基于 IaaS 云平台的无线传感网实时监测系统①

王 申, 关胜晓, 吴 勇, 曾逸琪

(中国科学技术大学 信息与科学技术学院, 合肥 230027)

摘 要: 本文基于中科大 IaaS 云平台设计并实现了一种无线传感网实时监测系统. 利用 6LoWPAN 建立可自组网的无线传感器网络, 系统中各个传感器节点将传感器采集的数据发送至汇聚节点, 汇聚节点通过网络将数据传送至中科大云平台服务器中进行记录和分析. 本文采用 Nginx 作为 Web 服务器, uWSGI 应用服务器和 Tornado 应用服务器分别处理用户的非实时数据和实时数据请求. 采用 WebSocket 技术实现实时数据的传输, 提高了系统实时性. 测试结果表明本文对构建可靠且可灵活部署的实时监测系统具有指导意义.

关键词: 云平台; 无线传感网; 监测系统; Websocket

WSN's Real-time Monitoring System Based on IaaS Cloud Platform

WANG Shen, GUAN Sheng-Xiao, WU Yong, ZENG Yi-Qi

(Information and Science and Technology, University of Science and Technology of China, Hefei 230027, China)

Abstract: Based on USTC'S IaaS cloud platform, this paper designs and implements WSN's real-time monitoring system. The 6LoWPAN is used to build wireless self-organized sensor network, and each sensor node in system transmits data collected by sensors to sink node, then the sink node sends data to the server on USTC's cloud platform through network to record and analyze. In this paper, we adopt Nginx as a web server, and the non-real-time and real-time data is handled by uWSGI application server and Tornado application server respectively. The WebSocket technology is used to achieve real-time data transmission which improves the performance of system in real-time. The test results show that this paper has a certain guiding significance for the construction of real-time monitoring system with good reliability and flexible deployment.

Key words: cloud platform; wireless sensor network(wsn); monitoring system; WebSocket

随着无线传感网的发展, 实时监测系统被应用在各种领域, 并且对系统的可靠性和实时性要求越来越高. 现在的监测系统多是融合 GPRS 和无线传感网[1-4], 将无线传感网采集的数据通过 GPRS 传输到数据监控中心, 数据监控中心对数据进行处理, 之后通过短信等形式反馈给用户. 类似的方案虽有效, 但因参考文献中大部分的数据中心, 由普通的计算机充当服务器, 运行的稳定性得不到保证. 因采用 GPRS 业务和 SMS 业务, 延时无法预估, 实时性也有待商榷. 文献[5]中采用云计算解决方案, 着重构建环境监测云平台原型系统和兼容各类传感器数据、文件数据, 性能稳定可靠, 但是同样没有对系统的实时性展开讨论.

从目前的文献来看[2,3,6-8], 大部分对子系统进行管理的方案是在现场搭建服务器, 用户只需要直接输入 IP 地址或者域名, 直接就可以获取现场的数据并且实现对现场的控制. 一般的直接采用的是嵌入式系统[9]或 PC 机作为服务器. 由于资源有限, 处理业务的能力受到影响, 而且如果要部署类似的多个子系统, 则需要管理多个 IP 地址或者域名, 且每个机器运行相同的业务逻辑, 造成资源的浪费.

本文设计的实时监测系统基于 IaaS 云平台, 在云平台上实现主要的业务逻辑, 辅助的业务逻辑分配给现场各个子系统, 降低系统复杂度的同时, 可对现场业务逻辑的灵活配置, 同时降低对现场服务器性能的

① 收稿时间:2015-09-29;收到修改稿时间:2015-11-19 [doi:10.15888/j.cnki.csa.005184]

要求. 在云平台管理分散在各地子系统, 实现集中化管理. 感知层的数据采集后, 通过汇聚节点将数据交给本地服务器, 连接着云平台的本地服务器将数据直接同步到云端数据库. 本系统采用 B/S 架构, 无需对客户端软件进行维护, 可跨平台在多终端使用, 用户直接使用浏览器即可获取系统的数据. 本文采用 Nginx 作为 Web 服务器, uWSGI 应用服务器和 Tornado 应用服务器分别处理用户的非实时数据和实时数据请求, 采用最新的 HTML5 中的 WebSocket 协议要保证数据的实时性[10]. 一旦云端数据库中的数据更新, Tornado 应用服务器会将最新的实时数据通过 WebSocket 协议推送给浏览器, 而非实时的业务逻辑保持不变, 提高通信效率.

