·体系结构与软件技术· 文章编号：1000-3428(2016)09-0015-06 文献标志码：A 中图分类号：TP311

多核平台下 Esper 数据流管理系统性能分析研究

王洪亚，张华庆，刘晓强

（东华大学 计算机科学与技术学院，上海 201620）

摘 要： Esper 事件处理系统可用于复杂事件处理与数据分析，适用于处理大量历史的或实时的消息和事件流。分析多核计算平台下基于 Esper 引擎所构建的数据流处理系统，介绍基于 Esper 引擎实验平台的设计与实现，给出完整的查询语句和测试用例，并使用该实验平台对多核平台下 Esper 引擎的性能进行测试，通过实时监控和离线数据分析给出系统的各项性能指标。实验结果表明，Esper 数据流系统对多核平台并不能够提供良好的支持。

关键词： 事件流处理；复杂事件处理；多核平台；Esper 引擎；数据流管理系统

中文引用格式： 王洪亚，张华庆，刘晓强. 多核平台下 Esper 数据流管理系统性能分析研究[J]. 计算机工程，2016，42(9)：15-20.

英文引用格式： Wang Hongya，Zhang Huaqing，Liu Xiaoqiang. Performance Analysis Research of Esper Data Stream Management System Under Multi-core Platforms[J]. Computer Engineering，2016，42(9)：15-20.

Performance Analysis Research of Esper Data Stream Management System Under Multi-core Platforms

WANG Hongya，ZHANG Huaqing，LIU Xiaoqiang

（School of Computer Science and Technology，Donghua University，Shanghai 201620，China）

【Abstract】 Esper event processing system can be used for complex event processing and data analysis，and it is suitable for handling a large number of historical or real-time news and event stream. This paper analyzes the data stream processing system based on Esper engine under multi-core computing platforms. It describes the design and implementation of experimental platform，designs complete query statements and test cases，and uses the experimental platform to test the performance of Esper engine under the multi-core computing platforms. Real-time monitoring and offline data analysis are used to test the system performance. Experimental results show that Esper data stream processing system can not provide better support under the multi-core platforms.

【Key words】 Event Stream Processing(ESP)；Complex Event Processing(CEP)；multi-core platform；Esper engine；data stream management system

DOI： 10.3969/j.issn.1000-3428.2016.09.003

1 概述

复杂事件处理(Complex Event Processing，CEP)是一种新兴的基于事件流的技术，用于处理事件并从事件流中发现复杂的模式。事件流处理(Event Stream Processing，ESP)目的是从事件流中获得有用的事件，进而从中获得有用信息。CEP 和 ESP 系统在流数据处理和实时响应方面非常出色。

由于功耗的原因，处理器的主频难以提升，因此目前处理器的发展主要有 2 个方面：(1)单核内部架构的优化，提高计算过程效率；(2)在单一的处理器中增加更多的核心，通过并行来提高性能。目前架构的优化经过近年的发展，已经有相当大的提升，在可预见的一段时间内，暂时还看不到能带来有显著效率提升的架构变化；而核心数在可预见的一段时间内可以有较大提升。因此，多核计算机在目前应用非常普遍。

Esper 是开源 CEP 引擎中比较优秀的一款，它有 Java 版本和. NET 版本(NEsper)[1]。Esper 是可以用来处理大量消息和事件的应用程序，它可以从不

基金项目： 国家自然科学基金资助项目(61370205)；上海市自然科学基金资助项目(13ZR1400800)；中央高校基本科研业务费专项基金资助项目。

作者简介： 王洪亚(1976－)，男，教授，主研方向为数据库技术、实时计算、移动计算；张华庆，硕士研究生；刘晓强，教授。

收稿日期： 2015-09-06 **修回日期：** 2015-11-11 **E-mail：** hywang@dhu.edu.cn

同的角度来过滤和分析事件并且在适当的条件下做出实时的响应。因此,Esper 可以应用在股票系统、风险监控系统等要求实时性比较高的场景。近年来,有一些国外文献对 Esper 的性能做了研究分析[2-4]。Esper 官方数据显示,在双 2 GHz CPU 的 Intel 系统测试环境下,处理速率超过每秒 500 000 个事件;在 VWAP 基准测试中有 1 000 个语句的情况下,引擎延时平均小于 3 μs[5]。然而官网的统计数据只是针对一些相对简单的查询得到的结果,复杂查询以及多核情况下 Esper 的性能情况如何却不得而知,现有文献也并未对该问题进行讨论。

