Stream and Complex Event Processing
Discovering Existing Systems: T-Rex

G. Cugola    E. Della Valle    A. Margara

Politecnico di Milano    Vrije Universiteit Amsterdam

cugola@elet.polimi.it    a.margara@vu.nl
dellavalle@elet.polimi.it
T-Rex: an Engine for TESLA

Define

From

Where

Consuming

CE(Att₁ : Type₁, ..., Attₙ : Typeₙ)

Pattern

Att₁ = f₁(..), ..., Attₙ = fₙ(..)

e₁, ..., eₘ
The Ingredients: Selection ...

- Selection of a single event
  - \( A(x>10) \)
  - \texttt{Timer()}
The Ingredients: ... Sequences ...

- Selection of sequences
  - $A(x>10)$ and each $B$ within 5 min from $A$
  - $A(x>10)$ and last $B$ within 5 min from $A$
  - $A(x>10)$ and first $B$ within 5 min from $A$
- Generalization
  - n-first / n-last
The Ingredients: ... Sequences ...

- TESLA allows *-within operators to be composed with each other:
  - In chains of events
    - A and each B
      within 3 min from A
      and last C
      within 2 min from B
  - In parallel
    - A and each B
      within 3 min from A
      and last C
      within 4 min from A
The Ingredients: ... Parameters ...

- Parameters can be added between events in a pattern
  - $A(a=x)$ and each $B(a=x)$
    - within 3 min from $A$
  - and last $C(a=x)$
    - within 4 min from $A$
SELECTION
Selection

• Generic problem of matching events against constraints on their content

• Used twice in CEP
  • To select the set of rules a primitive event is relevant for
  • To select the set of sinks a (composite) event needs to be delivered to
    • Classical Publish / Subscribe Content-Based Matching
Content-Based Matching

Events → Content-Based Matching → Sinks

Attribute: Light=50, Room=Bedroom, Sender="Sensor1"

Predicate: (Smoke=true and Room = “Kitchen”) or (Light>30 and Room="Bedroom")

Filter → Constraint
Related Work

• Two kinds of algorithms
  • Tree-Based
  • Counting
Tree-Based Algorithms

- Build a decision tree
  - Inner nodes: conditional statements
  - Leaves: sinks

```
A>0
  Yes
  B>8
    Yes
    C>2
      Yes
      S1, S2, S4
    No
    B≤2
      Yes
      S1, S3
  No
```
Counting Algorithms

• For each filter ...
• ... count how many constraints have been satisfied so far
Counting Algorithms

A=12
B=20

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>A&gt;10</td>
<td>F1</td>
</tr>
<tr>
<td>B=20</td>
<td>F1</td>
</tr>
<tr>
<td>B&gt;15</td>
<td>F2</td>
</tr>
<tr>
<td>C&lt;30</td>
<td>F2</td>
</tr>
<tr>
<td>D=20</td>
<td>F3</td>
</tr>
</tbody>
</table>

Filter | Size | Count | Predicate |
-------|------|-------|-----------|
F1     | 2    | 0 1 2 | S1        |
F2     | 2    | 0 1   | S1        |
F3     | 1    | 0     | S2        |

F1: A>10 and B=20
F2: B>15 and C<30
F3: D=20
Idea: Exploit Parallel Hardware

• PCM: Parallel Content-Based Matching
  • Both multi core CPUs ...
    • OCM – OpenMP
  • ... and Nvidia, CUDA GPUs
    • CCM
Programming GPUs: CUDA

• General purpose parallel computing architecture by Nvidia
  • New instruction set
  • **New programming model**
  • Programmable using high-level languages
    • Cuda C (a C dialect)
Programming Model: Basics

• The GPU acts as a coprocessor with its own separate memory space

• Copy data to/from the GPU memory is expensive
  • Communication through the PCI-Ex bus
    • Bandwidth but also latency!
  • Serialization of data structures
    • Keep them simple!
Typical Workflow

1. Allocate memory on device
2. Serialize and copy data to device
3. Execute one or more *kernels* on the device
4. Wait for the device to finish processing
5. Copy results back

Stream & Complex Event Processing - Discovering Existing Systems: T-Rex
Programming Model: Fundamentals

- **Single Program Multiple Threads** strategy
  - A single *kernel* is executed by multiple threads in parallel

- Threads are organized in *blocks*

- The runtime provides a *blockId* and a *threadId* variable, to uniquely identify each running thread
  - Accessing such variables is the only way to differentiate the work done by different threads
Hardware Matters!

