StreamCloud: A Large Scale Data Streaming System

Gulisano, Vincenzo
Jimenez-Peris, Ricardo
Patino-Martinez, Marta
Valduriez, Patrick

Rokey Ge

Outline

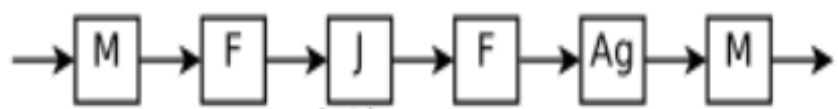
- The need for Data Stream Processing
- Current Stream Processing Engines
- Introducing StreamCloud
- Scalability, transparency, portability
- Evaluations
- My thoughts

Data Streaming

- Applications that require real time processing of data streams
  - Financial data analysis
  - Sensor network data
  - Military command & control
- Store and process can't deal with the high volume and low latency requirements
- Stream processing engines (SPEs)

Data Streaming

- Data stream: infinite append-only sequence of tuples
- Queries are defined over one or more data streams
- Each query is a network of operators
  - Stateless: filter, map, union
  - Stateful: join, aggregate (computation over sliding windo



a) Abstract query

Data Streaming

- Emerging applications are pushing the limit of SPEs
  - Network monitoring, fraud detection
- Distributed SPEs
  - Distribute queries, or operators to individual nodes
- Parallel SPEs
  - Same queries or operators on different nodes in parallel

SPEs

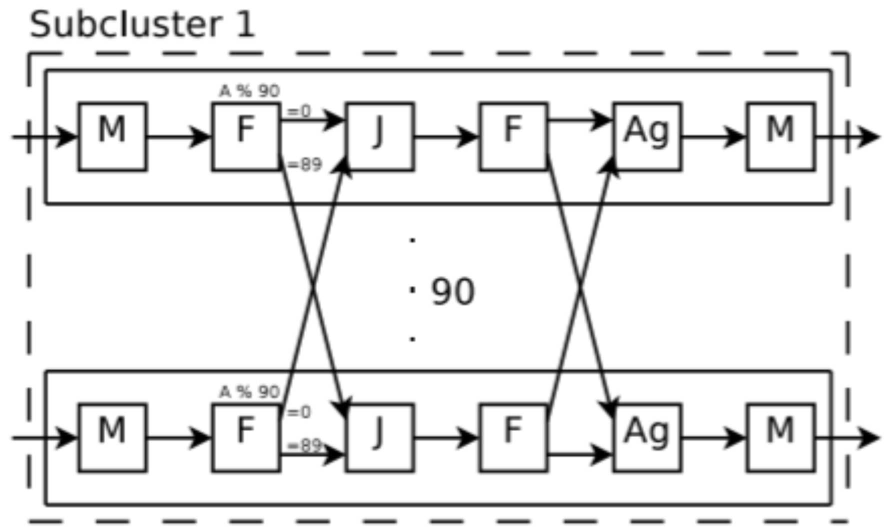
- Aurora [D.J.Abadi et al]
  - Splitting the load across several nodes running the same operator.
  - Data stream go through single nodes,bottlenecks.
- Flux [M.A.Shah et al]
  - Exchange parallel operator, specific to SPEs
- Limited evaluations
  - Simulated, limited scope

StreamCloud

- A data stream processing system
- Scalability: scale with respect to the data stream volume
- Transparency: parallelisation of queries without user intervention
- Portability: independent of underlying SPE

Scalability

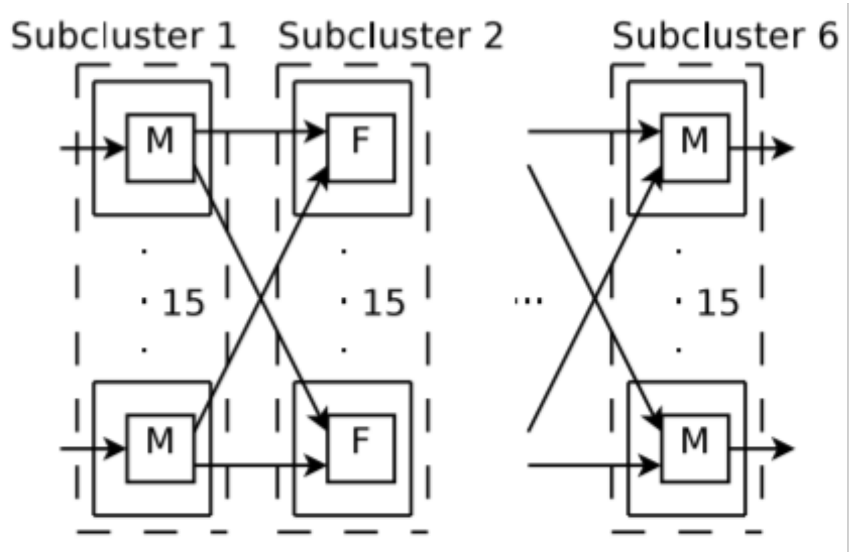
- Query cluster strategy
  - Full query allocated to a subcluster of nodes
  - Nodes execute on a subset of input
  - Communication across nodes, at least for each stateful operator



