CS 525
Advanced Distributed Systems
Spring 2010

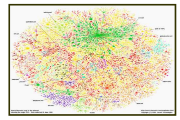
Indranil Gupta (Indy)

Distributed Monitoring

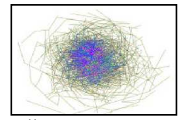
March 30, 2010

Acknowledgments: Steve Ko, Jin Liang, Ahmed Khurshid,Abdullah Al-Nayeem

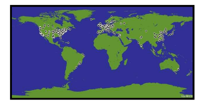
Distributed Applications run over Many Environments



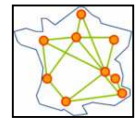
The Internet



Gnutella peer to peer system



PlanetLab



Grids, Datacenters, Cloud Computing

Distributed applications
- Large scale
  1000's of nodes
- Unreliable
  nodes and network
- Churned
  node join, leave, failure

Management and Monitoring of Distributed Applications

Monitoring of nodes, system-wide or per-node

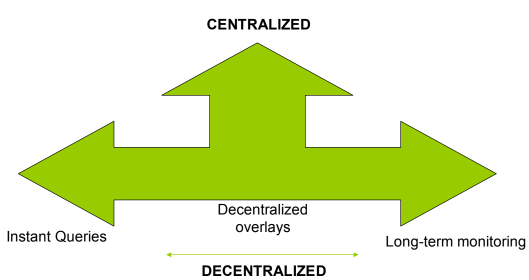
- Many applications can benefit from this, e.g.,
  - DNS, cooperative caching, CDN, streaming, etc., on PlanetLab
  - Web hosting in server farms, data centers, Grid computations, etc.
- A new and important problem direction for next decade
  - [CRA03], [NSF WG 05], [IBM, HP, Google, Amazon]
- Goal more end-to-end than cluster or network management
- Today typically constitutes 33% of TCO of distributed infrastructures
  - Will only get worse with consolidation of data centers, and expansion and use of PlanetLab

What do sysadmins want?

- Two types of Monitoring Problems:
  I. Instant (on-demand) Queries across node population, requiring <u>up-to-date answers</u>
     - {average, max, min, top-k, bottom-k, histogram etc.}
     - for {CPU util, RAM util, disk space util, app. characteristics, etc.}
       - E.g., max CPU, top-5 CPU, avg RAM, etc.
  II. Long-term Monitoring of node contribution
     - availability, bandwidth, computation power, disk space

Requirements:
- Low bandwidth
  - For instant queries, since infrequent
  - For long-term monitoring, since node investment
- Low memory and computation
- Scalability
- Addresses failures and churn
- Good performance and response

Existing Solutions: Bird's Eye View



Existing Monitoring Solutions

- Centralized/Infrastructure-based
- Decentralized

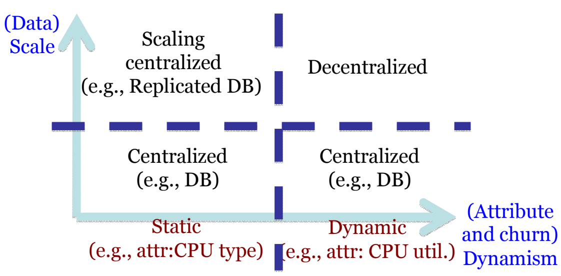
Existing Monitoring Solutions

- Centralized/Infrastructure-based: user scripts, CoMon, Tivoli, Condor, server+backend, etc.
  – Efficient and enable long-term monitoring
  1. Provide stale answers to instant queries (data collection throttled due to scale)
     – CoMON collection: 5 min intervals
     – HP OpenView: 6 hours to collect data from 6000 servers!
  2. Often require infrastructure to be maintained
  – No in-network aggregation
  – Could scale better
- Decentralized

Existing Monitoring Solutions

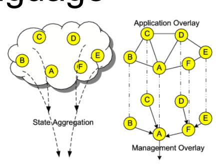
- Centralized/Infrastructure-based
- Decentralized: Astrolabe, Ganglia, SWORD, Plush, SDIMS, etc.
  – Nodes organized in an overlay graph, where nodes maintain neighbors according to overlay rules,
    - E.g., distributed hash tables (DHTs) – Pastry-based
    - E.g., hierarchy - Astrolabe
  – Can answer instant queries but need to be maintained all the time
    - Nodes spend resources on maintaining peers according to overlay rules => *Complex failure repair*
  – Churn => node needs to change its neighbors to satisfy overlay rules
  – Can you do a quick and dirty overlay, without maintenance?