• Lots (lots!) of cores …
• … but specialized for data parallelism

• Threads organized into groups, called warps
• Maximum performance only if all threads in a warp agree on the execution path
  • Conditional statement can create divergence!
Hardware Matters!

- Lots (lots!) of cores ...
- ... limited space for cache 😞
Hardware Matters!

• Lots (lots!) of cores ...
• ... limited space for cache 😞
  • In some architecture caching is controlled by the programmer
  • Hierarchies of memories
    • Global (DRAM, GB) → Shared (SRAM, block level, KB) → Local (Registers)

• Access to GPU memory often a bottleneck
  • Threads with contiguous ids should access contiguous memory regions
  • Hardware combines them into memory-wide accesses
PCM

- Constraints with the same name are stored in an array
  - Contiguous memory regions
- When processing an event E, the CPU selects all relevant constraint arrays
  - Based on the name of the attributes in E
PCM

- Bi-dimensional organization of threads
  - On GPU, one thread for each attribute/constraint pair
  - On CPU, different rows are processed sequentially
- On GPU, filters count updated with an atomic operation
PCM - Results
Performance: Throughput

Stream & Complex Event Processing - Discovering Existing Systems: T-Rex
Performance: Throughput

![Graph showing throughput performance for different systems.](image)

- **Throughput**
  - Throughput: 26
  - Stream and Complex Event Processing
  - Discovering Existing Systems: T-Rex

- **Graph Details**
  - Processing Rate (k events/s)
  - Input Rate (k events/s)
  - CCM
  - OCM
  - SFF
  - SFF ME
Results

• What is the time needed to install subscriptions?
  • Need to serialize data structures
  • Need to copy from CPU memory to GPU memory
  • But data structures are simple!

• Memory requirements?
  • Less than 1GB in all our tests
  • Not a problem for a modern GPU
PCM – Lessons Learned

• Algorithm designed for GPUs
  • Very simple data structures (matrix)
  • No pointers

• Benefits on CPU as well
  • Better use of data locality and cache

• Even better performance in case of bursts
  • Several events waiting to be processed
  • Can be processed in parallel
SEQUENCES
Related Work

• Sequences apparently similar to regular expressions
  • Detect sequence A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D
• Use algorithms based on automata
Problems

• Multiple selection
  • A
    and each B within 5 min from A
    and each C within 5 min from B
  • C1 C2 C3 B1 B2 A1
  • We need one automaton (instance) for each possible valid sequence!
    • C1 B1 A1
    • C2 B1 A1
    • ...
Problems

• Traditional approach
  • Duplicate!
  • C1 C2 C3 B1 B2 A1
    • Lot of memory!
    • Lots of copies!

Stream & Complex Event Processing - Discovering Existing Systems: T-Rex
Problems

• Parameters
  • A(x=$a)
    and last B within 5 min from A
    and last C(x=$a) within 5 min from B
  • We do not know which C can be useful
    • Until we receive an A
    • We have to save all of them!
  • We create a lot of automata that are later discarded
    • Single selection becomes expensive as multiple selection
From Automata to Columns

• Our first implementation (T-Rex 1.0) was based on automata
  • AIP (Automata-based Incremental Processing)
• We developed an entirely new algorithm in T-Rex 2.0
  • CDP (Column-based Delayed Processing)
  • We keep all the events
    • Stored in time-ordered columns
• We defer the processing
  • Until we receive a potential sequence terminator
CDP - Example

• A
  and each B within 5 min from A
  and each C within 5 min from B
• C1 C2 C3 B1 B2 A1
Evaluation - CPU

Processing Time (microsec)
Multiple Selection Policy

Processing Time (microsec)
Single Selection Policy

Stream & Complex Event Processing - Discovering Existing Systems: T-Rex
Evaluation – CPU & GPU

Stream & Complex Event Processing - Discovering Existing Systems: T-Rex
Performance

![Graph showing performance of different selection methods]

- AIP Multiple Selection
- CDP CPU Multiple Selection
- CDP GPU Multiple Selection
- AIP Single Selection
- CDP CPU Single Selection
- CDP GPU Single Selection

Processing Time (ms) vs Number of Rules
Performance

![Graph showing processing time vs number of rules for different selection methods.]

