文章编号:1671-4598(2017)03-0055-05 DOI:10.16526/j.cnki.11—4762/tp.2017.03.016 中图分类号:TP399 文献标识码:A

基于 Storm 的城市消防联网远程监控系统的实时数据处理应用

杨素素
(首都师范大学 信息工程学院,北京 100048)

摘要:针对城市消防联网远程监控系统中实时信息数据逐渐增长而引出的大数据问题,传统的消防系统无法实时、高效地处理消防实时数据的问题,提出了一种基于云计算和 Storm 实时数据处理系统的解决方案;对于开源的 Storm 框架进行需求和性能分析,实现对其技术架构上的改进,并结合消防系统的特点,提出一套高实时性、高可扩展性的消防联网监控中心的数据实时处理的体系架构,同时也进行了云计算平台的搭建,利用心跳检测机制保证各个监控单位的实时性连接;研究表明,基于云计算和 Storm 平台架构完全适用于消防联网监控中心的实时消防数据的处理,具有高效性、高可靠性、性能显著等特性。

关键词:Storm;消防预警;大数据实时处理;云计算

City Fire Remote Minitor Network System Based on Storm for Real—time Data Processing Application

Yang Susu
(College of Information Engineering, Capital Normal University, Beijing 100048, China)

Abstract: In the city fire control network system, the data of real—time information has become larger and larger. The traditional fire control system cannot deal with the problem of real time data with high efficiency. We analyze the requirements and performance of the open source Storm framework, and implement the improvement on the technical architecture and the characteristics of the fire system. We put forward a set of system architecture for real—time processing of data in high real—time and high scalability of the fire control network monitoring center. At the same time, the construction of the cloud computing platform, using the heartbeat detection mechanism to ensure the real—time performance of the monitoring unit. The research shows that, based on cloud computing and Storm platform, the architecture is fully applicable to the fire control system, which has the characteristics of high efficiency, high reliability, high performance and so on.

Keywords: Storm; fire protection pre—warning; big data real—time processing; cloud computing

0 引言

近年来,各种火灾事故频发,有交通工具火灾事故、生产生活中的火灾事故、工业生产中的火灾事故。80%以上的这些火灾事故给人们带来很惨重的损失。据分析调查发现,很多我们生活中常见的火灾事故都是可以预先发现,并及时处理和疏散来减少损失的。这就对于火灾危险性进行预警分析和评价是十分重要的。

为了解决火灾及时报警、设备故障及时发现及时巡检,提高城市的消防减灾能力,各个城市都在努力建设消防联网平台。消防联网覆盖的范围越来越广,消防检测设备的监控探头和烟雾感应、温度感应检测等相关设备的检测数据量也逐渐增大,尤其是消防联网传感设备中的采集的数据对于火灾的预判和分析来说越来越重要,各个地区的上百万台设备传输过来的海量数据的处理也越来越迫切,大数据的实时处理问题也越来越突出。

现有的城市消防联网远程监控系统主要研究的是对消防数据的采集、以及采用更先进的传输设备和更可靠的传输协议对数据进行传输以保证数据的可靠传输。然而系统地解决消防预警困难,不仅仅需要数据的实时可靠传输,我们还需要对这些实时传感数据进行实时地分析和预测,实时监控、远程查看和远程控制,以达到火灾的及时预警、及时指挥、消防设备故障及时发现和整修。

现有的大数据处理方式分为流处理[1-3]和批处理两种,数据批处理主要是针对静态数据和变化比较慢的一些数据集的处理。对于高实时性要求的系统,特别是为了应对不断增长的数据,如消防的检测设备中实时检测到异常数据、设备每时每刻的工作状态的判断等都要求我们更多地关注流数据的实时处理。然而,现如今,现有的显著的流处理计算系统有 Linkedin 的 Kalka[4] 系统、Twitter 的 Storm[5] 系统、Yahoo 的 S4[6](simple scalable streaming system)系统、Facebook 的 Data Freeway and Puma[7] 系统、MillWheel[8]、Spark Streaming[9] 和 Photon[10] 等,流处理技术已经作为传统数据库成品输送管道中很重要的一部分[11-13]。本文主要在消防监控中心采用基于云计算平台和 Storm 实时数据处理的系统设计,并对系统整体性能进行测试。