针对上述问题,本文设计一种基于 Esper 引擎的 Benchmark 实验平台,将各种查询语句分为四大类,展开对 Esper 引擎在多核平台下性能的研究与分析,帮助开发者更好地了解 Esper 引擎在多核平台下的优劣势,以便于开发基于开源 Esper 引擎的更有效的流处理系统。

2 相关研究

近年来,有国内外文献在数据流管理系统的性能和对基于 Esper 引擎的 CEP 系统方面进行了研究。文献[2]通过很多小型测试用例对比了 3 种不同的数据流管理系统的性能,其中一种就是基于 Esper 引擎的。文献[3]针对 CEP 系统设计了一个 CEPBen 的查询测试用例,展开对于过滤、转换和模式匹配查询的各种测试,并且主要从负载量和吞吐量两方面对结果进行了分析。文献[4]针对 S,S4 和 Esper 3 种数据流系统构建了 70 多个不同的应用场景进行测试,结果表明 3 种系统各有优劣。文献[6]就如何构建低延迟、高吞吐且持续可靠运行数据流式计算系统提出了一些策略和方法。文献[7]通过一个系统实例详细讲解了大数据流式计算的一些关键技术。文献[8]针对轻量级和可扩展的 CEP 系统进行了研究和讨论。文献[9]介绍了基于 Esper 引擎的高效外汇数据分析系统的构建,详细描述了 Esper 引擎是如何嵌入到该系统中并进行工作的。文献[10]介绍了在处理大量事件流时应如何进行数据的分析和挖掘。文献[11]提出了 2 款开源 CEP 引擎的很多用于测试模式匹配查询性能的测试用例。文献[12]设计的 BiCEP 和文献[13]设计的 SPECjms2007 都是基于 CEP 系统的一些压力和性能测试用例,BiCEP 中主要从可持续吞吐量、响应时间、共享计算和相似性搜索等方面进行了更加深入的分析研究,而 SPECjms2007 是一个基准测试用例,通过提供标准的工作量来对相似竞争产品的性能进行评估。文献[14]描述了基于 WebLogic 事件的 CEP 系统性能的研究,主要针对 CEP 系统中延时情况的探究。文献[15]主要针对 CEP 系统在提高吞吐量性能等方面进行了研究分析。

3 实验设计

3.1 Benchmark 实验平台设计

Benchmark 实验平台分为 Server 端和 Client 端两部分。Client 端主要负责多个数据流的生成、组装和发送,Server 端集成了 Esper 引擎,负责数据流的接收、查询语句的注册和查询结果的输出。图 1 所示为 Esper 引擎的工作机制。

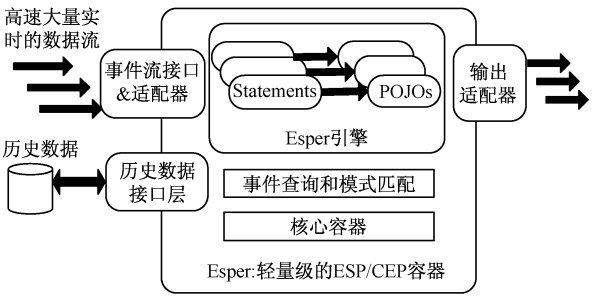

图 1 Esper 引擎的工作机制

Client 主要由 Socket 连接模块、事件生成模块和数据流传输模块 3 个部分组成。首先由 Socket 连接模块初始化主机端口,建立与 Server 端的连接,然后由事件生成模块对不同的事件进行组装,组合成数据流交给数据流传输模块进行打包发送。

