

# On the Declarativity of Declarative Networking

Yun Mao

AT&T Labs – Research

Oct, 2009

Big Sky, MT

# Declarative Languages are popular!

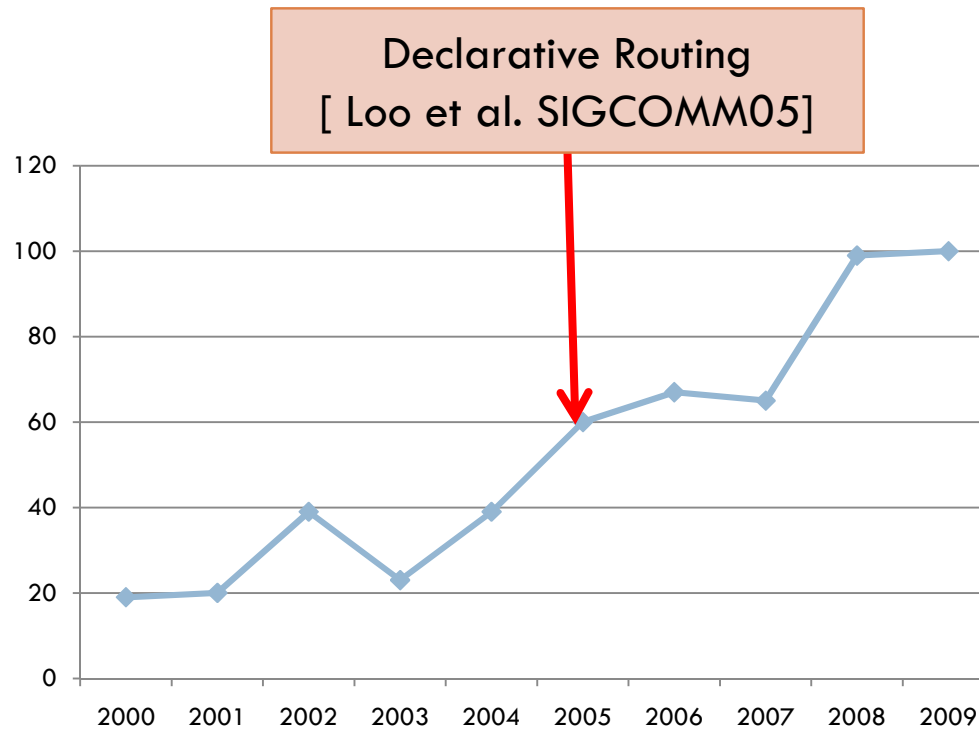
2

- More popular than you might think!
  - ▣ The most widely used (computer) language is a declarative language  
HTML
  - ▣ The most widely used programming language is still declarative  
Excel spreadsheet

# Declarative approach in networking/systems research

3

- “declarative and (network or networking or system)”
  - ▣ Search in all abstracts in ACM digital library



\*The number of 2009 is extrapolated.