- AIP Multiple Selection
- CDP CPU Multiple Selection
- CDP GPU Multiple Selection
- AIP Single Selection
- CDP CPU Single Selection
- CDP GPU Single Selection
CDP – Lessons Learned

- Simple memory layout
  - Arrays
- No need to duplicate
- No need to perform useless computation
  - Invalid sequences are immediately discarded
- We can easily maintain multiple indexes over the elements
  - E.g., for parameterization
- Makes parallel processing possible
  - But convenient only when a lot of data is available
  - Can we automatically and dynamically choose the best architecture?
HANDS ON
Architecture

Virtual Machine

- TRex Server
  - Uses
    - TRex Library
    - Processing Engine (CDP)

- TRex Client (Library + Example)
How to Get T-Rex?

- [https://www.dropbox.com/sh/8p3ahgaguhcuhp8/u3u0jZ66rM](https://www.dropbox.com/sh/8p3ahgaguhcuhp8/u3u0jZ66rM)

- **Includes**
  - T-Rex Library
  - T-Rex Server
  - Virtual Machine Image
    - Server already configured
  - T-Rex Client Library (Java)
    - Command line client
How to Define a Rule?

• We want to create a new rule “Rule1”
• Create a new Rule1 class
  • In “Source/test/” directory
  • Both .hpp and .cpp files
• Include the Rule1.hpp file in file /Source/test.hpp
• Install the rule to the engine in Main.cpp
  • In method addBootstrapTestRules()
    Rule1 rule;
    engine.processRulePkt(rule.buildRule());
• Invoke engine.finalize() after all rules have been installed
Current Limitations

- No rule parser
  - Need to write rules in the server
  - Tutorial in the next slides
  - Examples in the code
Define Composite_ID();
From Simple_ID (Val > 0)

Constraint constr[1];
strcpy(constr[0].name, "Val");
// We also have FLOAT, STRING, and BOOL
constr[0].type = INT;
// We also have LT (Less Than), EQ (Equal), DF (Different From)
// For strings we have EQ, DF, IN (Includes)
constr[0].op = GT;
// Use the correct type! intVal, floatVal, stringVal, boolVal
constr[0].intVal = 0;

RulePkt* rule= new RulePkt(false);
// Constr is the array of constraints, 1 is its size
rule->addRootPredicate(Simple_ID, constr, 1);

CompositeEventTemplate* templ= new CompositeEventTemplate(Composite_ID);
rule->setCompositeEventTemplateTemplate(templ);
Sequences

Define Composite_ID()
From Simple_ID (Val > 0) and
Last Simple_ID_2() within 5 min from Simple_ID and
Each Simple_ID_3() within 10 min from Simple_ID_2

Constraint constr[1];
strcpy(constr[0].name, "Val");
constr[0].type = INT;
constr[0].op = GT;
constr[0].intVal = 0;

RulePkt* rule = new RulePkt(false);
// This is the first predicate added: predicate 0
rule->addRootPredicate(Simple_ID, constr, 1);
// We do not pass any constraint (2nd and 3rd argument are constraint array and size)
// 4th argument is the id of the reference predicate
// We have “from Simple_ID”, whose id is 0
// The id of this new predicate is 1 (we increase every time)
rule->addPredicate(Simple_ID_2, NULL, 0, 0, 5*60, LAST_WITHIN);
// We have “from Simple_ID_2”, whose constraint is 1
rule->addPredicate(Simple_ID_3, NULL, 0, 1, 10*60, EACH_WITHIN);

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID);
rule->setCompositeEventTemplate(templ);
Sequences