1 适用于城市消防联网远程监控系统的云计算平台

目前消防行业里,随着科技技术的迅猛发展,各种各样具备先进性能的探测器和消防设备层出不穷,消防单位内所安装设备的种类大大丰富,所产生的信息量也是爆发式的增长,一

收稿日期:2016-01-12; **修回日期**:2016-02-24。
基金项目:国家科技支撑计划项目(2013BAH19F01)。
作者简介:杨素素(1990-),女,硕士研究生,主要从事云计算与数据实时处理系统及应用方向的研究。

个普通中等城市的数据量已经达到了很大的规模，而且随着城市建设的推进，数据量必然会达到大数据级别。城市消防远程监控系统是将监控中心与各个消防单位之间建立网络通讯，并结合运用地理信息系统、数字视频监控等现代网络信息技术，在监控中心内对所有在此联网内的建筑物的火灾报警情况进行实时监控、监测、对设备的运行情况进行集中监督和管理。同时国标要求：地级市以上城市（含地级，全国共 286 个），应设立城市消防远程监控系统；县级城市宜设立城市消防远程监控系统。

1.1 目前城市消防联网远程监控系统所受到的限制

目前我国的城市消防联网远程监控系统还处于厂家各自建设阶段。通常消防单位中消防监控系统的信息数据存储在各自的数据库中，城市消防控制中心对各个监控单位的数据库进行轮询，然后对查询到的数据进行处理，最后显示在页面上。这种方式在面对海量大数据的时候就显得力不从心，而且存在诸多问题：

（1）效率低：当前大多数的城市消防联网远程监控系统，都是消防单位将各自的数据存储在自己的数据库中，城市消防监控中心通过不断的查询各个监控单位的数据库来获取数据，然后对数据进行处理，最后显示在页面上。随着监控单位数据库数据量的不断增加，数据库的检索效率也会不断降低，系统效率很难确保。

（2）实时性差：由于当前的城市消防联网远程监控系统的效率低，消防中心得到数据的实时性就无法确保，至少会有几十秒左右的延迟，这几十秒的时间对于火灾预警来说至关重要，甚至关乎人的生命。对于消防监控中心来说数据的实时性是需要解决的最大的问题。

（3）维护难度大：当前的城市消防联网远程监控系统需要消防监控中心去不断的查询消防单位的数据库，从而给各个节点上的服务器都带来很大的压力，容易导致宕机。同时，系统的容错能力也较差，实际使用中系统的维护难度较大。

因此，高实时性、高效率、高性能的云计算集群架构的搭建是解决以上问题的最好方式，适用于城市消防联网远程监控系统的云计算平台的搭建是刻不容缓的。

1.2 城市消防联网远程监控系统的云计算平台的构建

本系统采用如图 1 所示的系统框架，根据消防行业的实际情况，消防单位之间没有必要建立数据连接，每个消防单位都直接和城市消防监控中心建立连接。采用心跳检测机制来保障中心实时地监控消防单位的消防系统中主要服务器的运行状况，消防监控中心实时采集数据并通过服务器集群进行高效分析和计算，最后显示在可视化界面上。

1.3 城市消防联网远程监控系统的云计算平台高实时性的体现

城市消防联网远程监控系统是一个需要实时反馈各个消防单位状态信息的系统，需要不断的获取消防单位数据服务器上的最新数据。因此数据采集的速度也直接影响系统能否做到高实时性，各消防单位采用规定好数据格式的 log 文件存储实时数据，并存储在数据服务器中，消防中心只读消防单位 log 文件上的新增数据，大大减小数据传输量。消防单位中的 log 文件按照按天存放，有效控制单个 log 文件的大小。这样，相较于传统数据库检索方式，数据采集做到了高效稳定。

1.4 城市消防联网远程监控系统的云计算平台高可靠性的体现

消防中心的服务器集群通过一个主服务器 master 服务器控制，集群中的服务器每隔一段时间就会发送一条状态信息给 master 服务器，一旦服务器集群中的某个服务器出现故障，master 服务器收不到该服务器的心跳信息，会将该服务器所处理的工作及时分配给别的服务器处理。并且 master 节点也有一个备用服务器，一旦 master 服务器宕机，备用服务器会立即投入使用。同时消防监控中心和消防单位之间也有一套心跳检测机制，消防单位中的服务器设备每隔一段时间就会发送一条状态信息给消防中心，如果状态信息接收不正常，则通知相应的监控单位对相关服务器设备进行及时地检修，保证平台的稳定运行。这种机制也保证了消防监控中心和消防单位之间保持可靠接连。