Another Bird's Eye View



MON: Instant Queries for Distributed System Management

MON Query Language



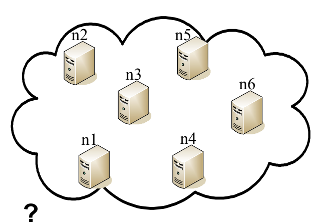
1. select **avg**(<resource>) [where <condition>]
2. select **top k** <resource> [where <condition>]
3. select **histo**(<resource>) [where <condition>]
4. select <resource list> [where <condition>]
5. select **grep**(<keyword>, <file>) [where <condition>]
6. select **run**(<shell command>) [where <condition>]
7. **count** and **depth**: number of nodes in, and tree-depth of, overlay
8. **push** <file>

- <resource> = either
  – system metric(s), e.g., CPU, RAM, disk, or
  – application-based metrics, e.g., average neighbor delay (p2p streaming), number of neighbors (partitionability of overlay), etc.
- <condition> = any boolean expression over the system resources
  – E.g., CPU > 50

MON: Management Overlay Networks

Supports
- Instant queries
  – Need instaneous answers
  – Inconsistency ok
- push software updates

- Basic Idea: Ephemeral overlays
  1. For each query, build an overlay on-demand
  2. Use overlay to complete query
  3. Do not maintain on-demand overlay



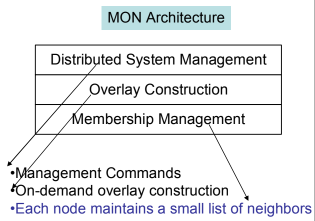
Why On-Demand?

Maintained overlays, e.g., DHT-based overlays

- maintenance bandwidth
- complex failure repair

On-demand approach is:

- Simple
- Light-weight
- *Suited to management*
  - Sporadic usage
  - Amenable to overlay reuse

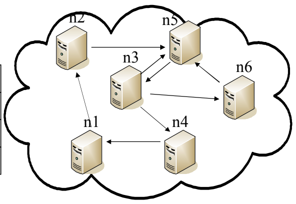


Membership by Gossip

- Partial membership list at each node (asymmetric)
- Contains random few other nodes; fixed in size (log(N))
- Periodic membership exchange [SCAMP01] [SWIM02] [TMAN04] [CYCLON06]
  - Measure delay
  - Detect failure: use *age* (last heard from time) to eliminate old entries - O(log(N)) time for spreading failure information [EGHKK03, DGHIL85]
- Weakly consistent – may contain stale entries

Membership list at node n3

node	IP:Port	RTT
n4	128.X.X.X:6020	90
n5	192.X.X.X:6020	20
n6	64.X.X.X:6020	30

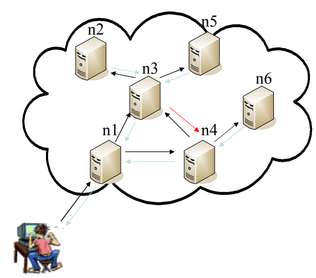


On-demand trees: Randomized Algorithms

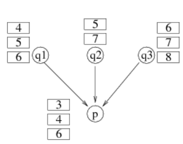
- Simple algorithm (randk)
  - Each node randomly selects *k* children from its membership list
  - Each child acts recursively
- Improved Algorithm (twostage)
  - Membership augmented with list of "nearby" nodes
    - Nearby nodes discovered via gossip
  - Two stage construction of tree
    - First *h* hops – select random children
    - After *h* hops – select local children
- DAG construction: similar to tree
- Weakly consistent membership list is ok
  - Retry prospective children
  - Or settle for fewer than *k* children

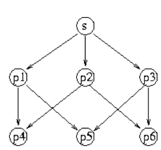
Tree Building Example

(k=2)



Software Push





- Receiver-driven, multi-parent download
- DAG structure helps: bandwidth, latency, failure-resilience

