1): 6

[4] Michael K, Miller K W. Big data: New opportunities and new challenges. Computer, 2013, 46(6): 22-24

[5] Neumeyer L, Robbins B, Nair A, et al. S4: Distributed stream computing platform//Proceedings of the 10th IEEE International Conference on Data Mining Workshops. Sydney, Australia, 2010: 170-177

[6] Mayer R, Koldehofe B, Rothermel K. Meeting predictable buffer limits in the parallel execution of event processing operators//Proceedings of the 2nd IEEE International Conference on Big Data. Washington, USA, 2014: 402-411

[7] Zacheilas N, Kalogeraki V, Zygouras N, et al. Elastic complex event processing exploiting prediction//Proceedings of the IEEE International Conference on Big Data. Santa Clara, USA, 2015: 213-222

[8] Matteis T D, Mencagli G. Proactive elasticity and energy awareness in data stream processing. Journal of Systems & Software, 2017, 127(5): 302-319

[9] Li Guo-Jie, Cheng Xue-Qi. Research status and scientific thinking of big data. Bulletin of Chinese Academy of Sciences, 2012, 27(6): 647-657(in Chinese)
(李国杰，程学旗. 大数据研究:未来科技及经济社会发展的重大战略领域——大数据的研究现状与科学思考. 中国科学院院刊，2012，27(6)：647-657)

[10] Wang Yuan-Zhuo, Jin Xiao-Long, Chen Xue-Qi. Network big data: Present and future. Chinese Journal of Computers, 2013, 36(6): 1125-1138(in Chinese)
(王元卓，靳小龙，程学旗. 网络大数据：现状与展望. 计算机学报，2013，36(6)：1125-1138)

[11] Sun Da-Wei, Zhang Guang-Yan, Zheng Wei-Min. Big data stream computing technologies and instances. Journal of Software, 2014, 25(4): 839-862(in Chinese)
(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)

[12] Qi Kai-Yuan, Zhao Zhuo-Feng, Fang Jun, Ma Qiang. Real-time processing for high speed data stream over larger scale data. Chinese Journal of Computers, 2012, 35(3): 477-490 (in Chinese)
(亓开元，赵卓峰，房俊，马强. 针对高速数据流的大规模数据实时分析方法. 计算机学报，2012，35(3)：477-490)

[13] Zhang Peng, Liu Qing-Yun, Tan Jian-Long, et al. SPSPS: A scalable parallel-distributed stream processing system. Acta Electronica Sinica, 2015, 43(4): 639-646(in Chinese)
(张鹏，刘庆云，谭建龙等. 流水行云：支持可扩展的并行分布式流处理系统. 电子学报，2015，43(4)：639-646)

[14] Sun D, Zhang G, Yang S, et al. Re-Stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. Information Sciences, 2015, 319: 92-112

[15] Heinze T, Pappalardo V, Jerzak Z, et al. Auto-scaling techniques for elastic data stream processing//Proceedings of the 30th IEEE International Conference on Data Engineering Workshops. Chicago, Illinois, USA, 2014: 296-302

[16] Zhou B, Luan Z, Wu J, et al. Using paralleled-PEs method to resolve the bursting data in distributed stream processing system//Proceedings of the 16th IEEE International Conference on Computational Science and Engineering. Sydney, Australia, 2013: 1324-1331

[17] Castro Fernandez R, Migliavacca M, Kalyvianaki E, et al. Integrating scale out and fault tolerance in stream processing using operator state management//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, USA, 2013: 725-736

[18] Gulisano V, Jimenez-Peris R, Patino-Martinez M, et al. Streamcloud: An elastic and scalable data streaming system. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(12): 2351-2365

[19] Martin A, Brito A, Fetzer C. Scalable and elastic realtime click stream analysis using StreamMine3G//Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems. Mumbai, India, 2014: 198-205

[20] Heinze T, Jerzak Z, Hackenbroich G, et al. Latency-aware elastic scaling for distributed data stream processing systems//Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems. Mumbai, India, 2014: 13-22