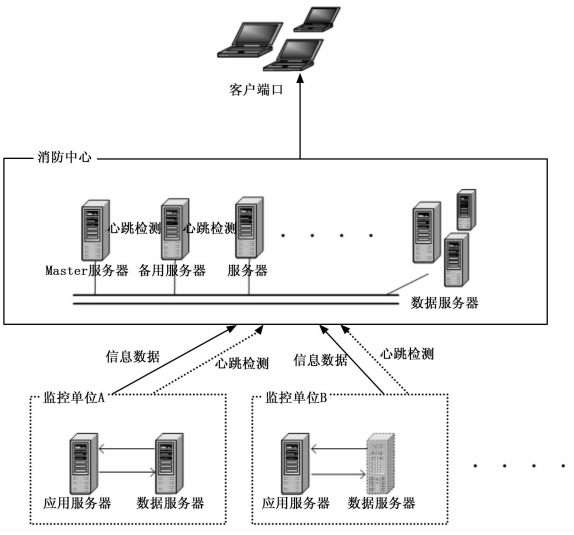

图 1 系统架构图

1.5 城市消防联网远程监控系统的云计算平台高可扩展性的体现

系统中的消防单位的识别信息都存在消防中心的 master 服务器的配置表中，对消防单位的编辑，只需修改配置表消防单位的信息。同时，如果发现集群中服务器的平均负载率过高（超过 80%），那么可以很方便的向集群中添加新的服务器，同样是在 master 服务器的配置表中添加新并入服务器的信息。

本文的系统平台进过实验，为城市消防联网远程监控系统提供了经济，稳定，高效，可靠，和方便扩展的云计算平台。

2 城市消防联网远程监控系统系统中数据的实时处理

本文针对消防系统中海量数据的实时采集和处理需求提出基于 flume、kafka、Strom 集群的城市消防联网实时监控系统。采用模块设计，每个组件完成自己特定的业务功能。本章主要介绍消防监控系统的组成部分、体系架构设计、和系统客户端结果展示等内容。

2.1 系统的组成部分

Flume 是一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统。本系统通过部署在消防单位上数据服务器上的 flume_agent，实时的采集 log 文件中新增的信息数据。

kafka 是一种高吞吐量的分布式发布订阅消息系统。因为需要在 flume 采集信息数据和 storm 处理数据之间建立缓冲中间件，所以使用 kafka 接受 flume 采集的数据，同时将数据以

消息队列的形式输送给 storm。

Storm 是一个开源的、大数据处理系统，旨在用于分布式实时处理且与语言无关。其使用核心在于拓扑结构的设计，我们需要把 kafka 中的消息数据传到到 storm 里，再照着设计好的拓扑结构给消防中心服务器集群中的机器分发数据进行处理，因此根据实际需求确定好机器数量是整个系统处理效率的关键。Storm 运行在一个分布式的集群架构上，Storm 集群由一个主节点和多个工作节点组成。

2.2　系统体系架构设计

城市消防联网远程监控系统采用四层架构：数据采集层、数据接入层、数据处理层和数据输出层。如图 2 所示。

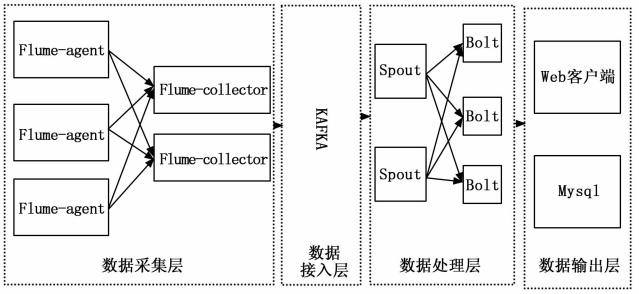

图 2　Storm 下的体系架构示意图

2.2.1　数据采集层

使用 Flume 实现分布式日志收集系统进行日志采集：log 文件中海量数据的高效收集是本系统的基石，本系统使用的 Flume 来进行日志采集。Flume 本身采用了分层架构：分别为 agent，collector 和 storage，Agent 负责日志收集工作，Collector 层负责接收 Agent 层发送的日志，并且将日志根据路由规则写到相应的 Store 层中，Store 层负责提供永久或者临时的日志存储服务，或者将日志流导向其它服务器。

