Flink 环境下基于负载预测的弹性资源调度策略

李梓杨[1,2]，于炯[1,2]，王跃飞[3]，卞琛[4]，蒲勇霖[2]，张译天[1]，刘宇[1]

（1. 新疆大学软件学院，新疆 乌鲁木齐 830008；2. 新疆大学信息科学与工程学院，新疆 乌鲁木齐 830046；
3. 成都大学计算机学院，四川 成都 610106；
4. 广东金融学院互联网金融与信息工程学院，广东 广州 510521）

摘 要：为了解决大数据流式计算平台中存在计算负载剧烈波动，但集群因资源不足而遇到性能瓶颈的问题，提出了 Flink 环境下基于负载预测的弹性资源调度（LPERS-Flink）策略。首先，建立负载预测模型并在此基础上提出负载预测算法，预测集群负载的变化趋势；其次，建立资源判定模型，以判定集群出现资源瓶颈与资源过剩的问题，由此提出弹性资源调度算法，制定弹性资源调度计划；最后，通过在线负载迁移算法执行调度计划，实现高效的节点间负载迁移。实验结果表明，该策略在负载剧烈波动的应用场景中有较好的优化效果，实现了集群规模和资源配置对负载变化的及时响应，降低了负载迁移的通信开销。

关键词：流式计算；资源调度；负载预测；性能瓶颈；Flink

中图分类号：TP311

文献标识码：A

doi: 10.11959/j.issn.1000−436x.2020195

Load prediction based elastic resource scheduling strategy in Flink

LI Ziyang[1,2], YU Jiong[1,2], WANG Yuefei[3], BIAN Chen[4], PU Yonglin[2], ZHANG Yitian[1], LIU Yu[1]

1. School of Software, Xinjiang University, Urumqi 830008, China
2. School of Information Science and Engineering, Xinjiang University, Urumqi 830046, China
3. College of Computer Science, Chengdu University, Chengdu 610106, China
4. College of Internet Finance and Information Engineering, Guangdong University of Finance, Guangzhou 510521, China

Abstract: In order to solve the problem that the load of big data stream computing platform fluctuates drastically while the cluster was suffering from the performance bottleneck due to the shortage of computing resources, the load prediction based elastic resource scheduling strategy in Flink (LPERS-Flink) was proposed. Firstly, the load prediction model was set up as the foundation to propose the load prediction algorithm and predict the variation tendency of the processing load. Secondly, the resource judgment model was set up to identify the performance bottleneck and resource redundancy of the cluster while the resource scheduling algorithm was proposed to draw up the resource rescheduling plan. Finally, the online load migration algorithm was proposed to execute the resource rescheduling plan and migrate processing load among nodes efficiently. The experimental results show that the strategy provides better performance promotion in the application with drastically fluctuating processing load. The scale and resource configuration of the cluster responded to the variation of processing load in time and the communication overhead of the load migration was reduced effectively.

Key words: stream computing, resource scheduling, load prediction, performance bottleneck, Flink

收稿日期：2020−04−20；**修回日期**：2020−07−20

基金项目：国家自然科学基金资助项目（No.61862060, No.61462079, No.61562086, No.61562078）；新疆维吾尔自治区自然科学基金项目（No.2017D01A20）；新疆大学博士生科技创新项目（No.XJUBSCX-201902）

Foundation Items: The National Natural Science Foundation of China (No.61862060, No.61462079, No.61562086, No.61562078), The Natural Science Foundation of Xinjiang Uygur Autonomous Region of China (No 2017D01A20), The Doctoral Innovation Program of Xinjiang University(No XJUBSCX-201902)

1　引言

随着信息化时代与互联网的高速发展，智能家居、智能医疗、智能工业、智能汽车等物联网[1]场景下产生的数据量日益增多，并与互联网共同成为各行各业大数据的主要来源。希捷（Seagate）公司与互联网数据中心（IDC, Internet Data Center）联合发布的《数据时代 2025》白皮书中预测，2025 年全球数据量将达到 163 ZB，其中，超过 25%的数据为实时数据。

由此可见，对大规模数据的实时分析和处理具有非常广阔的应用前景。就目前的研究成果来看，以 MapReduce[2]为代表的批量计算框架[3-4]的计算时延较高，无法满足业务的实时性要求。流式计算将处于活动状态的数据持续发送至各工作节点，计算和传输不间断发生，中间结果仅存在于内存而不需要持久存储，为实时数据的处理提供了良好的解决方案[5]。作为一款开源、高性能、分布式的数据流处理平台[6]，Flink[7]已经成为流数据处理领域的通用计算平台。与传统的流式计算平台 Apache Storm[8]相比，Flink 支持有状态的流式计算[9]和 Exactly-Once 的容错机制[10]；与新兴流式计算平台 Apache Heron[11]相比，Flink 具有更成熟的平台架构和更广泛的产业基础。因此，Flink 已经成为集批处理与流处理为一体的统一数据分析平台，得到学术界和产业界的广泛关注。

然而，Flink 的发展也面临一系列的挑战。首先，Flink 和传统流式计算平台一样，均面临负载波动环境下资源不足导致性能下降的问题。其次，作业执行中集群的规模和算子的并行度是固定不变的，但计算负载是波动变化的，如果计算资源无法满足计算负载的需求，就会严重影响计算的实时性，因此弹性资源调度策略的研究非常重要。最后，现有研究仅以当前的计算负载为依据制定弹性资源调度计划，无法适应未来一段时间的负载变化，从而导致调度滞后的问题。因此，结合机器学习算法实现动态弹性资源调度策略，将是未来研究的新热点[12]。但目前的主流预测方法对数据波动的抗干扰能力不足，在负载剧烈波动的流式计算环境中往往难以达到足够的预测准确性。此外，目前主流的资源判定模型[13-14]主要从拓扑结构的全局角度判定资源与负载的相对关系，无法从局部发现集群的性能瓶颈，无法适用于弹性资源调度的应用场景。

针对上述问题，本文提出 Flink 环境下基于负载预测的弹性资源调度（LPERS-Flink, load prediction based elastic resource scheduling in Flink）策略，本文的主要贡献总结如下。

1) 提出将机器学习算法应用于流式计算平台底层优化的思想，并在此基础上提出基于时间序列的预测算法，在负载剧烈波动的场景中取得较高的预测准确性，为提出在线弹性资源调度策略提供了模型支撑。

2) 在负载预测模型的基础上提出负载预测算法，根据已知的负载变化规律预测未来的变化趋势，从而将作业的拓扑结构抽象为预测网络模型，为提出基于负载变化趋势的弹性资源调度策略提供数据基础。

3) 在资源判定模型的基础上提出资源判定算法，针对集群存在的资源瓶颈与资源过剩的问题，制定相应的弹性资源调度计划，动态改变集群的规模与作业的拓扑结构，优化集群的资源配置。

4) 针对弹性资源调度开销过高的问题，提出基于同步计算和异步状态数据拉取的在线负载迁移算法，通过在线迁移计算负载和状态数据，实现对用户透明的高效弹性资源调度。

2　相关研究

在负载波动环境下，因集群资源不足产生性能下降的问题是目前流式计算平台面临的主要技术挑战，其中 Storm 平台现有的资源调度策略存在调度开销高和调度滞后的问题，新生 Heron 平台未提供针对负载波动场景的解决方案。因此，在负载剧烈波动环境下，集群因资源不足而遇到性能瓶颈的问题，是流式计算研究面临的主要难点。

针对上述问题，国内外学者分别从不同的角度提出了解决方案。目前，现有的研究成果主要基于以下 3 种方案来解决该问题：1) 通过优化任务调度策略提高集群性能；2) 通过降低通信开销提高作业的执行效率；3) 通过提出弹性资源调度策略突破集群的性能瓶颈。

首先，传统的流式计算平台研究大多关注优化集群的任务调度策略，通过提高集群性能降低计算时延。文献[14-15]分别根据作业拓扑结构的有向无环图模型提出 2 种不同的任务调度策略，在提高集群性能、降低计算时延的同时降低了集群的运行能耗[13,16]，取得了一定的优化效果。文献[17-20]分别

通过分析作业的拓扑结构及特点，从不同角度提出了相应的任务调度策略。然而，在面对持续波动的计算负载时，优化任务调度策略只能提高集群的资源利用率，并不能从根本上解决集群资源不足的问题。其次，降低通信开销是提高作业执行效率的另一种有效方案。文献[21]针对 Storm 平台提出的 T3-Scheduler，通过寻找集群中数据传输量较大的 2 个任务，并将其部署在同一个节点上，有效降低了任务之间的通信开销。文献[22]提出将机器学习算法应用于流式计算平台优化的思想，通过监控与集群传输性能有关的指标，提出基于机器学习的任务重部署策略，有效降低了节点之间的数据传输开销。文献[23-25]以降低集群的通信开销为目标，分别提出不同的任务调度策略，但这些调度策略有可能造成节点的 CPU 或内存资源利用率急剧上升，进而导致节点资源溢出的问题，在计算负载急剧上升的场景中无法提供有效的解决方案。

与上述 2 种方案相比，弹性资源调度策略通过动态增加计算资源突破集群性能瓶颈，是解决集群性能问题的最有效方案[26]。文献[27]提出一种适用于 Storm 平台的弹性资源调度策略，通过区分有状态数据流和无状态数据流的应用场景，分别处理 2 种情况，实现了高效的弹性资源调度。文献[28-29]提出基于强化学习的弹性资源调度策略，通过监控集群性能指标并执行强化学习算法，实现集群规模的动态自适应变化，创新性地提出将机器学习算法应用于分布式计算平台优化的思想，但该策略在产业界的大规模应用仍面临一定挑战。文献[30-34]从不同的角度提出弹性资源调度策略，但大多适用于传统的 Storm 平台，受限于平台架构的区别而无法应用于 Flink 平台。

