Complex event processing system

Background

This report is to summarize different complex event processing system with a focus on their different features.

Non-deterministic Finite automata(NFA)

NFA is the most common used method for evaluating CEP queries.

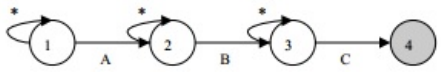

Figure 1: An example NFA for processing the sequential pattern A followed by B followed by C

1. SASE

1. [High-Performance Complex Event Processing over Streams] sigmod'06
2. [Recognizing patterns in stream with imprecise timestamps] vldb'10
3. [Optimizing Expensive Queries in Complex Event Processing] sigmod'14

The SASE system primary focus RFID industry.

1.1 Language features:

- **sequencing(SEQ)**:This is not *new*, idea come from "active database" community.
- **negation(!)**:Extends negation from active database to be more flexible to use in event sequence.
- **parameterization(WHERE):** This is used to distinguish with the simple predicates that compare to a constant. It adds feature to correlating events via value-based constraints.
- **windowing(WITHIN)**: sliding windows for imposing additional temporal constraints

1.2 Query plan:

presented a query query plan-based approach to implementing this language, which uses native operators.

1.2.1 Native operators

- **Sequence scan and construction (SSC)**: SSC handles the *positive* components of SEQ, internally, SSC contains
  - a sequence scan operator(SS->).
  - sequence construction operator(SC<-).
- **Negation(NG)**: NG handles the *negative* components of a SEQ, e.g. for SEQ(A,!B,C) this operators checkes if there's a 'b' event that arrived between the 'a' and 'c' event.
- **Window(WD)**:For *each event sequence*, it checks if the temporal difference between the first and lasts event is less than the specified window T.
- **Selection(σ)**:works similar to relational query processing, but include *parameterized* ones.

1.2.2 Event sequence Matching

The author adopt NFA to represent the structure of an event sequence:

Sequence Scan and Construction

For each **SSC** sub-sequence type, an NFA is created by mapping successive event types to successive NFA States. Like figure 2, state 0 is the starting state,state 1 is for successful recognition of an A event.. state, denoted using two concentric circles, is the accepting state of the NFA.

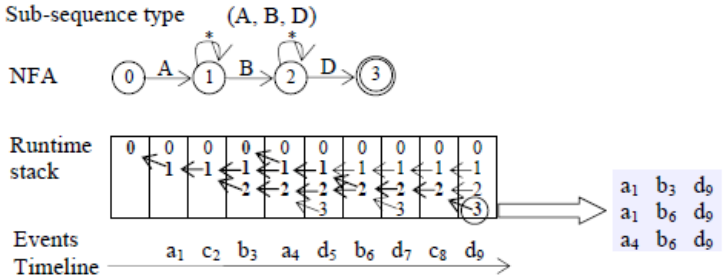

Figure 2: NFA-based Sequence Scan and Construction

Once an accepting state is reached during sequence scan, **sequence construction** is invoked to create the event sequences that the most recent event has completed.**Notice that**: This is actually because of the special feature of SASE: selection policy (i.e. contiguity/skip till next/skip till any).

The *solution* proposed by author is: Extract from the runtime stack a single-source DAG that starts at an instance of the accepting state in the rightmost cell of the stack and traverses back along the predecessor pointers until reaching instances of the starting state. Event sequence can then be generated by enumerating all possible paths from the source to the sinks of the DAG.

Negation

NG handles the negative components of the SEQ construct in a query which have been ignored by SSC. For each input event sequence, NG performs two tasks for each negative component:

Task 1: Check if an event of the type specified in the negative component appeared in a specific time interval.

Task 2: If such an event exists, check if it satisfies all the relevant predicates.

Any event passes both will cause the sequence to *False*.

Compile:Time interval for task (1)

- SEQ(A,!B,C): Time interval is defined as (A.timestamp, C.timestamp);
- SEQ(!A,B):the window size T is used to set interval to be (B.timestamp - T, B.timestamp).
- SEQ(A,!B):Interval is set to(A.timestamp, A.timestamp+T), **in addition**, the negation operator is marked as "postponed by T", which indicates to the runtime system that the evaluation of each event sequences needs to be postponed by a period of length T after its arrival.

Runtime system support

In addition to the possible postponed evaluation, it provides indexing support,

> i.e. given a time interval, retrieving all the events that occurred in the interval can be supported by using a standard relational indexing techniques.