Flume 具有的：(1) 高可靠性；(2) 易扩展及管理性 等优点是本系统采用其作为日志采集工具的主要原因。

其中 (1) 高可靠性体现在：当系统中消防单位服务器节点出现故障时（例如 agentA 出现了故障），其数据服务器上的日志文件能够被传送到备用数据服务器节点上（BackAgent1）而不会丢失。同时，Flume 提供了 3 种级别的可靠性保障，从强到弱依次分别为：end－to－end（收到数据 agent 首先将 event 写到磁盘上，当数据传送成功后，再删除；如果数据发送失败，可以重新发送。），Store on failure（这也是本系统采用的策略，当数据接收方服务器发生崩溃时，将数据写到本地，待恢复后，继续发送），Best effort（数据发送到接收方后，不会进行确认）。

(2) 易扩展及管理性：Flume 采用了三层架构，分别为 agent，collector 和 storage，每一层均可以水平扩展。其中 Agent 层中，每个监控单位的日志服务器部署一个进程，负责对监控单位的日志收集工作；Collector 层部署在消防中心服务器上，负责接收 Agent 层发送的日志，并且将日志根据路由规则写到相应的 Store 层中；Store 层负责将日志存储在本系统的下一个模块：kafka。

此外，本系统在 Agent 层向 Collector 层放送数据的时候使用负载均衡策略，将所有的日志均衡地发到所有的 Collector 上，达到负载均衡的目标，避免了单点故障的问题。

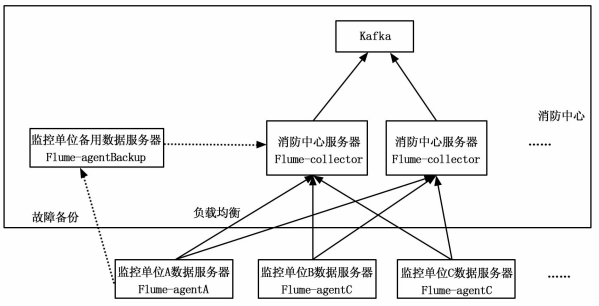

图 3　flume 系统架构图

2.2.2　数据接入层

由于 flume 采集日志数据的速度和 storm 处理数据的速度是不同的，所以本系统使用 kafka 作为中间的一个缓冲模块。kafka 能够实时的收集反馈信息，并能够支撑较大的数据量，通过磁盘数据结构提供消息的持久化，这种结构对于即使数据达到 TB＋级别的消息，存储也能够保持长时间的稳定，且具备良好的容错能力。本系统使用 kafka 统一接收 flume 采集的数据，将消息提交给系统的下一个模块，storm 集群进行数据的处理。

2.2.3　数据处理层

storm 架构组成如图 4 所示。其中有 3 种重要节点：Nimbus 节点、Supervisor 节点和数据库节点。Nimbus 就相当于 Hadoop 中的“JobTracker”，是用户和 Storm 系统沟通的关键点，Nimbus 主要负责发布和协调拓扑中的任务执行。Nimbus 是无状态的，如果 Nimbus 服务失败了，工作节点仍然可以持续地进一步工作，只是无法进行任务调度，直到 Nimbus 重新复活。Supervisor 运行在各个 Storm 节点上，它接受 Nimbus 分配的任务并根据任务启动和停止属于自己管理的工作进程。Supervisor 也是快速失败的，Nimbus 和 Supervisor 所有的状态都存储在 Zookeeper 中，这样设计是 Storm 具有恢复能力的关键。对分析后的结果持久化，可以用 MySQL 存储，同时 MySQL 采用主从复制架构，来避免数据的丢失。

