基于流网络的 Flink 平台弹性资源调度策略

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

（1. 新疆大学信息科学与工程学院，新疆 乌鲁木齐 830046；2. 新疆大学软件学院，新疆 乌鲁木齐 830008；
3. 广东金融学院互联网金融与信息工程学院，广东 广州 510521；4. 中国民航大学计算机科学与技术学院，天津 300300）

摘 要：为了解决大数据流式计算平台中存在计算负载波动上升，但集群无法有效应对负载变化的问题，提出了基于流网络的 Flink 平台弹性资源调度策略（FAR-Flink）。该策略首先建立流网络模型并通过构建算法计算每条边的容量值，其次通过弹性资源调度算法确定集群性能瓶颈并制定动态资源调度计划，最后通过基于数据分簇和分桶管理的状态数据迁移算法，实施调度计划并完成节点间的高效数据迁移。实验结果表明，该策略在状态数据复杂的应用场景中有较好的优化效果，在满足计算时延约束的前提下提高了集群的吞吐量，缩短了状态数据迁移的时间。由此可见，FAR-Flink 策略有效提升了集群对负载波动的响应能力。

关键词：流式计算；资源调度；弹性集群；负载迁移；Flink

中图分类号：TP311

文献标识码：A

doi: 10.11959/j.issn.1000−436x.2019173

Flow-network based auto rescale strategy for Flink

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

1. School of Information Science and Engineering, Xinjiang University, Urumqi 830046, China
2. School of Software, Xinjiang University, Urumqi 830008, China
3. College of Internet Finance and Information Engineering, Guangdong University of Finance, Guangzhou 510521, China
4. School of Computer Science and Technology, Civil Aviation University of China, Tianjin 300300, China

Abstract: In order to solve the problem that the load of big data stream computing platform is increasing with fluctuation while the cluster was not able to rescale efficiently, the Flow-network based auto rescale strategy for Flink was proposed. Firstly, the flow-network model was set up and the capacity of each edge that was calculated by self-learning algorithm. Secondly, the bottleneck of the cluster was acquired by maximum-flow algorithm and the resource rescheduling plan was drawn up. Finally, the resource rescheduling plan was executed and the stateful data was migrated efficiently by the data migration algorithm based on the strategy of data partitioning by bulk and bucket. The experimental results show that the strategy can effectively provide performance promotion in the application with complex stateful data. It improved the throughput of the cluster and reduced the time overhead of the data migration on the premise of satisfying the latency constrain of the application, which means that the strategy promotes the scalability of the cluster efficiently.

Key words: stream computing, resource scheduling, elastic cluster, load migration, Flink

---

收稿日期：2019−03−22；**修回日期**：2019−07−24

基金项目：国家自然科学基金资助项目（No.61862060, No.61462079, No.61562086, No.61562078）；国家科技部科技支撑基金资助项目（No.2015BAH02F01）；新疆维吾尔自治区自然科学基金资助项目（No.2017D01A20）；新疆维吾尔自治区高校科研计划基金资助项目（No.XJEDU2016S106）

Foundation Items: The National Natural Science Foundation of China (No.61862060, No.61462079, No.61562086, No.61562078), The Science and Technology Support Projects of Ministry of National Science and Technology (No.2015BAH02F01), The Natural Science Foundation of Xinjiang Uygur Autonomous Region of China (No.2017D01A20), Educational Research Program of Xinjiang Uygur Autonomous Region of China (No.XJEDU2016S106)

1 引言

随着云计算、物联网、人工智能等新技术的不断发展，自动驾驶、智慧城市、智能工业等新产业和新服务模式的不断兴起和发展，人们的生活方式也变得更舒适、更便捷。根据希捷公司发布的调查结果，预测到2025年全球数据量将高达163 ZB，其中25%的数据需要被实时计算和处理，这些数据主要应用于物联网[1]和人工智能等相关领域。

随着数据规模和计算复杂度的提升，以MapReduce[2]和弹性分布式数据集（RDD, resilient distributed dataset）[3]编程模型为代表的批处理[4]模式已经无法满足应用的实时性要求，大数据流式计算应运而生。流式计算将计算任务抽象为数据流模型，具有实时性、易失性、突发性、无序性和无限性的特征[5]，能够提供大规模数据的分布式实时处理，已经分别得到学术界和产业界的广泛关注。其中，Apache Flink是目前应用最广泛的新兴流式计算平台。然而，面对不断波动变化的计算负载，Flink平台存在可扩展性和可伸缩性较差，应对负载波动的能力不足，已经成为制约平台发展的严峻而又亟待解决的问题，受到开源社区的重点关注。

针对上述问题，本文提出基于流网络的Flink平台弹性资源调度策略（FAR-Flink, Flow-network based auto rescale strategy for Flink），并将其应用于Flink平台。本文的主要贡献如下。

1) 提出基于流网络对流式计算拓扑进行建模的思想，并在此基础上提出流网络划分等概念，定义了上下游节点的划分方式，为提出弹性资源调度算法和弹性资源调度策略提供了模型的支撑。

2) 提出流网络的容量值构建算法，采用初值定义和反馈调节相结合的方式，确定流网络每条边的容量值，为保障弹性资源调度算法的优化效果奠定基础。

3) 在流网络模型的基础上提出弹性资源调度算法，在合理分配新增计算负载的同时定位集群性能瓶颈，生成弹性资源调度计划，通过扩充计算节点突破瓶颈，提高集群整体性能。

4) 提出一种状态数据分簇和分桶管理的思想，并在此基础上提出状态数据高效迁移策略，有效提高了集群执行检查点和数据恢复的效率，降低执行弹性资源调度计划的时间开销。

2 相关研究

面对大数据流式计算平台存在突发性[6]问题，现有的流式计算平台均未提供默认的弹性资源调度机制，计算平台存在可伸缩性和可扩展性不足的问题，已经得到学术界和产业界的广泛关注。目前，现有的研究成果主要基于以下4个出发点来解决该问题：1) 通过弹性资源调度策略提高集群的可伸缩性；2) 通过优化任务调度策略提高集群的性能；3) 通过优化数据分配策略提高作业的执行效率；4) 通过减小网络通信开销以降低计算时延。

通过弹性资源调度策略提高集群资源的可伸缩性，是提高集群性能最有效的方法。其中，文献[6]提出适用于Nephel[7]数据流处理平台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并行度，并通过任务迁移实现集群资源的动态伸缩，但其任务迁移过程中的网络传输开销较大。文献[8]通过监控集群的性能指标，建立针对Storm平台无状态数据流的弹性资源调度策略。文献[9]通过提出分布式弹性资源管理协议，实现集群规模对输入负载的快速响应。文献[10]通过提出高效的状态数据管理策略，实现集群在横向和纵向上的可扩展性。文献[11]提出基于动态参数优化的弹性资源数据流处理平台。此外，文献[12-16]分别从资源分配、任务调度、负载优化等不同的角度，提出数据流弹性资源调度策略。

优化任务调度策略是另一种提高集群性能的有效方法。文献[17]提出优化流式作业的通用原则，实现了透明的拓扑优化策略。文献[18]提出对流式计算拓扑的图优化策略，在提高吞吐量的同时降低时延。文献[19]通过对关键路径节点进行重分配，在降低计算时延的同时减少系统能耗。此外，文献[20]通过优化任务拓扑结构的方式，有效提高了集群性能。

优化数据分配策略是提高作业执行效率的有效方式，其中最典型的策略是负载均衡。文献[21]通过实现上下游节点算子的灵活迁移和动态链接[22]，应对内存不足造成的背压（backpressure）问题。文献[23]提出自定义的代价模型，在邻近代价阈值时启动分区映射算法[24]，实现节点间计算负载的最优分配。文献[25]将流式计算拓扑定义为流网络模型并从中寻找优化路径，从而提高集群吞吐量。此外，文献[26-27]也分别提出不同的负载均衡策略，

以提升集群的整体性能。

通过降低网络传输开销以提高传输性能，也是提高作业执行效率的有效方式。文献[28]提出根据计算任务的不同类型动态调节缓冲池的分区大小，有效降低传输开销以提高传输性能。文献[22]通过对并行度相等的上下游算子进行动态链接，有效减少了节点间的数据通信。文献[29]通过将节点内进程间网络通信转化为线程间通信，有效降低了通信开销。此外，文献[30-31]分别从不同角度降低数据传输的通信开销，提高集群性能。

其中，通过弹性资源调度策略提高集群的可伸缩性，能够从根本上解决负载波动导致集群响应能力不足的问题，但现有研究成果多针对 Storm 平台提出，受平台内核之间的差异而无法直接移植于 Flink。因此本文提出适用于 Flink 平台的弹性资源调度策略，与现有研究成果的不同之处介绍如下。

1) 基于流网络对集群拓扑结构进行抽象，采用初值定义和反馈调节相结合的方式构建模型，准确评估了节点的计算能力和集群的性能瓶颈，并在此基础上制定了最优的弹性资源调度计划。

2) 重点考虑有状态数据流的任务调度问题，提出状态数据分桶管理的思想，有效提高了状态数据的迁移效率。

3) 通过分析 Flink 平台内核的结构和特点，结合状态数据管理和检查点机制，克服平台内核之间的差异，提出了适用于 Flink 的弹性资源调度策略。

4) 在实验中，提出通过监控数据传输量评估数据迁移效率的方法，验证了状态数据迁移算法的优化效果。在此基础上，通过对比实验得出主流弹性资源调度策略的优缺点及其适用场景。

3　问题建模与分析

本节从有状态流式计算的特点出发，建立了流网络模型和状态数据管理模型。首先通过分析有向无环图（DAG, directed acyclic graph）中节点容量与流量的数学关系，建立流网络模型，为弹性资源调度算法提供了模型的支撑。此外，通过分析现有状态数据管理策略的不足，指出影响状态数据迁移效率的主要原因，为设计高效的状态数据迁移算法提供了理论依据。