此外，文献[35]提出基于流网络的离线弹性资源调度策略，通过建立流网络模型定位集群性能瓶颈，并通过动态增加计算节点突破瓶颈，是本文研究的前提和基础工作。但该策略使用离线的负载迁移算法，存在调度滞后和调度过程中作业停滞的问题，影响计算的实时性。文献[36]提出适用于 Flink 平台内核 Nephel 的响应式资源调度策略，通过建立数学模型计算每个算子的最优并行度，并通过任务迁移实现集群资源的动态伸缩，与本文的研究目标紧密相关，但其负载迁移过程中的网络传输开销较大。文献[37]提出的 SRA（state and runtime-aware）-Stream 是一种有状态数据流的弹性资源调度策略，该策略既考虑计算负载与响应时间的数学关系，也考虑状态数据迁移的问题，是一种非常高效的弹性资源调度策略，但该策略仅适用于 Storm 平台。现有的研究成果大多面临以下问题：1) 以当前的计算负载为依据进行弹性资源调度，忽略了未来负载可能出现的波动变化，存在调度滞后的问题；2) 离线的调度策略本身产生过高的资源开销，存在调度过程中性能下降的问题；3) 现有研究多适用于传统的 Storm 平台，受限于平台架构之间的区别而无法移植于目前主流的 Flink 平台。

针对上述问题，本文提出基于负载预测的在线弹性资源调度策略，并将其应用于 Flink 平台。本文与现有研究成果的不同之处总结如下。

1) 现有研究成果大多基于当前时刻的计算负载进行弹性资源调度，存在调度滞后的问题。本文将机器学习算法应用于流式计算平台的底层优化，基于负载预测结果进行提前调度，实现集群规模和资源配置对负载变化的及时响应。

2) 现有研究成果大多在计算资源不足时增加节点以避免出现资源瓶颈，但忽略了在资源过剩时出现的资源浪费问题。本文通过定位集群的资源瓶颈与资源过剩，动态扩大和缩小集群规模，从而优化集群的资源配置。

3) 现有研究成果大多采用离线资源调度的方式，调度过程中存在数据堆积和时延上升的问题。本文提出基于异步状态数据拉取的在线弹性资源调度策略，有效解决调度过程中集群性能下降的问题。

3 问题建模与分析

针对大数据流式计算的突发性特征[5]，为了应对计算负载的波动变化，本节分别建立了基础逻辑模型、负载预测模型和资源判定模型。首先，通过基础逻辑模型对作业的拓扑结构进行抽象，量化集群资源与负载的数学关系；然后，通过负载预测模型预测计算负载的变化趋势；最后，通过资源判定模型定位集群的资源瓶颈与资源过剩，为实现在线弹性资源调度策略提供理论依据。

3.1 基础逻辑模型

作为目前主流的分布式数据流处理平台，Apache Flink 采用主从式架构，如图 1 所示。主节点为 JobManager，负责管理计算资源、接收用户提交的作业并根据其拓扑结构部署任务。工作节点为

TaskManager，负责根据 JobManager 的调度，执行计算任务并完成数据处理。

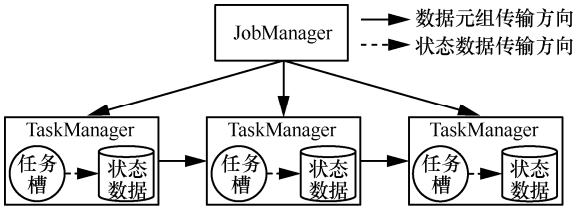


图 1　Apache Flink 基础架构

在 TaskManager 中，每个任务槽（TaskSlot）都是一个线程，负责执行任务并完成数据处理，其计算的中间结果可能存储在内存或轻量级数据库等存储介质中，作为节点的状态数据。因此，主节点以任务槽为基本单位进行资源管理，每个任务槽代表任务执行中可能用到的 CPU、内存等计算资源。在作业执行过程中，待处理的数据从源点发出，依次经过计算节点的处理，最终计算结果在汇点被持久化存储。节点能够处理数据的最高速率为节点的计算能力，从源点实际发出数据的速率为当前的计算负载，为了应对计算负载的波动变化，建立流网络模型，量化集群计算能力与计算负载的数学关系。

预测网络模型如图 2 所示。将流式计算的拓扑结构定义为有向无环图模型，其中，节点表示处理数据的任务槽，边表示节点之间的数据传输链路。为了量化计算能力与计算负载的关系，将每个节点能够处理的最高计算负载定义为对应边的容量 $c(v_i, v_j)$，通过文献[35]的流网络构建算法可以计算每条边的容量取值。节点实际的计算负载定义为当前时刻的流量 $f(v_i, v_j)$，从而将有向无环图模型转化为流网络模型。针对现有弹性资源调度策略存在调度滞后的问题，通过建立负载预测模型，预测未来负载的变化趋势。

3.2　负载预测模型

假设一段时间内，以固定的时间间隔采集到源点发送数据的速率为 $F=\{f_1, f_2, \cdots, f_t\}$，且样本的均值为 μ，方差为 σ^2。为了使用差分整合移动平均自回归（ARIMA, autoregressive integrated moving average）模型进行负载预测，要求计算负载的均值和方差保持基本稳定，即未来的计算负载与之前的负载取值具有相关关系，这样的数据样本是平稳的。通过建立不同时刻计算负载之间的函数关系，可以预测未来计算负载的变化趋势。

然而，在实际应用中，计算负载是随着时间的推移而剧烈波动的，通常无法满足预测模型对样本的平稳性要求。因此，通过分别计算 t 时刻与 $t-1$ 时刻负载的差值，可得到计算负载的一阶差分，即

$$f_t^1 = \begin{cases} 0, & t=1 \\ f_t - f_{t-1}, & t>1 \end{cases} \tag{1}$$

其中，f_t 为 t 时刻源点发送数据的速率。当一阶差分仍然存在剧烈波动时，计算 t 时刻与 $t-1$ 时刻的差值，得到计算负载的二阶差分。依次类推，负载的 d 阶差分为

$$f_t^d = \begin{cases} 0, & t=1 \\ f_t^{d-1} - f_{t-1}^{d-1}, & t>1 \end{cases} \tag{2}$$

其中，f_t^{d-1} 为上一阶差分中 t 时刻的样本取值。通过数据预处理的方式，将原本剧烈波动的计算负载变得平稳，使不同时刻的样本之间具有明显的相关

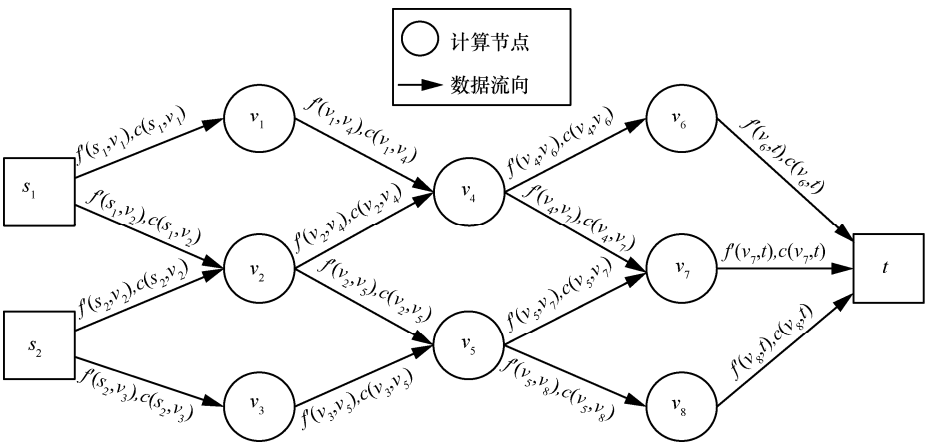


图 2　预测网络模型

关系，从而满足预测模型对样本的平稳性要求，并通过相关系数衡量样本的平稳性。

在计算负载的 d 阶差分序列上，t 时刻的负载 f_t^d 与 $t-k$ 时刻的负载 f_{t-k}^d 之间的自相关系数为

$$\mathrm{ACF}(k)=\rho_k=\frac{\mathrm{Cov}\left(f_t^d,f_{t-k}^d\right)}{\mathrm{Var}\left(f_t^d\right)} \tag{3}$$

其中，$\mathrm{Cov}\left(f_t^d,f_{t-k}^d\right)$ 为 t 时刻与 $t-k$ 时刻负载的协方差，$\rho_k\in[-1,1]$。

特别地，如果 $\rho_k>0.5$，说明不同时刻的负载取值之间具有很强的相关关系，即当前的 d 阶差分序列能够满足预测的平稳性要求。通过建立负载回归模型，能够描述不同时刻负载取值之间的依赖关系。

定义 1 负载回归模型。计算负载的 d 阶差分序列中，第 t 时刻的负载取值为 f_t^d，之前第 $t-i$ 时刻的样本取值为 f_{t-i}^d，则 t 时刻与之前 i 时刻负载的函数关系为

$$f_t^d=\sum_{i=1}^{p}\lambda_i f_{t-i}^d+\varepsilon_i+\mu_t \tag{4}$$

其中，λ_i 是模型的参数，μ_t 是常数项，λ_i 和 μ_t 均可通过最大似然估计等方法计算得出；ε_i 是预测过程中可能产生的随机波动，可通过负载平均模型消除由波动产生的误差。

由定义 1 可知，负载回归模型建立了不同时刻的计算负载之间的函数关系，从而可以根据过去一段时间的负载变化规律，预测未来负载的变化趋势。然而，利用负载回归模型进行预测的过程中，仍有可能因 ε_i 而产生误差。因此，通过建立负载平均模型，消除负载变化过程中可能产生的随机波动，从而降低误差并提高预测的准确性。

定义 2 负载平均模型。计算负载的 d 阶差分序列中，用 ε_t 表示 t 时刻计算负载可能产生的随机波动，则

$$\varepsilon_t=\sum_{i=1}^{q}\theta_i\varepsilon_{t-i}+\varepsilon_0+\mu_t \tag{5}$$

其中，ε_{t-i} 表示 $t-i$ 时刻负载的随机波动。通过建立不同时刻负载变化的函数关系，消除随机波动，从而建立更准确的负载预测模型。

定义 3 负载预测模型。计算负载的 d 阶差分序列中，通过消除样本产生的随机波动，并将式(5)代入式(4)，可得预测 t 时刻的负载取值为

$$f_t^d=\sum_{i=1}^{p}\lambda_i f_{t-i}^d+\sum_{i=1}^{q}\theta_i\varepsilon_{t-i}+\varepsilon_0+\mu_t \tag{6}$$

