

TelegraphCQ: Continuous Dataflow Processing for an Uncertain World

University of California, Berkeley

Presented by: Tai Hu

1

Outline

- Introduction
- Adaptive building blocks
- Initial CQ Approaches
- TelegraphCQ
- Open issues
- Conclusion

2

Introduction

- Increasingly pervasive networks are leading towards a world where data is constantly in motion
- Query processors based on adaptive dataflow is necessary
- TelegraphCQ is an extension to the Telegraph project
- Handling large streams of continuous queries over high-volume, highly-variable data streams

3

Introduction - Continue

- Traditional data processing environment is not suitable for motion data
- All challenges stem from
 - Large scale
 - Deeply-networked nature
 - Unpredictability of the environment
 - Need for close interaction with users

4

Data Movement Implies Adaptivity

- Streaming data
 - Pushing, instead of pulling
 - Have to be processed on the fly
 - No statically stored information about data
- Continuous Queries (CQ)
 - Queries are continuously active
 - In traditional database system, query initiate data access operations
 - In CQ, data initiate query processing operations

5

Data Movement Implies Adaptivity - Continue

- Shared processing
 - Avoid blocking or interrupt dataflow
 - Processing each query individually can be slow and wasteful of resources
 - Queries should have some commonalities
- Other Sources of Unpredictability
 - Deeply networked environment
 - User may need to adjust the query on the fly based on the previous result

6

Telegraph - an Ancestor of TelegraphCQ

- Designed to provide adaptivity to individual dataflow graphs
- Two new prototypes to extend Telegraph to support shared processing over streams
 - CACQ
 - PSoup

7

Limitation on Telegraph

- Restricted their processing to data that could fit in memory
- Did not investigate scheduling and resource management issues for queries with little or no overlap
- Did not explicitly deal with the notion of QoS for adapting to resource limitation
- Did not explore opportunities for varying the degree of adaptivity to tradeoff flexibility and overhead

8

Adaptive Building Blocks

		Request Parsing, Metadata			
		SQL	Explicit Dataflows	Catalog	
Modules	Query Processing				Inter-Module Comm API "Fjords"
	Join	Select	SteM	Project Group Aggregation	
		Sort	TransitiveClosure	DupElim	
	Adaptive Routing				
	Eddy	FLuX	Juggle		
		Ingress and Caching			
	File Reader	Sensor Proxy	P2P Proxy		
		TeSS (screen scraper)			

Figure 1 - Telegraph Architecture

9

Ingress and Caching

- Interface with external data sources
- HTML/XML screen scraper (TeSS)
- Proxy for fetching data from peer-to-peer networks (TeleNap)
- Could access data either locally or remotely

10

Query Processing

- Routing tuples through query modules
- Pipelined, non-blocking operators
- A special type of module known as a State Module (SteM)

11

Adaptive Routing

- Constructing a query plan that contains adaptive routing modules
- Optimize the plan on a continuous basis while a query is running
- Eddy and Juggle
- All these modules communicate with each other by using Fjords

12

Adaptive Processing with Eddies SteMs

- Continuously route tuples among a set of other modules according to a routing policy
- A partially or completely commutative set of modules whose inputs and outputs are connected to the Eddy
- Eddy could intercept tuples that flow into and out of these modules, observing the module behavior and choosing the order that tuples take through the modules

13

Implement a Symmetric Hash Join

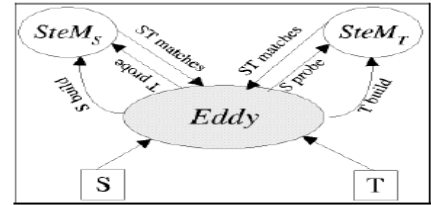


Figure 2 - Eddy and SteMs

14

Fjords – InterModule Communication

- We just talked about this detailed information about Fjords in the previous paper

15

Flux: Scaling Up Dataflow Processing

- Act as an exchange module and provide load balancing and fault tolerance
- Load balancing is provided via online repartitioning of the input stream and the corresponding internal state of operators on the consumer side
- Replicate an operator's internal state and in-flight data for fault-tolerance

