Cologne: A Declarative Distributed Constraint Optimization Platform

Changbin Liu
UPenn -> AT&T Labs Research

Joint work with Lu Ren (Bank of America), Boon Thau Loo (UPenn), Yun Mao (AT&T Labs Research), Prithwish Basu (Raytheon BBN Technologies)
Motivation

• **Distributed systems management**
  – Configure system parameters
  – Operators performance objectives
  – Constraints in the deployment environment
  – E.g., cloud resource orchestration, wireless network configuration
Motivation

Resource management $\Rightarrow$ constraint optimization problem (COP)

$$\begin{align*}
\text{Goal} & \quad \text{Variables (output)} \\
\min f(x) & \quad x = [x_1, x_2, \ldots, x_n] \\
\text{S.t.} & \quad g_i(x) \leq 0, i = 1, \ldots, p \\
\end{align*}$$

Operator objectives

System constraints

System states

Optimization goal and constraints

Automation

Management commands

Optimization output
Motivation

• Automation logic: complex

• Manual approach with imperative languages
  – Multi-thousand lines of code
  – Tedious, expensive, error-prone
  – One-time use, difficult to maintain and customize

• Distributed optimizations
  – Scalability
  – Autonomy: management constraints across administrative domains
    (e.g., federated cloud)
Declarative Policy Language: Colog

- **Declarative**: specify “what” instead of “how” to do it
- **Rule-based, high-level abstraction**
- **Compact**: orders of magnitude code size reduction
- **Based on Network Datalog (NDlog)**
  - A distributed variant of Datalog
  - Originate from declarative networking [CACM’09]
  - Distributed execution by a declarative query engine
- **Support distributed optimizations**
  - Integrate NDlog with constraint solving capability
  - Multiple local solvers coordinate with each other
  - Each local solver handles a portion of the whole problem, and they together achieve a global objective
Outline of the Talk

• General platform
  – **Cologne**: COnstraint LOGic engiNE
  – Code available at: netdb.cis.upenn.edu/coLOGne

• Two (very different) use cases
  – Cloud resource orchestration [SOCC’11, USENIX’12]
  – Wireless network configuration [COMSNETS’12]
Infrastructure-as-a-Service Cloud

- **IaaS Cloud**
  - Provide cloud infrastructure services: virtual machines (VMs), virtual block devices, VPNs
  - Widely adopted, e.g. Amazon Elastic Compute Cloud (EC2)

- **Cloud resource orchestration**
  - Provision, configure, manage, decommission
    - Compute: VM spawn / start / stop / migrate / destroy
    - Storage: replicate, deallocate
    - Network: allocate IP, create VPN
Automation

– Cloud services are increasingly sophisticated
  • Globally distributed data centers

– Provider: operational objectives
  • Load balancing, cost reduction

– Customer: service level agreements (SLAs)
  • Latency and bandwidth of web services

Manual: tedious, expensive, error-prone
COPE (Cloud Orchestration Policy Engine)

**Core idea**: cloud resource orchestration → constraint optimization problem

\[ \text{min } f(x) \]
\[ x = [x_1, x_2, \ldots, x_n] \]
\[ \text{S.t.} \]
\[ g_i(x) \leq 0, i = 1, \ldots, p \]

**Goal**

**Variables (output)**

**Constraints**

- **Provider operational objectives**
- **Customer SLAs**
- **System states**
- **Optimization goal and constraints**
- **COPE**
- **Orchestration commands**
- **Optimization output**
COPE Architecture

Deployment mode:
- Centralized
- Distributed (one for each data center)

Communication and Distributed optimization
- Autonomy
- Scalability
Colog Syntax

- **Network Datalog (NDlog)**
  - A distributed variant of Datalog used in declarative networking
  - Rule form: $p \leftarrow q_1, q_2, ..., q_n$.
  - Distributed execution by a declarative query engine
  - Use cases: routing, overlay networks