3.1　流网络模型

在分布式数据流处理平台中，用户定义的计算逻辑是封装在算子中的，并由各工作节点并行化执行算子的计算逻辑，各工作节点被称为对应算子的实例。由此可知，当集群的计算资源总量和拓扑结构确定后，其所能承担的最高计算负载也随之确定，计算资源不足将导致集群性能急剧下降。

因此，为了解决集群资源不足导致的性能问题，将流式计算的拓扑结构定义为流网络模型。在该模型中，将节点能够处理的最高计算负载定义为容量 $c(v_i, v_j)$，当前时刻节点实际处理的负载定义为流量 $f(v_i, v_j)$，通过构建对应的增进网络并寻找优化路径，实现计算负载的最优分配，从而定位集群的性能瓶颈并制定相应的弹性资源调度计划。

定义 1　流网络。如图 1 所示，设集群拓扑为单源有向无环图 $G=(V, E)$，其中 $V=\{v_1,v_2,\cdots,v_n\}$ 是图中所有节点的集合，$s\in S$ 是流网络的源点，$t_i\in T$ 是汇点，$E=\{(v_i, v_j) \mid i,j\in[1,n], n=|V|\}$ 是有向边的集合，(v_i, v_j) 表示节点 v_i 与 v_j 间的传输链路。且 $\forall(v_i,v_j)\in E$ 有 $c(v_i, v_j)\geqslant 0$，表示边 (v_i, v_j) 允许数据传输速率的最大值，称为边 (v_i, v_j) 的容量；$f(v_i, v_j)$ 表示

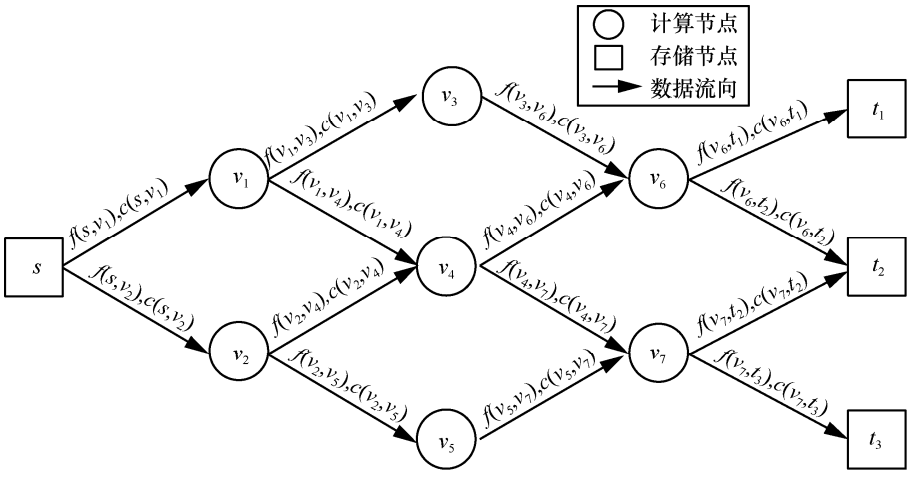


图 1　流网络模型

边(v_i, v_j)当前时刻的实际传输速率，称为边(v_i, v_j)当前的流量。此外，当前时刻从数据源点向网络输入数据速率的和记为该网络的流量，即

$$|f|=\sum_{v_i\in V} f(s,v_i) \tag{1}$$

在流网络中，容量与流量的单位均为 tuple/s，且有 $0\leqslant f(v_i, v_j)\leqslant c(v_i, v_j)$恒成立。

通过定义流网络模型，将流式计算的 DAG 拓扑转化为单源有向无环图，用容量 $c(v_i, v_j)$表示边(v_i, v_j)允许数据传输的最大值，并用流量 $f(v_i, v_j)$表示边(v_i, v_j)实际的传输速率。其中，每条边的容量值表示对应节点的处理能力和数据链路的传输能力，只有准确计算每条边的容量值才能有效评估集群的性能指标，从而实现集群资源的最优分配。因此，通过 4.1 节提出的流网络构建算法，采用数学计算与反馈调节相结合的方式，计算网络中每条边的容量大小，构建整个流网络的形态，对弹性资源调度算法的执行是至关重要的。

定义 2 增进网络。设有流网络 $G=(V, E)$，且 f 是 G 的一个流，则 $G_f=(V_f, E_f)$是流网络 G 由 f 产生的增进网络，其形态如图 2 所示，其中 $\forall v_i\in V$ 有 $v_i\in V_f$，$\forall(v_i,v_j)\in E$ 在 G_f 中对应的增进容量函数 $c_f(v_i, v_j)$为

$$c_f(v_i,v_j)=\begin{cases} c(v_i,v_j)-f(v_i,v_j), & (v_i,v_j)\in E \\ f(v_j,v_i), & (v_j,v_i)\in E \\ 0, & \text{其他} \end{cases} \tag{2}$$

其中，$c(v_i, v_j)$为原网络中边的容量函数，$f(v_i, v_j)$为原网络中边的流量函数。

由定义 2 可知，增进网络表示在原流网络中，每条边上流量提升的空间。因此在式(2)中，对于所有与原网络同向的边，其容量值为原网络中容量与流量的差值；对于所有与原网络反向的边，其容量值与原网络的流量值相等。这样，在增进网络中寻找一条从源点到汇点的无环路径，表示原流网络中提升流量的空间，即提升集群吞吐量的空间。

定义 3 优化路径。设有流网络 $G=(V, E)$，$G_f=(V_f, E_f)$是 G 由流 f 产生的增进网络。则在 G_f 中从源点 s 至任意汇点 t_i 的一条无环路径 $p:s\rightarrow v_i\rightarrow t_i$ 都是原网络 G 的一条优化路径。其中，集合 $P=\{(v_i, v_j)\mid$ 边(v_i, v_j)在路径 p 上$\}$为优化路径上边的集合，则该路径对应的递增量为

$$|f_p|=c_f(p)=\min\{c_f(v_i,v_j)\mid(v_i,v_j)\in P\} \tag{3}$$

其中，$c_f(v_i, v_j)$为边(v_i, v_j)在增进网络中的容量值。因此，将 f_p 作用于 f，得到流 f 在路径 p 上的一个递增流，记为 $f'=f\uparrow f_p$，且对应边的流量为

$$f'(v_i,v_j)=(f\uparrow f_p)(v_i,v_j)=\begin{cases} f(v_i,v_j)+|f_p|, & (v_i,v_j)\in P \\ f(v_i,v_j)-|f_p|, & (v_j,v_i)\in P \\ f(v_i,v_j), & \text{其他} \end{cases} \tag{4}$$

其中，$f(v_i, v_j)$是原网络中对应边的流量。由于递增量表示优化路径上每条边均能够提升流量的空间，因此选取该路径上增进容量的最小值。

设流网络 $G=(V, E)$的一个流为 f，$G_f=(V_f, E_f)$为 G 由流 f 产生的增进网络，路径 p 为 G_f 中的任意一条优化路径，则流 f 在路径 p 上的递增流 $f\uparrow f_p$ 也是原网络 G 的一个流，且其流量为 $|f'|=|f\uparrow f_p|=$

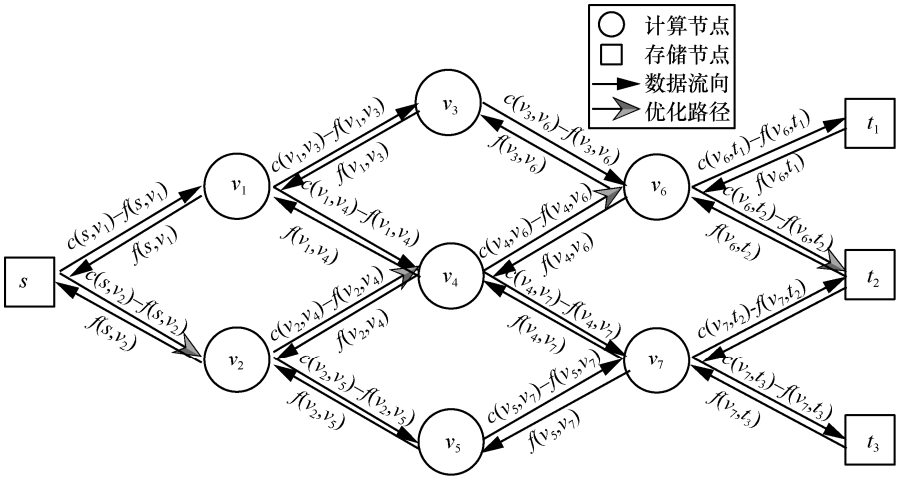


图 2 增进网络模型

$|f|+|f_p|>|f|$。

由定义 2 和定义 3 可知，增进网络描述了在原网络中，每条边在容量限制下可能提升流量的空间，则优化路径表示提升网络流量、合理分配堆积数据的有效方案。因此，当源点产生堆积数据时，沿着优化路径的方向分配负载，就一定能够提高集群的吞吐量。当增进网络中不存在优化路径时，集群达到其吞吐量的峰值，则一定存在一个算子成为整个集群的性能瓶颈。

3.2　资源优化配置模型

根据流网络模型可知，通过准确计算每条边上容量与流量的取值，量化集群计算能力与计算负载的数学关系，从而分析集群当前时刻的资源分配和使用情况。然而，当集群因计算资源不足而导致性能下降时，需要定义资源优化配置模型，通过定义流网络的划分来寻找集群的性能瓶颈，从而制定最优的弹性资源调度计划。