其中，f_{t-i}^d 为 $t-i$ 时刻的负载取值；ε_{t-i} 为 $t-i$ 时刻负载的随机波动；λ_i、θ_i 和 μ_t 为待求解的参数，一般通过最大似然估计求解参数，确定预测模型。

由定义 3 可知，在负载预测模型中，λ_i、θ_i 和 μ_t 可通过最大似然估计等方法计算得出，而三元组(d, p, q)则需要在建立模型之前通过外部输入确定。其中，d 表示对负载序列进行差分计算的阶数，p 表示当前时刻的负载取值与之前 p 时刻的取值有相关关系，q 表示当前时刻的负载波动与之前 q 时刻负载波动有相关关系。本文通过自相关系数及偏自相关系数，确定三元组(d, p, q)取值。

设计算负载的 d 阶差分序列中，t 时刻的负载 f_t^d 与 $t-k$ 时刻负载 f_{t-k}^d 的自相关系数为 ρ_k，则 t 时刻与 $t-k$ 时刻负载的偏自相关系数为

$$\mathrm{PACF}(k+1,k+1)=\varphi_{k+1,k+1}=\frac{\rho_{k+1}-\sum_{j=1}^{k}\varphi_{k,j}\rho_{k-j+1}}{1-\sum_{j=1}^{k}\varphi_{k,j}\rho_j} \tag{7}$$

其中，

$$\varphi_{k,j}=\varphi_{k-1,j}-\varphi_{k-1,k-1}\varphi_{k-1,k-j},\ j=1,2,\cdots,k \tag{8}$$

由式(3)可知，负载的自相关系数用于描述 f_t^d 和 f_{t-k}^d 之间的相关关系。但事实上，这 2 个负载取值的相关性会受到中间样本 $f_{t-1}^d,f_{t-2}^d,\cdots,f_{t-k+1}^d$ 取值的影响。偏自相关系数是在剔除这些样本取值影响的前提下，单纯描述负载 f_t^d 和 f_{t-k}^d 的相关关系。因此，在一定的置信区间下观察 ρ_k 和 $\varphi_{k,k}$ 的取值，就可以确定参数 p 和 q 的理想取值范围。

此外，为了描述预测负载与真实负载的偏离程度，得到尽可能接近真实负载的预测值，假设真实的负载序列为 $f_t\in F$，预测得到的负载取值序列为 $f_t'\in F'$，则负载的预测误差为

$$\delta=\frac{\sum_{f_t\in F}\left|f_t-f_t'\right|}{\sum_{f\in F}f_t} \tag{9}$$

由式(9)可知，预测误差描述了预测值与真实值之前的差别。

参数选择依据如表 1 所示。相关系数收敛于 0 表示该系数基本趋于稳定并无限接近于 0；相关系数满足置信区间，表示在该置信区间内样本的相关

系数都满足设定的取值范围。置信区间描述了在特定区间范围内的样本都满足表 1 设定的取值条件，置信区间过大会导致计算时间开销过高且出现过拟合的现象，置信区间过小出现欠拟合导致预测的准确性降低。综上所述，通过在平稳的负载序列上计算样本的自相关系数和偏自相关系数作为参数选择的依据，就可以确定参数 p 和 q 的理想取值范围。

表 1　参数选择依据

模型(参数)	自相关系数	偏自相关系数
负载回归模型(p)	p 阶后取值收敛于 0	p 阶后满足置信区间
负载平均模型(q)	q 阶后满足置信区间	q 阶后取值收敛于 0
负载预测模型(p, q)	q 阶后取值收敛于 0	p 阶后取值收敛于 0

3.3　资源判定模型

在负载预测模型的基础上，为了定位预测网络中可能出现的资源瓶颈与资源过剩，作为提出弹性资源调度算法的依据，需要进一步定义预测网络的划分和最小节点。预测网络划分如图 3 所示。

设预测网络 $G=(V, E)$，s 是预测网络的源点，t 是汇点。则该预测网络的一个划分 $D=(X, Y)$将节点集 V 分为 2 个集合 X 和 Y，其中 $Y=V-X$，使 $s\in X$，$t\in Y$，且 $X\bigcap Y=\varnothing$，$X\bigcup Y=V$。对于作业拓扑结构中的任意算子 O，$\forall v_i,v_j\in O$ 有 $v_i,v_j\in X$ 或 $v_i,v_j\in Y$，即同一个算子的不同实例不横跨任意一个划分。该划分 $D=(X, Y)$的容量值为

$$c(D)=c(X,Y)=\sum_{v_i\in X}\sum_{v_j\in Y}c(v_i,v_j) \tag{10}$$

且下一时刻该划分的预测流量值为

$$f'(D)=f'(X,Y)=\sum_{v_i\in X}\sum_{v_j\in Y}f'(v_i,v_j) \tag{11}$$

由此可知，预测网络中每个算子对应唯一的划分，表示集群中可能存在的资源瓶颈或资源过剩。

定义 4　最小节点。设预测网络 $G=(V, E)$有一个划分 $D=(X, Y)$，其中，$v_i\in Y$，则 v_i 的容量为所有对应输入边容量的累加和，记为

$$c(v_i)=\sum_{v_j\in V}c(v_j,v_i) \tag{12}$$

其中，$c(v_j, v_i)$是 v_i 的输入边(v_j, v_i)的容量值。在划分 $D=(X, Y)$中容量值最小的一个节点称为 D 的最小节点。

由定义 4 可知，预测网络的划分代表了集群中可能存在的资源瓶颈和资源过剩，如果一个划分代表的算子中存在资源过剩，则该划分的最小节点是非必要的计算节点。通过迁移负载并删除该节点，可以优化集群的资源配置，减少不必要的资源开销。因此，通过预测负载的变化趋势并判定资源与负载的数学关系，可以准确定位集群的资源瓶颈、资源过剩以及非必要节点。

假设预测网络 $G=(V, E)$存在一个划分为 $D=(X, Y)$，且存在一个节点 $v_i\in Y$ 是 D 的最小节点。假设当前时刻的预测流量为 f_t'，接下来 2 个时刻的预测流量分别为 f_{t+1}'和 f_{t+2}'。若存在 $f'(X,Y)\geqslant\alpha c(X,Y)$ 且 $f_t'<f_{t+1}'<f_{t+2}'$，说明预测计算负载大于设定的阈值，且预测得出计算负载呈持续上升趋势，D 会因为计算资源不足而成为集群的性能瓶颈，因此需要

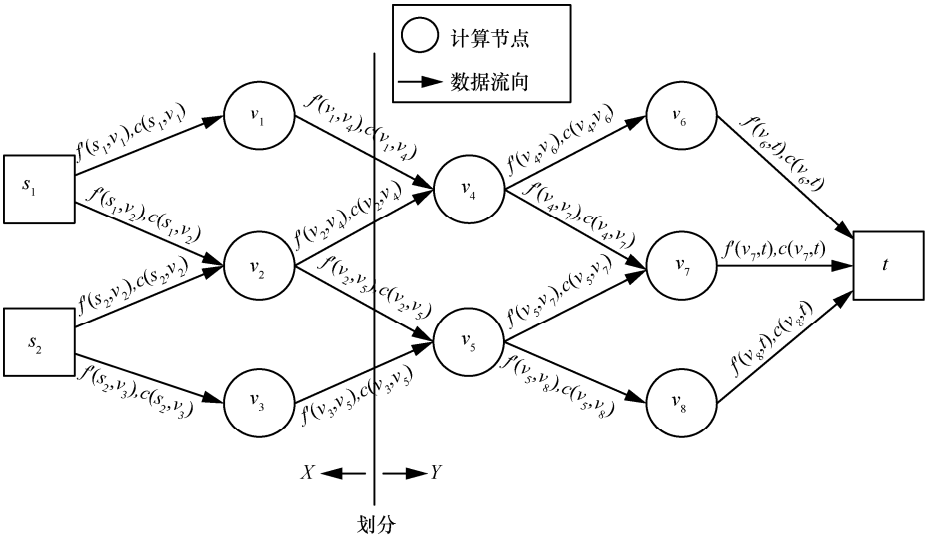


图 3　预测网络划分

通过增加计算节点，扩大 D 所对应算子的并行度，以保证集群性能。反之，若存在 $f'(X,Y)<\alpha\left(c(X,Y)-c(v_i)\right)$ 且 $f'_t>f'_{t+1}>f'_{t+2}$，则 D 中存在资源过剩，由于 v_i 是 D 的最小节点，$c(X,Y)-c(v_i)$ 表示在 D 中删除最小节点后剩余节点的计算能力，当 $f'(X,Y)<\alpha\left(c(X,Y)-c(v_i)\right)$ 时，说明删除 v_i 后，剩余节点的计算能力仍然大于设定的负载阈值，且预测得出计算负载呈持续下降趋势，因此节点 v_i 是 D 中冗余的计算资源，通过删去 v_i 可以优化集群的资源配置，避免出现资源浪费。由此可知，通过定义预测网络的划分和最小节点，设立判定集群资源配置的条件作为制定弹性资源调度计划的依据，并通过在线负载迁移算法可以提高弹性资源调度策略的执行效率。

4 在线资源调度策略

在第 3 节建立相关模型的基础上，本节提出基于负载预测的在线弹性资源调度策略，通过预测计算负载的变化趋势，定位集群的资源瓶颈与资源过剩，并通过在线负载迁移实现高效的弹性资源调度。该策略主要分为以下 4 个步骤，具体执行流程如图 4 所示。

