Leveraging Distributed Publish/Subscribe System for Scalable Stream Processing

Yongluan Zhou, Kian-Lee Tan, Feng Yu
National University of Singapore
Distributed Stream Processing Engines

- Applications:
  - Sensor network, network management
- Multiple widely distributed servers
- Process multiple continuous queries simultaneously
Distributed Stream Processing Engines

- Existing engines:
  - minimizing communication cost
  - construct the overlay paths for each query
Distributed Stream Processing Engines

- **Advantages:**
  - Efficient complex query processing

- **Disadvantages**
  - non-scalable:
    - Sources need keep track of all the queries
  - Hard to exploit the sharing of communication
Distributed Publish/Subscribe Systems

- Applications:
  - stock tickers, sports tickers, news feed etc.
- Content-based stream delivery to a large number of clients
- Backed by a number of brokers
Distributed Publish/Subscribe Systems

- **Advantages:**
  - Scalable
    - Each node is not aware of all actual clients
    - Data are routed based on their contents
  - Efficient in data comm
    - Comm is naturally shared

- **Disadvantages:**
  - Can only process simple data transformation and filtering
COSMOS

- Some applications require both strengths
  - Financial monitoring, stock monitoring, etc
  - a large number of users
  - complex query processing

- COSMOS (COoperative and Self-tuning Management Of Streaming data)
  - A two layer architecture to bridge these two types of systems and leverage their strengths
Two Layer Architecture

- Query layer
  - Employ stream processing engines at each node to process the queries
  - Leverage the data layer to disseminate source streams and result streams
  - Distribute the queries to the servers for processing

- Data layer
  - Utilize distributed pub/sub systems for efficient data delivery
Two Layer Architecture

SPE

Query Wrapper

Query Management

Query Overlay

Unicast Network

Query Streams

Profiles

Data Wrapper

Profiles

Content-Based Network

Overlay Network Optimizer

Data Streams

Data Streams

Transformed Queries

Queries

Queries

Queries
Query Distribution

- Load balancing:
  - to minimize the queueing time
  - maximize the system utility

- Minimizing comm cost of transferring
  - source streams to the processors
  - result streams to the users
  - two issues to be addressed:
    - disseminate each message to as few nodes as possible: minimize the overlap of data interests
    - Avoid transferring data along link with long distance: maintain flow locality
Problem Modeling

Simple Query Graph

- One vertex for each query
  - weight = the query’s load (% CPU cycles)
- One edge between each pair of queries with overlapped data interest
  - Weight = the rate of the data interested to both queries (bytes/sec)
Problem Modeling

- Simple Query Graph Partitioning
  - Balance vertex weights of each partition;
  - Minimize the weighted edge cut;
Problem Modeling

- Extended Query Graph
  - Add one vertex for each processor: p-vertex (weight=0)
  - Add one edge between each processor and its local query (weight = result stream rate)
Problem Modeling

- Extended graph partitioning
  - Each partition has exactly one p-vertex;
  - Balance vertex weights of each partition;
  - Minimize the weighted edge cut;
Graph Partitioning

- We transformed an allocation problem to a graph partitioning problem.
- It is a NP-hard problem.
- It is extensively studied in a lot of areas.
- New challenges in our context:
  - Different graph semantics
  - Allocating partitions to the processors
  - High comm cost in a WAN
  - Vertices could change very fast
Enhance the Scalability

- Adopt a hierarchical coordinator architecture
Hierarchical Graph Partitioning

1. Construct Query Graph
2. Graph Coarsening
3. Graph Partition
4. Construct Query Graph
5. Graph Coarsening
6. Graph Partition
7. Process Query
Graph Coarsening Algorithm
Graph Coarsening Algorithm
Experiments

- Experiment configuration
  - GT-ITM generates a network topology with 4096 nodes
  - Transit-Stub model
  - 100 sources, 256 processors
  - A simulator implemented in C
  - 20 groups of queries with different data hotspot
    - Zipfian distribution (theta=0.8)
Experiments

- Comparing to three algorithms
  - Naive
    - allocate the queries to their local processor
  - Greedy (centralized):
    - Distribute the queries one by one
    - Allocate each query s.t. the current comm cost is minimized and load limit of each processor is not violated
  - Centralized graph partitioning
Communication Cost

- Naive performs the worst
- Hierarchical and centralized graph partitioning perform similarly
- More benefits for graph partitioning algorithms with more queries
Running Time

- Hierarchical partitioning outperforms centralized in both total time and response time.
- More benefits with more queries.
Conclusion

- Proposed a new architectural design for a scalable stream processing system
  - retains the loose coupling, easy-to-deploy and comm efficient merits of a distributed pub/sub system
  - obtains the processing capabilities of a distributed stream processing engine
- A load distribution scheme is proposed for the new architecture
  - The problem is modeled as a graph partitioning problem
  - A hierarchical partitioning is proposed to enhance the scalability
Thank You!