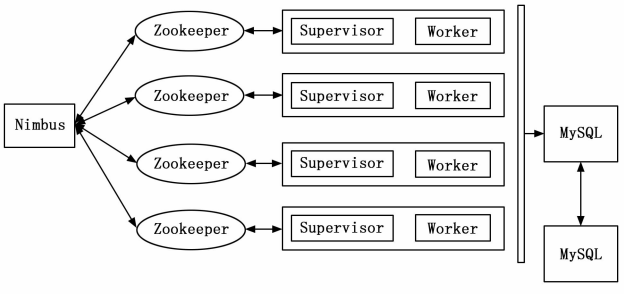

图 4　消防预警系统 Storm 架构组成示意图

Storm 数据处理架构由流元组组成的拓扑，有着明确的分层架构：Spout 和 Bolt，Spout 是拓扑中数据的来源，它会从外部数据源中读取数据，并发送给处理元组的 Bolt，并通过拓扑中的系统及组件 Acker 追踪从 Spout 中流出的元组的处理路径，如果在用户设置的最大超时时间内这些元组没有被完全处理，那么 Acker 就会告知 Spout 该消息处理失败了，相反会告知 Spout 消息处理成功了。这个 Spout 代表着整个元组的完全处理，如图 5 所示。因此可以说 Storm 记录容错原理保障了消息的可靠处理。

表 1 数据处理量和处理时间表

处理 log 数据	5W	10W	15W	20W	25W	30W	35W	40W	45W	50W
处理数据大小	4.01M	8.03M	13.02M	17.9M	21.3M	25.62M	28.74M	33.65M	36.33M	40.51M
处理时间	4.3S	7.9S	10.2S	12.8S	14.6S	16.4S	17.2S	19S	21S	22.9S

从一个提供数据源的 Spout 或 Bolt 流到处理元组的 Bolt 有很多种方式，由流分组机制所定义，主要有：随机分组、字段分组、直接分组、全局分组等。本系统根据实际需求采用先字段分组后随机分组策略。把数据流先按照消息类型字段（消息类型分为火警，故障，联动，反馈等）分组，发送到不同的 Bolt 中，该 Bolt 再把接收到的某类型的数据流按照随机分组方式分发元组到其它的 Bolt 任务中，对该数据进行计数等其他操作。最后再由整合 Bolt 计算统计结果。

消防单位中的各类探测设备和其他电子设备，遇到设定情况时会发送自己的警报数据，同时，每时每刻也在发送自己自检数据，对这些数据的处理实时性要求比较高，而数据处理组件 Bolt 主要按字段分组过滤掉海量数据中正常的状态数据，处理实时性要求比较高的报警数据，例如火警，故障，联动等信息，然后将报警数据发送到 web 客户端显示以及批量插入数据库中。插入完成后，追踪组件 Acker 通知 storm 框架任务已经执行完成。

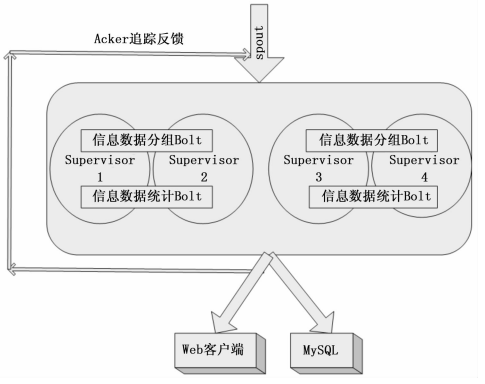

图 5 Bolt 组件处理过程

2.2.4 数据输出层

对于 storm 处理后的数据则需要显示在 web 客户端页面上以及将数据存储到数据库中，用于后期的对这些数据的统计查询。这里可以使用 mysql 分布式数据库系统来存储这些数据。这里不再详述。

3 客户端结果显示

使用 express＋socket.io 技术实现 web 客户端更新数据：Storm 处理完成数据处理后，通过 express＋socket.io 技术实现更新客户端信息界面，这里对于这个技术细节不再详细赘述。消防监控中心对页面上显示的问题进行处理。有效的监管各个消防中心的运行。

