doi:10.3969/j.issn.1000-436x.2012.z1.021

基于 S4 框架的并行复杂事件处理系统

陈皓[1,2]，李瑜[1,2]，虎嵩林[1]，梁英[1]
（1. 中国科学院 计算技术研究所，北京 100190；2. 中国科学院 研究生院，北京 100049）

摘 要：针对复杂事件处理技术在单机的吞吐量瓶颈，而现有通用并行框架不适合复杂事件处理系统的问题，在分析现有并行技术的基础上，设计了 14 种复杂事件处理操作符，提出了以操作符为单位并行的复杂事件处理系统框架，给出了基于操作符的事件流负载分流方法。并基于 S4 系统实现了并行复杂事件处理系统，提供比单机运行更高的可靠性和吞吐率。通过集群上的实验证明，在一定范围内，该系统的吞吐量可随集群节点数增加而线性增长。

关键词：复杂事件处理；海量数据；并行计算
中图分类号：TP302; TP393 **文献标识码**：B **文章编号**：1000-436X(2012)Z1-0165-05

Parallel complex event processing system based on S4 framework

CHEN Hao[1,2], LI Yu[1,2], HU Song-lin[1], LIANG Ying[1]
(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China;
2. Graduate University, Chinese Academy of Sciences, Beijing 100049, China)

Abstract: There was a throughput bottleneck for complex event processing in only one computer, and the existing general-purpose parallel framework was not suitable for complex event processing system. Based on the analysis of existing parallel technology, fourteen kinds of complex event processing operator and a parallel framework by operator for complex event processing system were proposed, the event stream load triage based on operator was given. S4 system for parallel complex event processing system was utilized, provided greater reliability and throughput than stand-alone operation. Through the cluster experiments show that, within a certain range, the throughput of the system can be increased with the number of cluster nodes linear growth.

Key words: complex event processing; mass data; parallel computing

1 引言

当今社会是一个信息爆炸的社会。由于离散数据源的增加，如标签、微博、传感技术等的发展，使得信息处理系统所需要进行处理的数据量大大增加。另外，社交网络、高频交易、实时搜索等新应用的出现，对处理速度的要求达到了传统处理系统的极限。由于海量数据是具有时效性的，最佳的解决方法不是先把数据在数据库中缓存起来再一批一批地处理，而是每当数据出现时便进行一次处理，实时地处理数据。复杂事件处理技术[1,2](CEP, complex event processing)作为信息处理系统的新兴技术，具有高吞吐量、短延迟和复杂计算的特点。它采用以事件驱动为基础的架构，可处理不同事件源、不同类型的事件。它通过分析事件间的关系，利用过滤、关联、聚合、排序等技术，将输入简单

收稿日期：2012-08-06
基金项目：国网信息通信有限公司科技基金资助项目（SGIT[2010]449）；广东省科技计划基金资助项目（2010B050100009）
Foundation Items: Science Plan of State Grid Information and Telecommunacation Co, Ltd(SGIT[2010]449); The Planned Science and Technology Project of Guangdong Province (2010B050100009)

事件同其他事件联系起来进行检测，在特定的上下文中分析计算，最终由简单事件产生输出复杂事件。

复杂事件处理系统所需处理的数据流量往往非常大，然而用户却希望实时地得到结果。面对海量而关系复杂的信息，系统还须进行快速地计算继而快速决策，这对系统的吞吐量提出了很高的要求，单一节点的处理速度往往难以满足需要。

MapReduce[3,4]是目前应用极为广泛的并行软件构架。它使用键值对描述数据，将复杂的运算分成“Map（映射）”和“Reduce（化简）”2 个独立步骤进行处理。Map 和 Reduce 都是对其输入数据中的元素进行独立地操作，所以这 2 个运算步骤都是可以并行的，但 Reduce 操作要等待所有 Map 操作完成才能开始。MapReduce 模型可以很容易地将多个通用批处理任务和操作在大规模集群上并行化。但 MapReduce 是批量处理数据的，它将输入数据切成小的片段，每隔一个周期就启动一次 MapReduce 任务。处理任务不是常驻服务，数据也不是实时流入，任务的分割难以满足数据流实时处理的需求。因此，MapReduce 框架不是并行 CEP 的合理方案。