16

Initial CQ Approaches

- CACQ was first continuous query engine to exploit the adaptive query processing framework of Telegraph
- CACQ is the modification of Eddies to execute multiple queries simultaneously
- Grouped filters – An index for single-variable boolean factors over the same attributes

17

Initial CQ Approaches - Continue

- PSoup extends the mechanisms developed in CACQ in two main ways
 - It allows queries to access historical data
 - It adds support for disconnected operation
- It treats data and queries symmetrically, thereby allowing new queries to be applied to old data and new data to be applied to old queries

18

Initial CQ Approaches - Continue

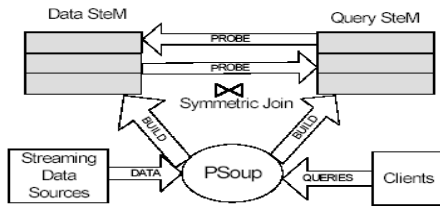


Figure 3 - PSoup

19

TelegraphCQ

- Scheduling and resource management for groups of queries
- Support for out-of-core data
- Variable adaptivity
- Dynamic QoS support
- Parallel cluster-based processing and distribution

20

Window Semantics In TelegraphCQ

- Lanmark – The older end of the window is fixed, while the newer end of the window moves forward with the arrival of new tuples in the stream
- Sliding – Both the ends move forward in unison with the arrival of new tuples
- Some examples

21

Examples for Window Semantics

```
for(t=initial_value; continue_condition(t); change(t)){
    WindowIs(Stream_A, left_end(t), right_end(t));
    WindowIs(Stream_B, left_end(t), right_end(t));
    ...
}
```

22

Examples for Window Semantics - Continue

```
ClosingStockPrices(
    long timestamp;
    char(4) stockSymbol;
    float closingPrice;)
```

23

Landmark Query

```
SELECT closingPrice, timestamp
FROM ClosingStockPrices
WHERE stockSymbol = 'MSFT' and
      closingPrice > 50.00
for (t = 101; t <= 1000; t++){
    WindowIs(ClosingStockPrices, 101, t);
}
```

24

Sliding Query

```

Select AVG(closingPrice)
From ClosingStockPrices
Where stockSymbol = 'MSFT'
for (t = ST; t < ST + 50; t += 5){
  Windows(ClosingStockPrices, t - 4, t);
}

```

25

Effect of Window Semantics on System Design

- Logical time vs. physical time
- Window type
- "Hop" size

26

TelegraphCQ Design Overview

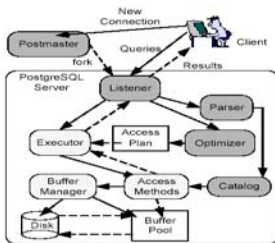


Figure 4 - Architecture of PostgreSQL.

27

TelegraphCQ Design Overview - Continue

- Chief Challenge in using PostgreSQL is supporting the TelegraphCQ features it was not designed for:
 - Streaming data
 - Continuous queries
 - Shared processing
 - Adaptivity

28

TelegraphCQ Architecture

- Font end
 - Listener
 - Catalog
 - Parser
 - optimizer
- Ingress Operator
 - Pull source
 - Push source
- The TelegraphCQ Excutor
 - Actual query processing

29

TelegraphCQ Architecture - Continue

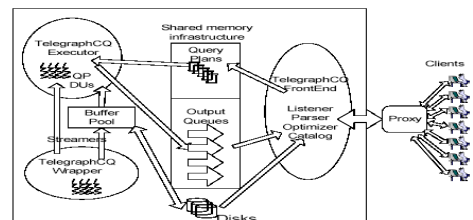


Figure 5 - TelegraphCQ Architecture

30



Discussion and Open Issues

- Query grouping and sharing
- Disk-based issues and QoS
- Egress modules
- Cluster and distributed implementations

31



Conclusion

- TelegraphCQ differs from other data stream and CQ projects due to its focus on extreme adaptivity and the novel infrastructure required to support such adaptivity

32