A Cloud-based Interactive Data Infrastructure for Sensor Networks

Tonglin Li[1], Kate Keahey[2], Rajesh Sankaran[2], Pete Beckman[2], Ioan Raicu[1,2]
[1]Illinois Institute of Technology, Computer Science Department
[2]Argonne National Laboratory, Mathematics and Computer Science Division
tli13@hawk.iit.edu, keahey@mcs.anl.gov, rajesh@anl.gov, beckman@anl.gov, iraicu@cs.iit.edu

ABSTRACT

Small specialized sensor devices capable of both reporting on environmental factors and interacting with the environment are becoming increasingly ubiquitous, reliable and inexpensive. This transformation has enabled domain sciences to create "instruments at large" – dynamic and often self-organizing groups of sensors whose outputs are capable of being aggregated and correlated to support experiments organized around specific questions. This calls for an infrastructure that can support remote administration of sensors, relies on protocols that can withstand unreliable communications, and extend storage capability that can scale to support many data producing sensors, many different data types, and many end user requests. In this work we present protocols and a cloud-based data store called "WaggleDB" that address the above challenges. The system efficiently aggregates and stores data from sensor networks and enables users to query the data sets. The "WaggleDB" data store incorporates a scalable multi-tier architecture with individually scalable layers toward overcoming the challenges.

Keywords

Sensor Network, Cloud, Data Infrastructure, Column-Oriented Database

1. INTRODUCTION

The last several years have seen a raise in the use of sensors, actuators and their networks for sensing, monitoring and interacting with the environment [1]. There is a proliferation of small, cheap and robust sensors for measuring various physical, chemical and biological characteristics of the environment that open up novel and reliable methods for monitoring qualities ranging from the geophysical variables, soil conditions, air and water quality monitoring to growth and decay of vegetation. Structured deployments, such as the global network of flux towers, are being augmented by innovative use of personal mobile devices (e.g., such as use of cell phones to detect earthquakes), use of data from social networks, and even citizen science.

As small, specialized sensor devices, capable of both reporting on environmental factors and interacting with the environment, become more ubiquitous, reliable, and cheap, increasingly more domain sciences are creating "instruments at large" – dynamic, often self-organizing, groups of sensors whose outputs are capable of being aggregated and correlated to support experiments organized around specific questions. In other words, rather than construct a single instrument comprised of millions of sensors, a "virtual instrument" might comprise dynamic, potentially ad hoc groups of sensors capable of operating independently but also capable of being combined to answer targeted questions. Projects organized around this approach represent important areas ranging from ocean sciences, ecology, and urban construction to hydrology.

2. Design and Implementation

2.1 Challenges and solutions

The online analysis needs of such "instruments at large" create the need for data store management system with the following properties:

Write scalability: The data store will need to be able to sustain many concurrent writes generated by thousands of sensor controller nodes that continuously send data to the database with the same quality of service, i.e., reliability (not losing any writes), time to acknowledge, etc. For achieving these goals, we propose to use a multi-layer architecture. A high performance load balancer is used as the first layer to accept and forward all write requests from sensor controller nodes evenly to a distributed message queue. In our system the message queue works as a write buffer and handles requests asynchronously. A separate distributed data agent service keeps pulling messages from the queue, preprocess it and then write to the data store. On the client side (sensor controller nodes), we adopt a collective sending method, which accumulates the sensor data and send a batch of message to the cloud periodically. Similar batch sending can be found in [6]. The client sending frequency is customizable so to meet the different needs of data freshness.

Support for various data types: Since sensor data can be of various types and sizes; there is no fixed scheme for the data formats from all the different types of sensor. Therefore we need a flexible data schema so to enable a unified API to collect and store the data, and to organize data in a scalable way for further use (query/analytics). To address this issue, we design a flexible and self-describing message data structure that easily fits into a large category of scalable distributed databases, called column-oriented databases (or BigTable-like data stores [2]), a type of NoSQL database [3,4,5]. This design enables us to elevate the rich features, performance advantage and scalability from column-oriented databases, as well as to define a unified data access API.

Transactional interaction between admin users and sensors: Administrators need to push commands or queries to sensor controllers for development or maintenance purposes. However sensor network connection is not reliable and the connection can be lost any time, which makes conventional remote login mechanism such as SSH fail to work. We use a database table to track the sessions between admins and sensor controllers. A session contains all communication events and their orders. Admin submit a series of commands and the nodes that will run these commands to the system database, sensor controllers check out the commands from the same database whenever they are online. The controllers' responses are also push to the database for admins.

Dynamic scalable services: The request rate can change in a wide range. The system needs to be able adjust its capability to satisfy multiple requests for processing with qualities of service. We distribute the functionality to independent tiers in the architecture, which is designed in such way that each tier can be scaled by

adding more independent resources provisioned on-demand in the cloud.

2.2 Architecture

We design a loosely coupled multi-layer architecture to boost the scalability while maintaining a good performance. As shown in fig 2, the system is composed of a sensor controller node and a data server that is both written to by the sensors and read from by the clients. On the server side, there are 5 layers of components, namely load balancer, message queue, data agent, database, and query execution engine. Each layer can be deployed on a dedicated or shared virtual cluster. If any layer becomes bottleneck, it can be scaled easily by simply adding more resource.