定义 4　流网络划分。设有流网络 $G=(V, E)$，其中 $s\in S$ 是流网络的源点，$t_i\in T$ 是汇点。则该网络的一个划分 $D=(X,Y)$将节点集 V 分为 X 和 Y 这 2 个集合，其中 $Y=V-X$，使 $s\in X$，$t_i\in Y$，且有 $X\bigcap Y=\varnothing$，$X\bigcup Y=V$。$\forall v_i,v_j\in O_i$ 有 $v_i,v_j\in X$ 或 $v_i,v_j\in Y$。则该划分 $D=(X,Y)$对应的容量记为

$$c(D)=c(X,Y)=\sum_{v_i\in X}\sum_{v_j\in Y}c(v_i,v_j) \tag{5}$$

该划分对应的流量记为

$$f(D)=f(X,Y)=\sum_{v_i\in X}\sum_{v_j\in Y}f(v_i,v_j)-\sum_{v_i\in X}\sum_{v_j\in Y}f(v_j,v_i) \tag{6}$$

其中，容量最小的划分为该流网络的最小划分。

如图 3 所示，流网络的一个划分将数据源点和汇点分在 2 个不同的集合中，且同一个算子的不同实例不横跨任何一个划分。因此，一个划分中容量和流量的关系反映了不同算子之间在资源配置和计算性能上的差异，为定位整个集群的性能瓶颈提供了可行的方案。

设流网络 $G=(V, E)$的一个流为 f，$G_f=(V_f, E_f)$是 G 由 f 产生的增进网络。当 G_f 中不存在任何优化路径时，f 是 G 的最大流，则至少存在一个划分 $D=(X,Y)$，使 $|f|=f(X,Y)=c(X,Y)$，且 D 是 G 的最小划分。

当增进网络中不存在任何优化路径时，原网络的流量达到最大值，且当前流量值与最小划分的容量值相等，则最小划分所对应的算子成为集群的性能瓶颈。因此，FAR-Flink 策略的核心思想为：首先，通过计算 DAG 模型中每条边的容量大小，将其转化为流网络模型；其次，通过计算对应的增进网络和优化路径，实现计算负载的最优分配，当集群性能达到瓶颈时，再通过寻找流网络的最小划分，制定弹性资源调度计划；最后，通过基于分簇和分桶的状态数据管理模型，实现状态数据高效迁移算法，并完成弹性资源调度计划的执行。

3.3　状态数据管理模型

根据资源优化配置模型，当数据源点的输入速率发生变化时，可制定最优的弹性资源调度策略。但由于 Flink 是有状态数据流处理平台，每个节点会保存当前的状态数据，为了实现高效的节点间状态数据迁移策略，降低弹性资源调度计划的执行开销，必须建立合理的状态数据管理模型，并在此基础上设计高效的数据迁移算法。在数据迁移过程

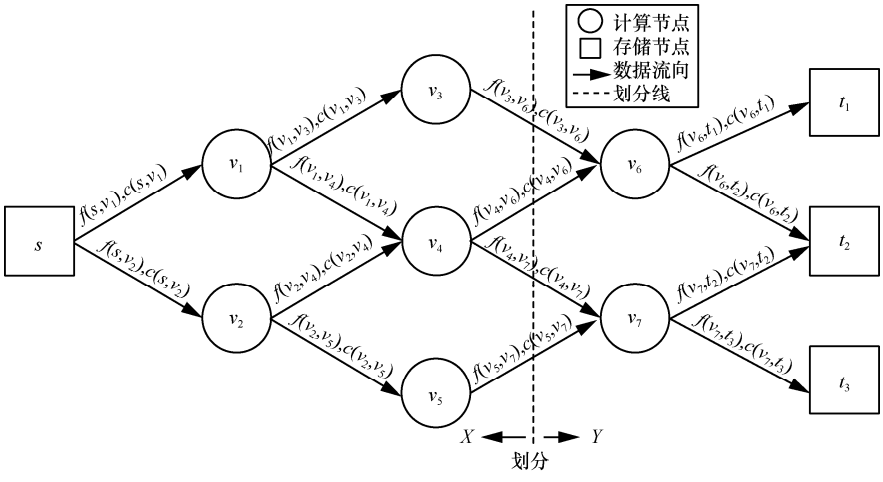


图 3　流网络划分示意

中，尽可能降低对 Hadoop 分布式文件系统（HDFS, Hadoop distributed file system）的访问频次，减少不必要的数据传输，从而有效提高动态资源调度策略的执行效率。

如图 4 所示，当集群由 v_1,v_2∈O_1 这 2 个实例扩充至 v_1,v_2,v_6∈O_1 这 3 个实例时，需要考虑原本由 2 个节点维护的状态数据 d_1 和 d_2 如何分配到 3 个节点中，以实现节点间计算任务的最优分配；此外，在对状态数据执行快照和重分配的过程中，都会频繁地访问 HDFS 执行数据读写的操作，这会占用大量的网络传输资源，进而影响节点间的数据传输性能。针对上述问题，提出高效的节点间状态数据迁移算法，能够有效提高动态资源调度计划的执行效率。

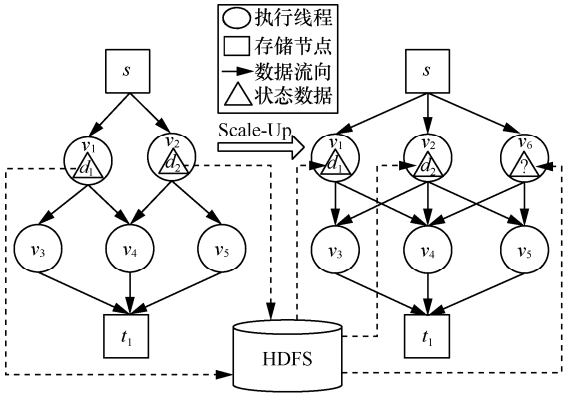


图 4 状态数据分配模型

设算子 O_n 待处理的数据为二元组 tuple_i=(key, value)，则该元组对应的簇为

$$\text{bulk}(\text{tuple}_i)=\text{hash}(\text{key}) \tag{7}$$

即具有相同 key 的 hash 值的数据元组属于同一个簇。在现有的 Flink 平台中，目前是按 key 的 hash 值不同，分簇对状态数据进行管理的，称为 KeyedState，是目前状态数据管理的主要策略。但由于数据元组有多种不同的 key，hash 函数发生碰撞的概率较低，通常状态数据的分布较为分散，在数据迁移过程中需要频繁访问 HDFS，占用大量的网络传输资源并引入较高的传输开销，进而影响了弹性资源调度算法的执行效率。因此，通过提出状态数据分桶管理的思想，并设计优化的状态数据迁移算法，能够有效减少节点对 HDFS 的访问频次，减少网络传输开销以提高算法性能。

4 动态资源调度策略

在第 3 节建立的相关模型基础上，本节提出了 FAR-Flink 策略，该策略分别包含流网络构建算法、弹性资源调度算法和状态数据迁移算法。该策略主要分为以下 3 个步骤，具体的执行流程如图 5 所示。

步骤 1 通过流网络构建算法计算每条边的容量大小，建立流网络模型。

步骤 2 通过弹性资源调度算法定位性能瓶颈，制定调度策略。

步骤 3 通过状态数据迁移算法执行调度策略，实现集群规模的弹性伸缩。

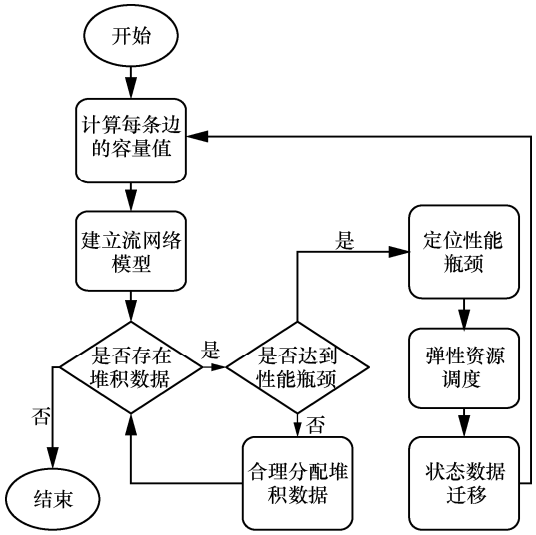


图 5 FAR-Flink 策略执行流程

4.1 流网络构建算法

根据流网络模型可知，准确评估网络每条边的容量大小对策略的执行效果是至关重要的。经实验分析得出，容量值与以下 4 个因素有关。

1) 节点间的网络传输性能和传输资源占用情况，传输资源不足将导致容量值减小。

2) 节点内的计算性能和计算资源占用情况，计算资源不足导致容量值减小。

3) 节点所承担计算任务本身的复杂程度，计算任务越复杂则容量值越小。

4) 下游节点的计算资源剩余情况，根据 Netty 组件的水位机制，下游节点的数据阻塞会引起反压现象，导致上游节点容量值下降。

在流式计算平台中，数据传输性能对计算的实时性影响较大，因网络拥塞导致数据在节点缓存中被滞留是流式计算平台面临的主要性能瓶颈。当一台 PC 机中关闭所有与 Flink 无关的进程时，在纯净的网络环境中，该节点实际可用于数据传输的网络带宽资源为

$$N_{v_i}(\text{Data}) = R_{v_i}^{B} - N_{v_i}(\text{System}) - N_{v_i}(\text{Heartbeat}) - N_{v_i}(\text{CK}) - N_{v_i}(\text{Other}) \quad (8)$$

其中，$N_{v_i}(\text{Data})$为实际可用于节点间数据传输的网络带宽资源；$R_{v_i}^{B}$为物理节点间原有的网络带宽资源总量，如节点间使用百兆带宽的网络连接，则$R_{v_i}^{B}$=100 Mbit/s≈12.5 MB/s；$N_{v_i}(\text{System})$为操作系统本身的静态网络传输开销；$N_{v_i}(\text{Heartbeat})$为进程间心跳信息的传输开销；$N_{v_i}(\text{CK})$为节点间歇性执行检查点（checkpoint）时向 HDFS 和 Zookepper 写入状态数据的传输开销；$N_{v_i}(\text{Other})$为其他随机因素可能产生的极少量传输开销。单位均为 MB/s。