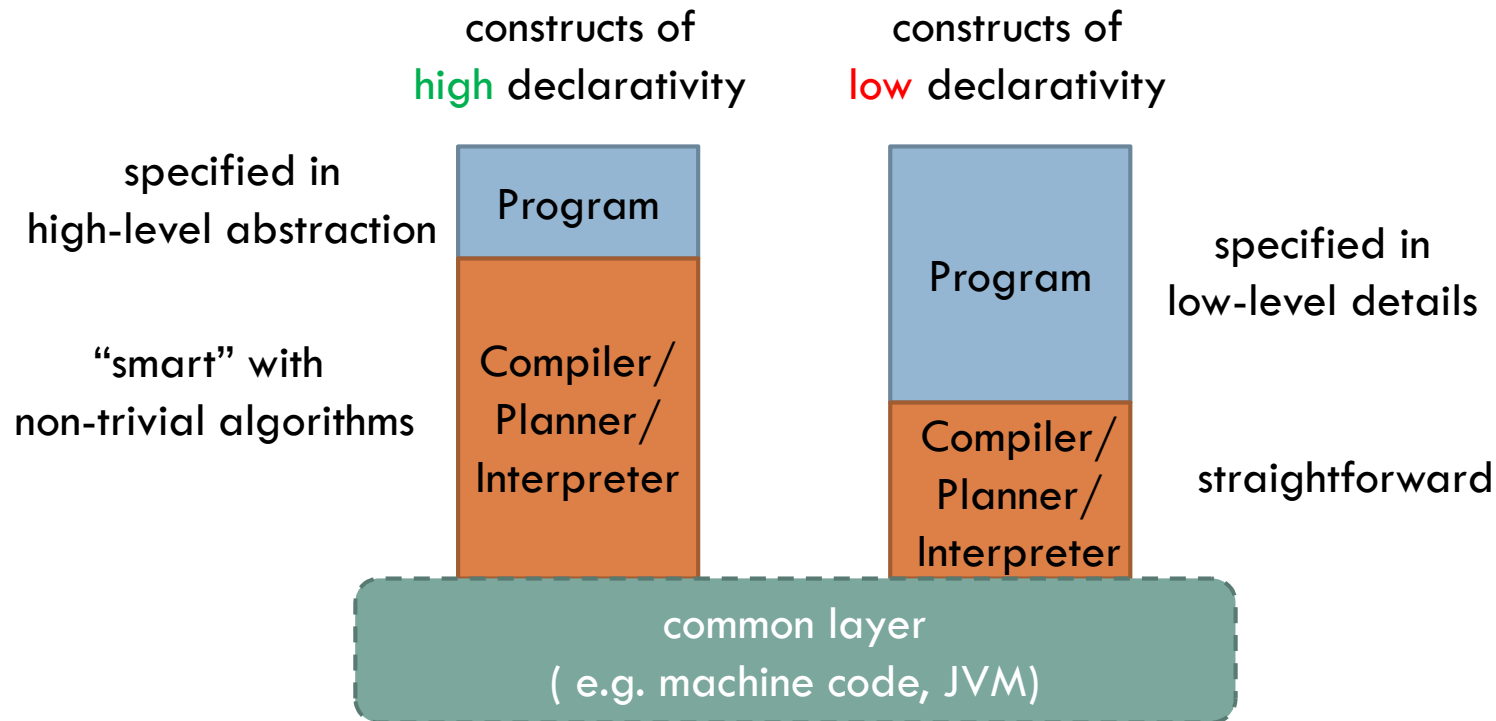
# What is the definition of “declarative”?

4

- By contrast: not imperative
  - ▣ But.. what is imperative?
- By intuition: specify “what”, not “how”
  - ▣ More of art than science
- By examples:
  - ▣ Imperative: C, C++, Java, C#, Perl, Python...
  - ▣ Declarative: Prolog, Datalog, Haskell, OCaml, SQL...
- Reality - many languages are hybrid.
  - ▣ Some constructs are declarative, some are not
  - ▣ Not exactly black vs. white
    - E.g. UPDATE in SQL

# What is “declarativity” in *this* talk

5



criteria: level of abstraction + “smartness” in compiler

# The rest of the talk

6

- Declarativity analysis in declarative networking
  - Focus: OverLog on P2 [Loo et al. SOSP 05; Condie et al. VLDB 08]
  - Result: The declarativity of declarative networking has been **decreasing!**
- Why?
- Does it matter?
- What should we do?



# OverLog rule classification

7

- Predicates: hard state tables vs. events
- Hard-state rules vs. event rules
  - H-H rule (hard-state rule)
    - $h() :- h1(), [h2(), \dots].$
  - H-E rule
    - $[delete] h() :- e() [, h1(), \dots].$
  - E-E rule
    - $e() :- e1() [, h1(), h2(), \dots].$
  - E-H rule
    - $e() :- h().$

# Rule Semantics

8

- Hard-state rules (H-H)

- Inherit the least model semantics from Datalog

$h() \Leftrightarrow h1(), [h2(), \dots]$ .

- Eventual consistency (if distributed)

- Liveness property

- Event rules (H-E, E-E, E-H)

- Event-condition-action

$h() :- e() [, h1(), h2(), \dots]$ .

