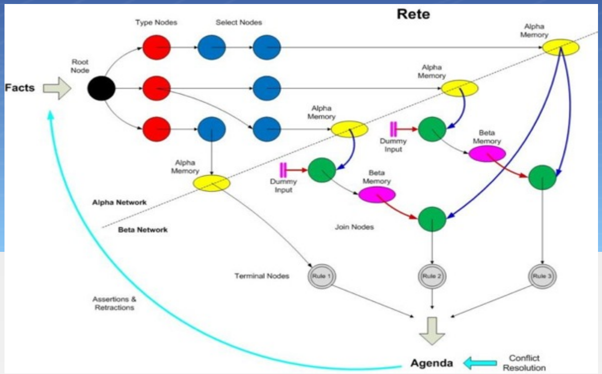
Optimising RETE for Stream Reasoning

Using the Storm framework

Motivation

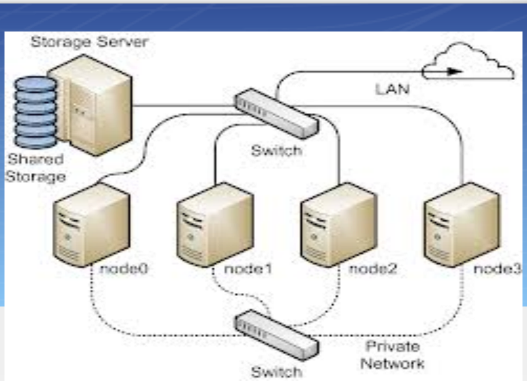
- We want to process (enrich and query) “big data”
- Data looks like “User256-likes-post2045” and is continuously generated
- DBMS can’t deal with the update rates
- We need a new type of system to apply ***persistent entailment rules*** and ***continuous queries*** on the data
- We want the system to run on a computer cluster and ***scale horizontally***

The RETE algorithm



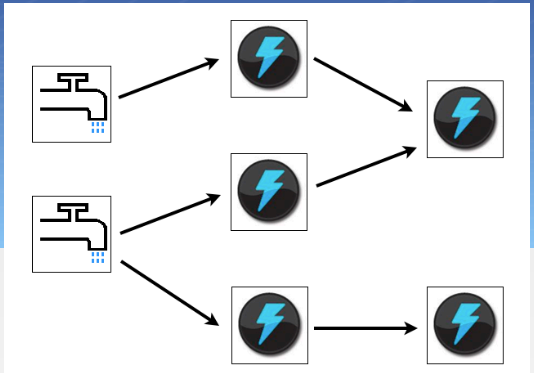
- ***A pattern-matching*** algorithm for implementing production rule systems
- Essentially a way of forming ***long-lived*** graphs of ***Filters and Joins***
- These graphs are called RETE-networks
- Tuples pass through the network ***once*** and partial matches are stored after each node (i.e. sacrificing memory for speed)
- A single network can implement many concurrent queries and therefore ***nodes can be shared***

RETE on a computer cluster



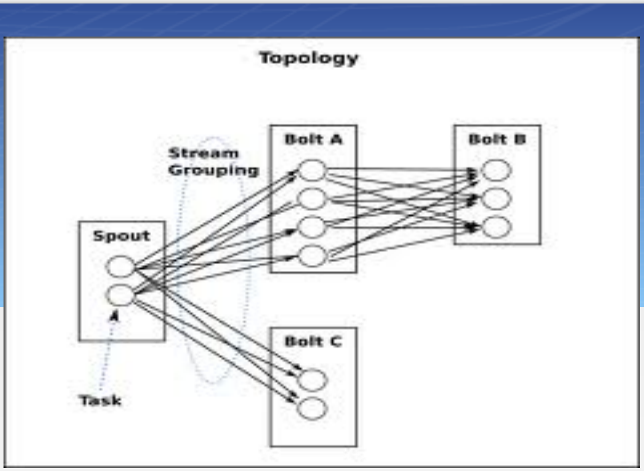
- In search of a framework to hide the complexity of the cluster
- Storm is a relatively new, open-source framework from Twitter
- Built specifically for ***scalable continuous computation***
- Storm applications are called ***Topologies*** and are similar to MapReduce jobs
- However they run indefinitely (waiting for tuples to process) until manually terminated