因此，该节点对应输入链路的容量值为

$$c(v_j, v_i) = \frac{N_{v_i}(\text{Data})}{|E_{ji}| \sum_{i=0}^{n-1} \text{size}(\text{tuple}.f_i)} \quad (9)$$

其中，$|E_{ji}|$为节点 v_i 对应输入边的数目，size(tuple.f_i)为数据元组中每个字段所占空间的大小，单位为 B。

根据上述分析，流网络构建算法的执行过程如算法 1 所示。

算法 1　流网络构建算法

输入　集群拓扑结构 T，最高计算时延阈值 θ

输出　流网络 G

1) for each $(v_i, v_j) \in G.E$　do

2)　calculate the capacity value of each edge $c(v_i, v_j)$　/*根据式(8)和式(9)计算每条边(v_i, v_j)的容量值*/

3) end for

4) if $S.f > 0$　then

/*在作业执行时，反馈调节节点容量*/

5)　for each $v_j \in G.V$　do

6)　if latency>θ and $f(v_i, v_j) \leqslant c(v_i, v_j)$ then

/*时延过高表明设定的容量值过大*/

7)　$c(v_i, v_j) \leftarrow c(v_i, v_j) - \eta$;

/*根据式(17)计算 η 取值并减小容量值*/

8)　else if latency $\ll \theta$ && $f(v_i, v_j) \approx c(v_i, v_j)$

/*吞吐量过低表明容量值过小*/

9)　$c(v_i, v_j) \leftarrow c(v_i, v_j) + \eta$; /*根据式(17)计算 η 取值并增大容量值*/

10)　end if

11)　end for

12) end if

13) return G

在算法 1 中，首先确定流网络中顶点和边的集合（步骤 1)和步骤 2)）；其次根据每个计算节点的网络传输资源，计算节点对应输入边的初始容量值（步骤 3)~步骤 6)）；最后在作业执行过程中，根据计算时延的平均值与阈值之间的关系，对流网络每条边的容量值进行反馈调节。当平均计算时延大于设定的阈值，且对应边的流量值仍小于容量时，表明设定的容量过大，则以 η 为步长减小对应边的容量；当平均计算时延远小于设定的阈值，但流量值已经接近容量时，表明对应边的容量值过小，则以 η 为步长增大对应边的容量。实验数据表明，通过网络开销计算出的初始容量值往往是数据传输的极限值，受计算开销等其他因素的影响，实际合理的容量值应小于初始值，因此需要通过反馈调节将容量值调整至合理的范围内。

4.2　弹性资源调度算法

通过流网络构建算法建立流网络模型后，当源点产生数据堆积时，通过弹性资源调度算法合理分配计算负载、定位性能瓶颈并生成弹性资源调度计划。

图 6 为弹性资源调度策略示意，假设图中有划分 $D=(X,Y)$，当且仅当

$$f(X,Y) \geqslant \lambda c(X,Y) \quad (10)$$

则 Y 集合中第一个算子 O_2 有可能成为集群的性能瓶颈，需要增加资源并扩大 O_2 的并行度，其中 λ 是当前划分 $D=(X,Y)$对应流量与容量的比值，且有 $0.85 \leqslant \lambda \leqslant 1$，即当一个划分对应的流量达到容量值的 85%时，算法认为该划分对应的算子可能成为集群的性能瓶颈。

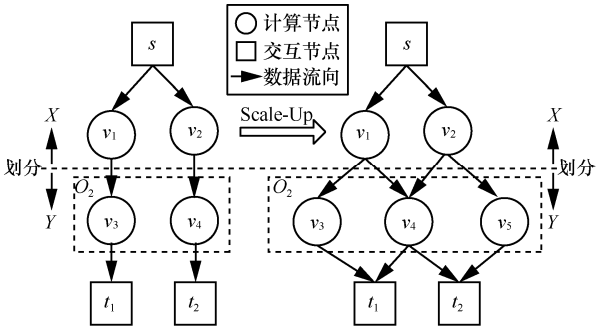


图 6　弹性资源调度策略示意

基于资源优化配置模型，弹性资源调度算法的执行过程如算法 2 所示。

算法 2 弹性资源调度算法

输入 流网络 G，流网络的一个流 f，数据源点每个分区的堆积值 lags[]

输出 完成任务调度后新的流网络模型 G'

1) for each $(v_i, v_j) \in G.V$ do

2) $c_f(v_i, v_j) \leftarrow c(v_i, v_j) - f(v_i, v_j)$;

3) $c_f(v_j, v_i) \leftarrow f(v_i, v_j)$;

4) end for

/*根据定义 2，计算每条边的增进容量函数*/

5) $P \leftarrow \mathrm{BFS}(G_f, G_f.s, G_f.T)$;

/*通过广度优先搜索，在增进网络中寻找一条优化路径*/

6) while too much accumulation AND P !=∅ do/*源点产生大量数据堆积，且优化路径存在*/

7) $|f_p| \leftarrow \min\{c_f(v_i, v_j) \mid (v_i, v_j) \in P\}$;

8) $G \leftarrow \mathrm{increaseFlow}(G.E, P, |f_p|)$;

9) $P \leftarrow \mathrm{BFS}(G_f, G_f.s, G_f.T)$;

10) end while

/*沿着优化路径的方向，提高每条边的流量*/

11) if too much accumulation then

/*当源点有数据堆积，且不存在优化路径时*/

12) for each $(X,Y) \in G.D[]$ do

13) if $f(X,Y) \geqslant \lambda c(X,Y)$ then

/*根据式(10)，判断划分流量与容量的关系*/

14) operator[].add($Y.O_1$);

/*根据定义 2，应增加 Y 集合第一个算子的并行度*/

15) end if

16) end for

17) end if

18) G' = stateMigration(operator[]);

/*通过算法 3 执行状态数据迁移，完成弹性资源调度*/

19) return G'

在算法 2 中，根据增进网络与优化路径的定义，首先由流网络 G 的一个流 f 生成对应的增进网络 G_f（步骤 1)~步骤 5)），当数据源点产生堆积时，在增进网络中寻找优化路径，并沿其方向提高原网络的流量（步骤 6)~步骤 11)）。当增进网络中不存在优化路径时，如果数据源点仍有堆积，则在流网络中寻找流量与容量的比值大于 λ（即满足式(10)）的划分，增加其 Y 集合中第一个算子的并行度（步骤 12)~步骤 18)），从而得到需要扩大并行度的算子。最后，通过调用状态数据迁移算法（步骤 19)），完成节点间的状态数据迁移，从而实现计算资源的动态调度。

4.3 状态数据迁移算法

在执行弹性资源调度策略时，节点间状态数据迁移会产生大量网络传输开销，但 Flink 状态数据管理策略并不适用于高效的数据迁移。因此，通过提出状态数据管理模型和迁移算法，降低数据传输开销，提高迁移效率。

定义 5 桶。设算子 O_n 待处理的数据为一个二元组 tuple_i=(key,value)，算子 O_n 共持有 k_n 个桶，则该元组对应的桶为

$$\mathrm{bucket}\left(\mathrm{tuple}_i\right) = \mathrm{bulk}\left(\mathrm{tuple}_i\right)\%k_n \tag{11}$$

则对应负责处理该元组的实例为

$$v_i = \frac{\mathrm{bucket}\left(\mathrm{tupe}_i\right) \cdot \left|P\left(O_i\right)\right|}{k_n} \tag{12}$$

其中，v_i 为算子 O_n 的第 i 个实例，$k_n \geqslant \left|O_n\right|$。这样就建立了数据桶与计算节点间的映射关系，将桶作为计算节点维护状态数据的基本单位。此外，参数 k_n 是算子 O_n 所持有桶的数目，用户可根据状态数据的规模和复杂程度进行适当的调整。

图 7 表示由状态数据簇到分桶管理的映射方式，其中 k_n=10，算子并行度由$|O_n|$=3 扩大至$|O_n|$=4。在状态数据迁移算法中，结合 Flink 支持的检查点（checkpoint）机制，在执行快照时将状态数据以桶为单位发送至 HDFS，同一个桶的数据存储在相邻的位置。当算子的并行度改变时，根据式(11)和式(12)重新计算每个桶到节点的映射关系，最后由节点从 HDFS 中拉取对应的状态数据，完成计算节点的状态数据恢复。

结合状态数据分桶管理的特点，可提出状态数据迁移算法如算法 3 所示。

算法 3 状态数据迁移算法

输入 需增加并行度的算子集 operator[]

输出 完成任务调度后新的流网络模型 G'

