Native Actors: How to Scale Network Forensics

Matthias Vallentin*, Dominik Charousset†, Thomas C. Schmidt‡, Vern Paxson§, Matthias Wählisch‡
*UC Berkeley, †HAW Hamburg, ‡FU Berlin, §ICSI Berkeley

Problem Statement
Network forensics today suffers from:
• Huge amount of activity to store for later inspection
• Numerous different data formats
• Separate analysis procedures for past and future activity

 ⇒ Time-consuming and complex process

Goals
Need a better approach with the following properties [1]:
• Interactive work flow
  – Sub-second response times
  – Iterative query refinements
• Scalable in terms of data and compute
  – Handle distributed ingestion and at high rates
  – Asynchronous query execution
  – Graceful aggregation of older data
• Expressive and easy to learn
  – Represent activity in a unified data model
  – Same procedures to analyze past and future data

Requirements
To achieve these goals, we need a platform that is:
1. Distributed: scale with number of nodes
2. Reliable: fault isolation & local recovery
3. Type-safe: check protocols statically at compile time
4. Adaptive: dynamic provisioning & deployment

 ⇒ Ideal fit for the actor model of computation

• Actor: primitive for parallel computations
• Network-transparent message passing
• Actors can dynamically spawn more actors

CAF offers building blocks meeting these requirements.

Use Cases

• Incident response
  – Goal: identify scope of security breach
  – Begins with a piece of intelligence
  – Ad-hoc, interactive analysis style
  – Concrete start, then widen scope

• Network troubleshooting
  – Goal: find root cause of failure
  – Only symptoms visible
  – Start broadly, then narrow scope

• Combating insider abuse
  – Goal: uncover policy violations
  – Attack: chain of authorized actions
  – Analysis style: “connect the dots”
  ⇒ Relate temporally distant events

CAF: C++ Actor Framework

Overview
A framework for building high-performance concurrent applications and distributing systems at scale [2]:

• Lightweight actor implementation
  – Actors have only a few hundred bytes overhead
  – Spawn millions of actors without performance penalty

• Type-safe messaging interfaces, checked at compile time
  – Actor protocol verified during development
  – No type errors at runtime, even in distributed scenarios

• Adaptive platform for heterogeneous systems
  – Actors can run on different nodes using different OSes
  – Actors can run on GPUs via OpenCL bindings

• Dynamic and extensible
  – Enables developers to deploy actors at runtime
  – Configurable scheduling to match application needs

Minimal Example

A simple example using a CAF actor to add two numbers:

```cpp
using server = typed_actor<replies_to<int,int>::with<int>>;
server::behavior_type adder() { return
  [](int a, int b) {
    return a + b;
  });
}

void run(server &)
  spawned_actor myself;
  myself->sync_send(x, 40, 2).await(?
                              [int result] {?
                                cout << x + 2 == result << endl;
                              }
                        [myself->send_exit(x, exit_reason::user_shutdown);
                        
    int main()
      run(spawn_typed(adder));
      await_all_actors_done();
      shutdown();
```