4 性能测试

本系统主要提升了城市消防联网远程监控系统在大数据处理过程中的实时性和可靠性，下面分别从这两个方面验证系统性能。

4.1 实时性性能评估

测试系统采用 4 台主机组建 storm 集群，CPU 4×3.3 G，4G 内存，千兆以太网卡，系统环境：Java1.6＋Kafka＋storm－0.9.1＋flume1.5＋MySQL5.1.69，集群中 1 个为 master 服务器，测试情况下采用 10 不同大小的 log 文件，分别包含为 5 万到 50 万行数据来模拟不同的数据量。系统每次处理一个文件。测试过程中实时记录集群中 master 服务器的 CPU 、I/O 和 内存占用情况以及其它 3 个节点服务器的硬件资源使用情况以及完成处理过程所用的时间。

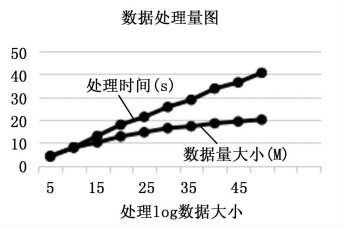

图 6 数据处理量

如表 1 和图 6 所示，从 5 W 到 50 W 行的数据，数据量线性增加，系统处理数据的所用时间呈缓慢增加。说明系统在处理大数据量时所用时间是可控的，实时性可以得到保证。同时我们看到无论是 master 服务器还是其它节点服务器的资源占用率都维持在正常水平。从这次实验我们看到数据处理时间还是不够理想，这跟实验中服务器的硬件水平和数量有很大关系。实际使用中，根据需要增加服务器的数量和提高硬件水平可以有效缩短处理时间，提高实时性。

4.2 可靠性性能评估

在这个实验中，我们预分配 8 台主机。在拓扑中每个组件的初始数目如下表 2。

表 2 组件的初始数目

组件	任务数
Spout	100
信息类型分组 Bolt	100
信息数目统计 Bolt	150
整合 Bolt	20

工作进程的数据设置为 30，并且一直保持在 30。我们在 8 台机器上同时运行拓扑任务。然后，我们等待了 15 分钟之后，关掉了其中的 1 台机器，之后每隔 15 分钟就关掉其中 1 台机器。实验的设置如表 3 所示。

表 3 实验实时配置展示表

时间(相对于实验开始)	机器数	工作进程数	工作进程数/机器数
0 分钟	8	30	4
15 分钟	7	30	4
30 分钟	6	30	5
45 分钟	5	30	6
60 分钟	4	30	7

最后，我们监视每分钟处理的元组的数目来表示整个架构的吞吐量，同时也监视着一个元组的平均处理延迟。吞吐量时

根据每分钟被追踪的元组的数据来衡量的。实验的结果如表 4 所示。

表 4　实验吞吐量和延迟响应数据表

时间段	机器数	每分钟平均吞吐量(百万)	每分钟平均延迟(毫秒)
0～15	8	6.8	7.8
15～30	7	5.8	12
30～45	6	5.2	17
45～60	5	4.5	25
60～75	4	2.2	45

正如图 7 所示，每当我们移除一组机器时，都会有一个临时的下降尖峰，但是随后很快地又恢复回来。同时，我们主要到，吞吐量每十五分钟下降一次，这意味着同样的拓扑被更少的机器处理。也正如图所示，吞吐量也很快地稳定回来。

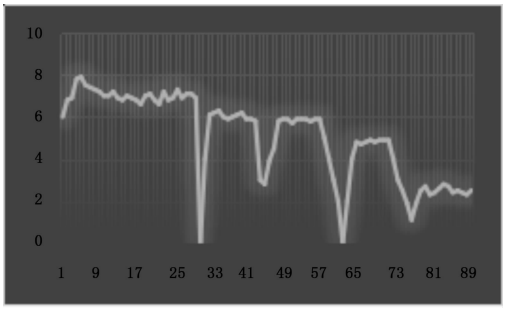

图 7　吞吐量测量图

如图 8 所示实验的延迟图，每关掉一组机器后，平均的响应延迟也增加了。但是我们同时也注意到了，在前几个 15 分钟内，延迟的时间是很短的，但是在最后两个 15 分钟内，延迟就相对来说有点大，但是，系统也都能够很快地稳定处理任务。