1) for each $O \in$ operator[] do

/*依次对每个需要增加并行度的算子执行状态迁移操作*/

2) for each $v \in O$ do

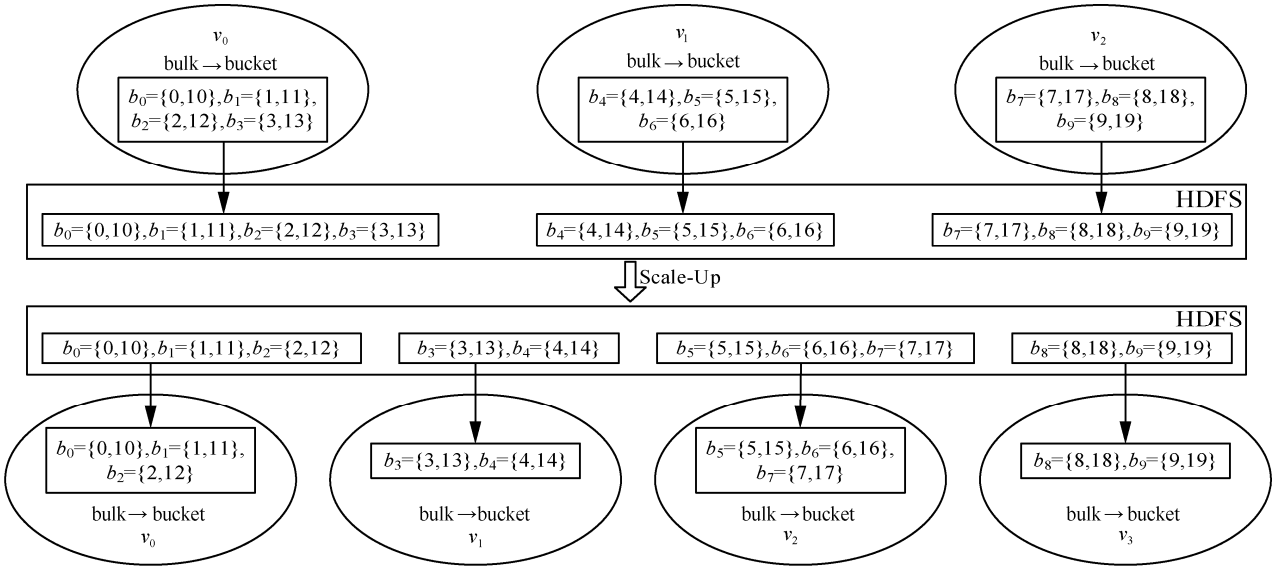


图 7　状态数据迁移示意

/*对每个节点执行检查点（checkpoint）流程*/

3)　handler ← v.sendData(state, HDFS);

/*将每个节点的状态数据发送至 HDFS，并记录状态数据对应的句柄*/

4)　v.sendData(handler, ZooKeeper);

/*将句柄信息保存在 ZooKeeper 中，以便于数据恢复*/

5)　bucket[++k] ← handler;

/*保存状态数据对应桶的信息*/

6)　end for

7)　O.add(v);

/*通过增加新的节点，扩大算子的并行度*/

8)　bucket[]←JobManager.rescale(bucket[],|O|);

/*基于式(11)和式(12)，由 JobManager 根据新的并行度，重新计算数据桶到计算节点的映射*/

9)　for each $v_i \in O$　do

/*对每个节点执行数据恢复（Restore）流程*/

10)　handler←v.getData(bucket[i],ZooKepper);

/*从 ZooKepper 中获取状态数据的句柄*/

11)　state ← v.getData(handler, HDFS);

/*分别从 HDFS 中拉取节点对应的状态数据*/

12)　end for

13) end for

14) flowNetwork_selfLearning(G');

/*在通过算法 1 重新计算每条边的容量值*/

15) return G'

在算法 3 中，首先对每个节点执行检查点操作（步骤 3)~步骤 7)）：将节点的状态数据发送至 HDFS，并将状态数据的句柄信息保存在 ZooKeeper 中，记录状态数据对应的桶。其次从资源池中获取一个新的计算节点，扩大对应算子的并行度，并由 JobManager 重新计算状态数据桶到计算节点的映射关系。再次根据新生成的集群拓扑结果，对每个节点执行数据恢复操作：从 ZooKeeper 中获取状态数据对应的句柄，并从 HDFS 中拉取对应的数据。最后再次执行算法 1，以修正每条边对应的容量值。

4.4　相关参数分析

在流网络构建算法中，参数 η 是动态调节容量值的步长。η 过大会导致调整幅度过于剧烈，η 过小会导致调整幅度不够，2 种情况均会导致最终的容量值不准确。因此参数 η 的计算方法如下。

根据定义 1，$c(v_i, v_j)$是边(v_i, v_j)每秒钟允许传输的最大元组个数，且 1 s=1 000 ms。根据文献[26]的时延统计算法，节点间传输数据的平均计算时延为

$$l_{v_j}=\frac{\sum_{i=1}^{n}\left(v_j.f_i-v_j.d_i\right)}{n} \tag{13}$$

其中，$v_j.f_i$ 是数据元组到达节点 v_j 的发现时间[26]，$v_j.d_i$ 是数据元组离开节点 v_j 的完成时间[26]。则容量值为对应节点在 1 000 ms 内所能传输的平均元组数目，将式(13)代入得

$$c\left(v_i,v_j\right)=\frac{1\,000}{l_{v_j}}=\frac{1\,000n}{\sum_{i=1}^{n}\left(v_j.f_i-v_j.d_i\right)} \tag{14}$$

由于 η 是调节容量值的步长，即容量函数的变化率，则对容量函数求导得

$$c'\left(v_i,v_j\right)=\frac{\partial c}{\partial l_{v_j}}=-\frac{1\,000}{{l_{v_j}}^2} \tag{15}$$

此外，根据容量值与计算时延的函数关系，通过记录的平均计算时延，求得调节前与调节后的容量值偏差应为

$$\mathrm{reg}_{v_j}=\left|\frac{1\,000}{l_{v_j}}-\frac{1\,000n}{\sum_{i=1}^{n}\left(v_j.f_i'-v_j.d_i'\right)}\right| \tag{16}$$

其中，$v_j.f_i'$ 和 $v_j.d_i'$ 分别为容量值调节后，采样得到的节点发现时间和完成时间[26]，这样就采集到了容量值调节后的平均计算时延。

因此，为了避免调整过于剧烈导致容量值出现抖动的现象，学习参数 η 取容量函数的导数与偏差的较小者，即

$$\eta=\min\left(\left|\frac{1\,000}{l_{v_j}}-\frac{1\,000n}{\sum_{i=1}^{n}\left(v_j.f_i'-v_j.d_i'\right)}\right|,\left|-\frac{1\,000}{{l_{v_j}}^2}\right|\right) \tag{17}$$

实验表明，通过式(17)分别计算流网络中每条边学习因子 η，再通过算法 1 调整对应边的容量函数，能够取得比较准确的容量值，流网络构建算法有比较好的优化效果。

4.5 算法性能分析

由于流式计算对平台的性能和实时性有很高的要求，因此弹性资源调度策略应具备很高的性能，且不能引入过高的时间开销。流网络构建算法首先遍历流网络的每条边，以确定其容量的初始值；再遍历每个节点，从而对边的容量值进行反馈调节，因此构建算法的时间复杂度为 $T(n)=O(|V|+|E|)$。

此外，弹性资源调度算法是基于广度优先搜索（BFS, breadth first search）实现的流量递增算法，已知 BFS 算法的时间复杂度为 $T(n)=O(|V|+|E|)$，且在算法 2 中，3 次循环的最高循环次数分别为流网络的节点数目、边的数目以及当前流量与弹性资源调度量的差值，因此弹性资源调度算法的时间复杂度为 $T(n)=O(|V|+(|V|+|E|)(f_{\max}-f)+|D|)$，由于流网络中边的数目大于节点数目和划分的数目，因此算法的时间复杂度为 $T(n)=O(|E|(f_{\max}-f))$。

数据迁移算法是典型的分布式并行化算法，其中的每个步骤都分别在不同的节点上执行，节点之间通过 ZooKeeper 进行统一协调，并且计算过程较为简单，其时间开销主要取决于状态数据的规模和节点间网络传输的性能，因此没有具体的数学计算式来表达其时间复杂度，本文 5.4 节通过实验验证了算法的时间开销较原系统有一定的下降。

5 实验与分析

为了验证 FAR-Flink 策略的有效性，本文通过设置对比实验，将 FAR-Flink 与原生 Flink 1.6.0 版本的系统形成对比，并与本文工作密切相关的 Elastic Nephel[6]策略形成对比，通过执行代表不同作业类型的典型 Benchmark，验证了算法的优化效果和执行开销，并分析了相关参数对算法的影响。

5.1 实验环境

实验搭建的集群由 21 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，在执行弹性资源调度计划动态扩展资源时启动备用节点，并部署相应的计算任务。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，由一个独立的节点执行流网络构建算法和弹性资源调度算法。其中，每个同构节点的硬件配置和软件配置分别如表 1 和表 2 所示。

表 1 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 RPM
网络	100 Mbit/s

表 2 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
Redis	5.0.2

同时，为了使 Flink 集群达到最优的计算性能，

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 3 所示。

表 3　　性能参数配置

配置项	参数值	说明
jobmanager.heap.size	2 048 m	主节点内存
taskmanager.heap.size	2 048 m	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	500 m	缓冲区上限
taskmanager.memory.segment-size	32 768	内存分块大小

5.2　构建算法性能测试

在流网络模型中，每条边的容量值应客观反映节点的计算能力和节点间的传输能力，容量值评估不准确可能会导致计算时延升高而无法满足计算的实时性要求。为了探究容量函数与计算时延的关系，实验为每条边设置相等的容量值，并不断递增。同时，基于 Flink Metrics 体系的 Latency Tracking 机制采集的计算时延如图 8 所示。

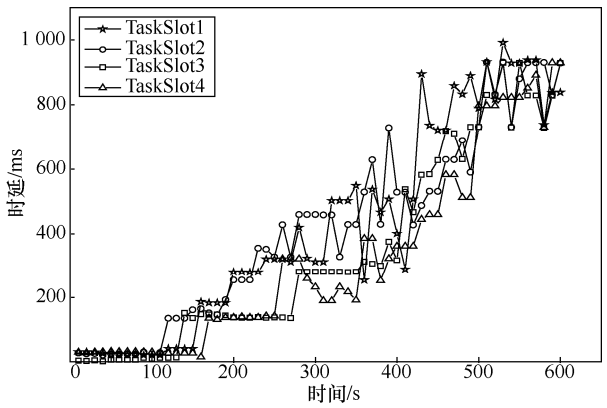


图 8　容量与计算时延关系

由图 8 可以看出，随着时间的推移，容量值持续增大，每个 TaskSlot 的计算时延均有一定的波动，但总体呈上升趋势，甚至出现阶跃上升的现象，且计算时延越高，其波动变化越剧烈。这表明过高地估计节点的计算能力，会导致数据元组被阻塞、计算时延升高，但在满足计算时延约束的前提下容量值总会被尽可能地增加，从而提高集群的吞吐量。因此，在当前实验环境中，当容量函数取 30 000~40 000 tuple/s 时，平均计算时延在 400 ms 以下，这是比较合理的取值区间。