1 系统概述

本文采用常用的物联网的三层结构[11]: 感知层、网络层、应用层. 系统总体框图如图 1 所示, 感知层的主要功能是完成数据采集; 而网络层主要就是借助现有的互联网资源实现现场服务器和云端服务连接, 实现用户访问 Web 应用, 传统的实时通讯所采用的技术大多是轮询机制, 本文采用 Html5 中的 WebSocket 协议来实现浏览器与服务器之间的通信, 保证数据的实时性的同时提高通信效率; 应用层的主要功能是根据底层采集的数据, 形成与业务相关基本数据存储到数据库中, 并完成对数据的处理和管理.

1.1 系统网络拓扑结构

系统整体的网络拓扑结构如图 2 所示, 共分为两大部分, 现场数据采集部分和用户远程访问部分.

现场数据采集部分由一个自组织、多跳的无线传感网络完成, 无线传感网络中包括中心汇聚节点(Sink 节点)和数据采集节点(Sensor 节点), Sensor 节点将采集的数据汇聚到 Sink 节点, Sink 节点使用 RS232 协议和现场服务器进行通信, 现场服务器将采集的数据存放到本地数据库的同时将记录同步到云端数据库, 以备其他业务逻辑使用. 图 2 中的 WSN_1 表示的是第一个无线传感网, WSN_N 表示的是第 N 个无线传感网, 本系统可将分布在不同地域的无线传感网不间断的采集实时的数据同步到云端数据库.

用户远程访问部分主要是通过利用现有的网络资源实现. 用户可以通过主流的浏览器在多终端实现访问, 因采用了响应式 Web 设计方法[12], 页面会根据用户行为以及设备环境进行调整和响应, 比如自动切换分辨率、更改要显示的图片尺寸以及启停相关脚本等, 以适应不同设备.

知识表示模块: 以 B/S 架构提供知识表示服务, 根据用户的输入从知识数据库中智能化搜索并生成用户需要的解决方案.