Storm[5]是一种分布式实时计算系统框架，它由一个主节点和多个工作节点组成。主节点用于分配代码、布置任务和故障检测。工作节点用于监听工作、开始并终止工作进程。它以有向图定义系统的逻辑拓扑结构，每个节点将以指定的进程数动态分布在集群中，而事件消息按照预定义的分组方式分发至逻辑节点对应的处理进程。作为编程框架，它实现了事件驱动的实时处理，但未提供对复杂事件处理中事件关系处理的支持，需要用户根据场景自行开发，其并行方式也需要用户按照经验自行配置，普通用户难以使用。

本文针对复杂事件处理对吞吐量的需求，将通用 CEP 系统搭建在使用 S4[6]并行处理框架的集群上，使用基于操作符的负载分流方法，将事件流分散到多处理节点上并行处理。既方便了普通用户对 CEP 系统的配置，又提高了 CEP 系统的处理效率。

2 并行 CEP 系统

复杂事件处理通常按照应用场景的需求，由一系列的事件生产者、处理代理、事件消费者组成。逻辑上是一个有向图，事件按照边进行传递，每个代理表示对于事件的一个操作。事件从生产者处被生产出来，最终流到了消费者，从而完成了处理过程。

把操作符看做对于复杂事件处理的基本功能单位。各种操作符组合，以共同完成对于复杂时间的处理。将事件处理中的处理逻辑抽象出来，组合成一个一个的事件代理，而不是按照具体应用程序定制处理过程，这种方法更为实用和强大。设计了 14 种操作符，这些操作符可过滤事件，对事件进行多种变形和提供流上的模式识别功能以及一些简单的系统辅助功能。不仅使单个操作符的功能具有实用性，而且使多个操作符组合后能够满足对于事件分析的需求。在操作符的基础上进行 CEP 场景建模，使普通业务人员也能高效地定义配置 CEP 系统。

S4（simple scalable streaming system）是一个去中心的、分布式的、可扩展的流式处理系统。类似于 Storm 的消息分组策略，S4 框架通过对流事件的关键字进行散列计算，得到处理事件的节点编号，再将事件发送到对应节点进行处理。通过修改 S4 框架的事件分流机制，基于操作符的不同，使用不同的分流策略，实现了并行的通用 CEP 系统。图 1 描绘了其系统结构。图中箭头方向为事件流向。处理节点通过网络连接在一起，单个处理节点由 CEP 引擎、事件接收器、事件发送器和事件分流器组成。

事件接收器用于接收其网络层发来的或处理节点内部的事件消息，它将按照事件流名称将事件放入不同的流中，交给 CEP 引擎处理；CEP 引擎以操作符为基本运行单位，事件根据其流的类型交由特定的操作符处理，处理结果将发往分流器；分流器应用负载分流策略，基于目标操作符和事件内容进行运算，决定该事件由哪个节点处理。如果是本地节点，事件将进入本地的接收器，重新发送给 CEP 引擎；如果是其他节点，将送至事件发射器，通过网络发送给其他节点；发射器负责将事件发送给集群上的其他节点。S4 初始版本只支持 UDP 的发送方式，将其改为带有队列的 TCP 方式，虽然增加了集群内部连接数，但防止了事件的丢失。

该系统通过分流器，将事件分流至不同的处理节点，使得事件流得以在集群中多机并行处理，提高了系统整体吞吐。在系统节点部署时，各个 CEP 引擎中使用相同的操作符配置。节点部署的物理机器可手工配置，也可通过 Zookeeper[7]协同自动分配。通过提供备用节点，还可提高系统的可靠性。