此外，在流网络构建算法中，参数 η 确定了调节容量函数的步长，对流网络的稳定性和收敛速率有决定性作用。如图 9 所示，当使用固定的 η 取值执行构建算法时，流网络中存在收敛速率和稳定性难以权衡的问题：较大的 η 值会导致容量值剧烈波动而无法趋于稳定，较小的 η 值会导致网络收敛时间较长（约 50~60 s）。但通过式(17)计算的动态 η 取值能够客观反映当前容量值与理想值的差，随着 η 的不断减小，流网络的容量值迅速收敛于理想值，并逐渐趋于稳定。

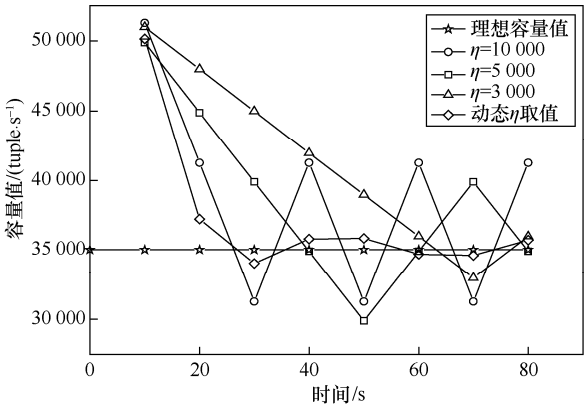


图 9　学习参数对比

5.3　FAR-Flink 性能测试

文献[6]提出针对 Flink 平台内核的弹性资源调度策略，与本文工作的研究目标最接近。为了验证 FAR-Flink 的优化效果，实验在 Flink 原系统、Elastic Nephel[6]和 FAR-Flink 上分别执行 WordCount、TwitterSentiment、IncrementalLearing 和 Streaming-Benchmarks 代表不同作业类型的标准 Benchmark，它们分别来自 Flink 源码中的示例和 Intel 等公司在开源社区 Github 上的贡献。实验结果分别如图 10~图 13 所示。

采用与 5.2 节相同的实验配置，分别在原系统和 FAR-Flink 上执行 WordCount 作业，并采集 Kafka 缓冲池中的数据堆积和计算时延，如图 10 所示。由图 10(a)和图 10(b)可知，计算时延与 Kafka 中的数据堆积呈正相关，即实验验证了数据堆积导致时延升高的理论推测是正确的。此外，由于原系统执行默认的资源调度策略，因资源不足导致计算时延不断升高。FAR-Flink 通过弹性资源调度算法合理分配 Kafka 中的堆积数据，并动态增加了算子

O_{FlatMap} 的并行度。但由于执行状态数据迁移有一定的时间开销，导致数据堆积有短暂的上升。从图 10(c)中可以看出，Elastic Nephel(EN)的时延上升时间较长。FAR-Flink 执行任务迁移过程中，持续时间比 EN 缩短了约 40 s。

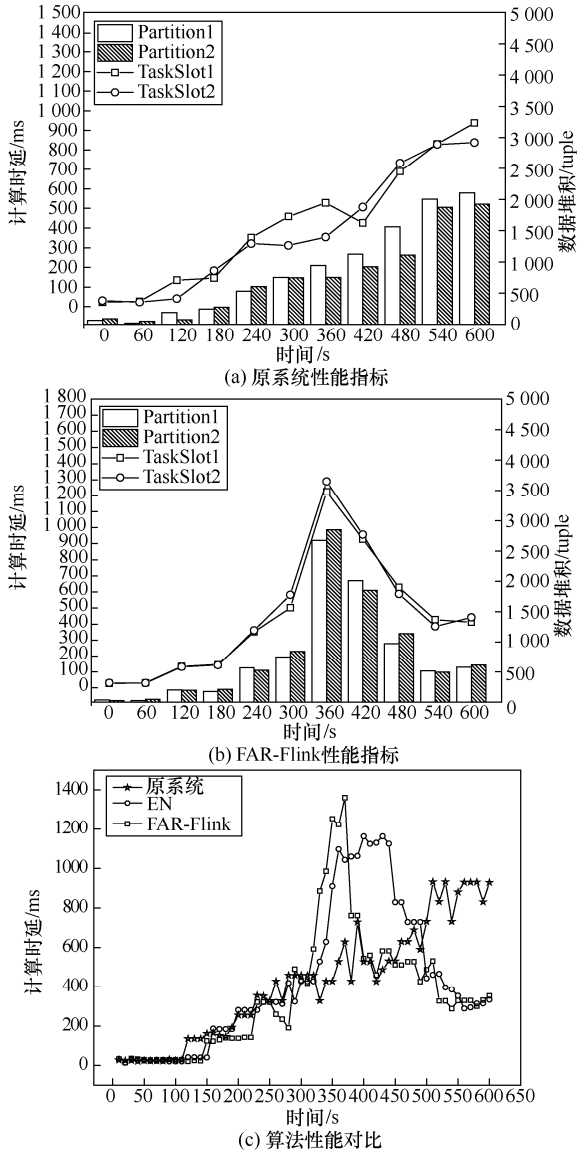


图 10 WordCount 作业执行效率对比

TwitterSentiment 是在生产环境中实际应用的标准 Benchmark。实验采用与文献[32]相同的实验配置，并以 10 s 为周期采集节点的内存使用率。得到如图 11 所示的实验结果。

由图 11 可知，随着计算负载的不断升高，EN 和 FAR-Flink 分别在第 540 s 和第 720 s 各增加了一个节点，扩大算子并行度并执行状态数据迁移。但由于作业执行中产生状态数据并占用大量内存资源，状态数据迁移过程产生了一定的时间开销：EN 系统的 2 次迁移分别消耗约 14 s 和 18 s，FAR-Flink 分别消耗约 7 s 和 13 s。但 FAR-Flink 迁移过程中吞吐量较低且内存资源消耗较高，这是因为执行 restore 操作需要一定时间，且迁移过程中产生大量的 HDFS 写操作。但随着计算资源的增加，系统的吞吐量有明显的上升，基本满足输入负载的要求，且内存资源占用降低至合理可接受的范围内，这说明 2 种弹性资源调度策略均有效提升集群性能。

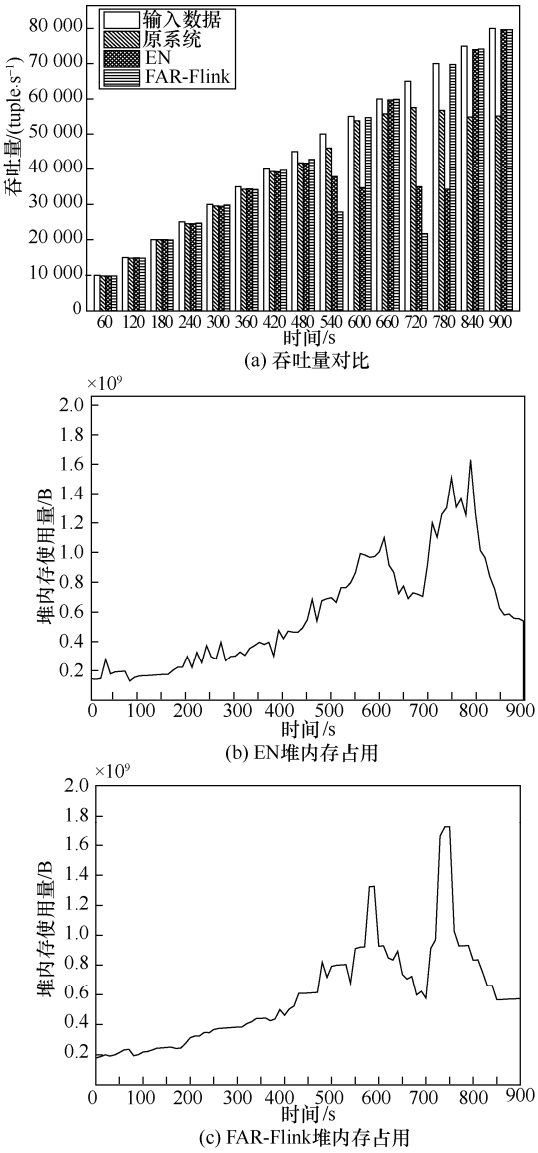
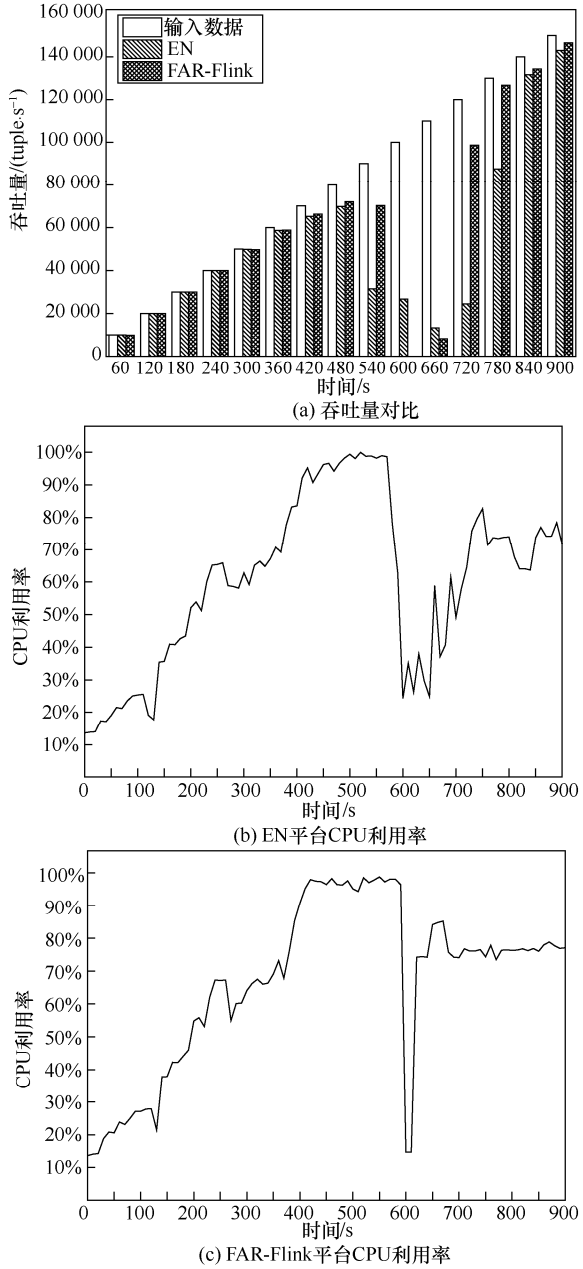


