

MON: Management of Large Scale Systems via On-Demand Overlay Networks

Presented by

Klara Nahrstedt

klara@cs.uiuc.edu

UIUC

Joint work with Jin Liang and Indy Gupta

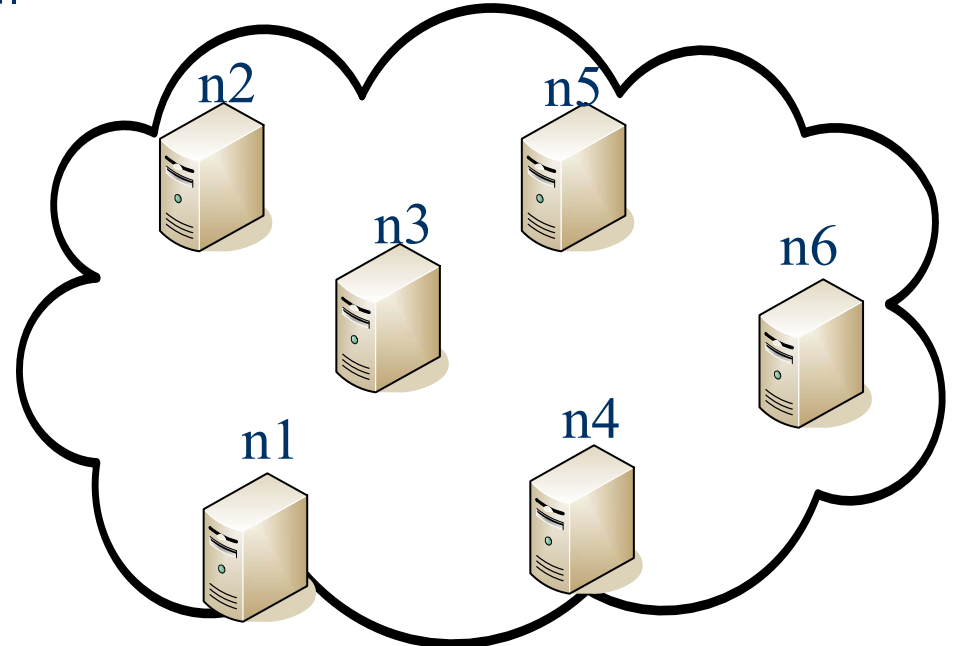
<http://cairo.cs.uiuc.edu/mon/>

Motivation

- Large dist. systems emerging
 - **Infrastructure**: PlanetLab, Grid, ...
 - **Apps services/experiments**: CDN, name, pub/sub...
- Difficult to manage
 - **Scale**: 100s or 1000s of nodes
 - **Failures**: node crash, route loop, disk full ...

Management Operations

- Instant status query
 - Avg CPU use (app runaway)?
 - Socket bind error (port conflict)?
- Software push