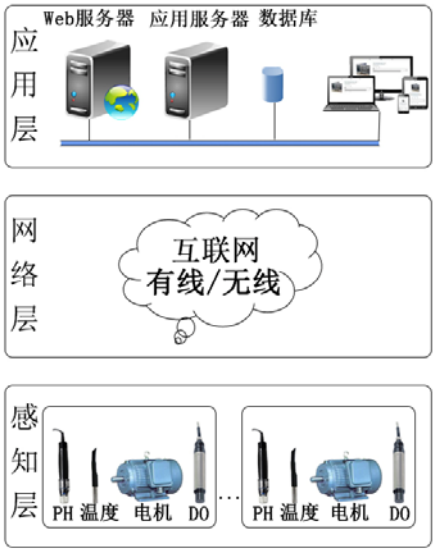

图 1　系统总体框图

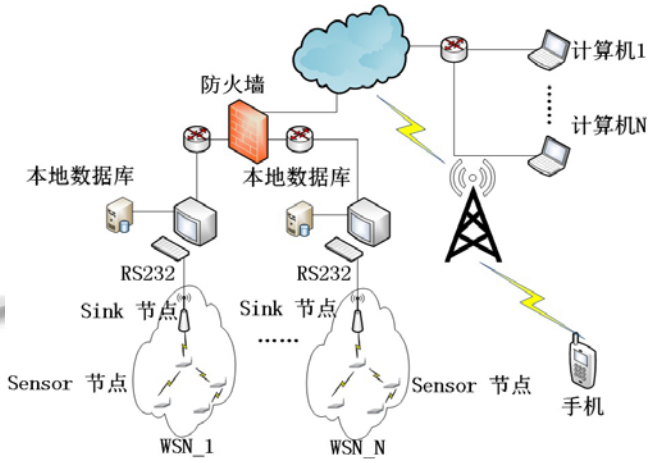

图 2　系统整体网络拓扑图

1.2 系统后台框架

系统的后台框架如下图 3 所示, 平台架设在科大云端, 平台上运行着 Web 服务器、应用服务器、Web 应用和数据库等等. 本文采用的操作系统是 Ubuntu 12.04 server, Web 服务器选择的是 Nginx, uWSGI 作为应用服务器处理非实时数据请求, Tornado 作为应用服务器处理实时数据请求. Web 应用采用 Django 框架[13]实现.

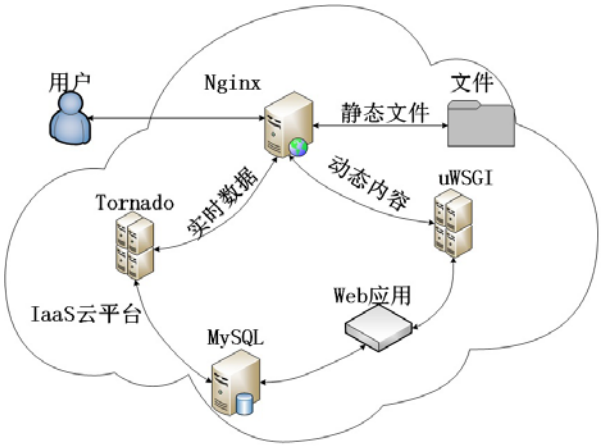

图 3 后台框架

整个通信流程如下, 首先 Web 服务器 Nginx 监听 80 端口, 客户端发出 HTTP 请求. 若 Nginx 发现客户端请求的内容是 css 文件、js 文件, 图片等文件, 则直接将请求的文件返回给用户; 若 Nginx 发现客户端请求的内容是是动态内容, 则将请求转发给 uWSGI 应用服务器, uWSGI 和 Web 应用进行交互, 从而将请求交给具体的 Web 应用进行处理, 处理后的结果按照原请求路径进行层级返回, 最终将数据返回给客户端.

2 无线传感网设计

本部分设计的主要目标是组建自组织、多跳的无线传感网络. 本文采用的硬件节点是 TI 公司的 CC2530 射频芯片, CC2530 是用于 2.4GHz 的片上系统解决方案, 有不同的运行模式, 满足本系统的低功耗的要求[14]. 这里使用 Contiki 作为 WSN 的操作系统平台.

2.1 无线传感网的网络拓扑

无线传感器网络节点往往是资源受限节点, 需要考虑到节点的特性, 例如微处理器计算能力相对较弱、RAM 和 ROM 大小受限制等, 同时要考虑到链路的属性, 比如链路时延、节点移动或死亡等网络条件. 节点需要动态地、自发地计算出适应性的路由. RPL 协议的设计目标就是基于数据采集的网络, 因此本文采用专为低功耗有损网络设计的 RPL 路由协议来解决网络节点移动问题. 如下图 4 所示, 其中黑色的节点是汇聚节点, 而白色的节点是普通节点.

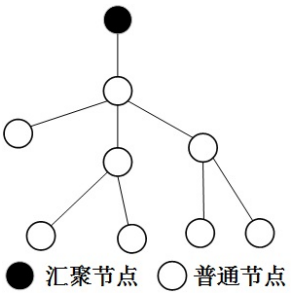

图 4 网络拓扑示意图

2.2 无线传感网节点设计

无线传感网络中包括两种类型的节点: 中心汇聚节点(Sink 节点)和数据采集节点(Sensor 节点). 这两种类型的节点, 使用相同的核心板, 不同的是数据采集底板, Sensor 节点上带有传感器接口, 而 Sink 节点无传感器接口, 只有与本地服务器通信的接口.

