ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Tackling network heterogeneity head-on

Timothy Roscoe

Networks and Operating Systems Group
ETH Zürich

Scene setting

- Different dimension of MAD networks:
  - Independent evolution
  - Arbitrary policies
  - Centralized standards
  - Tension between heterogeneity and interoperability
- Not addressing:
  - Cooperative / selfish behavior (but…)
  - Security (but see later)
  - Politics (beer conversation!)

Disclaimer

- I am going to talk about things I know nothing about!
- I am not going to present any results, or, indeed, work!
- I make no claims for originality here.
- I am not convinced this is a good idea (but I think it is!)
- Instead, I am “talking an idea for walk”
- Trying something out and looking for feedback…

Specific problem: resource heterogeneity in federated networked systems

- Scenario: GENI-like infrastructure (Combined utility computing and networking)
  - Resource providers:
    - Links
    - Processing
    - Storage
    - Etc.
  - Resource consumers:
    - Overlays
    - Applications

Specific problem: resource heterogeneity in federated networked systems

- How do domains and principals express:
  - What resources they have to ***offer***?
  - What their resource ***requirements*** are?
  - ***Commitments*** to provide resources?
- Old, hard problem:
  - Information is incomplete and may be restricted
  - Requirements are complex and span domains
  - Resources are heterogeneous and evolve over time

Specific example: current GENI design

- Current designs use an **`rspec`** for all three purposes
  - XML document giving resource values
  - Wildcards are used for resource requests/offers
  - Inherited from PlanetLab and Emulab
- Challenges:
  - Anticipating all hardware types & configurations
  - Expressing complex requirements as wildcards
  - Interpreting a hierarchical description of non-hierarchical concepts

Observation

- Much of the systems community fixated with static (even though extensible) formats:
  - IDLs, type systems, fixed schemas, etc.
- Separation between:
  - Schema / type defn (out of band, agreed in advance)
  - Concrete, semantics-free atomic values (in the msg)
- Result:
  - Clumsy specification of complex requests or offers

What should an advertisement look like?

- The kinds (*classes*) of object available
  - E.g. share of link, wavelength, NIC bandwidth, CPU share
- List of objects available (*instances)*
  - “Node with 4 processors”
  - “50 CHF for 1 hour of a full CPU”
  - “2 NICs, on the following links respectively”
- Constraints on allocation
  - “NIC bandwidth and buffer memory must be allocated together”
  - “No share less than 5%”
  - “Tradeoff between throughput and latency”
  - “By default you’re not worthy of more than best effort”

Exercise for the audience

- Do the same for:
  - Resource requests / requirements
  - Commitments for resources
- For extra credit:
  - Convince me that the right XML schema will surely solve all these problems…

Open vs. closed specifications

- Networking formats are *closed*
  - Expressing a new construct in the format requires new syntax or abuse of existing concepts
  - Example: BGP attributes
- Languages are *open* (to varying degrees)
  - Languages are generative of a range of descriptions
    - Sometimes called “constructors”
  - Extensions can often be expressed in the language itself

Why have networks always avoided languages?

- Historical reasons:
  - It was too complex at the time! (the 70s)
  - I needed something to work quickly!
- Prejudice:
  - What do these theoreticians know about networking?
- Fear:
  - I don't want to learn Prolog!
    - (or KL-ONE or FOL or PLANNER or BINDER or…)
- Practical considerations
  - My graduate students aren't logicians!

Back to the future: ANSAware (c.1988)

- Early DPE explicitly addressing *federation*
- ANSA Trading model specified:
  - Services described as arbitrary name/value pairs
  - Requests expressed in a *constraint language*
- Rich resource advertisement/negotiations
  - Across administrative boundaries
  - In the presence of incompose knowledge
- Mostly lost with SLP, WSDL, etc.

Various approaches (1)

- Databases and query languages
  - + Very fast evaluation
  - + Ready infrastructure
  - - No truth theory
  - - Not very expressive or generative
- Logic programming
  - + easy to distribute
  - +/- very expressive
  - +/- lots of hacks to go fast (& many implementations)

Various approaches (2)

- Constraint programming
  - + good match to problem space
  - - doesn't capture descriptive richness
  - - very computationally intensive
  - ? Parallelizable?
- Description logics
  - + highly predictable framework (proven results!)
  - + controllable tractability (but still complex!)
  - + work well with the web
  - - tools are complex (or are they?)

The good news: *Description Logics* is a whole *theory* of such languages

- View resource allocation as theorem proving?
- Solid theoretical foundation
- Evaluation tractability vs. Expressivity
  - Fundamental theoretical basis
- Efficient (well…) algorithms for evaluation
  - E.g. Tableaux algorithms
- Unification of logic programming, databases, knowledge representation…

Others are moving in this direction

- Web services:
  - RDF -> OWL (inc. OWL-DL)
- Cognitive Networks
  - DARPA project – use of DLs
- Security:
  - Authorization logics + theorem proving
- Information planes
  - Sophia: network as “logic soup” ☺
- Declarative networking
  - Integrates naturally with resource mgmt
  - Recent work on integrating access control languages

Concerns 1: Why bother?

- Getting distracted by ontologies!
  - I'm NOT going to mention the S****t*c W*b.
  - Metaphysics is a rathole – I'm a poststructuralist
- Isn't this overengineering?
  - Claim: we've reached the end of the road with tuples.
  - We are researchers: better to overshoot than undershoot
  - Problems with GENI and other systems are clear

Concerns 2: Avoiding intractability

- Restrict expressibility of language
  - Description logics: explicit tradeoff!
- Add procedural hints to guide evaluation
  - E.g cuts, evaluation order in Prolog
- Restrict evaluation semantics
  - C.f. databases
- We're systems people!
  - Fail and handle at a higher level
  - Hybrid systems: wire-in domain knowledge

Systems characteristics of the problem

- Datasets are small
  - Compare with database defns. of “large”
- Resource descriptions are rich, but not arbitrary
  - What language features are needed?
- Need to avoid DoS through complex expressions
  - See previous slide
- The problem is distributed
  - Need for distributed evaluation strategies
  - Possibly crossing administrative domains

Building a better white elephant?

- Resource frameworks have been built too many times before
- This time: concrete scenario
  - GENI resource federation
- Real implementation opportunity
  - Real code, real *users*
  - Applicable to other utility computing areas

(In)conclusion : I'd like to at least try:

- Using a richer logic to express resource:
  - Offers / Advertisements
  - Requests / Requirements
  - Commitments / Allocations
- Using DLs "in anger" to implement it all
  - Sound theoretical basis and framework
  - Subsequently wrecked by systems techniques
- Investigating "resource allocation as distributed theorem proving"
  - And the restrictions needed to make this work,

Thanks!

- Acknowledgements:
  - Katerina Argyraki (EPFL)
  - Mic Bowman (Intel)
  - Bryan Lyles (Telcordia)
  - Petros Maniatis (Intel)
  - John Wroclawski (ISI)