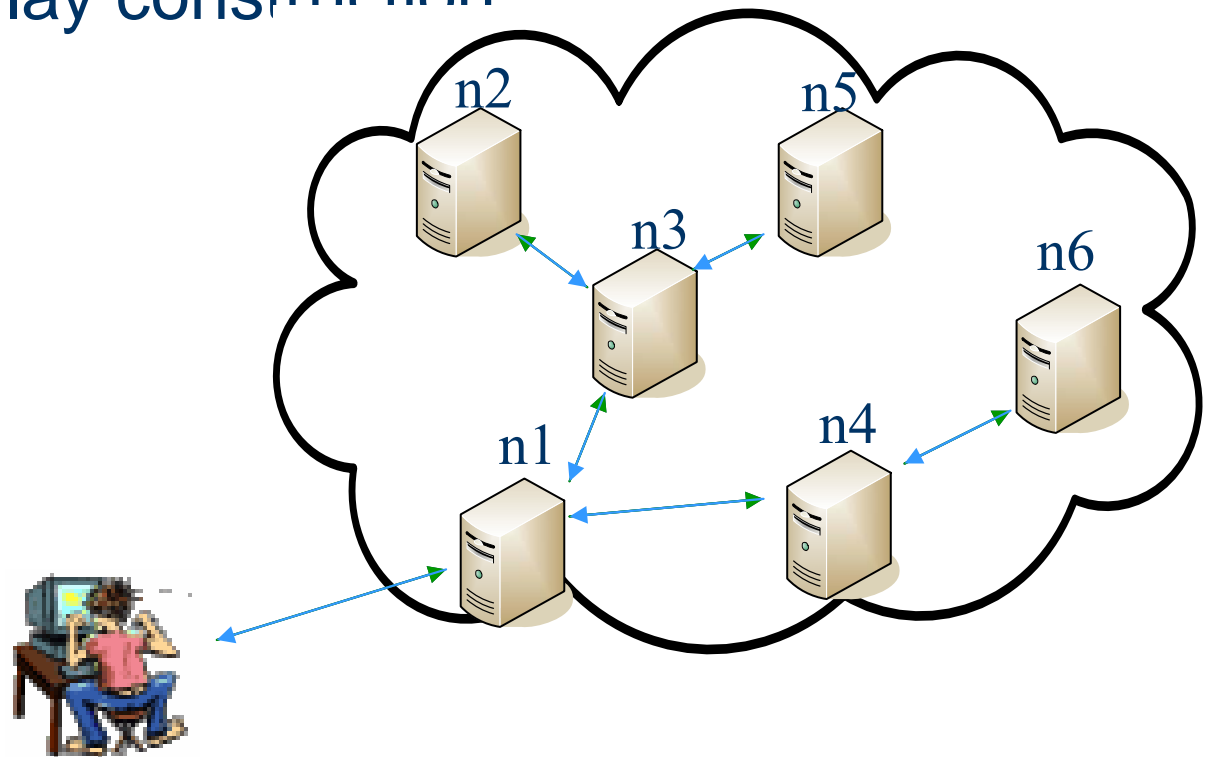


Existing Solutions

- **Centralized** (CoMon, GIS, scripts...)
 - **Efficient,**
 - Non-scalable, no in-network aggregation
- **Distributed but Persistent** (Astrolabe, Ganglia...)
 - **Scalable,**
 - Difficult to maintain, **complex failure repair**

MON: A New Approach

- Management Overlay Networks (MON)
 - Distributed management framework
 - On-demand overlay construction



On-Demand, Why?

- Simple
- Light-weight
- Better overlay performance
- *Suited to sys management*
 - Irregular/occasional usage
 - Short/medium term command execution

On-Demand, How?

Distributed System Management
Overlay Construction
Membership Management

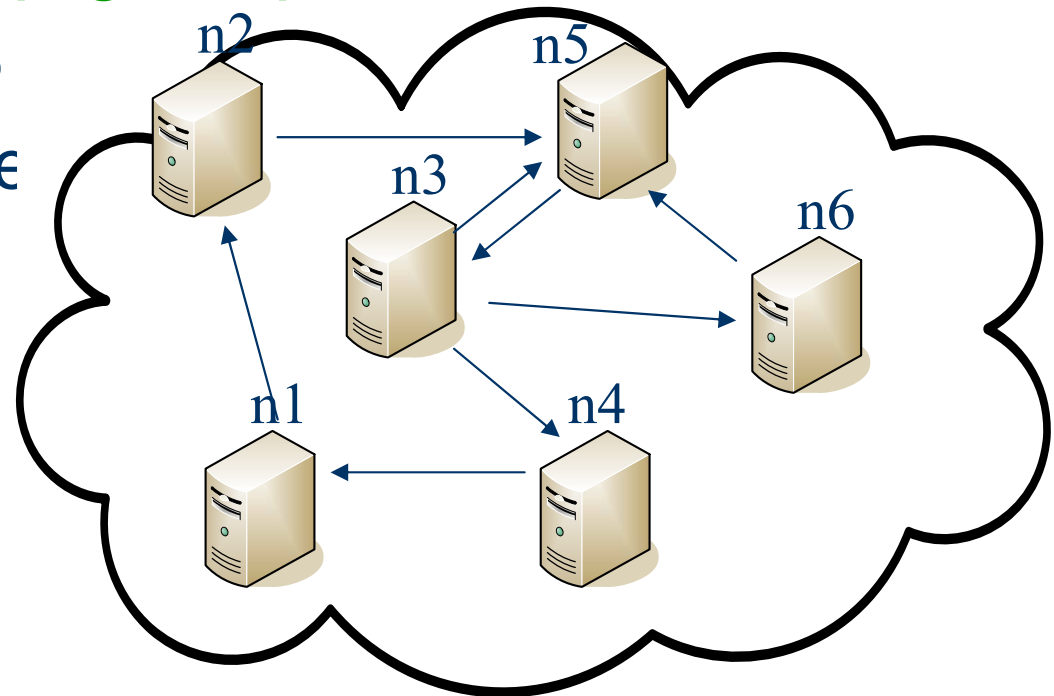
- Layered Architecture
 - Bottom: membership exchange
 - Middle: on-demand overlay construction
 - Top: cmd exec and results aggregation