步骤 1 建立负载预测模型并确定参数取值。

步骤 2 预测未来的负载变化趋势并建立预测网络。

步骤 3 定位集群中存在的资源瓶颈与资源过剩，制定相应的弹性资源调度计划。

步骤 4 执行弹性资源调度计划，在线迁移计算负载与状态数据。

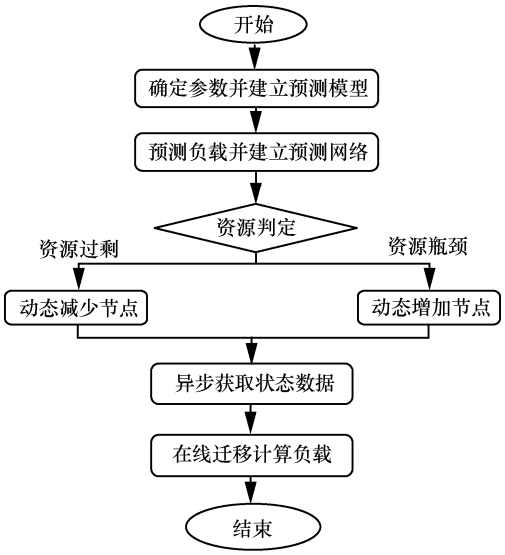


图 4 LPERS-Flink 策略执行流程

4.1 负载预测算法

为了实现集群规模对负载变化的及时响应，需要通过负载预测模型建立不同时刻计算负载之间的函数关系，根据过去的负载变化规律预测未来的变化趋势，从而根据资源与负载的数学关系定位集群的资源瓶颈和资源过剩。

由 3.2 节可知，在负载预测模型中，参数三元组 (d, p, q)的取值对模型的建立非常重要，选择合适的参数能够有效提高预测的准确性，降低模型的复杂度。本节通过衡量负载样本序列的平稳性，确定差分计算的次数求得 d 的取值；通过计算自相关系数 ρ_k 和偏自相关系数 $\varphi_{k,k}$，以表 1 为依据确定 p 和 q 的取值范围。为了在该范围内进一步确定参数的最优取值，需要通过贝叶斯信息准则（BIC, Bayesian information criterion）衡量预测的准确性和模型的复杂度，找到预测准确性最高且复杂度最低的预测模型。假设在一段时间内，以固定的时间间隔采集负载样本的序列为 F，则贝叶斯信息准则的计算式为

$$\mathrm{BIC}=k\ln(n)-2\ln(L) \tag{13}$$

其中，$k=p+q$ 为待求解参数的数目，$n=|F|$为负载序列中的样本总数，L 为预测函数（如式(6)所示）。由此可知，BIC 表示预测准确性与模型复杂度之间的权衡，在参数范围内寻找能够使 BIC 达到最小值的参数组合，即预测准确性最高且模型复杂度最低的参数取值，作为负载预测模型的最优参数。

接下来，通过建立预测函数，求解参数组 λ_i 和 θ_i 的取值，验证模型的准确性并不断执行预测，将作业拓扑的流网络转化为预测网络模型。根据上述思想，负载预测算法的执行过程如算法 1 所示。

算法 1 负载预测算法

输入 负载样本序列 F, 流网络 G

输出 预测网络 G'

1) F^d = difference(F, d); /*计算 d 阶差分使样本序列平稳*/

2) ρ_k=ACF(F); /*根据式(3)计算自相关系数*/

3) $\varphi_{k,k}$=PACF(F); /*根据式(7)计算偏自相关系数*/

4) range$_{p,q}$ = getRange(ρ_k , $\varphi_{k,k}$);

5) foreach p, q in range$_{p,q}$

6) (p, q) = minBIC(p, q); /*在参数范围内选择使 BIC 指数最小的 p, q 的组合*/

7) end foreach

8) model = ARIMA(d, p, q);　/*建立式(6)所示的负载预测模型*/

9) if　!verifyModel(model) /*验证预测准确性*/

10)　goto 1;

11) end if

12) foreach $(v_i, v_j) \in G.E$

13)　$(v_i, v_j).f'$= model.predict(v_i, v_j); /*通过预测下一时刻的负载取值，将流网络转化为预测网络*/

14) end foreach

15) return G'

算法 1 首先通过计算差分提高样本的平稳性（第 1)行）。其次，通过计算自相关系数和偏自相关系数确定参数的取值范围（第 2)行～第 4)行），并通过贝叶斯信息准则确定参数的最优取值（第 5)行～第 7)行）。再次，建立基于 ARIMA 的预测模型并验证预测的准确性（第 9)行～第 11)行）。最后，预测每条边上的未来负载，将流网络转化为预测网络模型（第 12)行～第 15)行）。

在时间复杂度方面，建立和验证模型的时间复杂度均为 $O(1)$，确定最优参数时需要遍历其取值范围内的所有值，复杂度为 $O(pq)$，建立预测网络的复杂度为 $O(|E|)$。因此算法 1 的时间复杂度为 $T(n)=O(pq+|E|)$。通常 p 和 q 的取值不超过 10，预测网络中边的数目不超过 500，因此算法 1 的时间复杂度较低。

4.2　在线资源调度算法

负载预测算法通过预测未来负载的变化趋势，将流网络模型转化为预测网络。此外，通过建立资源判定模型，在预测网络中定位集群的资源瓶颈与资源过剩，从而制定相应的弹性资源调度计划。

由 3.3 节可知，根据资源判定模型依次检查预测网络中的每一个算子，当算子对应的划分满足 $f'(X,Y) \geqslant \alpha c(X,Y)$，且预测负载持续上升时，该算子成为集群的资源瓶颈，需要增加节点并扩大算子的并行度。反之，当算子对应的划分满足 $f'(X,Y) < \alpha\left(c(X,Y) - c(v_i)\right)$，且预测负载持续下降时，该算子存在资源过剩，需要减去节点 v_i 并缩小算子的并行度，从而避免出现资源浪费的现象。

根据上述判定规则，在线资源调度算法的执行过程如下。

算法 2　在线资源调度算法

输入　集群拓扑结构 T, 负载序列 F

输出　完成资源调度后的预测网络 G'

1) G= buildFlowNetwork(T, latencyConstrain); /*调用文献[35]的构建算法，构建流网络*/

2) G' = predicLoad(F, G');　/*调用算法 1 获取预测网络*/

3) G' = maximizeFlow(G', f'_{t+1}); /*调用文献[38]的最大流算法优化负载分配*/

4) foreach D in G'

5)　$O = D$.operator;

6)　if $f'(X,Y) \geqslant \alpha c(X,Y)$ and $f'_t < f'_{t+1} < f'_{t+2}$ /*判定资源瓶颈并动态增加节点*/

7)　v = pool.getNode();

8)　O. addInstance(v);

9)　online_migration(O, pool); /*调用算法 3 进行在线负载迁移*/

10)　else if $f'(X,Y) < \alpha\left(c(X,Y) - c(v_i)\right)$ and $f'_t > f'_{t+1} > f'_{t+2}$ /*判定资源过剩并动态减少节点*/

11)　$v = D$.operator.removeInstance(v_i);

12)　online_migration(O, pool); /*调用算法 3 进行在线负载迁移*/

13)　v.shutdown();

14)　end if

15) end foreach

16) return G'

在线资源调度算法需要基于之前的基础工作来实现。首先，调用文献[35]的构建算法构建流网络模型（第 1) 行），并调用算法 1 将其更新为预测网络（第 2) 行）。其次，调用文献[38]的最大流算法优化负载分配（第 3) 行），最大化利用现有的计算资源。再次，根据资源判定模型定位集群的资源瓶颈与资源过剩，分别对资源瓶颈增加节点（第 6) 行～第 9) 行），对资源过剩减少节点（第 10) 行～第 13) 行），从而优化集群的资源配置。最后，调用在线负载迁移算法，完成在线弹性资源调度（第 9) 行和第 12) 行）。

在时间复杂度方面，流网络构建算法[35]的时间复杂度为 $O(|V|+|E|)$，最大流算法[38]的时间复杂度为 $O\left(|E|\left|f'_{t+1} - f_t\right|\right)$，负载预测算法的时间复杂度为 $O(pq+|E|)$，判定集群资源利用情况和制定弹性资源

调度计划的时间复杂度均为 $O(1)$，其中，线性级的时间复杂度可忽略。综上所述，在线资源调度算法的整体时间复杂度为

$$T(n)=O\left(|E|\left|f'_{t+1}-f_t\right|+pq+|V|+2|E|\right)=$$
$$O\left(|E|\left|f'_{t+1}-f_t\right|+pq\right) \tag{14}$$

实验结果表明，算法 2 的时间复杂度在合理可接受的范围内，且能够有效提升集群的性能。

传统的离线资源调度策略中，资源和任务的调度总是伴随着一定的时间开销，离线的任务、负载和状态数据迁移过程中作业有短暂的停滞，引发数据堆积和时延升高的问题。为了解决这一问题，本文提出在线负载迁移算法，从而实现高效、实时的弹性资源调度。

4.3 在线负载迁移算法

为了解决离线弹性资源调度策略存在数据堆积和时延升高的问题，本文提出在线负载迁移算法。通过同步执行任务、数据备份与缓存、异步状态数据拉取的方式，结合 Flink 平台现有的检查点策略，实现在线负载迁移技术。在线负载迁移过程如图 5 所示。

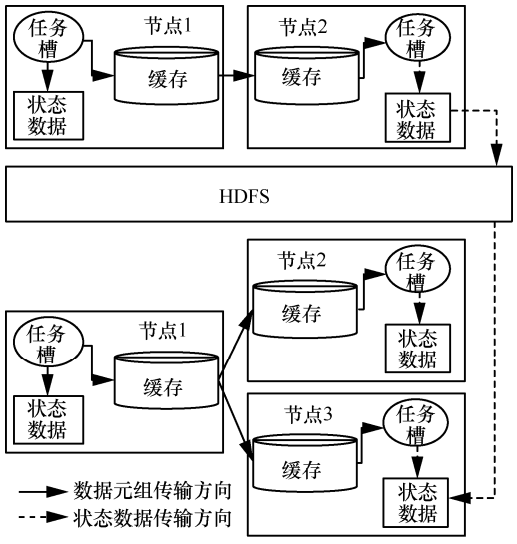


图 5 在线负载迁移示意

当算子从一个并行实例增加到 2 个时，在每个计算节点中添加数据缓冲区（buffer），用于临时存储待处理的数据。完成第一个 Checkpoint 后，节点的状态数据已经存储在 Hadoop 分布式文件系统（HDFS, Hadoop distributed file system）中。在下一次 Checkpoint 开始前，上游节点将待处理的数据复制 2 份分别发送至 2 个下游节点，原工作节点（图 5 中的节点 2）继续处理数据，实时产生计算结果并发送给下游节点。同时，新增节点（图 5 中的节点 3）以异步的方式从 HDFS 中拉取对应的状态数据，并通过同步处理缓冲区中的数据，执行状态的合并，但不输出计算结果。完成下一次 Checkpoint 后，上游节点将待处理的数据发送至新增工作节点，新增节点执行任务并向下游节点输出计算结果，从而实现用户无感知的透明负载迁移。具体步骤如算法 3 所示。