b) Parallel query: Query cluster strategy

Scalability

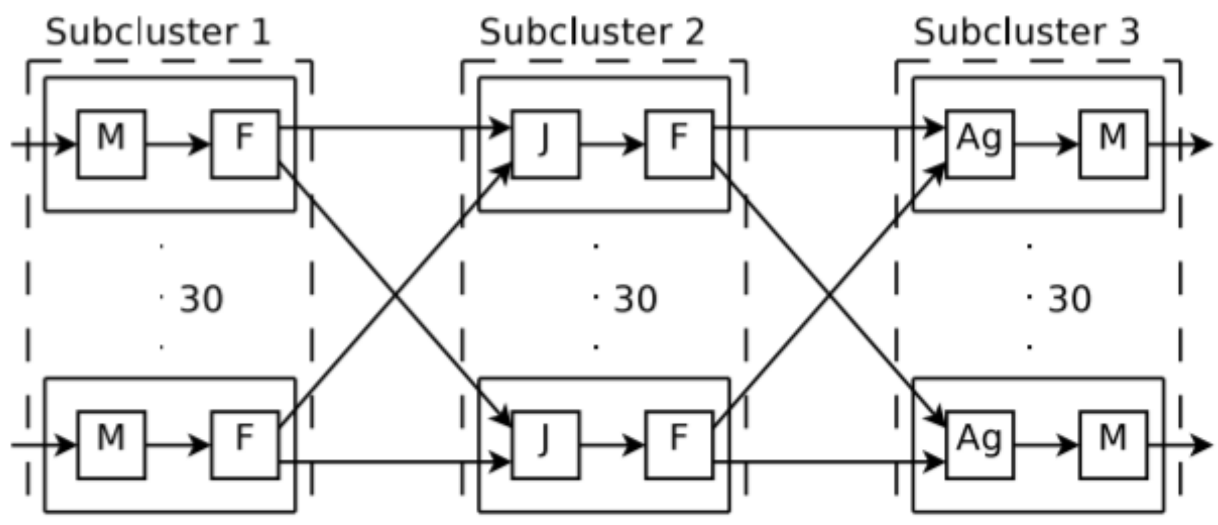
- Operator-cluster strategy
  - Each operator to a set of nodes
  - Communication between nodes of one subcluster to the next



c) Parallel query: Operator cluster strategy

Scalability

- Subquery-cluster strategy
  - Subquery: a stateful operator followed by stateless operators; or the whole query if no stateful operator
  - Subquery to nodes



d) Parallel query: Subquery cluster strategy

Scalability

- Subquery-cluster strategy
  - Minimum number of communication steps
  - Minimum fan out cost
- Parallelization of Staeless subqueries
  - Each input tuple can be processed by any node
  - Load balancer applies round-robin to distribute

Scalability

- Parallelization of Stateful Subqueries
- Join and Aggregate (group-by)
  - Each input stream split by LB into N substreams
  - hash(A)%N to distribute tuples
- Cartesian Join
  - Each tuple is sent to M=sqrt(N) nodes
  - %M to distribute