Define Composite_ID()
From Simple_ID (Val > 0) and Last Simple_ID_2() within 5 min from Simple_ID and Each Simple_ID_3() within 10 min from Simple_ID

Constraint constr[1];
strcpy(constr[0].name, "Val");
constr[0].type = INT;
constr[0].op = GT;
constr[0].intVal = 0;

RulePkt* rule = new RulePkt(false);
rule->addRootPredicate(Simple_ID, constr, 1);
rule->addPredicate(Simple_ID_2, NULL, 0, 0, 5*60, LAST_WITHIN);
rule->addPredicate(Simple_ID_3, NULL, 0, 0, 10*60, EACH_WITHIN);

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID);
rule->setCompositeEventTemplate(templ);
Parameters

Define

Composite_ID()

From Simple_ID (Val > 0, Par_1 = $a) and
Last Simple_ID_2(Par_2 = $a) within 5 min from Simple_ID

Constraint constr[1];
strcpy(constr[0].name, "Val");
constr[0].type = INT;
constr[0].op = GT;
constr[0].intval = 0;

RulePkt* rule= new RulePkt(false);
rule->addRootPredicate(Simple_ID, constr, 1);
rule->addPredicate(Simple_ID_2, NULL, 0, 0, 5*60, LAST_WITHIN);
// Parameter between state 0 with name Par_1
// and state 1 with name Par_2
rule->addParameterBetweenStates(0, "Par_1", 1, "Par_2");

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID);
rule->setCompositeEventTemplateTemplate(templ);
Negations

Define Composite_ID()
From Simple_ID (Val > 0) and Last Simple_ID_2() within 5 min from Simple_ID and Not Simple_ID_3() between Simple_ID and Simple_ID_2

Constraint constr[1];
strcpy(constr[0].name, "Val");
constr[0].type = INT;
constr[0].op = GT;
constr[0].intval = 0;

RulePkt* rule = new RulePkt(false);
rule->addRootPredicate(Simple_ID, constr, 1);
rule->addPredicate(Simple_ID_2, NULL, 0, 0, 5*60, LAST_WITHIN);
// Negation has no constraints (2nd and 3rd parameter)
// Negation is between states (predicates) 0 (Simple_ID) and 1 (Simple_ID_2)
rule->addNegationBetweenStates(Simple_ID_3, NULL, 0, 0, 1);

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID);
rule->setCompositeEventTemplate(templ);
Negations

Define Composite_ID() From Simple_ID (Val > 0) and Last Simple_ID_2() within 5 min from Simple_ID and Not Simple_ID_3() within 3 min from Simple_ID

Constraint constr[1];
strcpy(constr[0].name, "Val");
constr[0].type = INT;
constr[0].op = GT;
constr[0].intVal = 0;

RulePkt* rule = new RulePkt(false);
rule->addRootPredicate(Simple_ID, constr, 1);
rule->addPredicate(Simple_ID_2, NULL, 0, 0, 5*60, LAST_WITHIN);
// Negation has no constraints (2nd and 3rd parameter)
// Negation refers to state (predicate) 0 (Simple_ID): 4th parameter
// 5th parameter is the time window
rule->addTimeBasedNegation(Simple_ID_3, NULL, 0, 0, 3*60);

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID);
rule->setCompositeEventTemplate(templ);
Negations and Parameters

Define
From
Simple_ID (Val > 0) and
Last Simple_ID_2(Par_2 = $a) within 5 min from Simple_ID and
Not Simple_ID_3(Par_3 = $a) within 3 min from Simple_ID

Constraint constr[1];
strcpy(constr[0].name, "Val");
constr[0].type = INT;
constr[0].op = GT;
constr[0].intval = 0;

RulePkt* rule = new RulePkt(false);
rule->addRootPredicate(Simple_ID, constr, 1);
rule->addPredicate(Simple_ID_2, NULL, 0, 0, 5*60, LAST_WITHIN);
rule->addTimeBasedNegation(Simple_ID_3, NULL, 0, 0, 3*60);
// Parameter between state with id 1 for attribute "Par_2"
// and negation with id 0 (first negation defined) for attribute "Par_3"
rule->addParameterForNegation(1, "Par_2", 0, "Par_3");