Storm Topologies



- Storm’s three basic abstractions: ***Streams***, ***Spouts*** and ***Bolts***
- A Stream is an unbounded sequence of tuples with a ***predefined schema***
- Spouts read tuples from some source and generate one or more Streams
- Bolts subscribe to any number of streams and process tuples as they are received, possibly emitting new Streams

Stream groupings



- The workload is spread across multiple JVM instances running in parallel on the cluster
- Spouts and Bolts can be seen as having ***multiple instances*** running in parallel
- ***Stream groupings*** define which instance processes each tuple
- ***Shuffle grouping*** for Filters, ***Fields grouping*** for Joins, ***All grouping*** for cross-products

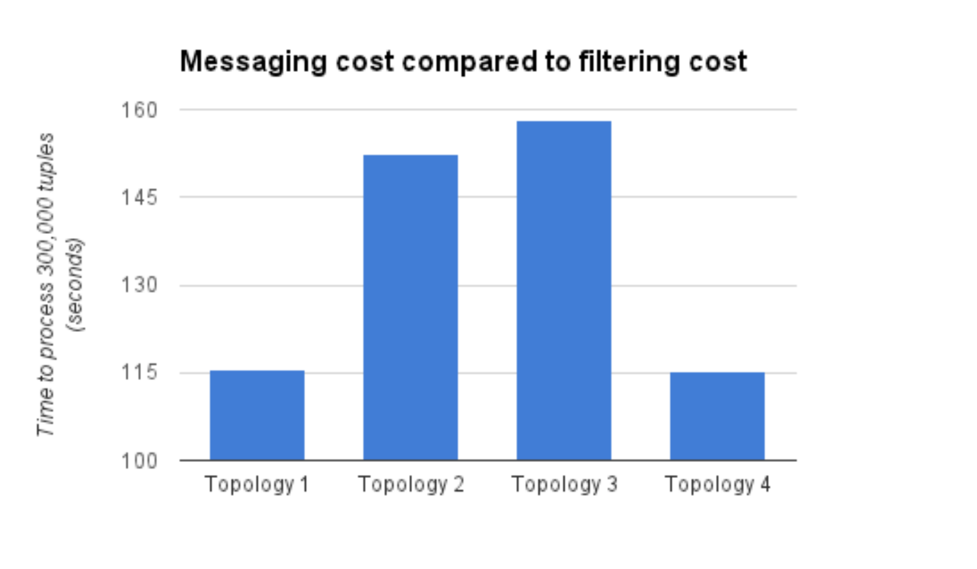
RETE on Storm

- TrendMiner (Sina et al, 2012)
- Single Spout, one Bolt per Rete operator
- Heavy message replication from Spout to Filter-Bolts
- Inter-query node sharing when queries have common filters
- Also Join-Bolt reordering to make cross-products smaller

Is node-sharing beneficial on Storm?

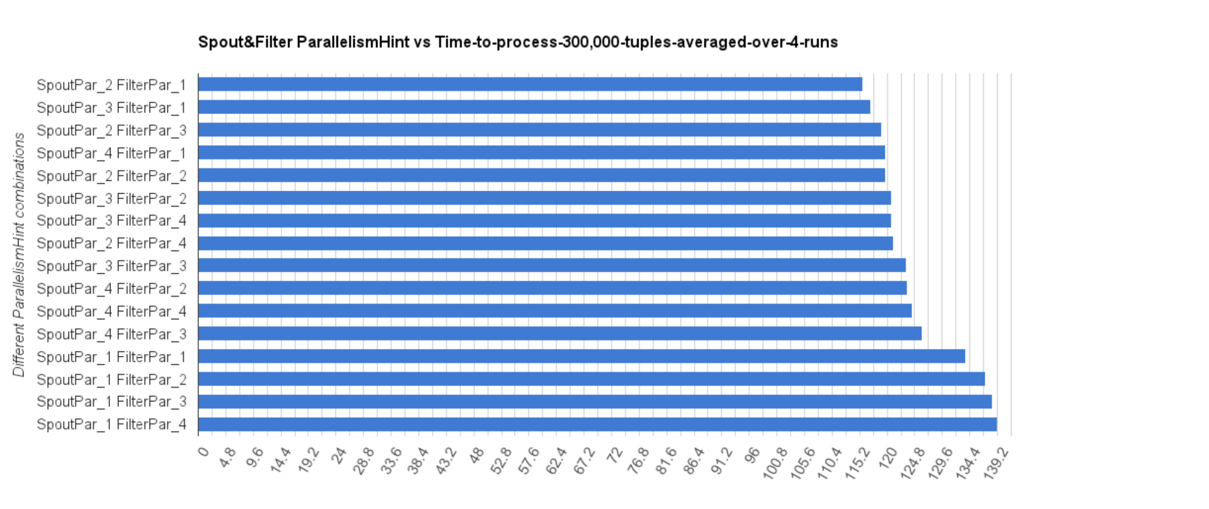
- In-process cluster simulator - Storm’s “local mode”
- Set debug mode “on” to see tuples being emitted
- ***Kestrel*** server for I/O queues
- ***Maven*** to build the project
- Used ***Python***, ***Perl*** and ***Bash scripting*** to run Topologies and ***measure throughput***

