5-3-2013

Network Technologies Used to Aggregate Environmental Data

Paul Stasiuk
Loyola University Chicago

Konstantin Läufer
Loyola University Chicago, klaeufer@gmail.com

George K. Thiruvathukal
Loyola University Chicago, gkt@cs.luc.edu

Recommended Citation

P. Stasiuk, K. Läufer, and G. K. Thiruvathukal. Network Technologies used to Aggregate Environmental Data: Research Poster. 2nd Greater Chicago Area System Research Workshop (GCASR), May 3, 2013, Evanston, IL, USA.

Network Technologies used to Aggregate Environmental Data

Dr. Konstantin Läufer, Dr. George K. Thiruvathukal, Paul Stasiuk

Department of Computer Science, Loyola University Chicago

Preparing people to lead extraordinary lives

Introduction

NOSWCEM or Loyola Weather Service(lws) project is a continuation of previous research conducted by Dr. George K. Thiruvathukal and Dr. Konstantin Läufer.

The goal of lws is to design and build a system of functioning environmental monitoring widgets that can intelligently and autonomously control the environment around them based on set thresholds and triggers. Ideally, the widgets will also have the ability to aggregate their data and easily display this data in various ways; through a user interface in the room that the widget is placed or via a web interface.

Methods

Agile software development

Building software based on iterative and incremental approaches with a focus on developing a working model as quickly as possible.

Communication Protocol Investigation

Investigating and benchmarking protocols that can be used to transport sensor data over networks.

Data Modeling

Developing a data model and queries that scale to growing databases.

Scalable Architecture

Developing and deploying an architecture that grows as sensors are added and is extendable through other interfaces.

Application Program Interface(API)

Developing a web based protocol that allows for others to build services around the data that is being collected.

Hardware

Raspberry Pi:

The Raspberry Pi is a micro-computer that fetches sensor data and pushes the data to the aggregation node(see architecture).

Phidget I/O Kit:

In place of developing new sensor interfaces directly on the Pi, we opted to use a simple I/O board that allows for sensors to be accessed programmatically.

Software

Python

While not the fastest language, prototyping is fast ,simple and flexible.

Flask

A Python Micro framework that allows for RESTful development of python web applications.

ZeroMQ

A message queuing protocol implementation.

MongoDB

A documents-based database that is being used to store persistent data.

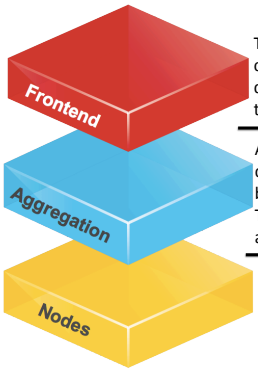
Acknowledgements

Special thanks go to my mentors Dr. Thiruvathukal and Dr. Läufer. Additionally, I would like to thank the Computer Science department for supporting my research in many ways. Finally, I would like to thank the communities at www.phidgets.com and www.raspberrypi.org for leaving a massive knowledge-base for me.

References

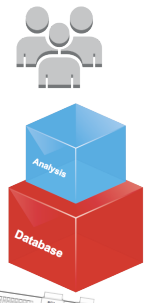
A detailed list of references can be found at: http://goo.gl/al18V

Architecture



The frontend presents data to the user as well as responds to API calls from developers. Currently, the Frontend layer is deployed directly on top of the aggregation layer, but can be separated out to scale as the application needs.

All data from the nodes is collected within the aggregation layer. The database(MongoDB) lives within this layer. The aggregation layer has been deployed on a scalable Software as a Service(SaaS) application The aggregation layer responds to requests from the frontend layer and performs analysis on the data that is being collected by the nodes.



Nodes are combinations of a host device that runs our application and a sensor kit. The nodes can perform basic functions such as collect data and respond to requests from the aggregation node for near real time data. The nodes are intended to be the primary interface between the users and the environment.

Application Program Interface(API)

Representative State Transfer(REST) interface for developing application with the data collected from the sensors using a few basic functions that allow for devices to be access over the web using different URI's.

/devices/device/settings/change – PUT

Leverages the HTTP PUT method to make changes to a devices settings. In order to make changes, a JSON object must be sent to the Frontend node for processing. This includes setting device thresholds such as toggling warnings for high temperature, and naming devices or making changes to the configuration of a device.

/devices/device/settings/json - GET

Retrieves all of the devices settings and thresholds in as a JSON data structure based on a device identification value. Requires that the device identification number is passed as an argument.

/devices/all/json – GET

Using the HTTP GET method, this returns a JSON data structure that includes a list of all of the devices currently registered with the service.

/devices/device/data/json – GET

Returns a JSON structure containing data within a date range for a specific device. Requires the passing of the

```
value_data= {
  phid: 1xC3123 ,
  ISO date: 07-03-2013,
  h:11,
  m:30,
  s:22,
  sensor_data:{
    0:0,1:0,
    2:0,3:0,
    4:0,5:0,
    6:0,7:0}
}
```

Benchmarking

Preliminary Benchmarking was done for both the communication between the nodes and aggregation layer and the database calls for the RESTful API.

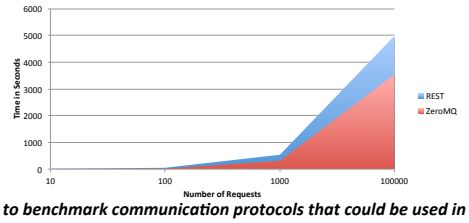
Communications

100 Requests with 100 Workers:
ZeroMQ:
Elapsed Time: 2.0682 seconds
REST:
Elapsed Time: 8.3849 seconds

1,000 Requests with 100 Workers:
ZeroMQ:
Elapsed Time: 19.6718 seconds
REST:
Elapsed Time: 38.0954 seconds

10,000 Requests with 100 Workers
ZeroMQ:
Elapsed Time: 316.2272 seconds
REST:
Elapsed Time: 526.9436 seconds

100,000 Requests with 100 Workers
ZeroMQ:
Elapsed Time: 3540.9639 seconds
REST:
Elapsed Time: 4987.3988 seconds



Discussion

The prototyping of the lws system gave us an opportunity to benchmark communication protocols that could be used in the transmitting of environmental data. Additionally, we were able to benchmark the use of a document based database's range queries based on compound indices.

Our system can not only scale, but also provide interaction between users and the environment on a more intelligent and extensible level than other environmental monitoring solutions. The open-source nature of the project allows for others to actively contribute to developing the system, further molding it to their needs. The currently deployed system includes a framework for extending functionality beyond simply environmental data collection. For example, controlling the environment based on thresholds set by the user; lights, temperature and humidity.

Our conclusions are twofold: (1) we have evidence that priority message queuing see's at least double the performance over HTTP based protocols for large amounts of queries from the nodes and (2) database range queries on compound time indices versus ISO times see only marginal performance increases; leaving us inconclusive the us of compound indices.