Server 端主要有 Socket 连接模块、数据流接收处理模块、查询语句注册模块、统计模块 4 个部分组成。Socket 连接模块主要功能是初始化主机端口,建立与 Client 端的连接。数据流接收处理模块的主要功能是将 Client 端打包发送过来的数据流,通过相应的方法策略进行拆分读取,然后识别不同的事件进行相应的处理。查询注册模块主要功能是从文件中批量读入各类查询语句,将不同的查询语句注册到在 Server 端封装好的 Esper 引擎中,执行不同的查询,并返回查询结果。统计模块主要功能是用来统计任务执行过程中系统资源的使用情况等性能指标。

3.2 数据流的构建

事件是数据流的一个基本单位,一个事件就是一个元组或结构体,代表一条记录。本文系统设计了 3 种不同的事件类型:MarketData, NewsData, UserData。

每种事件的结构如表 1 ~ 表 3 所示。

表 1　MarketData 事件结构

字段	类型	内容
ticker	String	股票号
price	double	股票的价格
volume	int	股票的成交量
country	String	股票所属国家

表 2　NewsData 事件结构

字段	类型	内容
newsId	String	新闻 ID
ticker	String	新闻报道的股票
date	String	新闻报道的日期
organisation	String	新闻媒体组织

表 3　UserData 事件结构

字段	类型	内容
userId	String	用户 ID
newsId	String	新闻 ID
ticker	String	股票号
holdvolume	String	股票持有量

整个数据流是由测试平台的 Client 生成的,3 种不同的事件记录随机发送,可以调整不同事件记录在数据流中的比例。数据流发送到 Server 端之后由 Server 端来对不同的事件记录进行处理:首先 Server 端通过读取事件记录的第一个字节,匹配其 ASCII 码值来识别该记录属于哪一类事件类型;然后针对不同的事件记录调用不同的方法进行解析处理,最后通过 Esper 引擎做连接查询等测试。

在该过程中,检测不同事件的方法通过事件记录各个字段的字节长度和类型匹配两方面进行验证,发现有任何不匹配的记录,即进行剔除操作。

3.3　性能指标

本文系统关注的性能指标主要是任务执行过程中系统资源的使用情况,主要有以下 3 个方面:

(1)CPU 核心使用率:任务执行过程中 CPU 的每个核心的使用率情况。

(2)内存使用率:任务执行过程中系统整体内存的使用率情况。

(3)吞吐量:能够正常执行任务的情况下 Server 端每秒钟所能处理最大的事件个数。

3.4　查询设计

事件处理语言(Event Processing Language,EPL)是 Esper 引擎中可以识别和执行的查询语言,它是一种类 SQL 语句,EPL 可以方便地对事件串流提供复杂的逻辑处理,使事件串流在内存中做模式匹配处理及查询的动作。实验中设计了比较完整的查询测试用例,一共包括四大类查询语句,查询语句的标号按照以下规则设置:

(1)第 1 类为比较简单的查询,该类查询语句以 S(Simple)开头,主要实现对单个数据流进行选择、投影、过滤等操作;第 2 类为聚类查询,该类查询语句以 A(Aggregation)开头,主要实现计数、求和、求均值、求最大最小值、求方差等聚合操作;第 3 类为连接查询,该类查询语句以 J(Join)开头,主要实现对数据流中不同的事件进行连接操作,本文共有3 种类型的事件,通过该操作求得对不同类型事件连接的结果;第 4 类模式为匹配查询,该类查询语句以 P(Pattern)开头,模式匹配查询也是 Esper 中非常灵活实用的查询语句,涉及到事件时序性的操作一般通过该类查询实现。

(2)EPL 中提供一种命名窗口的机制来处理数据流,有长度窗口和时间窗口两大类。功能基本类似但命名窗口不同的查询语句以下划线的形式来命名,比如查询语句 S2_1 和 S2_2,代表了一组功能相似窗口不同的 2 个查询,一个是时间窗口,另一个是长度窗口。

4　关键技术

4.1　Esper 构建流处理系统的方法

把 Esper 引擎集成到独立应用中需要以下 4 个必要的步骤:

(1)创建一个或几个 Java Event Class,用来定义事件的结构。

(2)获取一个 Esper 引擎实例,生成一个用 EPL 来表示的 Statement,使用引擎注册这个 Statement。

(3)生成一个 Listener(通过实现一个 Java 接口,该接口在 Statement 所得值为 true 会被触发),并把它跟 Statement 绑定起来。事件能以 Java 对象、XML 或 Map 的形式展现,当它们通过系统时,系统会评估 Statement 的值,并执行 Listener 中的逻辑。