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID);
rule->setCompositeEventTemplate(templ);
Composite Event Definition

Define Composite_ID(Result)
From Simple_ID (Val > 0)
Where Result = Simple_ID.Val

Constraint constr[1];
strcpy(constr[0].name, "Val");
constr[0].type = INT;
constr[0].op = GT;
constr[0].intVal = 0;

RulePkt* rule = new RulePkt(false);
rule->addRootPredicate(Simple_ID, constr, 1);
// The value is an integer, that refers to state 0 (Simple_ID) and attribute Val
OpTree* idTree = new OpTree(new RulePktValueReference(0, "Val"), INT);
// This value defines the attribute “Result”
templ->addAttribute("Result", idTree);

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID);
rule->setCompositeEventTemplate(templ);
Aggregates

Define Composite_ID(Result)
From Simple_ID [Val > 0]
Where Result = Avg(Simple_ID2().Val2) within 5 min from Simple_ID

Constraint constr[1];
strcpy(constr[0].name, "Val");
constr[0].type = INT;
constr[0].op = GT;
constr[0].intValue = 0;

RulePkt* rule = new RulePkt(false);
rule->addRootPredicate(Simple_ID, constr, 1);
// The aggregate does not have any constraint (2nd and 3rd parameters)
// Its reference point is State 0 (Simple_ID) (4th parameter)
// It is defined for the Simple_ID2 for value “Val2” and is an average (AVG)
rule->addTimeBasedAggregate(Simple_ID2, 0, NULL, 0, 5*60, "Val2", AVG);
// The value of Result refers to the value of the first define aggregate (Id = 0)
OpTree* idTree = new OpTree(new RulePktValueReference(0), INT);
templ->addAttribute("Result", idTree);

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID);
rule->setCompositeEventTemplate(templ);
Recursion

Define Composite_ID1()
From Simple_ID()

Define Composite_ID2()
From Composite_ID1()

-----------------------------------

RULE 1
-----------------------------------

rule->addRootPredicate(Simple_ID, NULL, Ø);

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID1);
rule->setCompositeEventTemplate(templ);

-----------------------------------

RULE 2
-----------------------------------

rule->addRootPredicate(Composite_ID1, NULL, Ø);

CompositeEventTemplate* templ = new CompositeEventTemplate(Composite_ID2);
rule->setCompositeEventTemplate(templ);
Talking with T-Rex

• How to communicate with T-Rex for publishing and subscribing to events?
  • Using the command line example
  • Writing your own Java program
    • Using the primitives offered by the T-Rex Client Library
Using the T-Rex Client Library

• Three steps
  • Create a connection manager
  • Send a subscription
  • Send a publication
Using the T-Rex Client Library

```java
// Create a TransportManager
TransportManager tManager = new TransportManager(true);

// Connect to the server
String serverHost = "127.0.0.1;"
int serverPort = 9000;
tManager.connect(serverHost, serverPort);

// Create a subscription for event 10
int subEventType = 10;
SubPacket sub = new SubPacket(subEventType);
// Add one or more constraints (e.g., Name > 10 and Value < 5)
sub.addConstraint(new Constraint("Name", ConstraintOp.GT, 10));
sub.addConstraint(new Constraint("Value", ConstraintOp.LT, 5));
// Send the subscription
tManager.send(sub);

// Create a publication for event 20
// with attributed (Attr1 = 10; Attr2 = "Hello World")
int pubEventType = 20;
PubPacket pub = new PubPacket(pubType);
pub.addAttribute(new Attribute("Attr1", 10));
pub.addAttribute(new Attribute("Attr2", "Hello World"));
// Send the publication
tManager.send(pub);
```
Using the T-Rex Client Library

• How to receive events for my subscriptions?
  • Implement the PacketListener interface
  • Implement two methods
    • notifyPktReceived(Packet pkt)
    • notifyConnectionError()
Using the T-Rex Client Library

• In the “main class”
  • Add the listener to the TransportManager
  • Start the TransportManager

```java
// In the "main" class
MyListener listener = new MyListener();
tManager.addPacketListener(listener);
tManager.start();
```