

Fjording the Stream: An Architecture for Queries over Streaming Sensor Data

Samel Madden, Michael J. Frandklin
University of California, Berkeley

Presented by: Tai Hu

Terminology

- Fjords – Framework in Java for Operators on Remote Data Streams
- Operator – a set of standard database modules
- Tuple – A collection of data which understood by operator
- Window – A time interval

2

Outline

- Introduction – why do we need a new architecture
- Background – Brief introduction for sensor environment and requirement for sensor query processing
- Solution – How Fjord works
- Experiment (will be skipped)
- Conclusion

3

Motivation

- If industry visionaries are correct, our lives will soon be full of sensors
- Sensors will be connected by wireless networks, each monitoring and collecting data about the environment at large
- Need a new architecture for queries over Streaming data

4

Why Do We Need a New Architecture?

- Properties of sensor
 - Unreliable network connection
 - Typically sample periodically and push immediately
 - No record of historical information
 - Power constraint (Most of them operate on the battery power)
 - Produce very large amount of data in real time

5

Standard Database vs. Sensor based Data Source

- Difference between Standard Database Source and Sensor Based Data Source
 - Sensors typically deliver data in streams. They produce data continuously, often at well defined time intervals without having been explicitly asked for that data
 - Sensors are fundamentally different from the over-engineered data sources typical in a business DBMS.

6

Proposed Solution to Problem

- An enhanced query plan data structure called Fjords
 - Allow users to pose queries that combine streaming, push based sensor source with traditional pull-based sources
 - Non-blocking and windowed operators
- A power-sensitive Fjords operators called sensor proxies
 - A mediators between the query processing environment and physical sensors

7

Difference to Other Works

- Providing the underlying systems architecture for sensor data management
- Enable other applications to be built on the top of this framework

8

Sensor Environment

- Physical sensor
 - A sensor consists of a remote measurement device that provides data at regular interval
 - A sensor may have some limited processing ability or may simply output a raw streams of measurements
 - No parsing on the queries or keeping track of which clients need to receive samples from them

9

Sensor Environment - Continue

- Sensor's proxy
 - Run on a fixed, powered, and well connected server with abundant disk and memory resource
 - An interface between physical sensor and the rest of the query processor
 - One machine is the proxy for many sensors.
 - Responsible for packaging samples as tuples and routing those tuples to user queries as needed

10

Sensor Environment - Continue

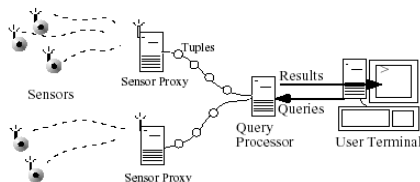


Figure 1. Environment For Sensor Query Processing

11

Sensor Environment – Continue

- Sensors do not participate in query processing
- Proxy can adjust their sample rate or ask them to perform simple aggregation before relaying data.

12

Next-Generation Traffic Sensor

- Tiny – Only 10 cm³ now. The ultimate goal is 1 mm³
- 8-bit microprocessors with small amounts of RAM running from 1 to 16MHz
- Transmitting at tens of kilobits per second with a range of few hundred feet.
- Capable of on board computation

13

Requirements for Sensor Query Processing

- Properties must be taken into account when designing the low level infrastructure
 - Limitations of Sensors
 - Streaming Data
 - Processing Multiple Queries

14

Limitations of Sensors

- Limited Resource, such as, battery capacity, communication bandwidth, and CPU cycles
- Data transmission is much more expensive than CPU cycles

15

Streaming Data

- Continuous, never ending streams of data
- Blocking operator cannot be used. (i.e. sorts, aggregates and join)
- Pull-based iterator model does not map well onto sensor streams
- Operators must process data only when sensors make it available

16

Solution

- Fjords: Generalized Query Plans for Streams
 - Operators and Queues
 - Flexible Data Flow
 - State Based Execution Model
 - Sensor Sensitive Operators
- Sensor Proxy
- Multiple Queries in a Single Fjord

17

Processing Multiple Queries

- In many sensor scenarios, multiple users pose similar queries over the same data streams
- A query processing system should be able to dynamically adjust sensor's sample interval and delivery rate