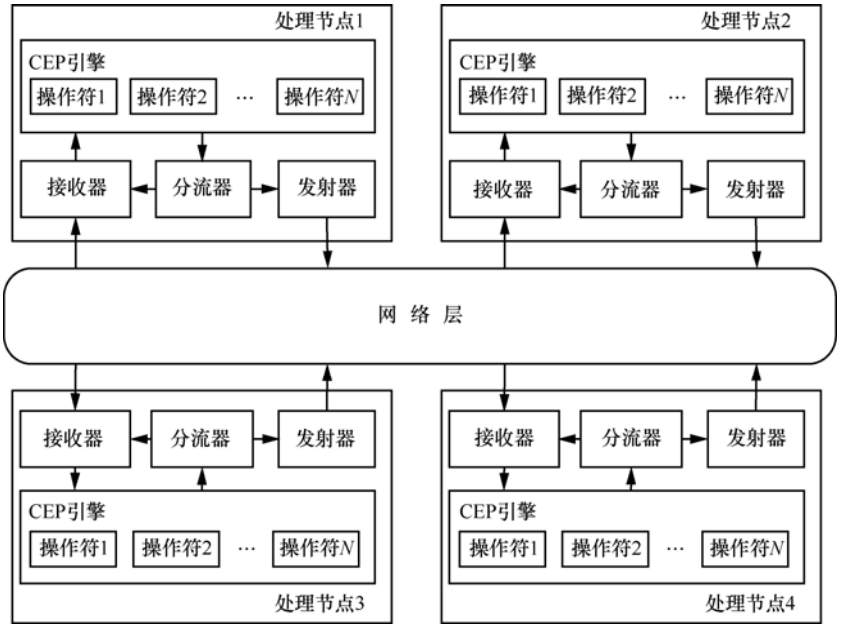


图 1　并行 CEP 结构(4 节点)

3　基于操作符的负载分流优化方法

当面对高速数据流时，如果能将负载均匀地分流到集群的各个节点，并行地处理事件流，则可以增加系统的吞吐量。然而复杂事件处理的特殊性在于事件往往不是孤立地存在，在处理中需要根据事件流上下文或其他事件流进行决策，单纯地拆分数据流到不同节点，会影响事件间的关系，导致处理出错。

基于操作符的负载分流方法，根据目标操作符的不同，使用不同的分流策略，实现动态分流。本节将给出 6 种操作符的分流方案。

操作符的输入流和输出流均是由事件组成的，将事件表示为(K, V)的集合。输入流事件表示为(K_i, V_i)，输出事件流表示为(K_o, V_o)，中间结果表示为(K_m, V_m)。每个操作符都是在输入流上进行计算，这个计算过程可以用函数 *operator* 来表示，最终将产生的结果放入输出流。操作符的这一处理过程用如下的表达式来表示

$$(K_i,V_i)\rightarrow operator\rightarrow(K_o,V_o)$$

1) 过滤操作符。首先使用分流器输入分散到各处理节点，然后每个处理节点分别进行原本的过滤操作，最后合并结果。这一分流过程可表示为

$$(K_i,V_i)\rightarrow\frac{filter_1\rightarrow(K_{1m},V_{1m})}{filter_2\rightarrow(K_{2m},V_{2m})}\rightarrow union\rightarrow(K_o,V_o)$$

2) 映射操作符。首先使用分流器输入分散到各处理节点，然后每个处理节点分别进行原本的映射操作，最后将结果合并。这一分流过程可表示为

$$(K_i,V_i)\rightarrow\frac{map_1\rightarrow(K_{1m},V_{1m})}{map_2\rightarrow(K_{2m},V_{2m})}\rightarrow union\rightarrow(K_o,V_o)$$

3) 分裂操作符。首先使用分流器输入分散到各处理节点，然后每个处理节点分别进行原本的分裂操作，最后须按原分裂目标合并结果，这一分流过程可表示为

$$(K_i,V_i)\rightarrow\frac{split_1\rightarrow\dfrac{(K_{1m}^1,V_{1m}^1)}{(K_{1m}^2,V_{1m}^2)}}{split_2\rightarrow\dfrac{(K_{1m}^1,V_{1m}^1)}{(K_{2m}^2,V_{2m}^2)}}\rightarrow\frac{\dfrac{(K_{1m}^1,V_{1m}^1)}{(K_{2m}^1,V_{2m}^1)}\rightarrow union_1}{\dfrac{(K_{1m}^2,V_{1m}^2)}{(K_{2m}^2,V_{2m}^2)}\rightarrow union_2}\rightarrow\frac{(K_o^1,V_o^1)}{(K_o^2,V_o^2)}$$

4) 合并操作符。首先使用分流器输入分散到各处理节点，然后每个处理节点分别进行原本的合并操作，最后将结果合并，这一分流过程可表示为

$$\frac{(K_i^1,V_i^1)}{(K_i^2,V_i^2)}\rightarrow\frac{union_1\rightarrow(K_{1m},V_{1m})}{union_2\rightarrow(K_{2m},V_{2m})}\rightarrow union\rightarrow(K_o,V_o)$$

