Partitioning Techniques for CEP

Background

In contrast to normal stream processing, CEP generally deploys stateful operators (SEQ, NEG etc.). Scaling CEP has several dimensions:

1. Handling large number of queries.
2. Queries that needs large working memory.
3. Handling a complex query that might not fit within a single machine. (Focus of this report)
4. Handling large number of events. (Focus of this report)

Motivation

This report focused on analysis the techniques for “Handling large number of events handling a complex query that might not fit within a single machine”.

> In DEBS’14 Grand challenge, The not-been solved tough question is:” For each house, calculate the percentage of plugs which have a median load during the last 24 hour greater than the median load of all plugs (in all households of all houses) during the last 24 hour, and the output must be re-calculated and output whenever a new event enters or leaves”. (The data is generate around 1000 event/second for each house), so the naive approach will require processing around 1000*43600*40 ~34 billion event/second…..

To handle such scenarios, we have to distribute the workload across many computers, either through data partitioning or task partitioning.

Stream partitioning

It exploits data parallelism by partitioning the input stream and sending the tuples of each partition (sub-stream) to a replicated sub-graph of the dataflow program. Data are split in the following way:

- Round-robin
- Value-based such as range
- Hash-partitioned, partitioned by a given key

methods	Merge step	Requirements	Pro	Con
RR	Require heavy post-processing	No requirements	Always applicable	Require heavy post-processing, cannot do much work.
ValueB	Depends	Can be partition by value, say, some numerical attribute	Generally applicable, possible to perform sequencing and aggregation on the	It’s sites between RR and H.P.
H.P	Simple merge	Given a partition key	No explicit , each sub-stream can be processed on independently	Not always applicable.

1. In DEBS’13, “RIP: Run-based Intra-query Parallelism for Scalable Complex Event Processing”.[#citation:3], further explored this idea, but instead of State-based, they proposed the Run-based, which means, a partial match remains on a single processing unit until it constitutes a match or it’s removed because it cannot end in a match anymore. Experiment shows to outperform the earlier approach. This work is very inspiring, in the sense that, they transfer the original partitioning problem into scheduling problem.

The author apply RR scheduling to feed the batch of each machine:

> Claimed by author: "Although different processing units will require different amounts of times, idle processing unit can be assigned to the next batch, and all processing units will be busy at a given time. Thus, different processing times do not create load imbalance across the processing units."

However, the method proposed only works for continuous matching strategy. And whether the data is duplicated or not is unclear.

comments

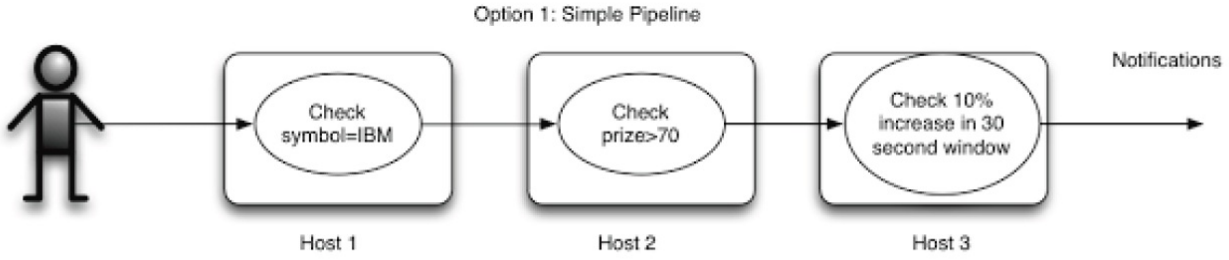
How to fix the con site of the methods? By introduce a novel algorithm? How to dynamically use those methods?

```
- Some stream use H.P?
- Some stream use RR?
```

1. my idea: partitioning by head event + within.

CEP Pattern partitioning

1. In DEBS'09, "Distributed event stream processing with non-deterministic finite automata" [#citation: 36] firstly explored this idea. The main idea behind is to segment each NFA state to be run on a separate node in parallel as a pipeline.
    - the idea is illustrate as following graph:



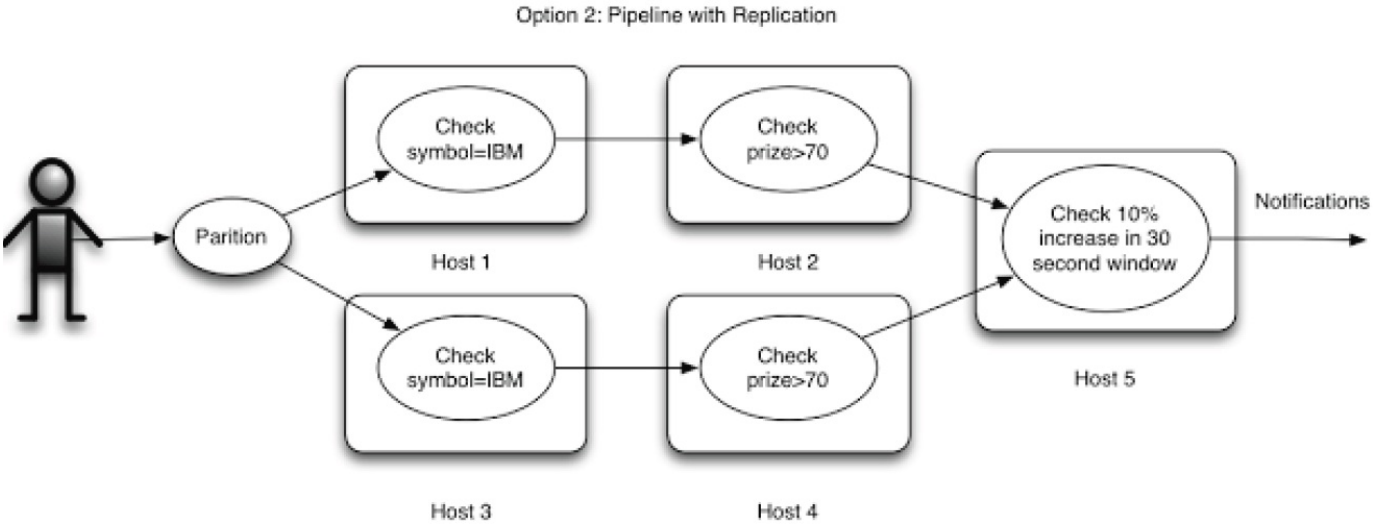
CEP Pattern query rewriting and partitioning.