图 11 TwitterSentiment 作业执行情况对比

为了验证 FAR-Flink 在高计算复杂度、高 CPU 资源占用场景下的优化效果，实验运行了 IncrementalLearing 作业，并以 10 s 为周期采集节点的 CPU 利用率，得到如图 12 所示的实验结果。



(a) 吞吐量对比

(b) EN平台CPU利用率

(c) FAR-Flink平台CPU利用率

图 12　IncrementalLearing 作业执行情况

由图 12 可知，集群执行弹性资源调度策略计划的过程需要一定的时间开销，EN 和 FAR-Flink 都在第 600 s 左右增加了一个计算节点，其中 EN 执行策略的时间开销较高（约 42 s），但执行过程中其他节点计算性能下降，集群吞吐量骤减。FAR-Flink 执行策略的时间开销较低（约 18 s），但执行 restore 过程中整个集群有极短暂的停滞，其中第 600 s 时检测输出数据量为 0 tuple/s，在短暂的时间内恢复了任务所有计算节点的状态数据，因此在下一个阶段的吞吐量值急剧回升。在资源利用方面，EN 在数据迁移过程中的 CPU 利用率值剧烈抖动，FAR-Flink 的 CPU 利用率值急剧下降后又快速回升，这都与其执行调度策略的过程相关，与吞吐量的检测结果是一致的。总体上，2 种策略均通过增加计算资源提高了集群性能，FAR-Flink 有效缩短了 EN 执行调度策略的时间，但其执行过程中计算任务有非常短暂的停滞。

为了进一步验证 FAR-Flink 在实际应用场景下的性能，实验采用 Yahoo 公司在 GitHub 上开源的 Streaming-Benchmarks[32]，并分别从吞吐量、计算时延、Kafka 数据堆积、堆内存利用率、CPU 利用率 5 个不同的维度，监测集群和作业的运行情况，监测结果如图 13 所示。

由图 13 可以看出，随着计算负载的持续上升，EN 和 FAR-Flink 系统都分别出现资源不足导致集群性能下降的问题。其中，EN 系统连续动态增加了 2 个计算节点，由于计算复杂且节点状态数据规模较大，数据迁移过程产生了较高的时间开销，在第一次数据迁移后集群吞吐量有少量的回升就立刻进入第二次迁移，整个迁移过程共持续约 273 s。FAR-Flink 首先通过弹性资源调度算法合理分配计算负载，并在第 600 s 和第 780 s 时分别动态增加一个计算节点，其数据迁移过程分别持续了 21.6 s 和 36.7 s，较 EN 系统的数据迁移时间有明显的下降。同时，2 种算法均通过动态增加计算资源有效提升了集群性能，其中 EN 系统的数据迁移时间较长，但迁移过程中集群可保证较低的吞吐量。FAR-Flink 系统能够在前期合理分配计算负载，且有效缩短了数据迁移过程的时间开销，但迁移过程中作业有极短暂的停滞，迁移完成后集群性能有较明显的上升。

综上所述，实验在 4 种资源调度策略下分别执行不同的标准 Benchmark，通过性能对比得出了不同算法的优缺点及适用场景，实验结果如表 4 所示。实验证明，FAR-Flink 通过合理分配计算负载、动态增加计算资源、降低数据迁移开销 3 种策略相结合的方式，有效提高集群性能。与原系统相比，在计算负载波动上升期间，针对不同类型的 Benchmark，集群吞吐量平均提高了 27.61%，状态数据迁移时间平均缩短了 45.36%，具有一定的优化效果。

5.4　数据迁移开销测试

为了准确评估 FAR-Flink 执行数据迁移过程产生的网络传输开销，验证数据迁移不会长时间占用过多的网络传输资源而影响集群性能，实验分别执

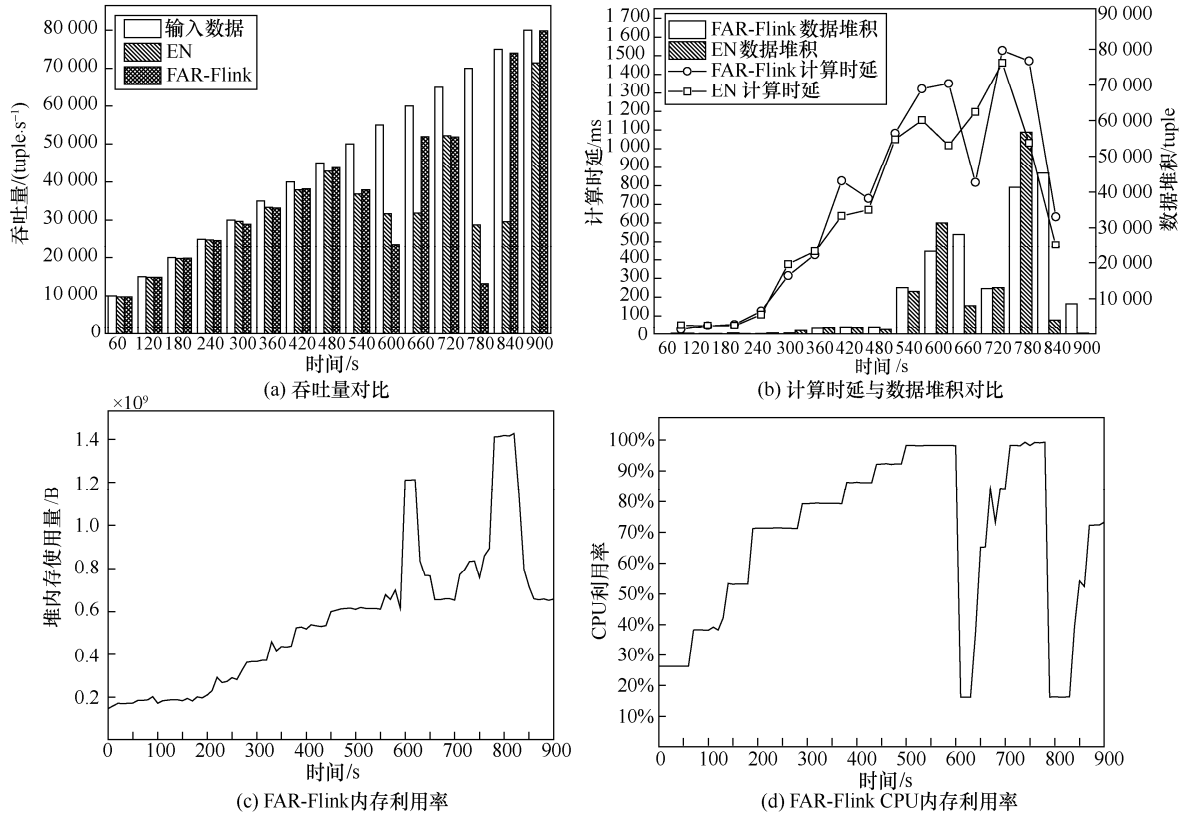


图 13　Streaming-Benchmarks 作业执行效率

表 4　　**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	通过数学模型计算得出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

行 TwitterSentiment 和 Streaming-Benchmarks 作业，并监测网络传输数据以评估网络传输开销，得到如图 14 所示的实验结果。

由图 14 可知，除去节点间可忽略的静态数据传输外，计算节点会依据相关配置周期性地执行状态数据快照，并向 HDFS 发送数据，在执行动态资源调度计划时，节点会从 HDFS 拉取相应的数据并执行状态数据的恢复操作。在 2 种策略执行过程中，需要传输的数据总量是基本相同的。但由于 EN 以块（bulk）为单位从远端拉取数据，其数据分布较为分散且存在大量碎片化数据，因此单位时间内的数据传输速率较低，传输时间较长。FAR-Flink 以桶（bucket）为单位从远端拉取数据，其数据分布较为集中且几乎没有碎片化数据，计算节点在短时间内集中拉取需要的数据，数据传输速率较高，传输时间较短。

综上所述，通过基于分簇和分桶的状态数据迁移算法，合理应对碎片化数据传输的问题，有效降低数据迁移的网络传输开销，通过提高传输效率缩短执行动态资源调度策略的时间。但实验结果表明，这种方式仍会产生一定的时间开销，在合理可接受的范围内对执行效率有一定的影响，如何进一