First experiment



- Confirm that simulation is realistic
- Do messages cost more than simple string comparisons?
- Yes.

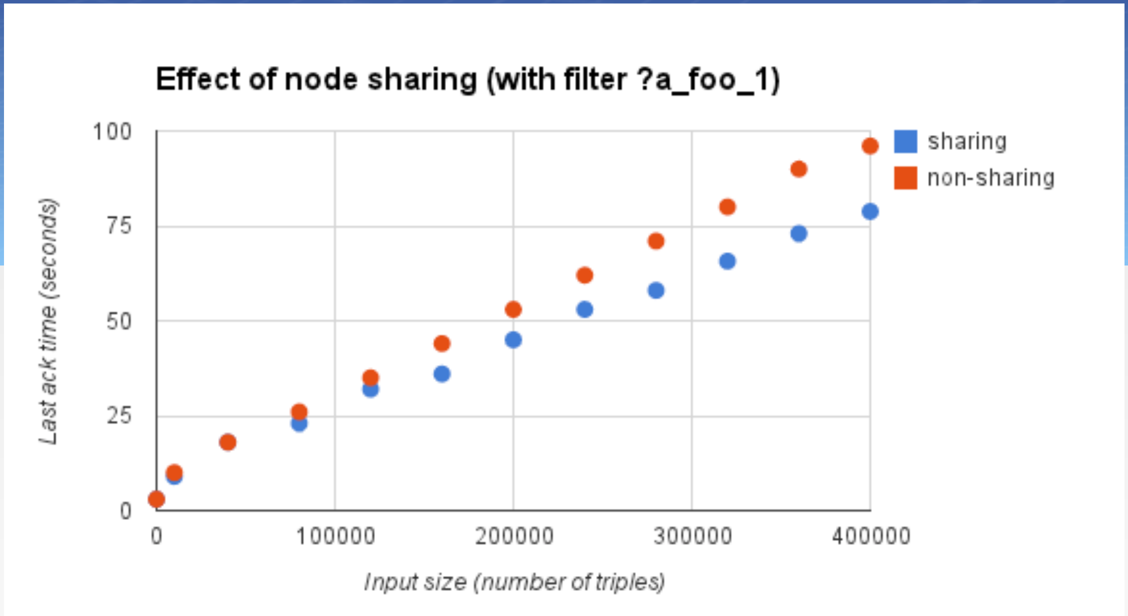
Second experiment

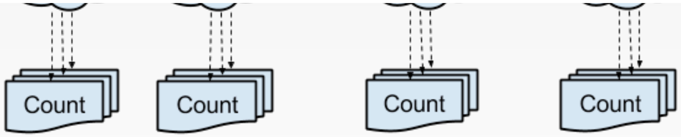


Spout&Filter ParallelismHint vs Time-to-process-300,000-tuples-averaged-over-4-runs

- ***Tuning the parallelism*** of the Topology
- Topology: a Kestrel-Spout, a Filter-Bolt and a Terminal-Bolt
- In local mode, optimal parallelism was: two instances for the Spout and one for each Bolt