5) 查询操作符。查询表和物化窗口都属于辅助表操作符，它们在内存中建立一块数据的缓存，供一个或者多个查询操作符进行插入、查询、更新或

删除操作。分流的方案是将插入表的输入流数据通过分流器分散到各处理节点，在每个节点上建立表，其他操作的输入流需要复制到每个节点，最后将结果合，并这一分流过程可表示为

$$(K_i,V_i)\rightarrow\frac{query_1^{QT/M1}\rightarrow(K_{1m},V_{1m})}{query_2^{QT/M2}\rightarrow(K_{2m},V_{2m})}\rightarrow union\rightarrow(K_o,V_o)$$

6) 聚集操作符。它根据 2 个及 2 个以上输入流的属性值相等性进行连接操作。使用分流器将输入流上键值属于同一个范围的事件分流到相同节点上，各节点再分别进行聚集操作最终将结果合并。这一分流过程可表示为

$$\frac{(K_i^1,V_i^1)}{(K_i^2,V_i^2)}\rightarrow\frac{gather_1\rightarrow(K_{1m},V_{1m})}{gather_2\rightarrow(K_{2m},V_{2m})}\rightarrow union\rightarrow(K_o,V_o)$$

在实际应用中，不需要将结果进行和并，因为操作符是连续进行流水线作业的，结果直接通过分流器进入下一个分流阶段。对于无法分流的操作符，如依据事件到达顺序进行计算的操作符，仍可配置分流器分流到指定的节点。

基于操作符特点进行动态地分流，既使分流工作对用户透明地自动完成，也使分流工作有针对性，有助于减轻处理节点的负载，整体上提高系统效率。

4 实验结果与分析

为了验证并行 CEP 系统的性能，在 8 台机器构成的集群上进行了实验，每台机器的配置为主频 1GHz，8 核处理器，8G 内存。

4.1 实验配置

实验主要测试的是事件处理吞吐量。它反应了系统在单位时间内能处理的事件的规模，用每秒钟处理的事件数量来量化表示。计算的方法是用总共处理的事件数量除以处理的时间。

数据来自国家电网智能园区的用电数据来模拟输入的事件流。事件输入流包括电表电量流、电价流和企业信息流 3 个。其中最重要的是电表电量流，它是每 10s 采集一次的电表读数。使用了 200 块电表一天内采集的所有读数，共 1 728 000 条数据。实验的方法是在是持续不停地向并行 CEP 系统中发送电表数据，在输出端进行吞吐量的度量。

4.2 实验结果

实验实时计算指定时间间隔内各个企业的电费。采用了 1~8 台机器对输入流进行相同逻辑的复杂事件处理。3 个查询所包括的操作符如图 2 所示，所测得的系统的吞吐量如图 3 所示。

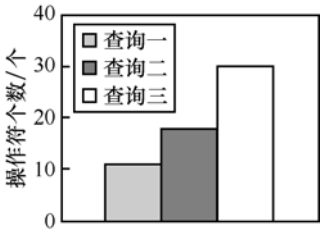


图 2 查询所包含的操作符数量

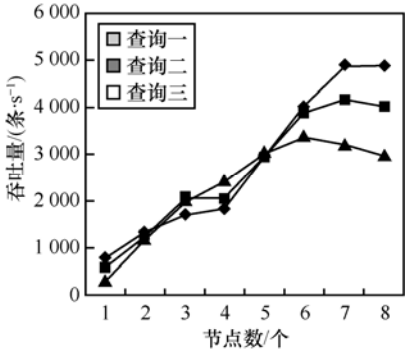


图 3 不同节点数的事件吞吐量

由图 2 和图 3 可见，查询一、查询二和查询三在节点数目增加的情况下吞吐量有显著的提升。并且，吞吐量当到达峰值后，由于节点的数目的增加会大大增加消息传递的通信开销，吞吐量会减少。由于吞吐量同查询的复杂程度相关，可见查询一的峰值吞吐量最大，而查询三的峰值吞吐量最小。而由于处理节点数目越多，节点间的通信开销所占比重越大。并且，受限于集群的网络环境，通信开销极大地影响了系统的吞吐量。所以，当节点的数目继续增加时，查询三的吞吐量减少得更为明显。

