Towards Correct and Efficient Program Execution in Decentralized Networks: Programming Languages, Semantics, and Resource Management

Karl Palmskog
palmskog@kth.se
KTH Royal Institute of Technology

2014-10-24
Problem: Scalable Program Execution

> $10^7$ objects

> $10^6$ nodes
Problem: Scalable Program Execution

> $10^7$ objects

> $10^6$ nodes
Vision: A Decentralized Program Execution Platform

Desirable properties:
- scalable
- abstract
- rigorous
- implementable
Application: Platform-as-a-Service Cloud Building Block

- End User
- Application
- Platform
- Infrastructure
- Cloud Provider
- Cloud User

PaaS
Towards Correct and Efficient Program Execution in Decentralized Networks

Introduction

Execution of Message Passing Programs in Networks
Three Problems

1. correspondence with network-oblivious executions
2. adapt executions to meet efficiency objectives
3. conversion of threaded programs to message-passing model
Contributions

Paper I: Location Independent Routing in Process Network Overlays

Paper II: Efficient and Fully Abstract Routing of Futures in Object Network Overlays

Paper III: ABS-NET: Fully Decentralized Runtime Adaptation for Distributed Objects

Paper IV: Decentralized Adaptive Power Control for Process Networks

Paper V: Dynamic Probabilistic Inference of Atomic Sets
## Contributions

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>theory</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>practice</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>correspondence</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>adaptability</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>conversion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
## Adaptability Mechanisms

<table>
<thead>
<tr>
<th></th>
<th>Paper I</th>
<th>Paper II</th>
<th>Paper III</th>
<th>Paper IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>object mobility</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>node startup/shutdown</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>message routing</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>task scheduling</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>message scheduling</td>
<td></td>
<td></td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>buffering</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>synchrony</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>add/remove links</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Objectives

<table>
<thead>
<tr>
<th></th>
<th>Paper I</th>
<th>Paper II</th>
<th>Paper III</th>
<th>Paper IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>correspondence</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>load balancing</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>network right-sizing</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>minimal messaging</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>low stretch</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>self-stabilization</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>compactness</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>confidentiality</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Abstract Behavioral Specification (ABS) Language

ABS Properties

- formal modeling language for concurrent, distributed systems
- based on active objects
- separate level for immutable data and pure functions
- objects accessed only through interfaces
The Abstract Behavioral Specification (ABS) Language

ABS Properties

- formal modeling language for concurrent, distributed systems
- based on active objects
- separate level for immutable data and pure functions
- objects accessed only through interfaces

Our Use of ABS

Tool for case studies, used while working in EU FP7 HATS project
Location Independent Routing in Process Network Overlays & Efficient and Fully Abstract Routing of Futures in Object Network Overlays

Adaptability Mechanisms:
- object mobility
- routing

Objectives:
- correspondence with network-oblivious behavior
- low stretch
- self-stabilization
1. defines fragments of ABS (µABS and mABS)
2. defines standard network-oblivious program behavior
3. introduces location independent routing for network-aware program behavior
4. proves *contextual equivalence* of the behaviors
Location Transparency in the Message Passing Model

Location Database

$o_0 \leftrightarrow u_0, o_1 \leftrightarrow u_1$

TCP/IP
Location Transparency in the Message Passing Model

Location Database

\[ o_0 \mapsto u_0, \; o_1 \mapsto u_1 \]
Location Transparency in the Message Passing Model

Location Database

\( \mathcal{O}_0 \leftrightarrow u_0, \mathcal{O}_1 \leftrightarrow u_3 \)

\( u_0 \)

TCP/IP

\( u_1 \)
Location Independent Routing

- routing tables with next hops at each node
- routing information exchange between neighbors
- up-to-date tables give optimal routes (stretch 1)
Location Independent Routing, continued

- routing tables with next hops at each node
- routing information exchange between neighbors
- up-to-date tables give optimal routes (stretch 1)
Location Independent Routing, continued

- routing tables with next hops at each node
- routing information exchange between neighbors
- up-to-date tables give optimal routes (stretch 1)
Location Independent Routing, continued

- routing tables with next hops at each node
- routing information exchange between neighbors
- up-to-date tables give optimal routes (stretch 1)
Towards Correct and Efficient Program Execution in Decentralized Networks

Location Independent Routing, continued

- routing tables with next hops at each node
- routing information exchange between neighbors
- up-to-date tables give optimal routes (stretch 1)
Location Independent Routing, continued