算法 3 在线负载迁移算法

输入 执行弹性资源调度的算子 Operator; 节点资源池 pool

输出 完成资源调度后的预测网络 G'

1) G'.waitForCheckpoint(); /* 执行第 1 次 Checkpoint*/

2) foreach O in Operator

3) v = pool.getNode();

4) master.remappingState(Zookeeper); /*重新计算状态数据到节点的映射关系*/

5) O.source.duplicateSend();/*上游节点同步发送 2 份待处理数据*/

6) v.buffer(data);

7) v.getState(HDFS); /*从 HDFS 中获取对应的状态数据*/

8) G'.waitForCheckpoint(); /*执行第 2 次 Checkpoint*/

9) O.source.singleSend();/*将待处理数据发送至新增节点*/

10) end foreach

11) return G'

算法 3 描述了动态增加节点时的负载迁移过程。对于每个需要执行弹性资源调度的算子，首先从资源池中获取一个计算节点（第 3)行），修改 Zookeeper 中状态数据到节点的映射关系（第 4)行）。然后，执行如图 5 所示的数据并行发送与缓存、HDFS 异步数据获取（第 5)行～第 7)行）。最后，在完成下一次 Checkpoint 之后，将上游发送的计算负载切换到新的计算节点中（第 8)行～第 9)行）。这样，集群在两次 Checkpoint 之间完成了弹性资源调度，且不影响正常的数据处理，调度过程中保证了计算的实时性。此外，动态减少节点与增加节点的过程基本类似，其区别在于，动态增加节点时，先从资源池获取节点，再迁移负载；动态减少节点时，先迁移负载，再向资源池归还节点。

在时间复杂度方面，首先，负载迁移是一个分布式算法，上述过程由多个节点并行完成，并由 JobManager 统一部署和监控，其时间开销很低。其次，由于算法遍历拓扑结构中每一个算子，其时间复杂度与拓扑结构中算子的数目相关，即 $T(n)=O(|G'.O|)$。实验证明，在线负载迁移算法有效降低了 LPERS-Flink 策略的时间开销，针对性解决了数据堆积和时延升高的问题。

4.4　算法实现与部署

在 Flink 的基本架构中，一个完整的流式作业通常从 Kafka 等数据源读取数据，由不同的工作节点协同完成计算，并将计算结果写入 HDFS 等存储系统中，通过 Checkpoint 机制将节点的状态数据存储在 HDFS 中。为了实现 LPERS-Flink 策略，需要监控节点的计算负载，预测负载的变化趋势，制定弹性资源调度计划并存储在 Zookeeper 中，最后通过在线负载迁移实施调度计划。

为了实现 LPERS-Flink，需要对 Flink 的基本架构进行改进，改进后的集群主要包含以下模块：数据持久化模块，用于存储待处理的数据或计算结果；数据处理模块，由 Flink 集群承担，主要依据用户定义的业务逻辑完成数据处理；负载预测模块，主要负责采集计算负载并预测变化趋势，形成预测网络模型并存储在资源协调器中；资源协调模块，负责同步各节点的执行状态，存储预测网络及资源调度计划。LPERS-Flink 部署架构如图 6 所示。

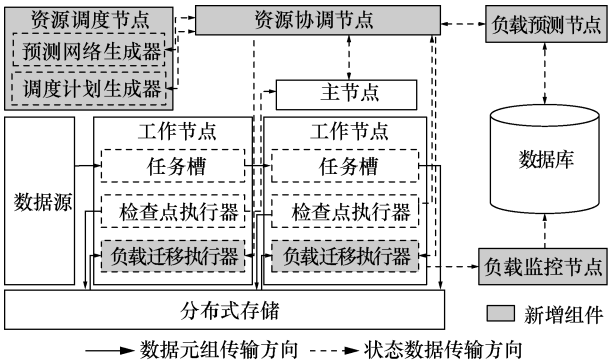


图 6　LPERS-Flink 部署架构

如图 6 所示，LPERS-Flink 在 Flink 原系统的基础上分别加入了负载监控节点、负载预测节点、资源协调节点、资源调度节点和负载迁移执行器。

1) 负载监控节点。负责监控和采集节点的计算负载，即节点输入数据的速率，并写入 MySQL 数据库。

2) 负载预测节点。执行算法 1，根据 MySQL 数据库中的负载序列，预测计算节点的负载变化趋势，形成预测网络并写入资源协调节点。

3) 资源协调节点。由 Zookeeper 集群承担，存储节点的元数据信息、预测网络模型以及资源调度计划。

4) 资源调度节点。执行算法 2，基于资源判定模型确定集群资源瓶颈与资源过剩，制定资源调度计划并写入资源协调节点。

5) 负载迁移执行器。执行算法 3，根据资源协调节点中的资源调度计划，从 HDFS 中拉取节点对应的状态数据并完成负载迁移，从而实现在线资源调度。

5　实验与分析

为了验证 LPERS-Flink 策略的有效性，本文对 LPERS-Flink 与 Flink v1.6.0 原系统及相关研究成果进行对比实验。在相同的配置环境下分别执行 3 个代表不同作业类型的标准 Benchmark，验证了负载预测的准确性和参数取值的合理性，讨论了 LPERS-Flink 策略的优化效果和执行开销。

5.1　实验环境

实验搭建的集群由 23 台同构的 PC 机组成。其中，Flink 集群包含一个 JobManager 节点、6 个 TaskManager 节点；资源池中包含 4 个 TaskManager 备用节点，在需要弹性增加节点时启动并部署计算任务；其他相关组件包括 3 个节点构成的 Hadoop 集群、3 个节点构成的 Kafka 集群和 3 个节点构成的 Zookeeper 集群。另外，由一个节点执行负载监控，一个节点执行负载预测，一个节点负责生成预测网络和资源调度计划。其中，每个节点的硬件配置参数和软件配置参数分别如表 2 和表 3 所示。

表 2　节点硬件配置参数

配置项	配置参数
CPU	Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz
内存	4 GB DDR3 1 600 MHz
硬盘	1 TB, 7 200 转/分
网络	100 Mbit/s

为了使集群达到最优性能，根据现有的软硬件环境，对 Flink 的相关配置参数进行了调整，其中重要的配置项及其参数值如表 4 所示。关于预测模

型的重要参数(d, p, q)的取值，在 5.2 节进行了详细介绍。

表 3 节点软件配置参数

配置项	配置参数
OS	CentOS 6.5
JDK	jdk1.8.0-181-Linux-X64
Apache Flink	1.6.0
Apache Hadoop	2.7.4
Apache Kafka	2.10-0.8.2.0
Apache Zookeeper	3.4.10
MySQL	5.7.27
MATLAB	R2014a (8.3.0.532) 64 bit

表 4 性能参数配置

配置项	参数值	说明
JobManager.heap.size/MB	2 048	主节点内存
TaskManager.heap.size/MB	2 048	工作节点内存
TaskManager.numberOfTaskSlots	2	节点线程数目
high-availability	Zookeeper	开启 HA 模式
state.backend	rocksdb	状态数据存储
state.backend.incremental	true	增量式快照
TaskManager.network.memory.fraction	0.2	缓冲区大小
TaskManager.network.memory.max/MB	500	缓冲区上限
TaskManager.memory.segment-size	32 768	内存分块大小

根据 3.2 节可知，在负载预测模型中，置信区间的设定对预测的准确性和预测算法的执行效率非常重要，为了确定合适的置信区间取值，实验分别在不同的置信区间下，根据式(9)计算负载的预测误差与预测时间，实验结果如表 5 所示。因此，实验选取的置信区间为[0.95, 1]，在该置信区间下的预测误差 δ=0.035，即 δ<0.05，满足预测的准确性要求，同时避免了过高的预测时间开销。

表 5 不同置信区间下的预测误差与预测时间

置信区间	预测误差	预测时间/s
0.80	0.107	1.3
0.85	0.083	2.5
0.90	0.057	3.1
0.95	0.035	3.3
1.00	0.034	7.4

文献[35]提出的 FAR-Flink（flow network based auto rescale strategy for Flink）策略是本文的前提和基础性工作。文献[36]提出的 EN（elastic nephel）策略与本文的研究目标紧密相关。为了验证 LPERS-Flink 策略的优化效果，实验将 LPERS-Flink 与 Flink 原系统（Flink v1.6.0）、FAR-Flink 和 EN 进行对比实验，在 4 个平台上分别执行 WordCount、TwitterSentiment 以及 Streaming-Benchmark 这 3 种代表不同作业类型的典型 Benchmark，分析了 LPERS-Flink 策略的优化效果和执行开销。

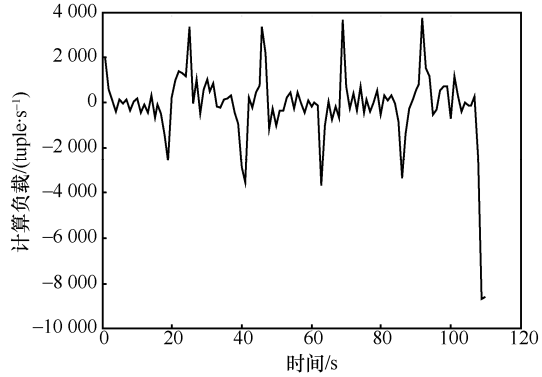
5.2 预测算法参数与性能

为了使用 ARIMA 模型进行负载预测，根据未来的负载变化趋势进行弹性资源调度，需要首先确定差分次数 d 的取值，再通过计算自相关系数和偏自相关系数确定 p 和 q 的取值，从而得到模型的参数三元组(d, p, q)，并建立负载预测模型。