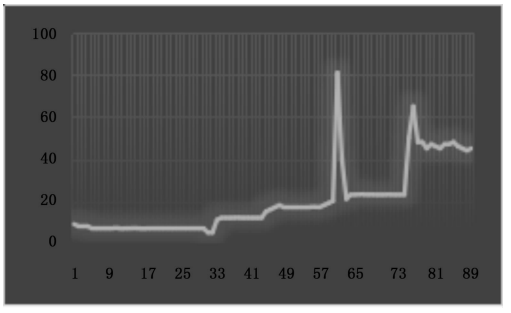

图 8　延迟测量图

总得来说，正如在这个实验中所示，Storm 对于机器故障具有很好的恢复能力。并且当面对机器故障时，也能够有效地稳定系统的性能。

5　结言

论文主要是研究消防监控中心对各个消防单位的数据进行实时采集和处理，对各个消防单位进行实时监控、警情及时发现、故障及时反馈。通过分析消防单位和消防主管部门的相应的需求和实际状况，我们发现利用基于云计算平台的 storm 集群系统能够有效地解决消防数据处理延迟以及系统可靠性问题。同时针对消防单位服务器或者是网络出现故障时能够及时发现，我们采用了心跳检测机制来保障监控中心实时地了解消防消防单位的状况，也保证了消防数据的准确无误。同时这套系统也有很高的经济性，消防单位的系统只需做少量修改，同时消防监控中心根据实际需求灵活配置服务器数量，

storm 架构在解决大量数据流的实时处理方面具有很好的性能，也是以后未来工作的研究重点，在之后的工作中，我们会更多地改进可视化的方法，提升相关部分的可靠性，也会更多地考虑实时处理和批处理的相结合等问题。

参考文献：

[1] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma, Jennifer Widom: STREAM: The Stanford Stream Data Manager [J]. IEEE Data Eng. Bull. 2003, 26 (1): 19－26.

[2] Minos N. Garofalakis, Johannes Gehrke: Querying and Mining Data Streams: You Only Get One Look [J]. VLDB 2002.

[3] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, Mitch Cherniack, Jeong－Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, Stanley B. Zdonik: The Design of the Borealis Stream Processing Engine [J]. CIDR 2005: 277－289.

[4] Apache Kafka, A high－throughput distributed messaging system. 2013 [EB/OL]. http://kafka.apache.org/design.html.

[5] Storm. 2013 [EB/OL]. http://storm.project.net/.

[6] S4 Distributed stream computing platform [EB/OL]. http://incubator.apache.org/s4/.

[7] Borthakur D, Sarma JS, Gray J, Muthukkaruppan K, Spigeglberg N, Kuang HR, Ranganathan K, Molkov D, Mennon A, Rash S, Schmidt R, Aiyer A Apache Hadoop goes realtime at Facebook [A]. Proc. Of the ACM SIGMOD Int" 1 Conf. on management of Data (SIGMOD 2011 and PODS 2011) [C]. Athens: ACMPress, 2011. 1071－1080. [doi: 10.1145/1989323.1989438].

[8] Tyler Akidau, Alex Balikov, Kaya Bekiroglu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, Sam Whittle: MillWheel: Fault－Tolerant Stream Processing at Internet Scale [J]. PVLDB, 2013, 6 (11): 1033－1044.

[9] Spark Streaming [EB/OL]. http://spark.incubator.apache.org/docs/latest/streaming－programming－guide.html.

[10] Rajagopal Ananthanarayanan, Venkatesh Basker, Sumit Das, Ashish Gupta, Haifeng Jiang, Tianhao Qiu, Alexey Reznichenko, Deomid Ryabkov, Manpreet Singh, Shivakumar Venkataraman: Photon: fault－tolerant and scalable joining of continuous data streams [A]. SIGMOD Conference 2013 [C]. 577－588.

[11] Mohamed H. Ali, Badrish Chandramouli, Jonathan Goldstein, Roman Schindlauer: The extensibility framework in Microsoft StreamInsight [J]. ICDE 2011: 1242－1253.

[12] Sankar Subramanian, Srikanth Bellamkonda, Hua－Gang Li, Vince Liang, Lei Sheng, Wayne Smith, James Terry, TsaeFeng Yu, Andrew Witkowski: Continuous Queries in Oracle [J]. VLDB 2007: 1173－1184.

[13] IBM Infosphere Streams [EB/OL]. http://www03.ibm.com/software/products/en/infosphere－streams/.