Declarative Overlays
CIS 700-005
20 February, 2007
Boon Thu Loo1, Tyson Condie1, Joseph M. Hellerstein2, Peter Maniatis2, Timothy Roscoe2, Ion Stoica1
University of California at Berkeley, Total Research Berkeley

Slides stolen from Boon’s SOSP and Job Talk ppts

Review: Declarative Networking
- A declarative framework for networks:
  - Declarative language: “ask for what you want, not how to implement it”
  - Declarative specifications of networks, compiled to distributed dataflows
  - Runtime engine to execute distributed dataflows
- Observation: Recursive queries are a natural fit for routing

Review: Thursday’s Class
- Packet routing algorithms
- Datalog w/ network operators (@)
  - @ says where update written/where data fetched
- Packet routing algorithms as Datalog queries
- What about more interesting routing...

Overlays Everywhere...
- Overlay networks are widely used today:
  - Routing and forwarding component of large-scale distributed systems
  - Provide new functionality over existing infrastructure
- Many examples, variety of requirements:
  - Packet delivery: Multicast, RON
  - Content delivery: CDNs, P2P file sharing, DHTs
  - Enterprise systems: PM Exchange

Overlays networks are an integral part of many large-scale distributed systems.

Problem
- Non-trivial to design, build and deploy an overlay correctly:
  - Iterative design process:
    - Desired properties → Distributed algorithms and protocols ← Simulation → Deployment → Repeat...
    - Each iteration takes significant time and utilizes a variety of expertise

The Goal of P2
- Make overlay development more accessible:
  - Focus on algorithms and protocol design, not the implementation
- Tool for rapid prototyping of new overlays:
  - Specify overlay network at a high level
  - Automatically translate specification to protocol
  - Provide execution engine for protocol
- Aim for “good enough” performance
  - Focus on accelerating the iterative design process
  - Can always hard-tune implementation later
Traditional Router

Declarative Router

Declarative Overlays

Traditional Overlay Node

P2 Overlay Node

Advantages of the P2 Approach

- Declarative Query Language
  - Concise/high level expression
  - Statically checkable (termination, correctness)
  - Ease of modification
  - Unifying framework for introspection and implementation
  - Automatic optimizations
    - Query and dataflow level
Data Model

- Relational data: relational tables and tuples
- Two kinds of tables:
  - Stored, soft state:
    - E.g. `neighbor(Source, Dest)`, `forward(Source, Dest, NextHop)`
  - Transient streams:
    - Network messages: `message(Recv, Dest)`
    - Local timer-based events: `periodic(NodeID, 10)`

Dataflow framework

- Dataflow graph
  - C++ dataflow elements
- Similar to Click:
  - Flow elements (mux, demux, queues)
  - Network elements (cc, retry, rate limitation)
- In addition:
  - Relational operators (joins, selections, projections, aggregation)

Query Language: Overlog

- “SQL” equivalent for overlay networks
- Based on Datalog:
  - Declarative recursive query language
  - Well-suited for querying properties of graphs
  - Well-studied in database literature
    - Static analysis, optimization, etc.
- Extensions:
  - Data distribution, asynchronous messaging, periodic timers and state modification

Query Language: Overlog

**Datagram rule syntax:**

```
<read> <condition1>, <condition2>, ..., <conditionN>
```

**Overlog rule syntax:**

```
<Action> = <event>, <condition1>, ..., <conditionN>
```


Example: Ring Routing

- Each node has an address and an identifier
- Every node knows its successor
- Objects "served" by successor
- Each object has an identifier
Ring State

- Stored tables:
  - node(NAddr, N)
  - suc(NAddr, Succ, SAddr)

Example: Ring lookup

- Find the responsible node for a given key k?
  \( n.\text{lookup}(k) \)

  \( \text{if } k \in (n, n.\text{successor}) \)
  \( \text{return } n.\text{successor}.\text{addr} \)