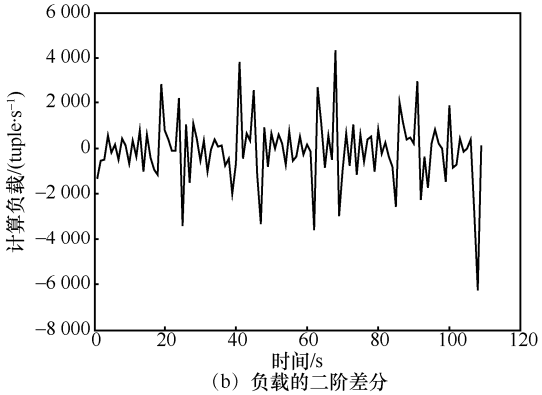
本节实验模拟了 Twitter 公司线上业务中负载剧烈波动的现象，TwitterSentiment 作业中建立负载预测模型过程的差分计算结果如图 7 所示。从图 7 中可以看出，负载的一阶差分序列无法满足预测模型对样本的平稳性要求，且序列中存在个别的奇异值。通过计算负载的二阶差分，负载序列在一段时间内基本趋于平稳，且奇异值明显减少，负载的自相关系数 ρ_k=0.68，满足预测模型对样本的平稳性要求。因此，可确定预测模型的第一个参数 d=2。

负载相关系数如图 8 所示，根据式(3)和式(7)分别计算了负载样本序列的 ACF 和 PACF 取值，反映了负载序列中不同时刻的样本之间的相关关系。由图 8 可知，在不考虑奇异值的前提下，ACF 在 2 阶之后满足置信区间，在 8 阶之后收敛于 0，因此 p 的取值范围为[2, 8]；PACF 在 2 阶之后满足置信区间，在 3 阶之后收敛于 0，因此 q 的取值范围为[2, 3]。

通过算法 1 计算 BIC 指数确定参数三元组(d, p, q)的最优取值为(2, 5, 3)，从而建立负载预测模型，采集连续 20 s 的预测负载和真实负载，结果如图 9 所示。图 9 反映了一段时间内，负载预测值与真实值的偏离情况，为了更直观地描述负载预测算法的准确性，将图 9 中关键时间点的负载预测值和真实值整理如表 6 所示。

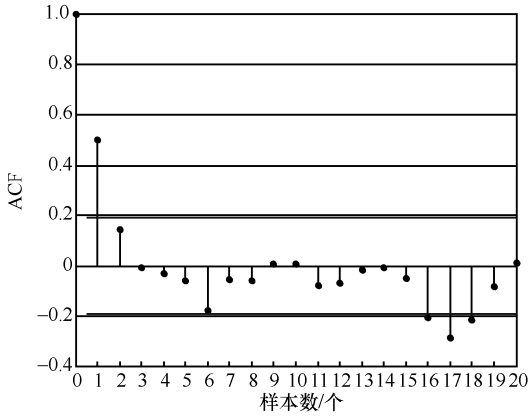


（a）负载的一阶差分

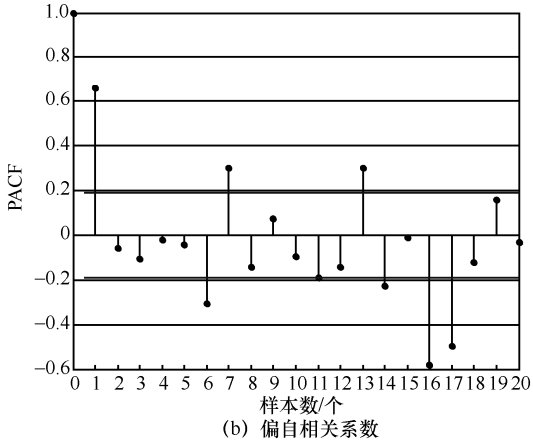


（b）负载的二阶差分

图 7　负载的差分计算结果



（a）自相关系数



（b）偏自相关系数

图 8　负载相关系数

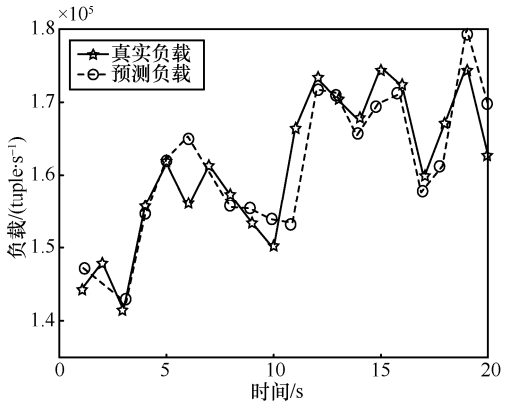


图 9　预测负载与真实负载

表 6　　预测负载偏差取值

时间/s	预测负载/(tuple·s^{-1})	真实负载/(tuple·s^{-1})
0	144 356	147 194
5	161 257	162 604
10	149 864	153 857
15	174 160	169 787
20	162 609	169 732

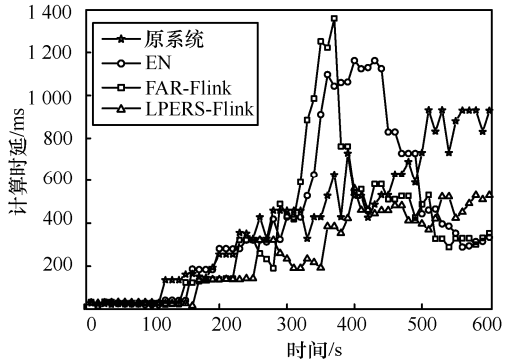
根据式(9)计算可得，在 0～20 s，负载的预测误差 δ=0.035，即 δ<0.05，且预测负载和真实值的变化趋势基本一致，预测结果与真实取值足够接近，预测负载能够反映未来负载的变化规律和近似取值，能够满足弹性资源调度策略对负载预测算法的准确性要求。因此，LPERS-Flink 策略能够以预测负载为依据制定相应的弹性资源调度计划，且参数(d, p, q)的取值是合理的。

5.3　LPERS-Flink 性能测试

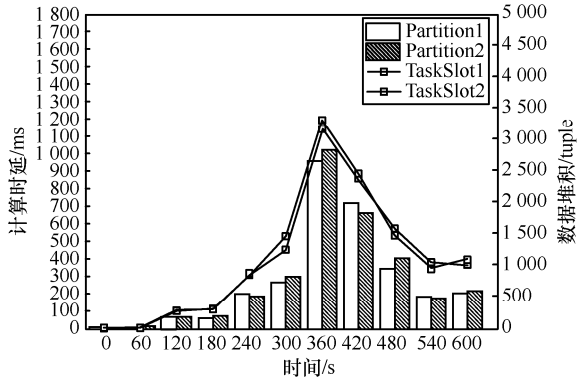
WordCount 是用于统计英文文本中单词频数的作业，作为 CPU 密集型的标准 Benchmark，其执行过程中消耗大量的 CPU 资源。实验在 4 个平台上使用相同的配置参数分别执行作业，每隔 10 s 采集一次计算时延及源点的数据堆积情况，WordCount 作业执行效率对比如图 10 所示。

由图 10 可知，LPERS-Flink 在 CPU 密集型的作业中有较明显的优势，随着计算负载的波动上升，原系统出现性能瓶颈，计算时延明显上升。EN 和 FAR-Flink 在弹性资源调度过程中，均存在调度滞后和开销过大的问题。然而，LPERS-Flink 通过在线负载迁移实现实时弹性资源调度，预先增加计算资源以应对负载的变化，其计算时延存在轻微的波动，但没有明显上升的趋势。在数据堆积方面，FAR-Flink 集群在弹性资源调度过程中出现了明显的性能下降，导

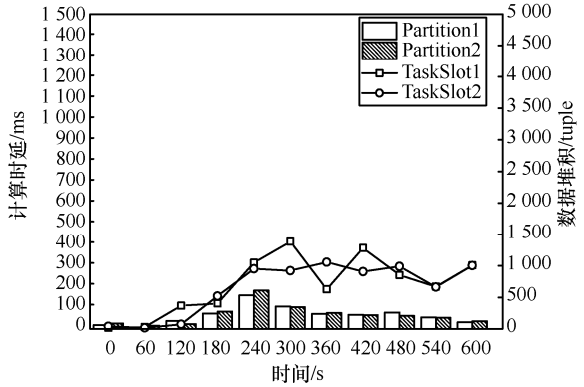
致数据堆积显著上升（300～360 s）；EN 由于调度过程中性能下降，导致计算时延上升；而 LPERS-Flink 根据预测负载提前改变集群规模，及时处理源点发出的数据，有效缓解数据堆积的问题，同时通过在线负载迁移算法实现对用户透明的弹性资源调度。



（a）计算时延对比



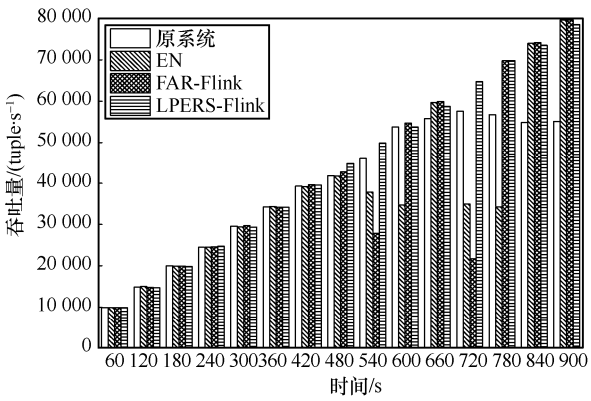
（b）FAR-Flink性能指标



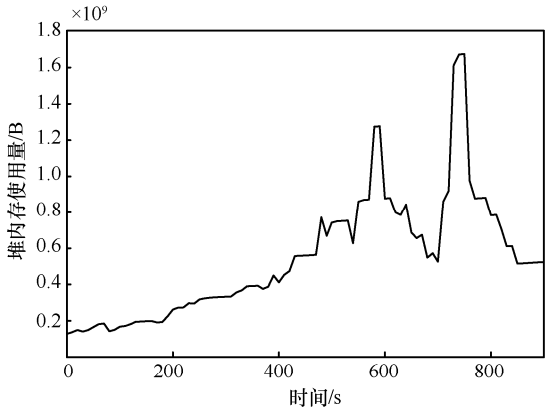
（c）LPERS-Flink性能指标

图 10 WordCount 作业执行效率对比

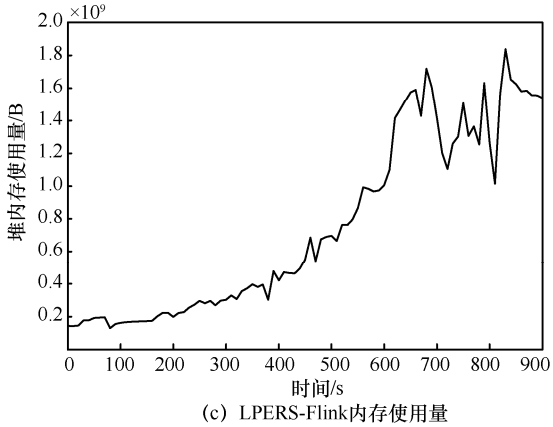
TwitterSentiment 是 Twitter 公司开发的根据用户发布的推文进行实时情感分析的作业，是实际应用场景中的标准 Benchmark，其计算过程中产生较复杂的状态数据，占用大量的内存资源。实验监测集群的吞吐量和节点的内存占用情况，TwitterSentiment 作业执行效率对比如图 11 所示。