(4)获取引擎运行接口 EPRuntime,调用sendEvent()方法向引擎发送事件。

4.2　线程池多线程技术

系统主要针对多核平台下 Esper 引擎的性能状况进行探究分析,因此,必然要用到线程池多线程技术。

一个典型的线程池包括以下 5 个部分:(1)线程池管理器,用于启动、停用和管理线程池;(2)工作线程,线程池中的线程;(3)请求接口,创建请求对象,以供工作线程调度任务的执行;(4)请求队列,用于存放和提取请求;(5)结果队列,用于存储请求执行后返回的结果。

线程池工作原理如下:

(1)线程池管理器通过添加请求的方法,向请求队列添加请求,这些请求事先需要实现请求接口,即传递工作函数、参数、结果处理函数以及异常处理函数。

(2)初始化一定数量的工作线程,这些线程通过轮询的方式不断查看请求队列,只要有请求存在,就会提取出请求,进行执行。

(3)线程池管理器调用方法查看结果队列是否有值,如果有值,则取出,调用结果处理函数执行。

如图 2 所示为整个线程池工作模型。

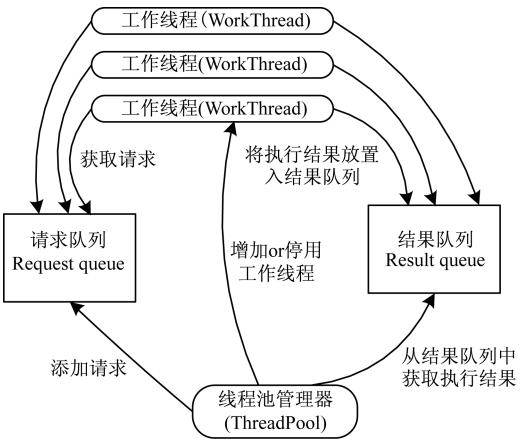

图 2 线程池工作模型

4.3 统计系统资源使用率的方法

整个系统运行的过程分 2 个部分:(1)对查询语句执行中的性能指标的实时监控;(2)对系统资源利用率的离线数据分析,离线数据是通过开源 Sigar 工具来辅助收集得到的。

Sigar 的全名是 System Information Gatherer And Reporter,中文名是系统信息收集和报表工具。Sigar 是一个开源的工具,提供了跨平台的系统信息收集的 API,可以收集的信息包括:操作系统信息,CPU 信息,内存信息,进程信息,文件系统信息,网络接口信息等。Sigar 集成到系统中只需要在项目中添加 sigar.jar 以及对应系统的库文件即可,64 位 Linux 操作系统下依赖 Libsigar-amd64-linux.so 库文件。

本文系统中主要统计 CPU 每个核心的利用率和内存的利用率等信息。通过调用 Sigar 的 getCpuPercList 方法可以找到每一个 CPU 的对象,然后通过 getSys,getUser,getCombined,getWait,getIdle 等方法可以获取到 CPU 系统使用率、用户使用率、总的使用率、当前等待率和当前空闲率等信息。通过调用 Sigar 的 getMem 方法可以获取到内存 Mem 类对象,通过该对象调用 getUsed,getTotal 等方法可以获取到内存的使用量和内存总量,通过两者的比值就可以得到内存的使用率。

5 系统功能与数据处理

5.1 系统环境

系统运行环境和开发工具如表 4 所示,其中,CPU 为 4 核心,3.40 GHz 的主频。Linux 内核版本为 Linux Ubuntu 3.5.0-17-generic。

表 4 系统运行环境和开发工具的版本号或型号

运行平台或工具	版本号或型号
Linux	Ubuntu12.10
CPU	Core i7-4770
RAM	DDR3 8GB
JDK	1.7.0
Eclipse	4.3.2
JFreeChart	1.0.6
Python	2.7.6