  \( \text{else} \)
  \( \text{return } n.\text{successor}, \text{lookup}(k) \)

Ring Lookup Events

- Event streams:
  - lookup(Addr, Req, K)
  - response(Addr, Owner)

Pseudocode → Dataflow "Strands"

Pseudocode:

```
node(Addr, Succ, SAddr)
return n.successor.addr
```

Network in Dataflow

Local Table

```
\text{lookup}(Addr, Req, K)
```

```
\text{response}(Addr, Owner)
```

```
\text{node}(n, n.successor)
```

```
\text{return } n.\text{successor}.\text{addr}
```

```
\text{lookup}(k)
```

Dataflow Strand

```
Event: \text{Incoming network messages, periodic timers}
Condition: \text{Process event using strands and elements}
Action: \text{Outgoing network messages, local table updates}
```

Pseudocode → Strand 1

```
\text{node}(NAddr, N)
\text{suc}(NAddr, Succ, SAddr)
```

```
\text{if } k \in (N, N.\text{successor}) \)
\text{return } n.\text{successor}.\text{addr} \)
\text{else} \)
\text{return } n.\text{successor}.\text{lookup}(k) \)

Event: \text{SENT lookup(NAddr, Req, K)}
Condition: \text{node(NAddr, N) \& suc(NAddr, Succ, SAddr) \& \text{K} \in (N, Succ)}
Action: \text{SEND response(Req, K, SAddr) to Req}
Pseudocode to Strand 1

**Event:** RECEIVE lookup(RAddr, Req, K)
**Condition:** node(RAddr, K) < nodex(SAddr, Succ, SAddr)
**Action:** SEND response(Req, K, SAddr) to Req

Dataflow strand

1. Join R in Addr = R
2. Join R in Addr = R
3. Redirect to ONNX
4. Redirect R to ONNX
5. Redirect R to ONNX

Strand Execution

Network-Addr-Flow

lookup

lookup

lookup

lookup

lookup

Local Tables

Network-Addr-Flow

lookup

lookup

lookup

lookup

lookup

Local Tables

Actual Chord Lookup Dataflow

P2-Chord

- Chord Routing, including:
  - Multiple successors
  - Stabilization
  - Optimized finger maintenance
  - Failure recovery
- 47 Overload rules
- 13 table definitions
- Other examples:
  - Nameko, flooding, routing protocols

Performance Validation

- Experimental Setup:
  - 100 nodes on Emulab testbed
  - 500 P2-Chord nodes
- Main goals:
  - Validate expected network properties
Sanity Checks
- Logarithmic diameter and state ("correct")
- BW-efficient: 300 bytes/s/node

Benefits of P2
- Introspection with Queries
- Automatic optimizations
- Reconfigurable Transport (WIP)

Introspection with Queries
With Atul Singh (Rice) and Peter Druschel (MPI)
- Unifying framework for debugging and implementation
  - Same query language, same platform
- Execution tracing/logging
  - Rule and dataflow level
  - Log entries stored as tuples and queried
- Correctness invariants, regression tests as queries:
  - "Is the Chord ring well formed?" (3 rules)
  - "What is the network diameter?" (5 rules)
  - "Is Chord routing consistent?" (11 rules)

Automatic Optimizations
- Application of traditional Datalog optimizations to network routing protocol (SIGCOMM 2005)
- Multi-query sharing:
  - Common "subexpression" elimination
  - Caching and reuse of previously computed results
  - Opportunistic sharing message propagation across rules

Open Questions
- The role of rapid prototyping?
- How good is "good enough" performance for rapid prototypes?
- When do developers move from rapid prototypes to hand-crafted code?
- Can we get achieve "production quality" overlays from P2?
- Security (prob. need to specify extra constraints, easy?)
- Fairness of query -> dataflow compilation for comparison
Future Work

- "Right" language
- Formal data and query semantics
- Static analysis
  - Optimizations
  - Termination
  - Correctness