Sink 节点主要完成对数据采集节点发送来的数据进行汇聚, 一个子系统内部只有一个 Sink 节点. 对无线传感网络而言, Sink 节点接收 WSN 中所有 Sensor 节点发送过来的数据并且将其发送互联网, 反之亦然. 对互联网而言, Sink 节点起着网关的作用, 每当它有一个数据包要发送的时候, 它将核对这个包的地址, 若该数据包属于 WSN 的子网范围内, 就将该包通过无线发送. 否则, 它将此通过串行连接的方式发送到现场服务器. 本系统中为避免发生溢出或覆盖问题, 需开辟大小合适的数据缓存, 缓存的大小将直接影响整个系统运行的稳定性.

Sensor 节点主要完成数据采集和数据转发等任务. 系统运行时, 首先要对系统的硬件进行初始化, 之后初始化操作系统, 最后进入运行状态周而复始的进行数据采集和转发.

3 实时通信流程

实时通信阶段工分为两个阶段, 第一阶段即处理 HTTP 请求阶段, 此时主要处理的是非实时数据, 具体的流程可参考 1.2 节, 此时用户已获取一个 Web 页面和静态文件, 但是没有具体的实时数据. 如果用户访问的页面如果涉及到具体的实时数据, 则进入第二阶段. 本文将发起 WebSocket 请求的单独用 JavaScript 脚本实现, 若客户查看实时数据, 则浏览器会执行 JavaScript 脚本发起 WebSocket 请求, Nginx 发现是实时数据请求, 则将请求转发给 Tornado 应用服务器,

Tornado 应用服务器响应请求. 进行握手连接成功之后, Tornado 应用服务器检查是否有数据更新, Tornado 应用服务器从数据库中读取实时数据推送给客户端. 浏览器获取动态页面和实时数据的方式如图 5 所示.

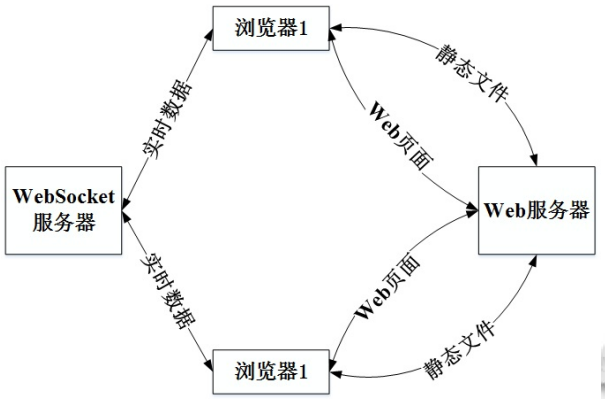

图 5　通讯流程

4　测试与分析

本文对平台进行简单的功能性、客户端兼容性、并发测试与实时性能测试, 参与测试的两台机器处于校园网中, 本测试环境中采用的客户机通过百兆的路由器连接到校园网中, 而服务器在网络中心. 整个校园的带宽对测试来说是一个很重要的影响因素, 因此本文中的测试大多是在晚上和夜间测试完成, 以保证测试中具有相对相同的、稳定的网络环境, 尽可能排除其他网络因素的影响. 测试机器的配置如表 1 所示.

表 1　测试机器配置表

	机器配置		
服务器(一台)	Ubuntu12.04 server	四核@2.67Ghz	**8G 四核**
客户机(一台)	Ubuntu12.04 desktop	双核四线程 @2.67Ghz	**2G 四核**

4.1 功能测试与客户端兼容性测试

系统可以实现预期功能, 如图 6、图 7 所示, 系统用户界面良好, 通过图表的形式可视化数据. 平台采用响应式 Web 设计, 页面呈现的内容会根据不同的终端进行相应的调整.