5.2 查询注册

系统通过一个 Query.txt 的文本文件来灵活配置注册到系统的查询语句。查询语句分为四大类:选择查询,聚合查询,连接查询和模式匹配查询,每一类查询都设计一些比较有代表性的查询语句,将 EPL 中比较灵活的窗口、强大的 pattern 模式匹配等功能都体现在设计的查询语句中。

配置文件中的记录格式包括 3 个部分:(1)查询语句的标识符;(2)查询语句原型;(3)数字代表该条查询语句注册到 Esper 引擎的数量。

5.3 系统工作流程

首先设置传输速率,定义统计输出文件的路径和文件名,然后启动系统的 Server 端程序,再启动系统 Client 端程序,让其建立 Socket 传输,Server 端会将配置文件中的所有查询注册进入引擎,执行各种查询。

对于任务执行过程中各项性能指标信息采用 2 种方式处理:一种是实时监控,通过图形界面及时反映系统资源的利用率情况;另一种是离线数据的统计分析,即将各类性能指标信息收集记录到文本中,之后再做分析研究。

实时监控采用 Sigar + JFreeChart 模式来实现。JFreeChart 是 Java 平台上的一个开发的图表绘制类库。通过两者结合,调用 Sigar 的 API 获取到系统信息,然后通过 JFreeChart 绘制的图表进行实时的显示。

图 3 和图 4 分别为任务执行过程中对 CPU 每个核心的利用率和 JVM 内存使用率进行的实时监控情况。

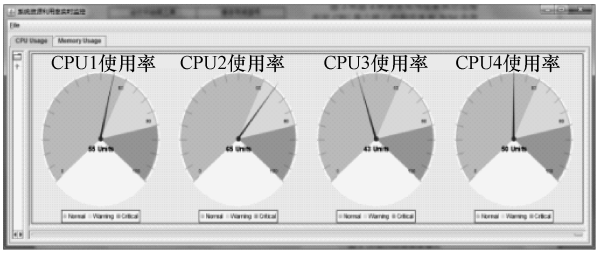

图 3　CPU 每个核心利用率情况

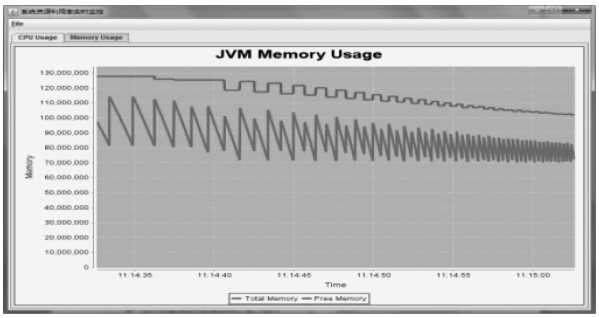

图 4　JVM 内存使用率情况

离线数据统计分析,首先通过系统的统计模块,将任务执行过程中要收集的系统性能指标信息进行统计保存在文本中,由此得到实验的原始数据。

5.4　数据处理

系统中生成的实验原始数据存放在是 txt 文件中,该文件的命名是由 4 个部分组成的:注册语句条数,窗口大小,输入速率和线程池队列长度。例如 3_10_50000_5. txt 代表是查询语句注册条数为 3、窗口长度为 10、输入速率为 50 000 events/s、线程池中队列长度为 5 时系统执行时得到的数据。

用写好的 Python 脚本对原始数据进行二次处理,分析研究各项性能指标。在第二阶段对实验原始数据进行处理的过程中用到了 NumPy, pandas, matplotlib, IPython, SciPy 等重要的 Python 库。

为了完成更加精准的测试,在数据处理过程中还有一个重要的问题就是要剔除部分不合适的数据,因此,在系统刚开始运行时,吞吐量远没有达到设定值时的一些记录可以剔除掉,此段时间为系统的预热时间,在系统达到正常状态下开始统计结果。

6　实验结果与分析

通过实验数据发现,Esper 引擎运行在多核平台下的性能并没有太多的提升,系统最大吞吐量变化不大,反而 CPU 使用率有了较大增加。可能原因是 CPU 核心数比较多时,线程池多线程的加锁控制以及管理代价等浪费了比较多的系统资源。实验结果还表明多核平台下增加线程数量和队列并没有导致 Esper 性能显著提高。