Scalability

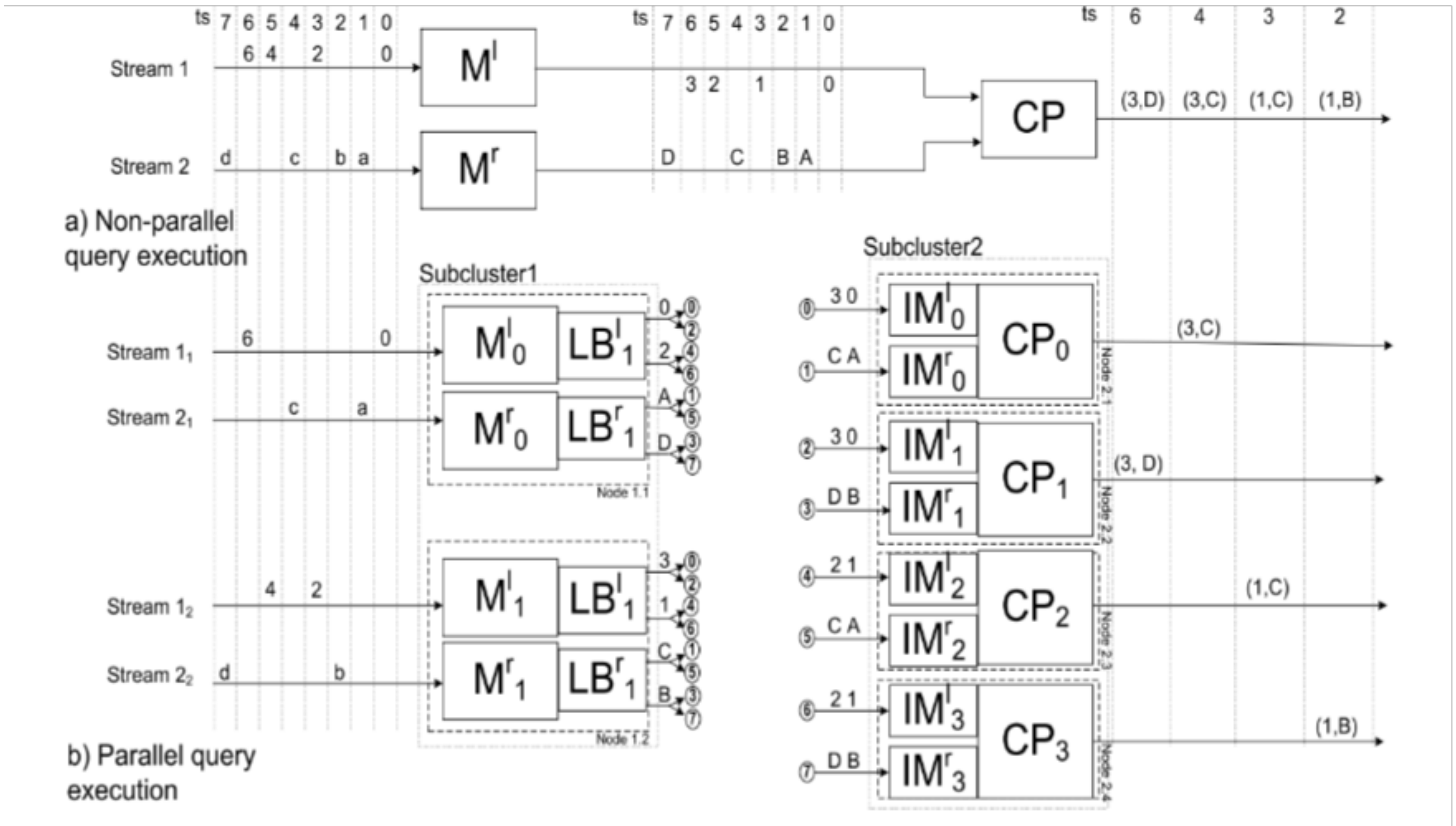


Fig. 4. Cartesian Product Sample Execution

- Transparency
  - Parallelization result should equal to non parallel version
  - Input Merger: takes timestamp ordered substreams from LB and generate ordered substream
- Optimisations
  - Merge stateful subqueries if they share same aggregation method
  - Merge union with IM, filter with LB

Evaluation

- Targets to measure the scalability
  - The number of processors
  - The window size
- Methodology
  - Increasing input loads for different configurations
  - StreamCloud instances process tuples until it overloads
  - Throughput: tuples/comparisons per second
  - CPU usage, queue length