Membership Layer

- Goal: connectivity, failure detection
- Design: membership gossip
 - Partial membership
 - Periodical exchange

Membership list at node n3

node	IP:Port	delay
n4	128.X.X.X:6020	9
n5	192.X.X.X:6020	2
n6	64.X.X.X:6020	3



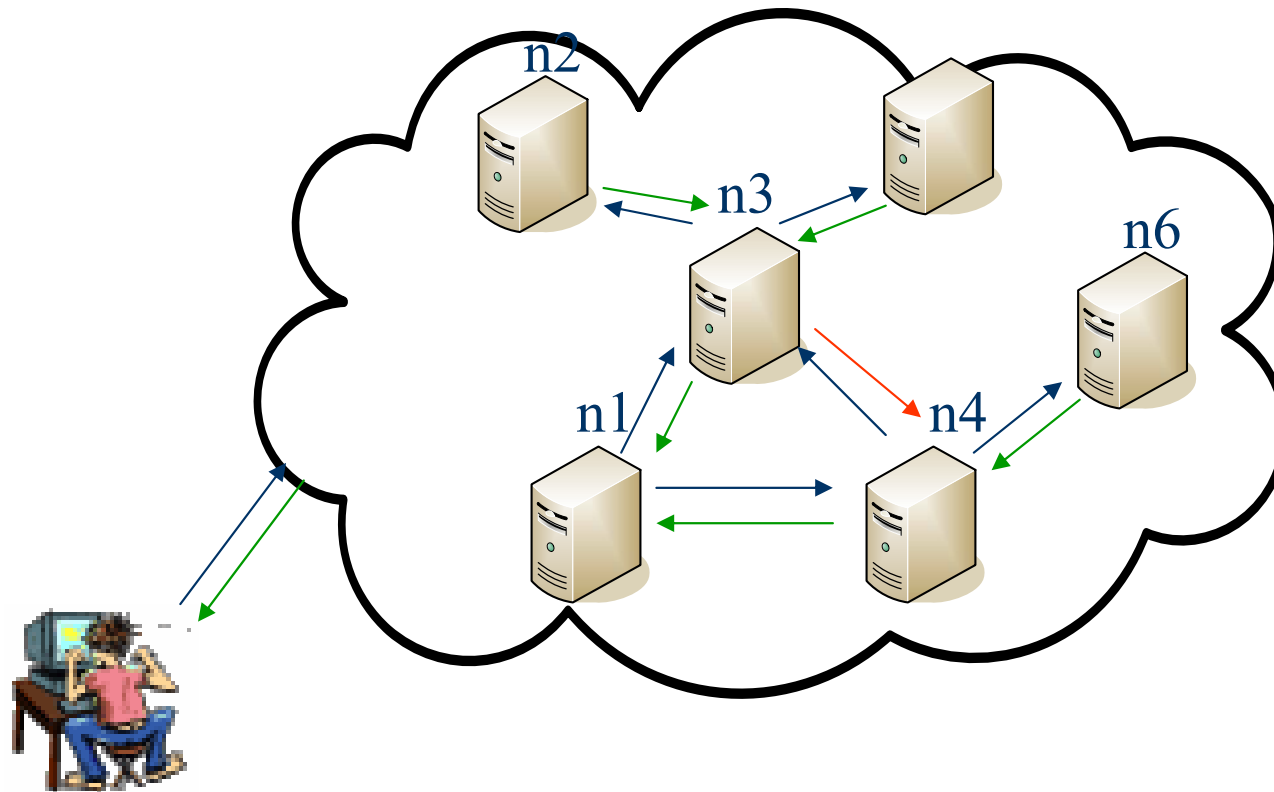
On-Demand Construction

- Problem:
 - Directed graph $G = (V, E)$
 - Create spanning subgraph (tree/DAG)
- Goal:
 - Efficient, quick construction
 - Ok with probabilistic coverage
 - $\Omega(E)$ messages for deterministic coverage

Randomized Algorithms

- Simple algorithm (**randk**)
 - Each node randomly selects k children
 - Each child acts recursively
- Improved Algorithm (**twostage**)