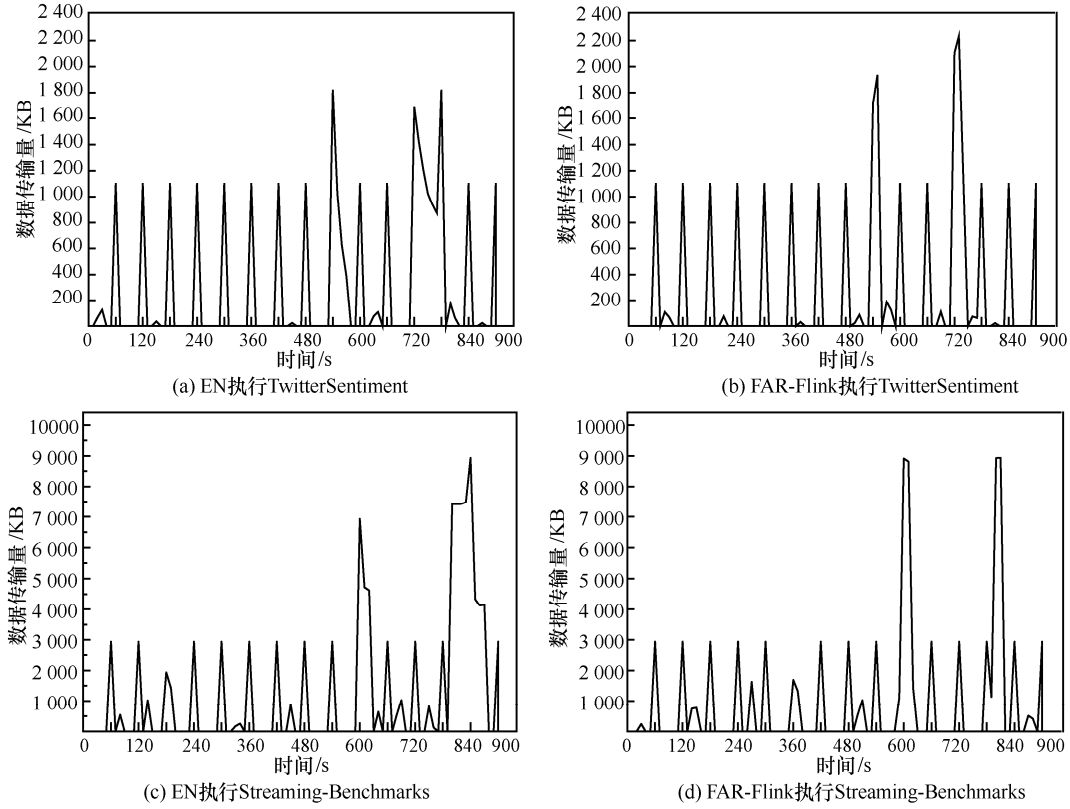


图 14　数据迁移的网络传输开销对比

步提高数据迁移效率以缩短迁移时间，将是下一步研究工作的主要方向。

6　结束语

作为集流处理与批处理为一体的统一大数据处理平台，Apache Flink 得到学术界和产业界的广泛关注，但其可扩展性和可伸缩性不足的问题，已经严重制约了平台的发展。本文提出了基于流网络的数据流动态资源调度策略，通过合理分配负载、动态增加资源、高效数据迁移 3 种策略相结合的方式，从根本上解决了因计算资源不足而影响集群性能的问题，并有效降低了网络传输的时间开销。

但本文算法也存在一定的局限性，首先，本文算法对 ZooKeeper 有很强的依赖性，需要 ZooKeeper 中保存相关的数据结构；其次，状态数据迁移过程中作业会有极短暂的停滞（约 2~3 s），但算法的执行开销在可接受的范围内。因此，未来的研究工作将主要集中于以下 3 个方面。

1) 降低资源调度策略对 ZooKeeper 的依赖程度，在限制单点计算和传输负载的前提下，尝试由 JobManager 统一提供资源调度和数据共享服务。

2) 通过提出新的状态数据管理和迁移策略，降低数据迁移开销，缩短调度计划的执行时间，提高执行效率。

3) 通过提出计算任务的热部署方案和节点间状态数据实时共享策略，实现对用户作业无感知的、完全透明的在线资源调度策略。

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(04):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. 大数据流式计算: 应用特征和技术挑战[J]. 大数据, 2015, 1(3): 99-105.
SUN D W. Big data stream computing: features and challenges[J]. Big Data Research, 2015, 1(3): 99-105.

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

[7] LOHRMANN B, WARNEKE D, KAO O. Nephele streaming: stream processing under QoS constraints at scale[J]. Cluster computing, 2014, 17(1): 61-78.

[8] VAN D V J S, VAN D W B, LAZOVIK E, et al. Dynamically scaling Apache Storm for the analysis of streaming data[C]//The 1st IEEE International Conference on Big Data Computing Service and Applications. IEEE, 2015: 154-161.

[9] GULISANO V, JIMENEZPERIS, RICARDO, et al. StreamCloud: an elastic and scalable data streaming system[J]. IEEE Transactions on Parallel & Distributed Systems, 2012, 23(12):2351-2365.

[10] WU Y, TAN K L. ChronoStream: elastic stateful stream computation in the cloud[C]// IEEE International Conference on Data Engineering. IEEE Computer Society, 2015:723-734.

[11] HEINZE T, ROEDIGER L, MEISTER A, et al. Online parameter optimization for elastic data stream processing[C]//The Sixth ACM Symposium on Cloud Computing. ACM, 2015: 276-287.

[12] MATTEIS T D, MENCAGLI G. Keep calm and react with foresight: strategies for low-latency and energy-efficient elastic data stream processing[C]//The 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. ACM, 2016: 1-12.

[13] HEINZE T, JERZAK Z, HACKENBROICH G, et al. Latency-aware elastic scaling for distributed data stream processing systems[C]//The 8th ACM International Conference on Distributed Event-Based Systems. ACM, 2014: 13-22.

[14] ZANG Z, RAO R N. DBalancer: a tool for dynamic changing of workers number in storm[C]//The 4th International Conference on Computer Science and Network Technology. 2016: 142-145.

[15] SHIEH C K, HUANG S W, SUN L D, et al. A topology-based scaling mechanism for Apache Storm[J]. International Journal of Network Management, 2017, 27(3): 1-12.

[16] LI J, PU C, CHEN Y, et al. Enabling elastic stream processing in shared clusters[C]//The 9th International Conference on Cloud Computing. 2016: 108-115.

[17] SUN D W, FU G, LIU X R, 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): 53-65.

[18] SUN D W, 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.

[19] XU J L, CHEN Z H, TANG J, et al. T-Storm: traffic-aware online scheduling in storm[C]//The 34th IEEE International Conference on Distributed Computing Systems. IEEE, 2014: 535-544.

[20] VAN D V J S, VAN D W B, LAZOVIK E, et al. Dynamically scaling Apache Storm for the analysis of streaming data[C]//The 1st IEEE International Conference on Big Data Computing Service and Applications. IEEE, 2015: 154-161.

[21] COLLINS R L, CARLONI L P. Flexible filters: load balancing through backpressure for stream programs[C]//The Seventh ACM International Conference on Embedded Software. ACM, 2009: 205-214.

[22] ASTRID R, HEISE A, HUESKE F, et al. SOFA: an extensible logical optimizer for UDF-heavy dataflows[J]. Information Systems, 2015, 52: 96-125.

[23] KWON Y C, BALAZINSKA M, HOWE B, et al. Skew-resistant parallel processing of feature-extracting scientific user-defined functions[C]//The 1st ACM symposium on Cloud computing. ACM, 2010: 75-86.

[24] 卞琛, 于炯, 修位蓉, 等. 基于迭代填充的内存计算框架分区映射算法[J]. 计算机应用, 2017, 37(3): 41-47.
BIAN C, YU J, XIU W R, et al. Partitioning and mapping algorithm for in-memory computing framework based on iterative filling[J]. Journal of Computer Application, 2017, 37(3): 41-47.

[25] 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.

[26] 李梓杨, 于炯, 卞琛, 等. 基于负载感知的数据流动态负载均衡策略[J]. 计算机应用, 2017, 37(10): 2760-2766.
LI Z Y, YU J, BIAN C, et al. Dynamic data stream load balancing strategy based on load awareness[J]. Journal of Computer Application, 2017, 37(10): 2760-2766.

[27] El K G F, WANAS N M, HEGAZI N H, et al. A dynamic load balancing framework for real-time applications in message passing systems[J]. International Journal of Parallel Programming, 2011, 39(2): 143-182.

[28] LOHRMANN B, WARNEKE D, KAO O. Massively-parallel stream processing under QoS constraints with Nephele[C]//The 21st International Symposium on High-Performance Parallel and Distributed Computing. ACM, 2012: 271-282.

[29] 鲁亮, 于炯, 卞琛, 等. 大数据流式计算框架 Storm 的任务迁移策略[J]. 计算机研究与发展, 2018, 55(1): 71-92.
LU L, YU J, BIAN C, et al. A task migration strategy in big data stream computing with storm[J]. Journal of Computer Research and Development, 2018, 55(1): 71-92.

[30] EWEN S, SCHELTER S, TZOUMAS K, et al. Iterative parallel data processing with stratosphere: an inside look[C]//The 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013: 1053-1056.

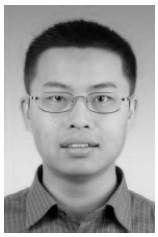
[31] PENG B, HOSSEINI M, HONG Z, et al. R-storm: resource-aware scheduling in storm[C]//The 16th Annual Middleware Conference. ACM, 2015: 149-161.

[32] KARIMOV J, RABL T, KATISIFODIMOS A, et al. Benchmarking distributed stream processing engines[J]. arXiv Preprint, arXiv: 1802.08496, 2018.

[作者简介]

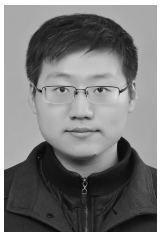
李梓杨（1993− ），男，新疆乌鲁木齐人，新疆大学博士生，主要研究方向为分布式系统、内存计算、流式计算。

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。



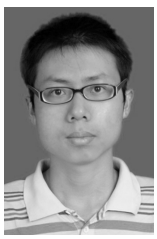
卞琛（1981− ），男，江苏南京人，博士，广东金融学院副教授，主要研究方向为分布式系统、内存计算、绿色计算。

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。



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。

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



鲁亮（1990− ），男，湖南湘潭人，博士，中国民航大学讲师，主要研究方向为分布式系统、内存计算、流式计算。