Third experiment



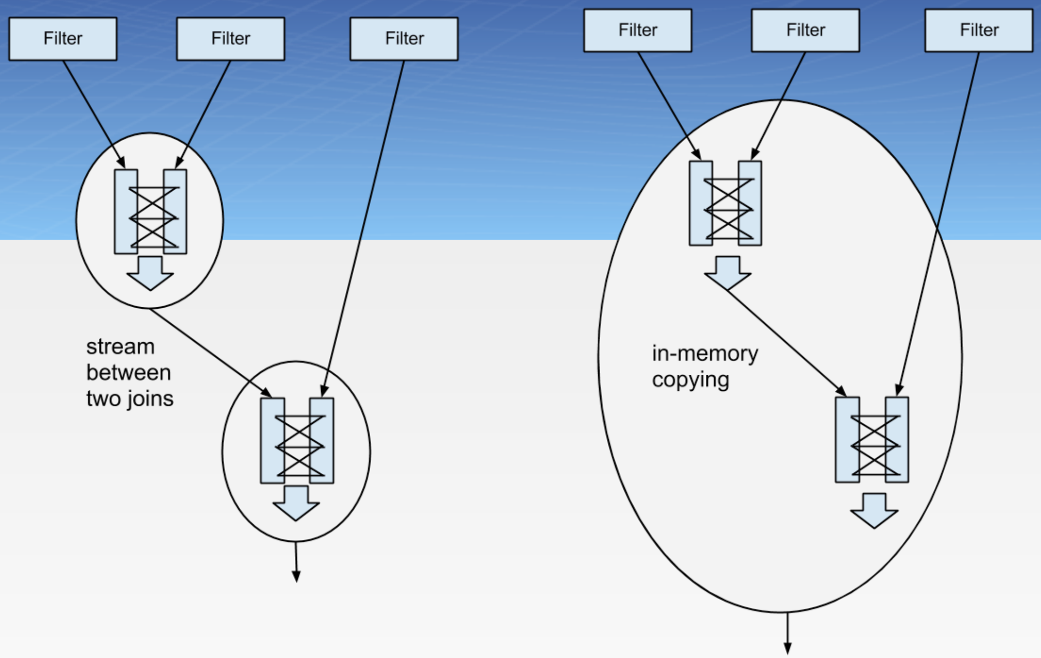


- Sharing two ***equivalent filters***
- Filters that block the same tuples
- Slope of the line is the throughput

Why not put all Filters in the same Bolt then?

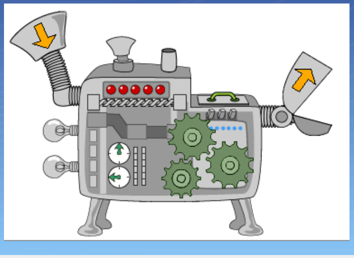
- **En-bloc filtering**: One bolt doing all filtering – one shuffle grouping to the spout
- Instead of 10 bolts emitting ***one stream each***, one bolt emitting ***all 10 streams***
- Join-Bolt subscribed to <Filter-Bolt-7, stream “default”> now subscribes to <Universal-Filter-Bolt, stream “7”>
- The Universal-Filter-Bolt internally iterates over the whole filter list
- Filter sharing can still happen internally
- The bolt’s can still run in parallel across the cluster

Can we do the same with Joins?



- To some extend, yes.
- Fields grouping is what limits us
- Or rather, the need to parallelise Join-Bolts without missing joins
- **Join clustering**: “If two joins in a cluster have a variable in common, all joins in the cluster should have that variable in common”

The delivered Topology Builder



- Takes a list of filters as input and creates Storm Topologies
- The "shape" of the Topology depends on whether various optimisations are turned on or off
- Optimisations: ***Equivalent-filter-sharing***, ***En-bloc-filtering***, ***Join-clustering***
- Shows that En-bloc filtering is superior to *external* equivalent filter sharing
- Show that Join-clustering is beneficial and that it doesn't break when parallelism is more than 1
- Not meant to be a fully working DSMS system (just a program for experimentation)
- I've also submitted the various scripts that can be used to run the Topologies and measure their throughput in local mode