CRUCIBLE: Towards Unified Secure On- and Off-Line Analytics at Scale

DISCS 2013

Peter Coetzee & Stephen Jarvis

Performance Computing and Visualisation,

University of Warwick

{P.L.Coetzee,S.A.Jarvis} @warwick.ac.uk
The Problem
The Problem

Data Science

- Timely Insight
- High-Value Information from High-Volume Data
- Volume; Variety; Velocity
- Purpose-Built Technology Stacks
  - Prototypes (R, Matlab, Python)
  - Online / Streaming (InfoSphere Streams, S4, Storm)
  - Offline / Batch (Data Cubes, MapReduce)
The Problem
The Problem

Can we justify writing *and maintaining* the same analytic three times?

How can we be sure that the implementations are equivalent?
CRUCIBLE: Approach
Progress towards a solution by providing:

- DSL + IDE Tooling targeting a range of environments
- Cell-Level Security Framework
- Runtime environments:
  - Online (InfoSphere Streams)
  - Offline (Hadoop + Accumulo)
  - Standalone (Local Mode)
process Filter extends SuperType {

    // Compile-time configuration
    config : int N = 1500000

    // Run-time mutable state
    state : {
        int foo = 0
        boolean bar = true
    }

    // 0..n output ports
    outputs : [FirstOutput, SecondOutput]

    // 0..n input streams
    input : Source.OutputName -> {
        // Processing Body...
    }
}
CRUCIBLE: DSL

- Compiles to Idiomatic Java (not ByteCode)
- Fully integrated with Java Type System
- Eclipse Integration
CRUCIBLE: DSL
CRUCIBLE: Standard Library
CRUCIBLE: Standard Library

Standard library of components

- Base PEs (e.g., Data Sources and Sinks, JSON load, XPath, Bloom Filters etc.)
- Shared State and Atomic Operations (in-JVM for local state or using ZooKeeper)
- Easily extended with domain-specific primitives: Can be implemented in any JVM language, including CRUCIBLE
CRUCIBLE: Security Labelling
CRUCIBLE: Security Labelling

Novel Cell-Level Security Label Management

- Assistance for security-conscious analytic developer. NOT fully hardened!
- Visibility defined by boolean label expression
  » “admin | superuser”
  » “(emea & sales) | management”
- Automatically expand labels based on usage in PE
  » “foo” + “bar” -> “foo & bar”
CRUCIBLE: Security Labelling
input : Source.OutputName -> { 
  val x = someInput + someState 
  MyOutput.emit('result' -> x) 
}
CRUCIBLE: Security Labelling

input : Source.OutputName -> {
    val x = someInput + someState
    MyOutput.emit('result' -> x)
}

Compiles to (abbreviated):

protected int $someState = 0;
protected final SecurityLabel someState$label = new SecurityLabel();

public void receive$Source$OutputName(final SecurityLabel someInput$label, final Integer someInput) {
    final int x = someInput + $someState;
    final SecurityLabel x$label = new SecurityLabel();
    x$label.expand(someInput$label);
    x$label.expand(this.someState$label);

    Pair<String,Integer> _mappedTo = Pair.<String, Integer>of("result", x);
    this.MyOutput.emit(Pair.<SecurityLabel, Object>of(x$label, _mappedTo));
}
CRUCIBLE: Security Labelling

```java
input : Source.OutputName -> {
    val x = someInput + someState
    MyOutput.emit('result' -> x)
}
```

Compiles to (abbreviated):

```java
protected int $someState = 0;
protected final SecurityLabel someState$label = new SecurityLabel();

class $Source$OutputName {
    public void receive(SourceOutputName, final SecurityLabel someInput$label, final Integer someInput) {
        final int x = someInput + $someState;
        final SecurityLabel x$label = new SecurityLabel();
        x$label.expand(someInput$label);
        x$label.expand(this.someState$label);

        Pair<String,Integer> _mappedTo = Pair.<String, Integer>of("result", x);
        this.MyOutput.emit(Pair.<SecurityLabel, Object>of(x$label, _mappedTo));
    }
}
```
input : Source.OutputName -> {
    val x = someInput + someState
    MyOutput.emit('result' -> x)
}