- **Two reserved keywords:** `goal` and `var`

- **Solver derivation rule:** $p \leftarrow q_1, q_2, ..., q_n$.
  - Derive new variables based on existing ones

- **Solver constraint rule:** $p \rightarrow q_1, q_2, ..., q_n$.
  - If $p$ is true, then the rule body must also be true
  - Restrict variable’s allowed values, limit the search space
Use Case: Centralized COPE

• Goal: load balancing within a data center
  – Via VM migration
  – One metric: minimize system-wide host CPUs variance

• Customizations:
  – Power reduction
  – Migration cost
  – Load consolidation

• Input: CPU and memory of VMs

• Constraints: resource availability

• Output: VM-to-host assignments ➔ migration commands
Use Case: Centralized COPE

**goal** minimize C in hostStdevCpu(C). // Goal
**var** assign(Vid,Hid,V) forall vm(Vid,Cpu,Mem), host(Hid,Cpu2,Mem2). // Variables

d1  hostCpu(Hid,SUM<C>) <- assign(Vid,Hid,V),
    vm(Vid,Cpu,Mem), C==V*Cpu. // Compute CPU standard deviation

d2  hostStdevCpu(STDEV<C>) <- host(Hid,Cpu,Mem),
    hostCpu(Hid,Cpu2), C==Cpu+Cpu2. // deviation

d3  assignCount(Vid,SUM<V>) <- assign(Vid,Hid,V). // Each VM is assigned
c1  assignCount(Vid,V) -> V==1. // to one and only one host

d4  hostMem(Hid,SUM<M>) <- assign(Vid,Hid,V),
    vm(Vid,Cpu,Mem), M==V*Mem. // Host memory constraint

c2  hostMem(Hid,Mem) -> hostMemThres(Hid,M), Mem<=M.

Translates to ~1000 lines of C++ code
Evaluation: Centralized COPE

- Cloud service: mini-EC2, with VM migration
- Workload derived from production traces in a large hosting company
  - VM spawn/start/stop, 15 hosts and 936 VMs
- Colog program runs every 10 minutes
Use Case: Distributed COPE

Follow-the-Sun cloud service

– When to migrate?
– Which and how many VMs to migrate?
– Cloud providers in different administrative domains?
Use Case: Distributed COPE

– **Goal**: minimize *total cost*, e.g. defined for:
  • Customers: communication (end-to-end latency)
  • Providers: operating + migration
– **Constraints**: data center capacity
– **Output**: VM migration decisions between data centers
Use Case: Distributed COPE

- **Distributed constraint optimization**
  - Two data centers negotiate VM migrations between them
  - Propagate decisions to neighbors for next-round computation

```
goal minimize C in aggCost(@X,C).                          // goal
var migVm(@X,Y,D,R) forall setLink(@X,Y), dc(@X,D).        // variables

    d1 aggCost(@X,C) <- aggCommCost(@X,C1),
        aggOpCost(@X,C2), aggMigCost(@X,C3),
        nborAggCommCost(@X,C4),
        nborAggOpCost(@X,C5), C==C1+C2+C3+C4+C5.         // cost definition

    d2 nborAggCommCost(@X,SUM<Cost>) <- link(@Y,X),       // distribution
        commCost(@Y,D,C), nborNextVm(@X,Y,D,R), Cost==R*C.
```

- Expose limited information  ➔  autonomy
- Approximate solution (divide-and-conquer)  ➔  scalability
Full Specification: Distributed COPE

Math formulations

Follow-the-Sun | Colog rules | Imperative (C++) |
--- | --- | --- |
Centralized | 16 | 1487 |
Distributed | 32 | 3112 |

100X reduction

Colog rules
Evaluation: Distributed COPE

- Data centers geographically distributed at different locations
- Random topology with an average network degree of 3
- Random cost: operating, communication, migration
- Simulated network environment in ns-3
- Solver execution time: $\leq 0.5$ seconds (average)
Outline of the Talk