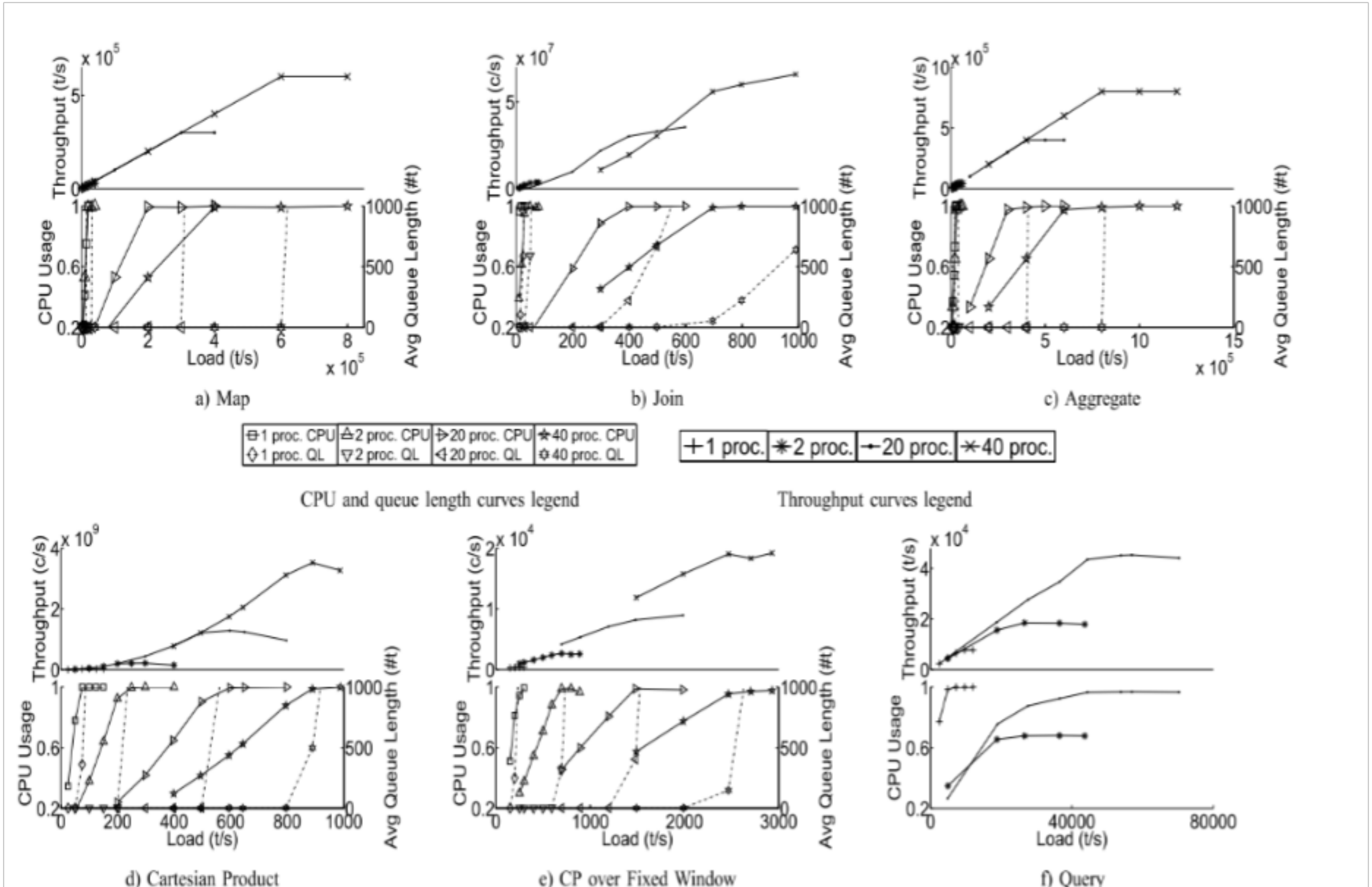
Evaluation setup

- 60 nodes with 160 cores
- Multiple instances of StreamCloud per node for multi-core nodes
- Baselines: centralised SPE on one node; two StreamClound instances on one node

Evaluation Plan

- Scalability of each individual operator
- Scalability of full queries
  - Comparison with query-cluster and operator cluster strategies
- Increase system size while maintain fixed window size to handle increased input node
- Scalability in terms of numbers of instances per node

Crazy charts



Crazy charts explained

- Operators scale well
- Subquery-cluster is 2.5 to 5 times better than query-cluster and operator cluster
- Scale with cores too
- Scalability maximised!

My thoughts ++

- Subquery-cluster strategy provides better scalability
- Load-balancer & Input-merger implemented with standard stream operators
- Detailed evaluations over real implementation (albeit crazy charts)

My thoughts --

- Other operators? (e.g. Bsort, ReSample)
- How does it handle network imperfections?
  - Delayed, missing, out-of-order data
  - Broken node
- Independence unproven. What about other SPEs?
- Evaluations do not contain comparison with other systems

Questions?

- ???
- ??
- ?