Compiles to (abbreviated):

```java
protected int $someState = 0;
protected final SecurityLabel someState$label = new SecurityLabel();

public void receive$Source$OutputName(final SecurityLabel someInput$label, final Integer someInput) {
    final int x = someInput + $someState;
    final SecurityLabel x$label = new SecurityLabel();
    x$label.expand(someInput$label);
    x$label.expand(this.someState$label);

    Pair<String,Integer> _mappedTo = Pair.<String, Integer>of("result", x);
    this.MyOutput.emit(Pair.<SecurityLabel, Object>of(x$label, _mappedTo));
}
```
CRUCIBLE: Security Labelling

```
input : Source.OutputName -> {
    val x = someInput + someState
    MyOutput.emit('result' -> x)
}
```

Compiles to (abbreviated):

```
protected int $someState = 0;
protected final SecurityLabel someState$label = new SecurityLabel();

public void receive$Source$OutputName(final SecurityLabel someInput$label, final Integer someInput) {
    final int x = someInput + $someState;
    final SecurityLabel x$label = new SecurityLabel();
    x$label.expand(someInput$label);
    x$label.expand(this.someState$label);

    Pair<String,Integer> _mappedTo = Pair.<String, Integer>of("result", x);
    this.MyOutput.emit(Pair.<SecurityLabel, Object>of(x$label, _mappedTo));
}
```
CRUCIBLE: Security Labelling
input : Source.OutputName -> {
  val x = someInput
  x.label += 'admin'
  x.label += someState.label
  MyOutput.emit('result' -> x)
}
CRUCIBLE: Security Labelling

input : Source.OutputName -> {
    val x = someInput
    x.label += 'admin'
    x.label += someState.label
    MyOutput.emit('result' -> x)
}

Compiles to (abbreviated):

protected int $someState = 0;
protected final SecurityLabel someState$label = new SecurityLabel();

public void receive$Source$OutputName(final SecurityLabel someInput$label, final Integer someInput) {
    final int x = someInput;
    final SecurityLabel x$label = new SecurityLabel();
    x$label.expand(someInput$label); // This expansion is automatic
    x$label.operator_add("admin"); // These two are manually applied
    x$label.operator_add(someState$label);
    Pair<String, Integer> _mappedTo = Pair.<String, Integer>of("result", x);
    this.MyOutput.emit(Pair.<SecurityLabel, Object>of(x$label, _mappedTo));
}
CRUCIBLE: Security Labelling

```java
input : Source.OutputName -> {
    val x = someInput
    x.label += 'admin'
    x.label += someState.label
    MyOutput.emit('result' -> x)
}
```

Compiles to (abbreviated):

```java
protected int $someState = 0;
protected final SecurityLabel someState$label = new SecurityLabel();

public void receive$Source$OutputName(final SecurityLabel someInput$label, final Integer someInput) {
    final int x = someInput;
    final SecurityLabel x$label = new SecurityLabel();
    x$label.expand(someInput$label); // This expansion is automatic
    x$label.operator_add("admin"); // These two are manually applied
    x$label.operator_add(someState$label);
    Pair<String,Integer> _mappedTo = Pair.<String, Integer>of("result", x);
    this.MyOutput.emit(Pair.<SecurityLabel, Object>of(x$label, _mappedTo));
}
```
CRUCIBLE: Security Labelling

input : Source.OutputName -> {
  val x = someInput
  x.label += 'admin'
  x.label += someState.label
  MyOutput.emit('result' -> x)
}

Compiles to (abbreviated):

protected int $someState = 0;
protected final SecurityLabel someState$label = new SecurityLabel();

public void receive$Source$OutputName(final SecurityLabel someInput$label, final Integer someInput) {
  final int x = someInput;
  final SecurityLabel x$label = new SecurityLabel();
  x$label.expand(someInput$label); // This expansion is automatic
  x$label.operator_add("admin"); // These two are manually applied
  x$label.operator_add(someState$label);
  Pair<String,Integer> _mappedTo = Pair.<String, Integer>of("result", x);
  this.MyOutput.emit(Pair.<SecurityLabel, Object>of(x$label, _mappedTo));
}
CRUCIBLE: Security Labelling

input : Source.OutputName -> { 
  val x = someInput 
  x.label += 'admin'
  x.label += someState.label
  MyOutput.emit('result' -> x)
}