4.2 并发测试

本次测试的是系统的并发能力. 设每次请求总量为固定值 5000, 并发用户数设置为 100~1000 变化, 步长设置为 50, 测试结果如图 8 所示. 当系统并发为 750 时, 开始出现失败请求, 每秒钟处理的请求数(吞吐率)稳定在 300 附近, 当系统并发为 800, 请求总量 5000, 失败请求总数 4550, 失败请求比例高达 91%, 说明服务器此时已经达到瓶颈. 此时测试过程中, 使用 nmon 记录到的系统的 CPU 负载较高. 测试前后内存变化不明显, 原因是为防止系统内存不足的情况发生, 在配置文件中对物理内存的最大使用和应用服务器子进程的个数做强制性限制.

图 6　系统页面

图 7　手机平台登录效果

从以上测试可以看出, 本系统可以满足一定的并发需求, 因为本系统主要针对的实时监测系统, 系统的稳定运行时先决条件, 所以用户群较小的情况下, 牺牲系统的稳定性去获取高并发性是不可取的. 因此本文中将并发用户数设置为固定值 300, 总的请求数设置为 100~5000 变化, 步长设置为 100, 让测试用例在一天里不同的时间段运行相同的时间, 然后获取 3 天的平均值. 如下图 9 所示, 系统的每秒钟处理的请

求数稳定在 300 左右, 性能稳定, 失败请求数为 0, 系统的负载也在正常的范围之内.

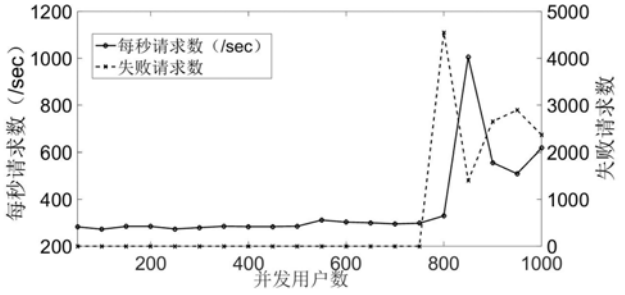

图 8 并发测试

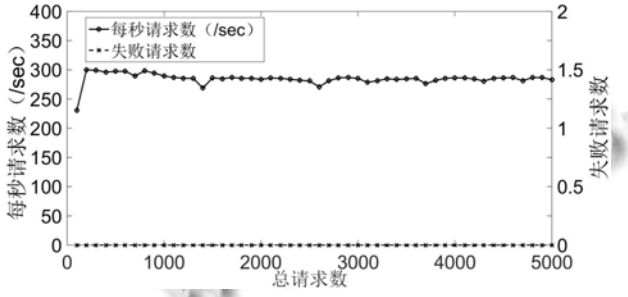

图 9 稳定性测试

4.3 实时性能测试

延迟是从服务器获取到信息到客户端接收到此信息需要一定的时间开销造成的. 在实时应用中, 延迟越小, 实时性能越优越, 本文采用平均延迟时间作为评价结果. 为了获取每条信息的延时时间, 本文中采用 NTP 协议对现场服务器、远程服务器和测试机进行时间同步, 使用 ntpupdate 指令将时间偏差 offset 的值调整到小于 0.1ms 可达到要求, 现场将采集的数据和时间戳一并写入本地数据库后, 同步到远程服务器数据库中. 服务器在将该记录发送给客户端后, 通过获取新的时间戳与记录中的时间戳相减即可得到本次发送的延时时间.

本文将对比以下两种方案: 方案一、单独使用 uWSGI 服务器处理所有类型请求; 方案二、采用 uWSGI 服务器与 Tornado 应用服务器分别处理非实时和实时请求. 其中方案一中分别采用 Ajax 轮询方式和 Ajax 长连接方式进行测试. 而方案二中, uWSGI 服务器仍按照正确方式处理非实时数据, Tornado 服务器采用 WebSocket 方式处理实时数据.