18

Fjords: Generalized Query Plans for Streams

- A generalization of traditional approaches to query plans
- Support for integrating streaming data that is pushed into system with disk-based data which is pulled by traditional operators

19

How a Fjords Works

- Each machine runs a single controller in its own thread
- Operators are instantiated by controller
- Operators are connected by queues

20

How A Fjords works - Continue

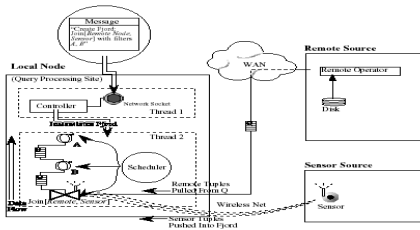


Figure 2. A Fjord: Join/Remote Node/Sensor/ with filters A and B

21

Operators and Queues

- Operators form the core computational unit of Fjords
- Each operator has a set of input queue and output queue
- Operator read tuple from input queue in any order and output any number of tuple to its output queue
- A dataflow architecture

22

Operators and Queues - Continue

- Responsible for routing data from one operator to another
- No transformation on the data
- Could connect either local operator or remote operator
- It is possible for a queue to fill

23

Flexible Data Flow

- Key advantage: Using a mixture of push and pull connections between operators
- Push and pull are implemented by the queue
- Push queue relies on its input operator to put data into it
- Pull queue actively requests that input operator produce data in response to a get call on the part of the output operator

24

State Based Execution Model

- Programming model for operators is based upon state machine
 - Each operator in the query plan represents a state in a transition diagram
 - A current state s and some set of inputs I causes the operator to transition to a new state (or remain in the same state)
 - $s \leftarrow \text{transition}[s, I]$
 - Implicit in this model of state programming is that operators do not block

25

State Based Execution Model - Continue

```
public class Select extends Module {
    Predicate filter; //The selection predicate to apply
    QueueIF InputQueue;
    ...
    public TupleIF transition(StateIF state) {
        MsgIF msg;
        TupleIF tuple = null;
        //Look for data on input queue
        msg = inputQueue.get();
        if (msg != null) {
            //If this is a tuple, check to see if it passes predicate
            if (msg instanceof TupleMsg) {
                filter.apply(((TupleMsg)msg).getTuple());
                tuple = ((TupleMsg)msg).getTuple();
            }
            else ... handle other kinds of messages ...
        }
        // adjust state: Select is stateless, so nothing to do here ...
        return tuple; //returning null means nothing to output
    }
}
```

(a) Selection Operator

```
public class PullQueue implements QueueIF {
    //Modifies this Queue owner
    Module below, above;
    StateIF bSt;
    ...
    public MsgIF get() {
        TupleIF tup = null;
        //Loop pulling from below
        while (tup == null) {
            tup = below.transition(bSt);
            // ... check for errors, etc ...
        }
        return new TupleMsg(tup);
    }
}
```

(b) Pull Queue

Figure 3. Code Snippet For Selection Operator and Pull Queue

26

Advantage of a State Machine Model

- Reduces the number of threads
 - Threads are very expensive on certain operating system
 - Lack of control on thread
 - Fjord scheduler mechanism enable dynamical control on all operators

27

Sensor Sensitive Operators

- Some traditional operator cannot be applied to the streaming data
- Incrementally or periodically output the results (Non-blocking)
- Window based operator (Blocking)

28

Sensor Proxy

- Shield the sensor from having to deliver data to hundreds of interested end-users
- Adjust the sample rate of the sensor based on user demand
- Direct the sensor to aggregate samples in predefined ways or to download a completely new program into the sensor
- Limit the number of copies of sensor tuples flowing through the query processor to just one per sample

29

Multiple Queries in a Single Fjord

- Instantiate streaming scan operators with multiple outputs that allocate only a single copy of every streaming tuple
- New queries over the same streaming source are folded into an existing Fjord rather than being placed in a separated Fjord

30



Conclusion

- Fjords architecture combines proxies, non-blocking operators and conventional query plans
- Sensor proxies serve as intermediaries between sensors and query plans, using sensors to facilitate query processing while being sensitive to their power, processor, and communications limitations