（a）吞吐量对比



（b）FAR-Flink内存使用量



（c）LPERS-Flink内存使用量

图 11 TwitterSentiment 作业执行效率对比

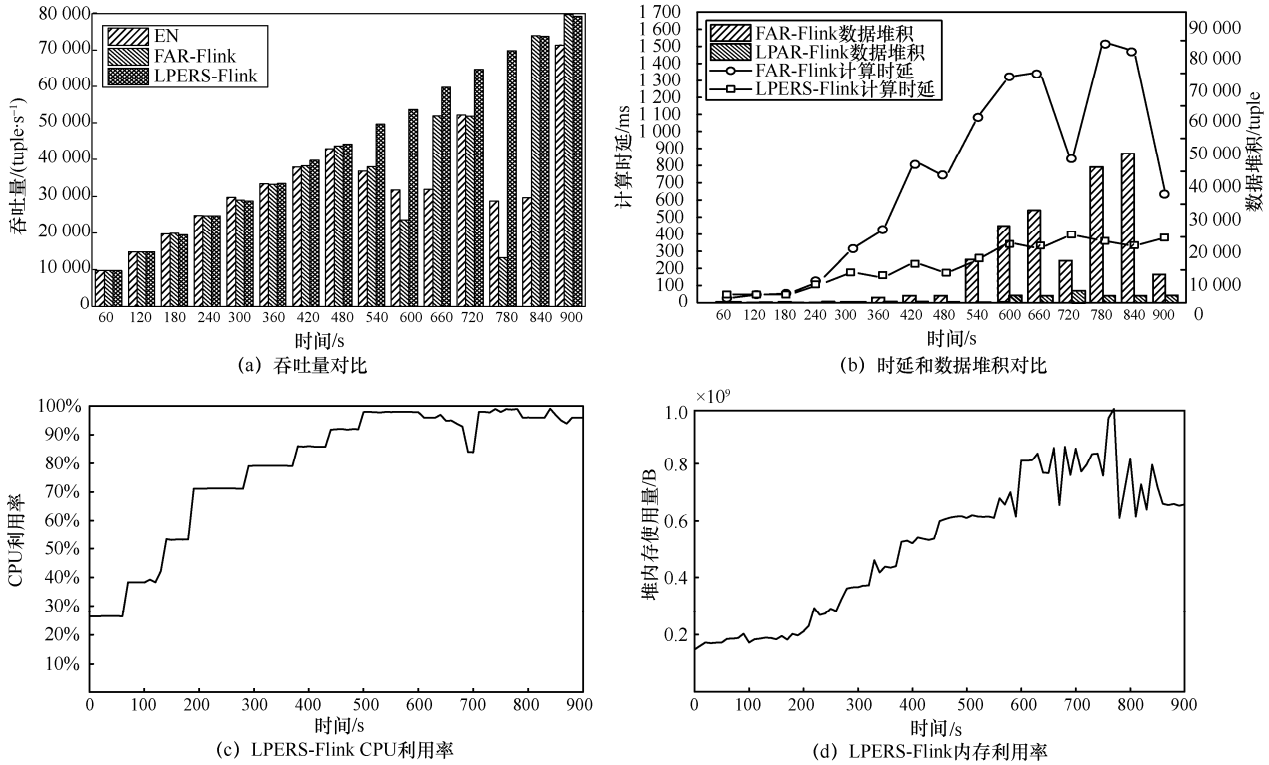
由图 11 可知，随着计算负载的持续上升，原系统遇到了内存资源的瓶颈，其吞吐量最高只能达到 57 563 tuple/s。EN 和 FAR-Flink 分别在 660～720 s 触发了弹性资源调度，但这是在负载急剧上升之后才发生的调度，存在调度滞后的问题。LPERS-Flink 在 580～640 s 触发调度，通过负载预测提前感知到负载上升的趋势，通过提前触发弹性资源调度及时响应负载的变化，集群的最高吞吐量达到 78 523 tuple/s。此外，执行离线调度的时间内集群性能明显下降，无法保证计算的实时性。LPERS-Flink 通过在线负

载迁移技术降低了负载迁移的通信开销，解决了集群性能下降的问题，执行调度的过程中仍然能保证计算的实时性。然而，LPERS-Flink 在调度过程中出现内存资源利用率轻微波动的现象，这是因为增加计算节点和拉取状态数据的过程占用了一定的内存资源，但并不影响正常的数据处理过程。

Streaming-Benchmark 是 Yahoo 公司开发的对广告数据进行实时分析的作业，其计算逻辑复杂且存在大规模状态数据，因此其执行过程中消耗大量的节点 CPU 和内存资源，是实时计算领域的标准 Benchmark。实验分别采集了作业执行过程中的吞吐量、计算时延、数据堆积、节点的 CPU 利用率和内存利用率，得到如图 12 所示的实验结果。

实验结果表明，在计算复杂度高的作业中，调度滞后和负载迁移开销大的问题严重影响集群性能，无法保证计算的实时性。如图 12(a)和图 12(b)所示，随着计算计算负载的波动上升，EN 和 FAR-Flink 分别在 660～720 s 发生弹性资源调度，存在明显调度滞后的问题。而 LPERS-Flink 在 540～600 s 触发弹性资源调度，预测算法使集群提前感知到了负载急剧上升的趋势，且预测负载与实际负载基本一致，保证了调度计划的准确性。此外，在执行弹性资源调度的过程中，EN 和 FAR-Flink 均因为触发弹性资源调整，在调度过程中出现了集群性能下降，导致其吞吐量分别降低至 28 741 tuple/s 和 13 257 tuple/s，而 LPERS-Flink 并未因调度而导致集群性能明显下降，其调度过程中的吞吐量能够达到 69 856 tuple/s。因此，在这一段时间内 EN 和 FAR-Flink 都分别出现数据堆积和时延升高的问题，无法满足计算的实时性要求，而 LPERS-Flink 基本没有数据堆积，且计算时延稳定在 300 ms 左右，这充分体现了在线负载迁移算法的优越性。在资源利用方面，如图 12(c)和图 12(d)所示，LPERS-Flink 的节点 CPU 利用率和内存利用率均随着计算负载的上升而上升，且在弹性资源调度的过程中出现轻微波动的现象，但这并不影响集群的整体性能。

综上所述，实验在一种弹性资源调度策略下分别执行 3 个不同的标准 Benchmark，通过对比不同的性能指标，得出算法的优缺点及适用场景，实验结果如表 7 所示。实验结果表明，LPERS-Flink 通过提前预测计算负载，提高了集群对负载变化的感知能力；通过提前执行弹性资源调度及时响应计算



(a) 吞吐量对比　(b) 时延和数据堆积对比　(c) LPERS-Flink CPU利用率　(d) LPERS-Flink内存利用率

图 12　Streaming-Benchmark 作业执行效率对比

表 7 对比实验结果

系统名称	调度策略	优点	缺点	适用场景
原系统	默认调度策略	支持 Exactly-Once 的有状态数据流处理	无弹性资源调度策略	计算负载稳定或小幅度波动
EN	通过数学模型计算每个算子合理的并行度，并动态增加计算资源	数据迁移过程中可同时执行计算任务	数据迁移过程的时间开销较高	负载持续上升，上升幅度较大，且状态数据规模不大
FAR-Flink	先合理分配上升的计算负载，再通过流网络模型检测需要增加并行度的算子，并动态增加计算资源	准确分配计算资源，有效降低数据迁移的时间开销	数据迁移时任务有极短暂的停滞(约 2～3 s)	负载持续上升，上升幅度较大，且状态数据规模较大
LPERS-Flink	根据负载预测结果提前执行弹性资源调度，并在线迁移计算任务和状态数据	提前响应计算负载的波动变化，避免调度滞后的问题，调度过程不影响集群性能	节点的资源利用率出现轻微波动	计算负载剧烈波动，且对计算的实时性要求高

负载的波动变化，避免调度滞后的问题；通过计算任务和状态数据的在线迁移，降低了调度过程中由数据传输产生的网络资源开销，进而降低了任务调度的时间开销，LPERS-Flink 的网络传输开销低于 EN 和 FAR-Flink，有效提升了弹性资源调度策略的执行效率。

6 结束语

随着大数据流式计算的不断发展，Apache Flink 已经成为数据分析领域的通用计算平台，得到学术界和产业界的广泛关注，但集群可扩展性和可伸缩性不足的问题成为制约平台发展的瓶颈。本文提出 Flink 环境下基于负载预测的弹性资源调度策略，通过准确预测负载变化、预先执行弹性资源调度以及在线负载迁移相结合的方式，从根本上解决了集群无法及时应对负载波动的问题。本文针对有状态流式计算场景，结合 Checkpoint 机制提出的弹性资源调度策略。由于 Checkpoint 机制是分布式计算平台容错策略的主流和通用思想，除 Flink 平台外，LPERS-Flink 策略同样适用于 Apache Heron 等所有支持 Checkpoint 机制的有状态流式计算环境。

但本文算法也存在一定的局限性。首先，LPERS-Flink 策略对 MySQL 和 Zookeeper 有很强的依赖性，需要依靠外部组件的协助实现负载预测和弹性资源调度。其次，执行在线负载迁移的过程中，节点的资源利用率会出现轻微的波动，但不影响集群的整体性能。因此，未来的研究工作将主要集中于以下 3 个方面。

1) 通过研究新的负载预测算法，进一步提高负载预测的效率和准确性，作为弹性资源调度的依据。

2) 降低调度策略对 MySQL、Zookeeper 等外部组件的依赖性，实现独立的弹性资源调度。

3) 通过优化在线负载迁移算法，减少调度过程中节点资源利用率的波动，提高集群稳定性。

参考文献：

[1] 彭安妮, 周威, 贾岩, 等. 物联网操作系统安全研究综述[J]. 通信学报, 2018, 39(3): 22-34.
PENG A N, ZHOU W, JIA Y, et al. Survey of the Internet of things operating system security[J]. Journal on Communications, 2018, 39(3): 22-34.