Compiles to (abbreviated):

protected int $someState = 0; 
protected final SecurityLabel someState$label = new SecurityLabel();

public void receive$Source$OutputName(final SecurityLabel someInput$label, final Integer someInput) {
    final int x = someInput;
    final SecurityLabel x$label = new SecurityLabel();
    x$label.expand(someInput$label); // This expansion is automatic
    x$label.operator_add("admin"); // These two are manually applied
    x$label.operator_add(someState$label);
    Pair<String,Integer> _mappedTo = Pair.<String, Integer>of("result", x);
    this.MyOutput.emit(Pair.<SecurityLabel, Object>of(x$label, _mappedTo));
}
CRUCIBLE: Security Labelling

In Summary:

– Cell-level boolean visibility expressions
  » Maintained automatically
  » With manual label expansion
– Applied uniformly across all runtimes
– Leverage native runtime support where available — CRUCIBLE integrates with wider system security context
CRUCIBLE: Runtimes
CRUCIBLE: Runtimes

Three key runtime environments:

- **Standalone**
  - Local mode, for development purposes

- **Online / Streaming**
  - IBM InfoSphere Streams

- **Offline / Batch**
  - Apache Accumulo
CRUCIBLE: Runtimes

Standalone Runtime

- Single JVM, one thread per PE
- Blocking queues
- Queues provide back-pressure to prevent thrashing
- Designed for local-mode development and testing of topologies
CRUCIBLE: Runtimes

Streaming Runtime

- Distributed real-time processing using InfoSphere Streams runtime

- Generate SPL topology representing CRUCIBLE PE interconnections, and CRUCIBLE containers to interface to Streams runtime

- Can manually tweak SPL to distribute load of a single PE across multiple instances
CRUCIBLE: Runtimes

Offline Runtime

- Process bulk data using Accumulo Iterators
- Push work to the tablet server, rather than pulling data to the client for processing
- An Iterator-Scanner pair per CRUCIBLE PE, with temporary tables in Accumulo to store intermediate data
- Native security label support ensures persisted data secure in wider security context
CRUCIBLE: Runtimes

In Summary:

– Execute identical Java code in multiple runtime containers

– Guarantee of equivalent message-passing semantics, and application of topology to data

– Extensible design permits integration with a variety of external platforms — e.g. Storm, HBase, Cassandra, or others as needed
CRUCIBLE: Performance
Caveat: Early performance results from un-optimised codebase
CRUCIBLE: Performance

Execution Time (ms)

Problem Size (Tuples)

- **Standalone**
- **Native Java**
- **SPL (Global)**
- **SPL (Local)**
- **Native SPL**
- **Accumulo (Global)**
- **Accumulo (Local)**
- **Native Accumulo**
CRUCIBLE: Summary
CRUCIBLE: Summary

Work on CRUCIBLE is ongoing. Thus far, we have demonstrated:

- High-level DSL and associated IDE Tooling
- Suite of runtime environments providing consistent execution semantics across on- and off-line data
- Consistent cross-runtime cell-level security labelling
CRUCIBLE: Summary

- Significant improvement in engineer productivity
  - Write analytics once
  - Run semantically equivalent analytics both on- and off-line
  - Reduce effort required to maintain cell-level security
- Imperfect performance, but scalability in line with native implementations
CRUCIBLE: Further Work
Next steps for CRUCIBLE development:

- Profile and optimise existing codebase
- Explore new runtime models to improve performance
- Language features to facilitate job reuse and subscription
- Expanded standard library
- IDE Debug tooling
- PE Fission & Fusion techniques
- Workload-based optimisation for alternative architectures
CRUCIBLE: Questions?

Many thanks for your attention

Please do get in touch:

{p.l.coetzee, s.a.jarvis}@warwick.ac.uk

CRUCIBLE is online, with downloads coming soon:

http://go.warwick.ac.uk/crucible