• General platform
  – *Cologne*: COnstraint LOGic engiNE
  – Code available at: netdb.cis.upenn.edu/cologne

• Two (very different) use cases
  – Cloud resource orchestration [*SOCC’11, USENIX’12*
  – Wireless network configuration [*COMSNETS’12*]
Cologne overview

• **Integration**
  – **General constraint solver:** top-down goal-oriented constraint solving, Gecode (www.gecode.org)
  – **Declarative query engine:** bottom-up distributed Datalog evaluation, RapidNet declarative query engine (netdb.cis.upenn.edu/rapidnet)

• **Compilation:** Colog $\rightarrow$ C++ (Gecode and RapidNet)

• **Interaction**
  – Tables (data) are shared between RapidNet and Gecode
  – RapidNet invokes Gecode modules for solving: `eInvokeSolver`
Colog Compilation

- **Static analysis**
  - Regular attribute: fixed value
  - Solver attribute: variable, determined after optimization
  - Identify solver predicates (containing solver attributes) that may have dependencies on solver predicates
  - Further syntactically differentiate solver derivation and constraint rules (“->” vs “<-”)

- **Solver rules (derivation, constraint)**
  - Variable assignment
  - Aggregation (SUM, MAX)
  - Table: join, select, project

- **Three core techniques**
  - Solver derivation rule execution
  - Solver constraint rule execution
  - Distributed solving

Gecode solver constraints
RapidNet operations
Solver Rules Execution

d1  hostCpu(Hid, SUM<C>) <- assign(Vid, Hid, V),  // Derive host CPU by
vm(Vid, Cpu, Mem),  // summing VMs CPU
  C == V * Cpu.

1. Join: assign(Vid, Hid, V) ▶️ vm(Vid, Cpu, Mem)
2. Assignment: C == V * CPU ➔ solver constraint, derives a new
   solver attribute C
3. Aggregation: SUM<C> ➔ solver constraint, derives a new
   solver attribute S == SUM<C>
4. Projection: ➔ hostCpu(Hid, S)

c2  hostMem(Hid, Mem) -> hostMemThres(Hid, M), Mem <= M.

Selection: Mem <= M used to prune solver search space
Distributed Solving

- Distributed constraint solving
  - Localization rewrite

\[
\begin{align*}
d2 & \text{nborAggCommCost}(\mathbf{X}, \text{SUM}<\text{Cost}>) \leftarrow \text{link}(\mathbf{Y}, \mathbf{X}), \quad \text{// distributed solver} \\
& \text{commCost}(\mathbf{Y}, \mathbf{D}, \mathbf{C}), \text{nborNextVm}(\mathbf{X}, \mathbf{Y}, \mathbf{D}, \mathbf{R}), \text{Cost}==\mathbf{R}*\mathbf{C}.
\end{align*}
\]

\[
\begin{align*}
d21 & \text{tmp}(\mathbf{X}, \mathbf{Y}, \mathbf{D}, \mathbf{C}) \leftarrow \text{link}(\mathbf{Y}, \mathbf{X}), \text{commCost}(\mathbf{Y}, \mathbf{D}, \mathbf{C}). \quad \text{// distributed Datalog} \\
d22 & \text{nborAggCommCost}(\mathbf{X}, \text{SUM}<\text{Cost}>) \leftarrow \text{tmp}(\mathbf{X}, \mathbf{Y}, \mathbf{D}, \mathbf{C}), \text{nborNextVm}(\mathbf{X}, \mathbf{Y}, \mathbf{D}, \mathbf{R}), \text{Cost}==\mathbf{R}*\mathbf{C}.
\end{align*}
\]
Summary

- **Declarative constraint optimization platform**
  - Declarative query engine + general constraint solver
  - Colog declarative policy language
  - Distributed optimization
  - Use cases
    - Cloud resource orchestration
    - Wireless network configuration
Cologne: A Declarative Distributed Constraint Optimization Platform

Thank you!

Code available at: netdb.cis.upenn.edu/cologne