 - Membership augmented with **local list**
 - Two stages
 - First h hops: random selection
 - Thereafter: local selection
- DAG construction: similar to tree

Tree Building Example



Instant Status Query

- Aggregate Queries

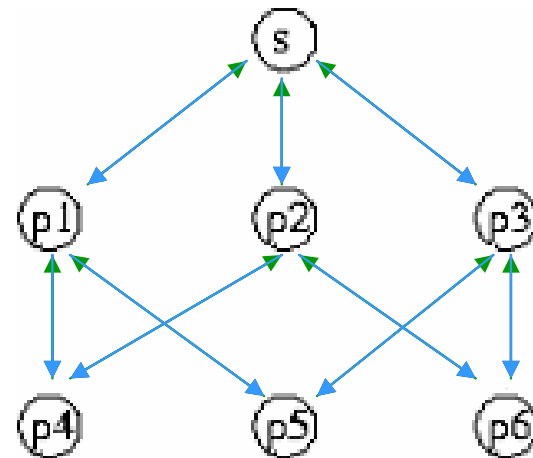
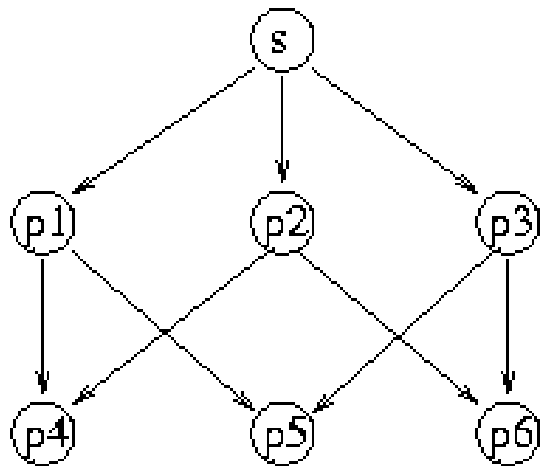
- `avg <resource>`
- `top <num> <resource>`
- `histo <resource>`

- Generic Filtering

- `filter <resource> \leq <value>`
- `filter grep <keyword> <file>`

Demonstration: <http://cairo.cs.uiuc.edu/mon>

Software Push



- Receiver-driven, multi-parent download
 - Parent **notify** availability,
 - Child **request** blocks
- DAG: better **path diversity** and **resilience**