1. In VLDB'08, "Plan-based Complex Event Detection across Distributed Sources" [# citation: 95], firstly explored the idea of "CEP Pattern query re-writing". the author present an approach for communication-efficient complex event detection over distributed sources. A multi-step detection plan for a complex event is generated on the basis of event frequency statics, postponing the monitoring of high frequency events to later steps in the plan. E.g. one "SEQ" is rewritten into several consequent sequence sub-operators.

2. In DEBS'09, "Distributed Complex Event Processing with Query Rewriting" [# citation: 70] Instead of considering only "how to distribute the original pattern (NFA, Query tree)", they consider the benefit of:

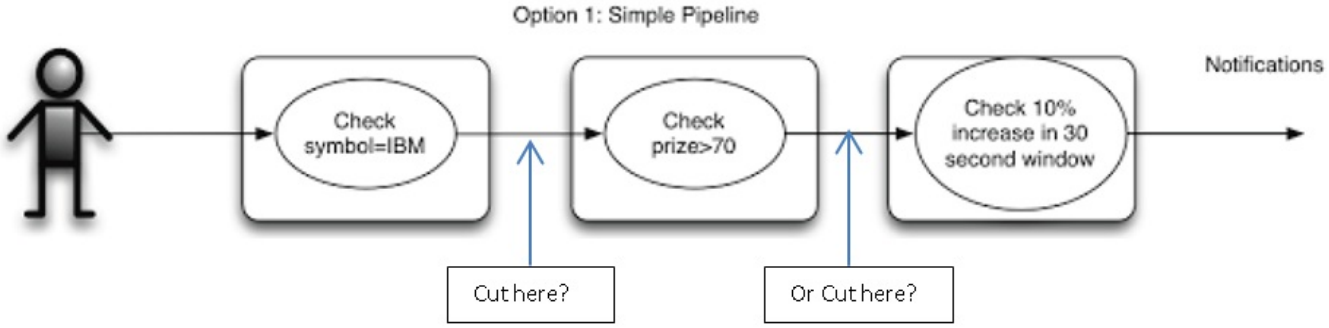
    1. rewriting the original pattern query into a more efficient equivalent manner and then
    2. distribute the operators. They addressed "Simple cases", (sequence, any) focus on the efficient use of CPU resources.
3. In DEBS'11, "Pattern Rewriting Framework for Event Processing Optimization" [#citation: 10] follows up the previous work, and further considered the "pattern assertion and policies" issue.

4. In DEBS'12, "Partition and compose: Parallel complex event processing" [#citation: 20] follow up, and provide a MatchRegex operator for System S, which 1) detects tuple patterns in parallel(through hash partition), 2) automatically translate the pattern into automaton, and 3) also implements the aggregations incrementally, idea like this:



    However it does not support "Dynamic repartitioning on demand"
5. In New Trends in Database and Information System II, 2015, "Partitioning for Scalable Complex Event Processing on Data Streams", author

proposed a cost-based optimization approach for determining the number of partitions as well as the split points in the queries to achieve better load balancing and avoid congestions of processing nodes in a cluster environment. The partition strategy used by author include: Hash partitioning for streaming data (A simplified map function), NFA state partitioning and combine. The author proposed a greedy algorithm to partition the query graph considering CPU utilization (should also have memory, network), but the experiment part is weird, he first test stream partitioning, <throughput V.S partition number(cores)>, so he get the “optimal partitions number”1 for each query, later, he shows the graph partitioning planning(cost model) are able to (closely) estimate “optimal partitions number” 2. It’s weird that, the first optimal partitions number is refer to stream partitioning, but the cost model is referring to operator partitioning.



CCX query optimizer from XXXXX (confidential)

CCX is a optimizer that translate original CCL projects(may consists of multiple queries) into a CCX query execution plan graph. The optimizer is different compare to the one in traditional database:

```
1. In traditional DB, optimizer optimize query plan tree.
2. In CCX, optimizer optimize query plan graph.
```

In short, the optimizer will automatically determine the parallelism part of the graph, and then apply parallelism or pipeline into this graph.