- routing tables with next hops at each node
- routing information exchange between neighbors
- up-to-date tables give optimal routes (stretch 1)
μABS program

```java
class Server() {
    serve(from, x) {
        r = foo(x);
        from!response(r)
    }
}

class Client(arg) {
    use(server) {
        server!serve(self, arg)
    }
    response(y) {
        ext!output(y)
    }
}

{ server, client,
  server = new Server();
  client = new Client(42);
  client!use(server)
}
```
Standard Behavior

\[
\left\{ \begin{array}{c}
o_0 \rightarrow o_1 \\
t_0 \end{array} \right. \quad o_1 \text{!serve}(o_0, 42)
\]
Standard Behavior, continued

\[
\begin{align*}
\{ & \quad o_0 \rightarrow o_1 \\
& \quad t_0 \\
& \quad o_1 \\
& \quad o_1, \text{serve}, o_0;42 \
\}
\end{align*}
\]
Standard Behavior, continued

\[\{ o_0 \rightarrow o_1 \mid t_0, t_1 \mid o_0!\text{response}(15) \}\]
Standard Behavior, continued

\[
\{ o_0 \rightarrow o_1, t_0, t_1, o_0, \text{response}, 15 \}
\]
Network-Aware Behavior

\[ o_1!serve(o_0, 42) \]
Network-Aware Behavior, continued

\[ o_1, \text{serve}, o_0; 42 \]
Network-Aware Behavior, continued

\( u_0 \) \( o_0 \) \( t_0 \) \( u_1 \) \( o_1 \) \( t_1 \)

\( o_0!\text{response}(15) \)
Network-Aware Behavior, continued

\[ u_0, o_0, t_0 \rightarrow o_1, u_1, t_1 \]

\[ o_0, \text{response}, 15 \]
Contextual Equivalence

**Definition**

An observation, or **barb**, $obs$ is a method call of the form $ext!method(v_1, \ldots, v_n)$.

**Definition**

Two well-formed runtime configurations are **contextually equivalent** if they are in a relation that:

- is closed under reductions
- is closed under context applications
- preserves barbs
Correspondence Result

Theorem

A well-formed network-oblivious runtime configuration is contextually equivalent to its network-aware counterpart.
ABS-NET: Fully Decentralized Runtime Adaptation for Distributed Objects

Adaptability Mechanisms:

- object mobility
- routing
- task scheduling & message scheduling
- buffering
- synchrony

Objectives:

- load balancing
- minimal messaging
- low stretch
- self-stabilization
Paper III Summary

1. defines network-aware behavior of full Core ABS
2. simulates behavior using TCP/IP on network topologies
3. measures progress towards objectives for strategies
Load Balancing and Minimal Messaging
Load Balancing and Minimal Messaging, continued
for each active object \( o \) do

\( u' \) is a neighbour chosen uniformly at random

\( l \) is the current load

\( l' \) is the last known load of \( u' \)

if \( l > l' + 1 \) then

send \( o \) to \( u' \) with probability \( 1 - l'/l \)
for each active object $o$ do
  $u'$ is a neighbour chosen uniformly at random
  $l$ is the current load
  $l'$ is the last known load of $u'$
  if $l > l'$ then
    send $o$ to $u'$ with probability $1 - l'/l$
Star Program

- Communication only between fringe and center objects
- Fringes and their centers need to be on nodes at close distance
Towards Correct and Efficient Program Execution in Decentralized Networks

Star Program on Grid

- Weighted neighbour’s load, msg intensity
- Berenbrink et al. w/ neutral, msg intensity
- Berenbrink et al.

k transitions vs. messages sent avg.
Star Program Object Distance Distribution

- **Berenbrink et al.**
- **Berenbrink et al. w/ msg intensity**

The graph shows the distribution of total distance for different numbers of objects. The x-axis represents the total distance, while the y-axis represents the number of objects.
Paper IV

Decentralized Adaptive Power Control for Process Networks

Adaptability Mechanisms:

- **node startup/shutdown**
- object mobility
- routing

Objectives:

- **network right-sizing**
- load balancing
- minimal messaging
- low stretch
- self-stabilization
Paper IV Summary

1. defines protocol for controlled startup/shutdown of nodes
2. gives evidence that object behavior is preserved and progresses
3. outlines network right-sizing procedures
Shutdown Protocol

$u_0$

$u_2$  Prepare

$u_1$  Prepare

$u_3$  Prepare
Shutdown Protocol, continued

```
\[ u_0 \]

\[ u_2 \quad \text{Ready} \quad \text{Ready} \quad \text{Ready} \quad u_3 \]
```
Shutdown Protocol, continued

\[ u_0 \]

Object(\( o_1 \))  Object(\( o_0 \))

\[ u_2 \]  \[ u_1 \]  \[ u_3 \]
Shutdown Protocol, continued

Graph:
- $u_0$
- $u_2$, with label "Shutdown"
- $u_1$, with label "Shutdown"
- $u_3$, with label "Shutdown"
Shutdown Protocol, continued