[21] Gedik B, Schneider S, Hirzel M, et al. Elastic scaling for data stream processing. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(6): 1447-1463

[22] Fu T Z J, Ding J, Ma R T B, et al. DRS: Dynamic resource scheduling for teal-time analytics over fast streams. Computer Science, 2015, 690(1): 411-420

[23] Lohrmann B, Janacik P, Kao O. Elastic stream processing with latency guarantees//Proceedings of the 35th IEEE International Conference on Distributed Computing Systems. Columbus, USA, 2015: 399-410

[24] Madsen K G S, Thyssen P, Zhou Y. Integrating fault-tolerance and elasticity in a distributed data stream processing system//Proceedings of the 26th International Conference on Scientific and Statistical Database Management. Aalborg, Denmark, 2014: 48

[25] Zhang F, Cao J, Khan S U, et al. A task-level adaptive MapReduce framework for real-time streaming data in healthcare applications. Future Generation Computer Systems, 2015, 43: 149-160

[26] Kumbhare A G, Simmhan Y, Frincu M, et al. Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure. IEEE Transactions on Cloud Computing, 2015, 3(2): 105-118

[27] Kumbhare A G, Simmhan Y, Prasanna V K. PLAStiCC: Predictive look-ahead scheduling for continuous dataflows on clouds//Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Chicago, USA, 2014: 344-353

[28] Kumbhare A, Frincu M, Simmhan Y, et al. Fault-tolerant and elastic streaming MapReduce with decentralized coordination //Proceedings of the 35th IEEE International Conference on Distributed Computing Systems. Columbus, Ohio, USA, 2015: 328-338

[29] Wu Y J, Tan K-L. ChronoStream: Elastic stateful stream computation in the cloud//Proceedings of the 31st IEEE International Conference on Data Engineering. Seoul, South Korea, 2015: 723-734

[30] Kalman R E. A new approach to linear filtering and prediction theory. Journal of Basic Engineering, 1960, 82D(1): 35-45

LI Li-Na, born in 1978, Ph. D. candidate, associate professor. Her research interests include stream data processing and wireless sensor network.

WEI Xiao-Hui, born in 1972, Ph. D., professor. His research interests include distributed system, etc.

LI Xiang, born in 1988, M. S. His research interests include stream data processing and distributed system.

WANG Xing-Wang, born in 1990, Ph. D. candidate. His research interests include cloud computing and mobile computing.

Background

In the big data stream-computing environment, the scalability of the system is an important factor of restricting its application. At present, the research on elastic resource allocation is still facing the challenge. Most of the work puts forward the "instant" response strategy, and not solves the problem of data loss caused by the adjusting delay. To this end, some strategies of "pre-adjustment" have been launched. But, the load-forecasting model in these strategies is based on the historical data, which influences the accuracy of the adjustment. Specially, the research is mainly to solve the problem of a single node of the application, rarely considering the problems of burst load and adjustment bumps of the downstream node, which caused by the resource adjustment of the upstream node. In this paper, we define and study on the step phenomenon and jitter phenomenon caused by the above problems, and the main challenge is to realize the rapid adjustment of the associated downstream node and effectively avoid its adjustment bumps. Thus, this paper first proposes a precise data correlation model for sensing the burst load of the downstream nodes, as the basis of its pre-adjustment; and then, this paper transfers the adjustment information of the upstream node and the feedback information of the downstream node by a bi-directional control mechanism; finally, this paper presents an upstream-downstream cooperative and elastic resource allocation strategy to support the optimal pre-adjustment of the downstream node, the feedback adjustment of the upstream node and avoiding the bumpy adjustment. Compared with the benchmark strategy, this strategy can completely solve the problems of the data loss of the associated node and avoid its adjustment bumps well, while being able to achieve high resource utilization and low system overhead.

The research is supported by the National Natural Science Foundation of China (No. 61170004), and the Project of Jilin Province Science and Technology Agency (No. 20140204013GX).