图 5、图 6 为单线程和多线程情况下 CPU 使用率的比较,且此时两者吞吐率是基本一致的。可见,线程池多线程的应用并没有导致吞吐量的提高。因此,多线程没有提高系统的吞吐量将成为后续研究的主要问题。

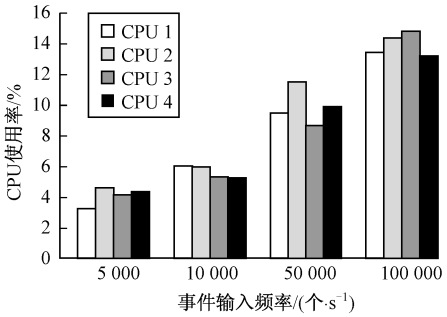

图 5　单线程 CPU 使用率随输入频率的变化

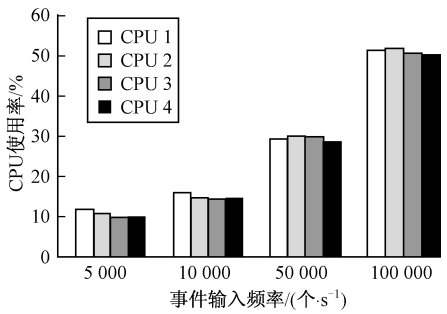

图 6　多线程 CPU 使用率随输入频率的变化

图 7 为不同输入频率在 1 min 的时间内对 CPU 方差的统计,因为系统有预热过程,所以要剔除掉部分时刻的数据,该实验中剔除了实验 1 min 前 10 s 的数据,因此,测试运行时间节点总共为 50 个。说明多核平台下随着数据流输入频率的增加,CPU 利用率的方差也随之增大,从而表明 CPU 不同核心利用率随输入频率的增大而越来越不均衡。

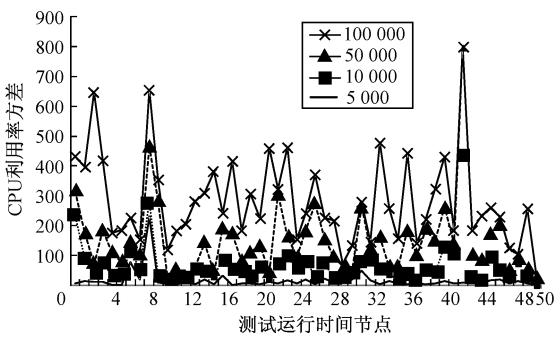

图 7　多线程 CPU 利用率方差随输入频率的变化

同时实验中还发现,在不采用线程池的情况下,CPU 方差较大,CPU 的每个核心利用率是非常不均

匀的。采用线程池多线程技术以后 CPU 的每个核心的利用率更加均匀,但是 CPU 每个核心的利用率都增大了很多,说明 Esper 引擎不能充分利用多核带来的优势,反而会添加额外的工作量。

在多核平台下 CPU 每个核心使用率在没有达到 100% 时整个系统吞吐率不再增加。为了分析其中的原因,实验中用 JProfiler 工具对 Server 运行的整个过程进行了监控。

JProfiler 是一个商业授权的 Java 剖析工具,它把 CPU、执行绪和内存的剖析组合在一个强大的应用中,它允许 2 个内存剖面评估内存使用情况和动态分配泄漏及 CPU 剖析,以评估线程冲突。JProfiler 提供 Eclipse 和 IntelliJ 等 IDE 的插件,该实验中将 JProfiler 工具集成到 Eclipse 中进行使用。用 JProfiler 对 Server 程序进行监控的结果如图 8 所示。