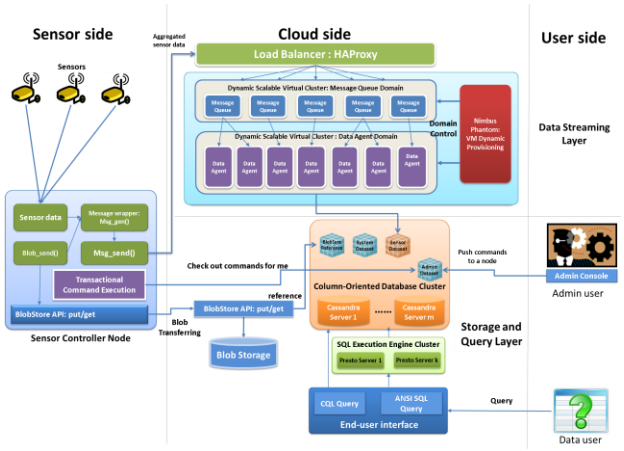

Figure 1. System architecture.

2.3 Implementation

We have implemented a functioning prototype of WaggleDB and deploy it on a public cloud platform, FutureGrid. We use RabbitMQ as the message queue, Cassandra as the database, HAProxy as the load balancer. All tiers are deployed on multiple VMs.

The prototype now has following features: accepting and storing data from different type of sensors, a blob store for potential big files from sensors such as full-spectrum cameras, transactional interaction between admin and sensor controllers, sensor and controller registration, CQL and SQL query on the database, dynamic scalability.

3. Performance Evaluation

We have conducted a preliminary performance evaluation of WaggleDB system on FutureGrid, which is an OpenStack based public cloud. Since we don't have many sensor controller nodes at this point, we run the clients on virtual machines and send random data in a tight loop to a WaggleDB queue server as a simulation.

In this experiment, we use the number of clients and message size as parameters. We measured request latency from client side, message queue processing bandwidth and requests throughput on the queue server side. Figure 2 shows that the latency slowly increases with the concurrency level but much better than. It is worth noting that the message size has little influence on latency. This indicates that the major parts of overhead consists connection creation and closing, but not network transferring. This is also proved by our bandwidth measurement, as shown in figure 3. Clients have proportionally high bandwidth when using bigger messages. With 32 clients sending messages of 10,000 bytes, the system reaches 50MB/s bandwidth, while it only has 59KB/s with 10 bytes messages. The request throughput benefits from adding from concurrent clients as well.

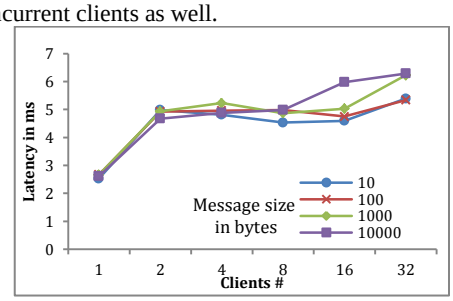

Figure 2. When the client scale increased by 32 times, the average latency only increased by 2.2 times. The differences between the latencies of 10 to 10k bytes message sizes were small.

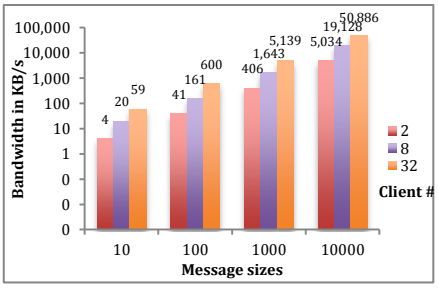

Figure 3. The bandwidth gain from bigger message size is close to that from adding more clients.

When adding concurrent clients up to 128, the latency increases rapidly for one server, while the systems that have more servers perform better. Single server is saturated at 32 clients scale, 2, 4 and server systems saturated at 64 and 128 clients respectively. 8 servers system performs still well. This implies excellent server scalability.

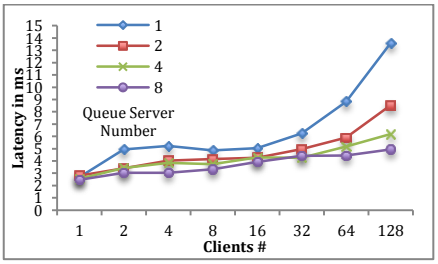

Figure 3. Server scalability. The more server in the system, the less latency increases.

4. Conclusion

The emerging sensor network usage calls for an infrastructure able to collect, store, query, and process data set from sensor networks. We present the challenges and gives tentative solution through WaggleDB, a cloud-based interactive data infrastructure for sensor networks. WaggleDB is elastic on both scales and data presentation, and shows excellent potential to scale each tier in the architecture.

5. REFERENCES

[1] Martinez, K., J.C. Hart, and R. Ong, Environmental Sensor Networks. Computer, 2004. 37(8): p. 50-56.

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat. 2006. Bigtable: a distributed storage system for structured data. OSDI '06.

[3] Jiri Schindler. 2013. Profiling and analyzing the I/O performance of NoSQL DBs. *SIGMETRICS Perform. Eval. Rev.*

[4] Tonglin Li, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke Wang, Anupam Rajendran, Zhao Zhang, Ioan Raicu. ZHT: A Light-weight Reliable Persistent Dynamic Scalable Zero-hop Distributed Hash Table, IPDPS2013.

[5] Tonglin Li, Raman Verma, Xi Duan, Hui Jin, Ioan Raicu. 2011. Exploring Distributed Hash Tables in High-End Computing, *SIGMETRICS Perform. Eval. Rev.*

[6] Tonglin Li, Ioan Raicu, Lavanya Ramakrishnan, Scalable State Management for Scientific Applications in the Cloud, BigData2014