$u_0$  

$u_2$  

Ack

$u_1$  

Ack

$u_3$  

Ack
Shutdown Protocol, continued

$u_2$

$u_1$

$u_3$
Node Shutdown State Machine

start → IDLE → TRANSACT → CLEAR → SHUTDOWN

start → ABORT
Shutdown Protocol Correctness

**Safety**
Whenever a node shuts down:
- no object is located on the node
- no object-related messages are incoming/outgoing

**Liveness**
If a node attempts to shut down, it will eventually shut down.
Shutdown Protocol Correctness

Safety
Whenever a node shuts down:
- no object is located on the node
- no object-related messages are incoming/outgoing

Liveness
If a node attempts to shut down, it will eventually shut down.

Evidence
- bounded safety model verified in the Spin model checker
- transition system safety lemmas proved in Coq proof assistant
**Dynamic Probabilistic Inference of Atomic Sets**

one step of a conversion process taking shared-memory programs to the message-passing model
class ArrayList {
    int size;
    Object[] entries;

    Object get(int i) {
        synchronized(lock) {
            if (0 <= i && i < this.size) {
                return this.entries[i];
            } else {
                return null;
            }
        }
    }

    void addAll(ArrayList o) {
        synchronized(lock) {
            this.size += o.size;
        }
        /*... copy elements ...*/
    }
}
Annotated Java Class

class ArrayList {
    atomicset L;
    atomic(L) int size;
    atomic(L) Object[] entries;

    Object get(int i) {
        if (0 <= i && i < this.size) {
            return this.entries[i];
        } else {
            return null;
        }
    }
}

void addAll(unitfor(L) ArrayList o) {
    this.size += o.size;
    /*... copy elements ...*/
}
Elements of Data-Centric Synchronization: Atomic Sets, Units of Work, and Aliases

**Atomic Set**

Group of fields in a class connected by a consistency invariant

**Example**

In the `ArrayList`

- **Invariant**: `entries[i]` valid if $i < \text{size}$
- **Atomic set**
  
  \[ L = \{ \text{size}, \text{entries} \} \]
### Atomic Set

Group of fields in a class connected by a consistency invariant

### Unit of Work

Method that preserves the invariant when executed sequentially

### Example

*In the ArrayList*

- Instance methods are units of work for all atomic sets of the object
- `addAll(ArrayList)` is a unit of work for the other list’s atomic set $L$
## Elements of Data-Centric Synchronization: Atomic Sets, Units of Work, and Aliases

### Atomic Set

Group of fields in a class connected by a consistency invariant

### Unit of Work

Method that preserves the invariant when executed sequentially

### Alias

Combines atomic sets

### Example

- **Class** `DownloadManager` with field `urls`
- **Field declaration:**
  
  ```java
  atomic(U)
  ArrayList urls|L=this.U|;
  ```
- Atomic set `U` now contains `urls.size` and `urls.entries`
Towards Correct and Efficient Program Execution in Decentralized Networks

Converting a Program to Atomic Sets Requires Understanding its Concurrency Structure

- Data-centric synchronization looks beneficial
- But: must *understand* old synchronization to convert it

Conversion Experience of Dolby et al.:

- Takes several hours for rather simple programs
- 2 out of 6 programs lack synchronization of some classes
- 2 out of 6 programs accidentally introduced global locks
Converting a Program to Atomic Sets Requires Understanding its Concurrency Structure

- Data-centric synchronization looks beneficial
- But: must *understand* old synchronization to convert it

Conversion Experience of Dolby et al.:

- Takes several hours for rather simple programs
- 2 out of 6 programs lack synchronization of some classes
- 2 out of 6 programs accidentally introduced global locks

Our Algorithm

Avoid conversion errors by automatically determining annotations from program traces using Bayesian probabilistic inference
Actorization

- atomic sets and aliases define actor boundaries
- transform program to use interfaces and initialize actors
- actorized program executes in cloud with preserved behavior
Actorization

- atomic sets and aliases define actor boundaries
- transform program to use interfaces and initialize actors
- actorized program executes in cloud with preserved behavior
Actorization

- atomic sets and aliases define actor boundaries
- transform program to use interfaces and initialize actors
- actorized program executes in cloud with preserved behavior

Problem

Units of Work spanning multiple actors
Actorization Steps

- Program
  - Annotations
    - Annotated Program
    - Actorized Program
      - Program Using Actor Library
## Conclusions

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>theory</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>practice</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>correspondence</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>adaptability</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>conversion</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Environmental Conditions to be Addressed

- node crash failures
- node Byzantine failures
- link failures
- honest-but-curious adversaries
- active node-corrupting adversaries