The author proposed an advanced technique called *partitioned indexing*. the idea is to *partition an event stream by timestamp.*

1.3 optimization for handling:

> A mechanism shared by all these optimizations is to index relevant events both in temporal order and across value-based partitions.

- large windows:
  - intra-operator optimization to expedite sequence scan and construction(SSC).
- reducing intermediate result size.
  - inter-operator optimization that strategically push predicates and windows down to SSC to reduce intermediate result sizes.

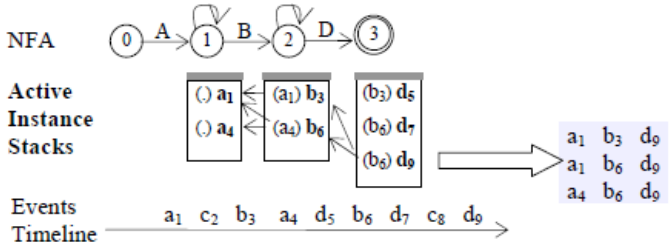

Figure 3: SSC using Active Instance Stacks

Sequence Scan

Instead of using a runtime stack to record the set of active states as an event arrives like Figure 2. An active *instance* stack is created at *each* NFA state to store the events that triggered transitions to this state like Figure 3. In addition, each instance maintain its own *most Recent Instance Previous stack(RIP)*, which tells that any instances in the A stack up to a_4 (i.e. a_1 and a_4)can be matched with b_6 if event sequences involving b_6 need to be created.

Sequence Construction

With active instance stack, the consturction is imply done by a DFS in the DAG that is rooted at this instance and contains all the virtual edges reachable (RIP)

Pushing Predicates Down

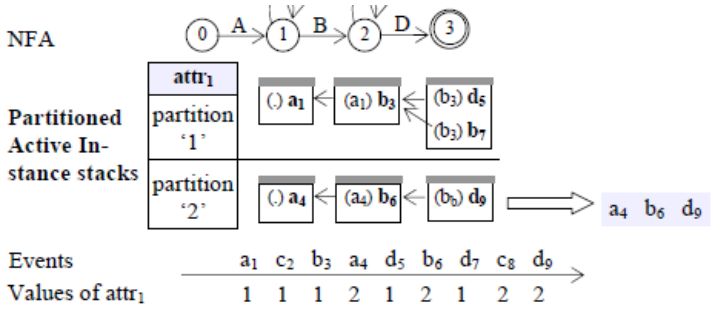

Figure 4: Partitioned Active Instance Stacks (PAIS)

Pushing an equivalence test down to SSC

the idea is partition the data by the 'equivalent attribute', so for instance, d_9 does not need to go back and check for b_3.

Pushing multiple equivalence test down to SSC

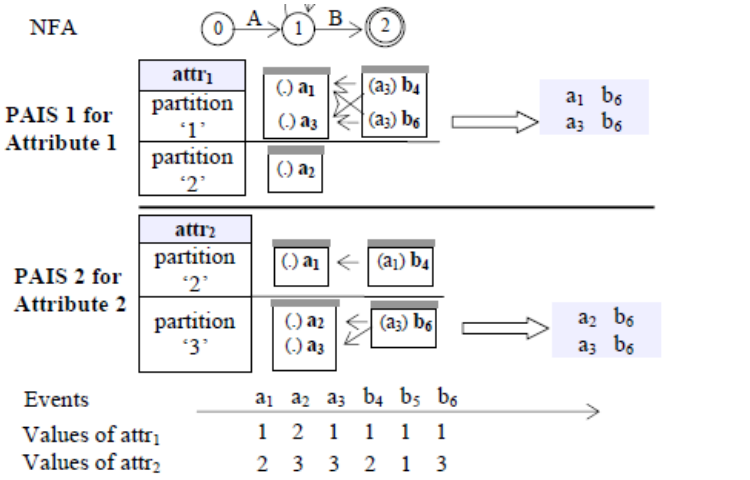

Figure 5: Multiple Partitioned Active Instance Stacks (Multi-PAIS)

like in Figure 5, we can partition again after partition.

Pushing Windows Down

Similarly, window constraint can also be evaluated early in SSC to reduce the number of event sequences created.

> detail are omitted by the author

- window in sequence scan can filter some of the events so they are not added to active instance stacks, and prunes expired instance from stacks.
- window in sequence construction searches those stacks and performs window checking on the fly for each event sequence to be generated.