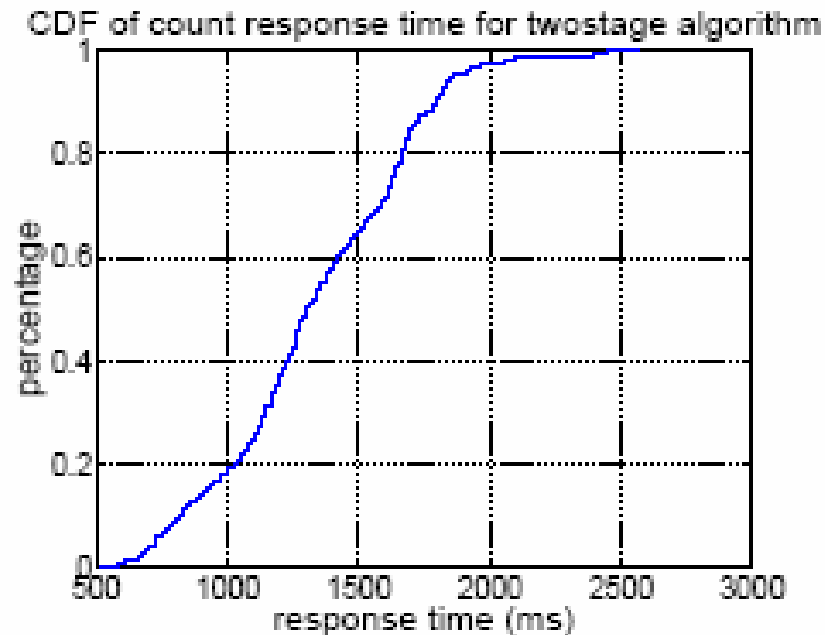
Tree Construction Performance

Table 1: Tree construction performance

	rand5	rand6	rand8	twostage
coverage	314.89	318.64	320.52	321.59
create time(ms)	3027.21	3035.46	2972.46	2792.03
count time(ms)	1539.19	1512.07	1369.92	1354.79

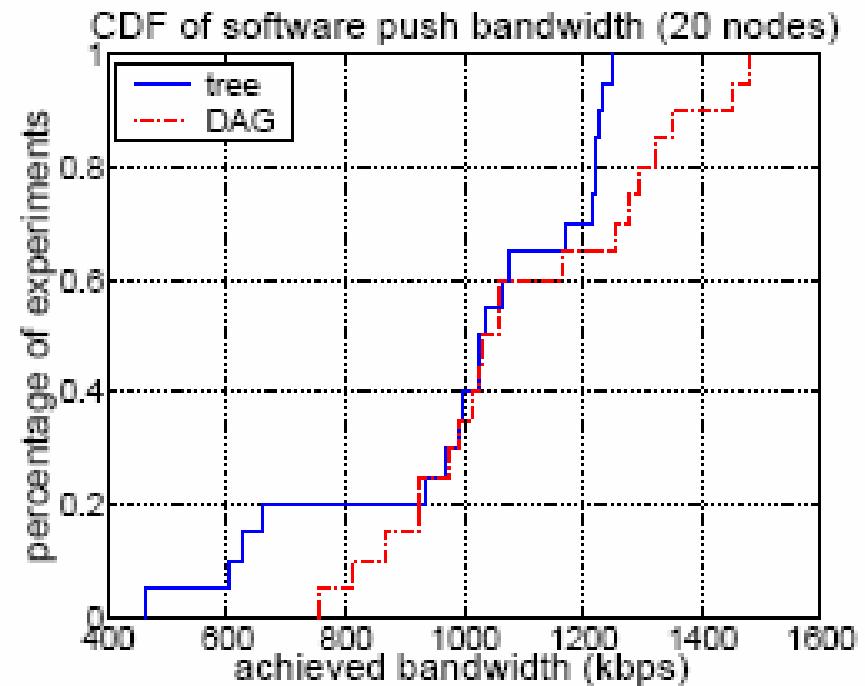
- Experiment on ~330 nodes, average of 200 trees
- Twostage has better **coverage** and **locality**

Tree Construction Performance



- CDF of count time of twostage
- Percentile 60th < 1.5s, 90th < 2s

Software Push Bandwidth



- Software push to 20 nodes
- DAG achieves larger bw than tree

Conclusion

- On-demand overlay
 - Alternative to long term maintainability
 - c.f. *software rejuvenation*
 - Suited to distributed system management
- MON: reasonable performance on PL
- Future work
 - Better construction algorithms
 - Better usability (**MON scripts!**)
 - Distributed log query!