$e() :- e1() [, h1(), h2(), \dots]$ .

- Take away: semantics of ECA rules and hard-state rules are very different

# Declarativity analysis

9

- High declarativity
  - Hard-state rules (H-H)
    - Abstraction: Datalog semantics
    - Compiler: distributed incremental view maintenance
- Low declarativity
  - ECA Rules
    - Abstraction: **not** Datalog semantics
    - Compiler: more mechanical in compiling ECA rules
  - Mixing H-H and H-E rules
    - r1 **h1**(X) :- h2(X).
    - r2 delete **h1**(X) :- e1(X).

# Case study

10

Protocol [venue & year]	total	H-H total/mixed/pure	E-E	H-E	E-H
Path vector [SIGCOMM 05]	4	4/0/4	0	0	0
Narada [SOSP 05]	17	0/0/0	7	9	1
Chord [SOSP 05]	46	2/2/0	20	22	2
Zyzyva [NSDI 08]	147	4/1/3	86	41	16
Coda [NSDI 09]	42	0/0/0	19	20	3

- ECA rules of low declarativity dominate in complex protocols

# Why?

11

- Language factor
  - ▣ Not expressive enough
- Compiler/Planner factor
  - ▣ Not smart enough
- Human factor
  - ▣ Do not think “declaratively”
  - ▣ Or, think ECA rules == hard-state rules in declarativity
- Implementation
  - ▣ Non-trivial algorithms → complex implementations → bugs, performance issues...

# Does declarativity matter?

12

- No.
  - ▣ As long as the language is handy, who cares?
  - ▣ Low declarativity → more controls by the programmer
- Yes!
  - ▣ An irreversible trend
    - Asm → C → Java ...
  - ▣ As researchers, we should look ahead by years!
    - Line between being wrong vs. early

# What could we do?

13

- Relatively easy ones
  - Language expressiveness: stratified negation
  - Implementation: more engineering effort
  - Human factor
    - syntactic salt to separate hard-state and ECA rules
    - Education!
- Hard one: smart compiler/planner
  - Planners in DBMS are already very complex
  - A holistic compiler for distributed systems?
    - Consistency, throughput, latency, power, etc.
  - “Programmer-assisted” compiler

# WIND (**W**ind **I**s **N**ot **D**atalog) language

14

- Separation between base tables and derived views
  - ▣ No more mixed H-H, H-E rules!
- ECA rules have different syntax
  - ▣ `on event()[,conditions] => action;`
  - ▣ `on insert/delete h()[,conditions] => action;`
- Virtual views
  - ▣ User-supplied query plan in ECA rules
- Specification
  - ▣ The liveness property in hard-state rules for verification
- Other features
  - ▣ Static typing, modularity, statically compiled code generation

# Open issues in readable syntax

15

- Not about declarativity but very important
  - ▣ Liskov in the Turing award lecture: “# read >> #write”
- Datalog syntax: sometimes too succinct
  - ▣ Implicit join, unintuitive aggregation
  - ▣ Tables with large # of attributes:  
h1(\_\_\_\_\_,X,\_\_\_\_\_,Y,\_\_\_\_\_)
- Hard-state tables are like global variables
- E-E rules are like distributed goto statement

# Conclusions and future directions

16

- Be aware of different levels of declarativities in declarative networking
  - ECA rules: compromise between declarativity and reality
  - WIND: an attempt in its very early age
- Future directions
  - Domain-specific planner as meta-programs
  - From conciseness to verification and parallelism
  - New domains (mobile, cloud) and new tasks (management)

# Thank you!

17

<https://mosaic.maoy.net>

Special thanks to Mary Fernandez, Divesh Srivastava, Trevor Jim, Pamela Zave, Wenchao Zhou, Changbin Liu, Mike Dahlin

# Chord

18

```
spec{
  succ(@NI, min<Dist>, SI) :-
    node(@NI, NodeKey),
    node(@SI, SuccKey),
    Dist := f_ring_dist(SuccKey, NodeKey),
    groupBy(NI);
}
```