对 6 种操作符的负载分流性能进行了评测。对于每个操作符，分别在不同分流的数目情况下进行了实验。以下实验结果均是在 4 个节点的分布式环境下得到的。

图 4 显示了不同的操作符所设计的分流方案所包含的操作符个数。图 5 显示了实验所测得的不同操作符在不同的分流方案下运行所得的吞吐量。由图 4 和图 5 可知，在分流的数量增加的情况下，系统的吞吐量有所提升。而由于分流数量增加，事件的通信开销会增加，所以当分流数量进一步增加的情况下，系统的吞吐量会降低。由于所涉及的操作符的运算开销相差不大，所以各个操作符的实验结果无明显区别。

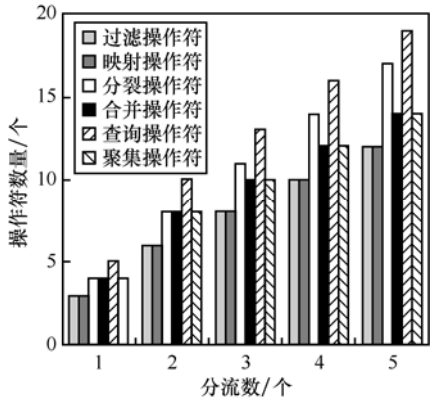


图 4　负载分流后所得的实际操作符数

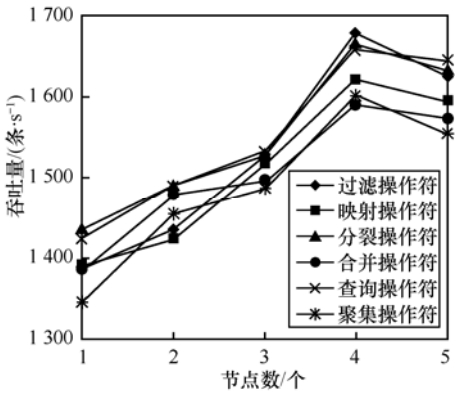


图 5　操作符在不同的负载分流方案下的吞吐量

实验结果表明，并行 CEP 系统的吞吐量在一定范围内随着集群处理节点数的增加线性增长，数倍于单机的处理能力。

5　结束语

本文针对复杂事件处理系统要求处理海量事件、进行复杂计算和简易直观建模的需求，设计了 14 种操作符给出了 6 个操作符的负载分流方案，然后在 S4 框架下搭建了并行 CEP 系统的方法，实验证明并行 CEP 系统能有效地提高系统的吞吐量。进一步工作将深入分析其他操作符的分流方案，提高系统的并行率。

参考文献：

[1] LUCKHAM D C, VERA J. An event-based architecture definition language[J]. IEEE Transactions on Software Engineering, 1995, 21(9): 717-734.

[2] LUCKHAM D C. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems[M]. Boston: Addison-Wesley Professional, 2002.

[3] DEAN J, GHEMAWAT S. Mapreduce: simplified data processing on large clusters[J]. Commun ACM, 2008, 51(1): 107-113.

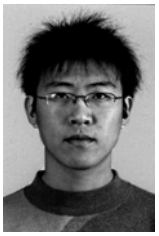
[4] CONDIE T, CONWAY N, ALVARO P, *et al*. Map reduce online[A]. Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation[C]. Berkeley, San Jose, CA, USA, 2010. 21.

[5] Storm wiki[EB/OL].http://github.com/nathanmarz/storm/wiki,2012.

[6] NEUMEYER L, ROBBINS B, NAIR A, *et al*. S4: distributed stream computing platform[A]. The 10th IEEE International Conference on Data Mining Workshops[C]. Sydney, Australia, 2010. 170-177.

[7] Zookeeper[EB/OL]. http://zookeeper.apache.org/,2012.

作者简介：



陈皓（1984-），男，黑龙江佳木斯人，中国科学院博士生，主要研究方向为分布式系统与中间件、服务组合、复杂事件处理。

李瑜（1986-），女，重庆人，中国科学院硕士生，主要研究方向为复杂事件处理。

虎嵩林（1973-），男，山西侯马人，中国科学院副研究员，主要研究方向为分布式系统与中间件、服务计算和事件计算。

梁英（1962-），女，山东潍县人，中国科学院高级工程师，主要研究方向为服务计算、企业信息集成和中间件等。