为了使数据更准确, 方案一每种方式各收集 1000 次延迟时间, 方案二的 WebSocket 方式收集 7000 次延迟时间, 实验结果如表 2 所示.

表 2 实时性能测试表

	方式	时间间隔(s)	总延时时间(ms)	次数	平均延时时间(ms)
方案一	Ajax 轮询	2	1173611	1000	1173.61
		1	610380	1000	610.38
		0.5	457343	1000	457.34
	Ajax 长连接		151536	1000	151.36
方案二	Websocket		369181	7000	52.47

根据上述结果可以看出, 方案一中, Ajax 轮询时间间隔分别为 2 秒、1 秒、0.5 秒时, 其平均延长时间为 1.173 秒、0.610 秒、0.457 秒. 实时性能与轮询时间间隔有一定关系, 若将轮询时间进一步减小, 会产生许多无效请求. Ajax 长连接方式在实时性能上优于 Ajax 轮询方式, 但是在测试过程中 Ajax 长连接的方式对 uWSGI 应用服务器造成的压力较 Ajax 轮询方式大. 而采用方案二的 WebSocket 方式, 实时性能明显优于方案一的各种方式, 而且采用此种方式分担了 uWSGI 服务器的压力. 所以采用方案二的本质是增加系统的复杂性和牺牲一定的内存来提高实时性能.

5 结语

本文结合无线传感数据采集和监测系统实际应用需求, 设计并实现了一种基于云平台的实时监控系统. 将实时数据和非实时数据的分离, 减小耦合, 使系统易于扩展. 将 WebSocket 技术应用在实时数据的传输, 提高实时性和通信效率. 系统具有运行稳定,界面友好, 易于维护,方便扩展等优点. 最后对监测系统整体性能进行测试和评估, 结果表明系统稳定、可靠, 实现预期功能.

参考文献

1 李慧娟.基于 GPRS 和 GPS 的机车信号远程监测系统研究[硕士学位论文].兰州:兰州交通大学,2014.
2 时晓艳.基于 ZigBee 和 GPRS 技术的列车远程监测系统设计[硕士学位论文].青岛:中国海洋大学,2012.
3 费永云.基于 GPRS 的农田环境远程监测系统设计[硕士学位论文].南京:南京农业大学,2013.
4 左俊平.基于物联网技术的电池组远程监测系统设计[硕士学位论文].哈尔滨:哈尔滨理工大学,2014.
5 刘宏宇.基于无线传感器网络的森林环境监测云平台研究与实现[硕士学位论文].北京:中国林业科学研究院, 2012.

6 赵小欢.基于 WSN 的水产养殖水质在线监测系统设计[硕士学位论文].哈尔滨:哈尔滨理工大学,2014.
7 吴美文.基于 Zigbee 技术的石油生产远程监测系统设计[硕士学位论文].曲阜:曲阜师范大学,2010.
8 刘钧鹏.基于物联网的电梯监控系统设计与实现[硕士学位论文].武汉:武汉理工大学,2014.
9 方朝阳.基于物联网的家庭火灾监控系统设计[硕士学位论文].重庆:重庆大学,2013.
10 Furukawa Y. Web-based control application using WebSocket. Web-based Control Application Using Websocket, 2011.
11 Gubbi J, Buyya R, Marusic S, et al. Internet of things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 2013, 29(7): 1645−1660.
12 Bryant J, Jones M. Responsive Web Design. Springer, 2012: 37−49.
13 Sanderson D. Programming google app engine: Build and run scalable web apps on google's infrastructure. O'Reilly Media, Inc. 2009.
14 Li ZF, Zhong HS. Distributed data acquisition system based on CC2530. Microcontrollers & Embedded Systems, 2011, 9: 40−42.