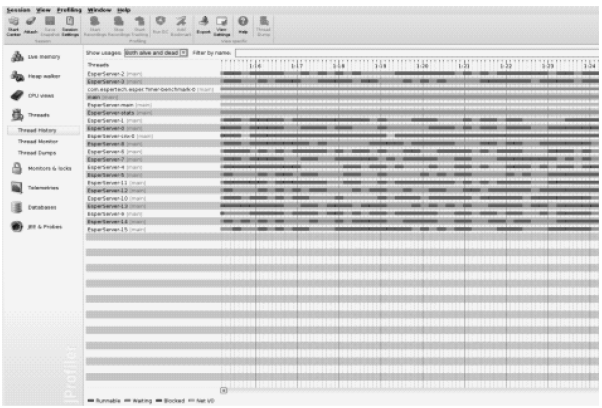

图 8 JProfiler 对不同线程状态的监控情况

从线程的执行状态中可以看到,系统资源不仅消耗在事件发送过程中,TCP/IP 传输以及事件对象的打包解析也消耗了不少的系统资源,同时这一部分的时间开支也是较大的。因此,构建数据流管理系统时也要考虑这方面的影响。至于多个线程的状态同时为 Runnable,却没有导致系统性能的提高,可能是由于 Esper 引擎内部资源不共享或加锁机制的一些设计缺陷原因造成的。

7 结束语

本文介绍了用于多核环境下 Esper 引擎性能的测试方法和测试平台,阐述了实验平台的设计、关键技术以及系统功能。本文的主要思路是探究 Esper 引擎集成的数据流管理系统在多核情况下的性能,因此,设计了多种结构类型的事件和非常完整的各类 EPL 查询测试用例。实验结果表明,在各类查询中多核的使用并没有在吞吐量性能上带来显著的提升。线程池多线程的使用使多个核心使用率更加平均的同时会增加额外的工作量,因此,在构建数据流管理系统时,如果处理的查询大多比较简单,不推荐使用线程池技术,反之,处理的复杂查询使用线程池技术会更加理想。

下一步将针对 Esper 引擎内部设计的一些缺陷,展开更加深入的分析研究,期望能够从 Esper 引擎的设计层面进行适当的改造,使其能够对多核 CPU 提供支持。

参考文献

[1] Esper[EB/OL]. [2015-10-21]. http://www.espertech.com/esper/.

[2] Mendes M R N, Bizarro P, Marques P. A Performance Study of Event Processing Systems[C]//Proceedings of LNCS'09. Berlin, Germany: Springer, 2009: 221-236.

[3] Li C, Berry R. CEPBen: A Benchmark for Complex Event Processing Systems[M]. Berlin, Germany: Springer, 2014.

[4] Dayarathna M, Suzumura T. A Performance Analysis of System S, S4, and Esper via Two Level Benchmarking[M]. Berlin, Germany: Springer, 2013: 225-240.

[5] EsperTech[EB/OL]. [2015-10-21]. http://www.espertech.com/esper/performance.php.

[6] 亓开元,赵卓峰,房 俊,等. 针对高速数据流的大规模数据实时处理方法[J]. 计算机学报,2012,35(3):477-490.

[7] 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,2014,37(4):839-862.

[8] Zappia I, Paganelli F, Parlanti D. A Lightweight and Extensible Complex Event Processing System for Sense and Respond Applications[J]. Expert Systems with Applications, 2012, 39(12): 10408-10419.

[9] 蔡昭权,吴文忠,卢庆武,等. 利用 HDF5 和 Esper 的高效外汇数据分析系统[J]. 计算机工程与科学,2011,33(4):159-163.

[10] Gaber M. Advances in Data Stream Mining[J]. Data Mining & Knowledge Discovery, 2012, 2(1): 79-85.

[11] Dekkers P. Complex Event Processing[D]. Nederland, Radboud: Radboud University, 2007.

[12] Bizarro P. BiCEP Benchmarking Complex Event Processing Systems[C]//Proceedings of Dagstuhl Seminar on Event Processing. Washington D. C., USA: IEEE Press, 2007: 458-467.

[13] Kounev S, Sachs K. Benchmarking and Performance Modeling of Event-based Systems[J]. Information Technology, 2009, 51(5): 262-269.

[14] White S, Alves A, Rorke D. Web Logic Event Server: A Lightweight, Modular Application Server for Event Processing[C]//Proceedings of the 2nd International Conference on Distributed Event-based Systems. New York, USA: ACM Press, 2008: 193-200.

[15] Isoyama K, Kobayashi Y, Sato T, et al. A Scalable Complex Event Processing System and Evaluations of Its Performance[J]. Ieice Technical Report Information Networks, 2012, 112: 31-36.

编辑 索书志