[2] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[3] 卞琛, 于炯, 修位蓉, 等. 基于分配适应度的 Spark 渐进填充分区映射算法[J]. 通信学报, 2017, 38(9): 133-147.
BIAN C, YU J, XIU W R, et al. Progressive filling partitioning and mapping algorithm for Spark based on allocation fitness degree[J]. Journal on Communications, 2017, 38(9):133-147.

[4] 卞琛,于炯,修位蓉, 等.内存计算框架局部数据优先拉取策略[J].计算机研究与发展, 2017, 54(4): 787-803.
BIAN C, YU J, XIU W R, et al. Partial data shuffled first strategy for in-memory computing framework[J]. Journal of Computer Research and Development, 2017, 54(4): 787-803.

[5] 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[J]. 软件学报, 2014, 25(4): 839-862.
SUN D W, ZHANG G Y, ZHENG W M. Big data stream computing: technologies and instances[J]. Journal of Software, 2014, 25(4): 839-862.

[6] ALEXANDROVE A, BERGMANN R, EWEN S, et al. The stratosphere platform for big data analytics[J]. The VLDB Journal, 2014, 23(6): 939-964.

[7] 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.

[8] TOSHNIWAL A, TANEJA S, SHUKLA A, et al. Storm @Twitter[C]// The 2014 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2014: 147-156.

[9] CARBONE P, EWEN S, FÓRA G, et al. State management in Apache Flink®: consistent stateful distributed stream processing[J]. Proceedings of the VLDB Endowment, 2017, 10(12): 1718-1729.

[10] PARIS C, GYULA F, STEPHAN E, et al. Lightweight asynchronous snapshots for distributed dataflows[J]. Computer Science, arXiv Preprint, arXiv:1506.08603,2015.

[11] KULKARNI S, BHAGAT N, FU M, et al. Twitter Heron: stream processing at scale[C]//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2015: 239-250.

[12] FLORATOU A, AGRAWAL A. Self-regulating streaming systems: challenges and opportunities[C]//Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics. New York: ACM Press, 2017: 1-5.

[13] SUN D, ZHANG G, YANG S, et al. Re-stream: real-time and energy-efficient resource scheduling in big data stream computing environments[J]. Information Sciences, 2015, 319: 92-112.

[14] 蒲勇霖, 于炯, 鲁亮, 等. Storm 平台下工作节点的内存电压调控节能策略[J]. 通信学报, 2018, 39(10): 97-117.
PU Y L, YU J, LU L, et al. Energy-efficient strategy for work node by DRAM voltage regulation in Storm[J]. Journal on Communications, 2018, 39(10):97-117.

[15] SUN D, FU G, LIU X, et al. Optimizing data stream graph for big data stream computing in cloud datacenter environments[J]. International Journal of Advancements in Computing Technology, 2014, 6(5): 53.

[16] 蒲勇霖, 于炯, 鲁亮, 等. 基于 Storm 平台的数据迁移合并节能策略[J]. 通信学报, 2019, 40(12): 68-85.
PU Y L, YU J, LU L, et al. Energy-efficient strategy for data migration and merging in Storm[J]. Journal on Communications, 2019, 40(12): 68-85.

[17] ZHANG C, CHEN X, LI Z, et al. An on-the-fly scheduling strategy for distributed stream processing platform[C]//IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications. Piscataway: IEEE Press, 2018: 773-780.

[18] SHUKLA A, SIMMHAN Y. Model-driven scheduling for distributed stream processing systems[J]. Journal of Parallel and Distributed Computing, 2018, 117: 98-114.

[19] CARDELLINI V, MENCAGLI G, TALIA D, et al. New landscapes of the data stream processing in the era of fog computing[J]. Future Generation Computer Systems, 2019, 99: 646-650.

[20] TANTALAKI N, SOURAVLAS S, ROUMELIOTIS M, et al. Linear scheduling of big data streams on multiprocessor sets in the cloud[C]//IEEE/WIC/ACM International Conference on Web Intelligence. New York: ACM Press, 2019: 107-115.

[21] ESKANDARI L, MAIR J, HUANG Z, et al. T3-Scheduler: a topology and traffic aware two-level Scheduler for stream processing systems in a heterogeneous cluster[J]. Future Generation Computer Systems, 2018, 89: 617-632.

[22] SILVA V A, DE-SOUZA F R, DE-ASSUNÇÃO M D, et al. Multi-objective reinforcement learning for reconfiguring data stream analytics on edge computing[C]//Proceedings of the 48th International Conference on Parallel Processing. New York: ACM Press, 2019: 106.

[23] LOUKOPOULOS T, TZIRITAS N, KOZIRI M, et al. A pareto-efficient algorithm for data stream processing at network edges[C]//2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). Piscataway: IEEE Press, 2018: 159-162.

[24] PAGLIARI A, HUET F, URVOY-KELLER G. On the cost of acking in data stream processing systems[C]//19th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. Piscataway: IEEE Press, 2019: 14-17.

[25] ZHOU S, ZHANG F, CHEN H, et al. Fastjoin: a skewness-aware distributed stream join system[C]//2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Piscataway: IEEE Press, 2019: 1042-1052.

[26] RÖGER H, MAYER R. A comprehensive survey on parallelization and elasticity in stream processing[J]. ACM Computing Surveys (CSUR), 2019, 52(2): 36.

[27] LIU S, WENG J, WANG J H, et al. An adaptive online scheme for scheduling and resource enforcement in Storm[J]. IEEE/ACM Transactions on Networking, 2019, 27(4): 1373-1386.

[28] RUSSO G R, CARDELLINI V, PRESTI F L. Reinforcement learning based policies for elastic stream processing on heterogeneous resources[C]//Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems. New York: ACM Press, 2019: 31-42.

[29] RUSSO G, NARDELLI M, CARDELLINI V, et al. Multi-level elasticity for wide-area data streaming systems: a reinforcement learning approach[J]. Algorithms, 2018, 11(9): 134.

[30] CARDELLINI V, PRESTI F L, NARDELLI M, et al. Towards hierarchical autonomous control for elastic data stream processing in the fog[C]//European Conference on Parallel Processing. Berlin: Springer, 2017: 106-117.

[31] MEHDI B, CÉDRIC T. A fully decentralized autoscaling algorithm for stream processing applications[C]//Auto-DaSP 2019-Third International Workshop on Autonomic Solutions for Parallel and Distributed Data Stream Processing. Berlin: Springer, 2019:1-12.

[32] RUSSO G R. Self-adaptive data stream processing in geo-distributed computing environments[C]//Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems. New York: ACM Press, 2019: 276-279.

[33] MENCAGLI G, TORQUATI M, DANELUTTO M. Elastic-PPQ: a two-level autonomic system for spatial preference query processing over dynamic data streams[J]. Future Generation Computer Systems, 2018, 79: 862-877.

[34] HIDALGO N, WLADDIMIRO D, ROSAS E. Self-adaptive processing graph with operator fission for elastic stream processing[J]. Journal of Systems and Software, 2017, 127: 205-216.

[35] 李梓杨, 于炯, 卞琛, 等. 基于流网络的 Flink 平台弹性资源调度策略[J]. 通信学报, 2019, 40(8): 85-101.
LI Z Y, YU J, BIAN C, et al. Flow-network based auto rescale strategy for Flink [J]. Journal on Communications, 2019, 40(8): 85-101.

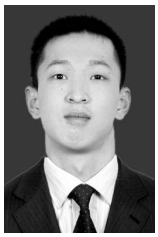
[36] LOHRMANN B, JANACIK P, KAO O. Elastic stream processing with latency guarantees[C]//2015 IEEE 35th International Conference on Distributed Computing Systems. Piscataway: IEEE Press, 2015: 399-410.

[37] SUN D, GAO S, LIU X, et al. State and runtime-aware scheduling in

elastic stream computing systems[J]. Future Generation Computer Systems, 2019, 97: 194-209.

[38] 李梓杨, 于炯, 卞琛, 等. 基于流网络的流式计算动态任务调度策略[J]. 计算机应用, 2018, 38(9): 2560-2567.
LI Z Y, YU J, BIAN C, et al. Dynamic task dispatching strategy for stream processing based on flow network[J]. Journal of Computer Application, 2018, 38(9): 2560-2567.

[作者简介]



李梓杨（1993− ），男，新疆乌鲁木齐人，新疆大学博士生，主要研究方向为分布式系统、内存计算、流式计算。

于炯（1964− ），男，北京人，博士，新疆大学教授、博士生导师，主要研究方向为网格计算、并行计算、分布式系统。

王跃飞（1991− ），男，新疆乌鲁木齐人，博士，成都大学讲师，主要研究方向为数据挖掘、机器学习。

卞琛（1981− ），男，江苏南京人，博士，广东金融学院副教授，主要研究方向为分布式系统、内存计算、绿色计算。

蒲勇霖（1991− ），男，山东淄博人，新疆大学博士生，主要研究方向为内存计算、流式计算、绿色计算。

张译天（1995− ），男，河南商丘人，新疆大学硕士生，主要研究方向为云计算、实时计算、分布式计算。

刘宇（1996− ），男，新疆克拉玛依人，新疆大学硕士生，主要研究方向为云计算、分布式计算。