Tree Construction Performance

PlanetLab slice of 330 hosts

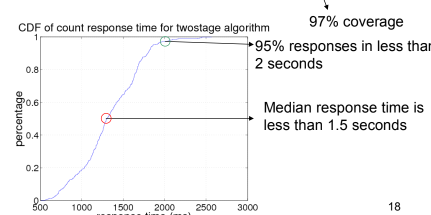
Table 1: Tree construction performance

	rand5	rand6	rand8	twostage
coverage	314.89	318.64	320.52	321.59

97% coverage

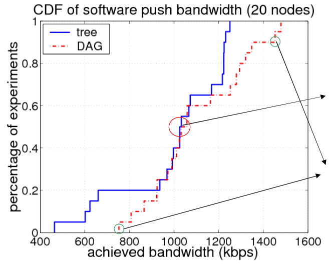
Bandwidth=10 Bps

10 s gossip interval
***k*=5 children**



Software Push Bandwidth

PlanetLab slice of 330 hosts

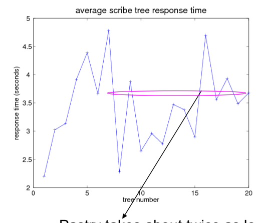


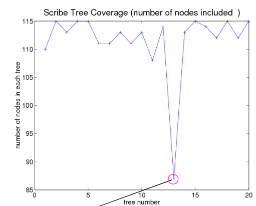
DAG median bandwidth about same as tree

But DAG faster overall due to replication

Comparison with DHT Tree

Scribe/SDIMS trees built over Pastry DHT. In a PlanetLab slice with 115 nodes.



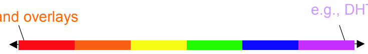


Pastry takes about twice as long to answer queries, does not ensure coverage when there are failures (or spends persistent bandwidth for routing table repair)

On-demand vs. Maintained Overlay

On-demand overlays

Maintained Overlays, e.g., DHT-based

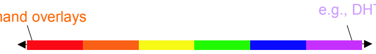


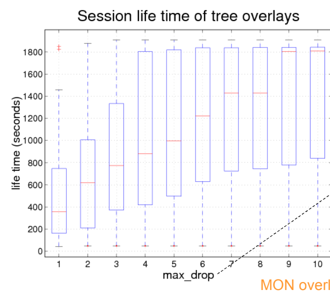
- Choice depends on Query Rate = *q* per second
- Maintained overlay
  - Neighbor Maintenance (up to date neighbors) = *B* Bps
  - Per-query cost for *k* child selection = *C* Bytes
  - Bandwidth = *B + q.C*
- On-demand approach
  - Weakly consistent neighbors = *B/m* Bps (*m > 1*)
  - Per-query cost = *m.C* Bytes
  - Bandwidth = *B/m + q.m.C*
- On-demand preferable when: *B/m + m.q.C < B + q.C*
  - *q < B/mC*
- In MON, *B* = 10 Bps, and C = 400 Bytes
- On-demand approach preferable for query rates of under one query every 40.4 seconds
  - Queries injected by users have think times of several minutes
  - Query rate far better than centralized collection tools which have periods of O(minutes)
  - Yet saves bandwidth

Spanning the Spectrum

On-demand overlays

Maintained Overlays, e.g., DHT-based





How long can an on-demand overlay survive without maintenance?

Function of max_drop= Maximum number of dropped nodes from overlay that application can tolerate *(kept track of by MON)*

MON overlays in PlanetLab (325 node slice) can be reused for up to 6-30 minutes

Discussion

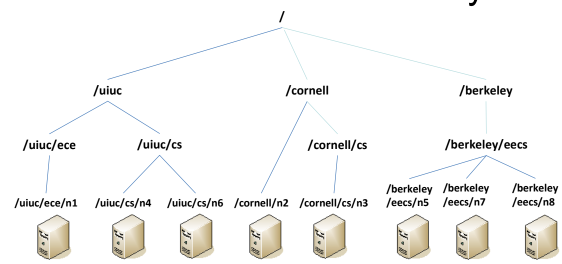
- Using partial DHTs to build better on-demand trees?
- On-demand DHTs?
- On-demand datastructures for anything?
- What about groups that do not span the entire system?

Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining

User Interface

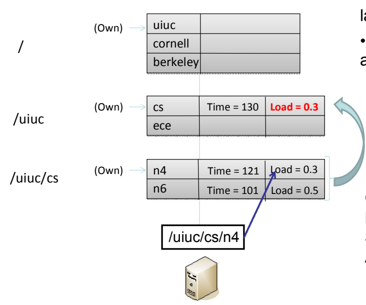
- Query Datacenter as a Database
  - SQL queries on the datacenter
- Each server contributes one or more tuples
- E.g.: grep for a file name
  - SELECT
    COUNT(*) AS file_count
    FROM files WHERE name = 'game.db'
- A "database" = A "Management Information Base" (MIB)
- Astrolabe = distributed MIB

Astrolabe Zone Hierarchy



- Zone ~ domain name
  - but customizable
- Zone hierarchy is determined by administrators
- Each host runs Astrolabe agent

Astrolabe = Decentralized MIB

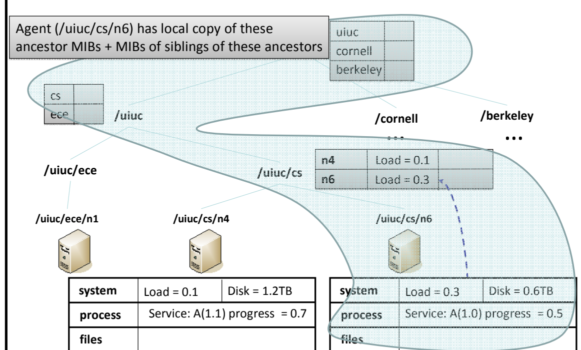


- Updates aggregated lazily up the tree
- Using a user-defined aggregation function

SELECT MIN(Load) as Load

Other aggregation funcs:
MAX (attribute)
SUM (attribute)
AVG (attribute)
FIRST(n, attribute)

Astrolabe's workings a little more complex



Gossip Protocol

- Conceptually:
  - Sibling zones gossip and exchange the MIBs of all their sibling zones
  - This propagates information upwards eventually
- Leaf zone: correspond to actual servers
- Internal node zone: collection of servers in that subtree zone
- In reality: Each agent, periodically, selects another agent at random, and exchanges information with it
  - If the two agents are in same zone, they exchange MIB information about that zone
  - If in different zones, they exchange MIB information about their least common ancestor zone
  - And then gossip for all ancestor zones

Gossip Protocol (continued)

- For efficiency, each zone elects a set of leader servers to act as representatives of that zone
  - Representatives participate in gossip protocol, then propagate information down to other servers in that zone (also via gossip)
  - Agent may be elected to represent multiple zones, but no more zones than its # ancestors
- How gossip happens at a representative agent:
  - Pick a zone (=ancestor) to gossip in
  - Pick a child of that ancestor
  - Randomly pick one of the contact agents from that zone
  - Gossip messages pertaining to MIBs of that zone, and all its ancestor zones (up to the root)
  - Gossip results in merge of entries, based on timestamps (timestamps assumed global)

Etcetera

- Eventual consistency guarantee for data: Each update eventually is propagated. If updates cease, everyone converges.
- AFC (aggregation function certificates): programmable aggregation functions; propagated throughout system
- Membership protocol: similar to gossip-style membership + ~ Bimodal Multicast
- Experimental results: simulations; see paper
- Astrolabe (or a variant of it) is rumored to be running inside Amazon's EC2/S3 cloud

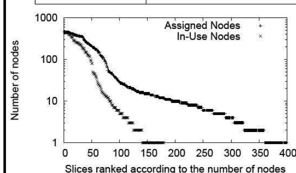
Discussion

- Non-leaf zones: have representative leaders vs. make all descendants responsible?
  - What are the tradeoffs?
- Up to date-ness of answers to queries?
- Truly on-demand querying system?
- Timestamps assumed to be global – why may this be ok/not ok?

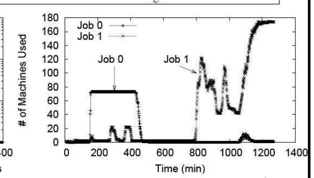
Moara: Flexible and Scalable Group-Based Querying System

Querying Groups of Nodes

Tasks	Queries
Resource Allocation	Average utilization for servers belonging to (i) floor F, (ii) cluster C, (iii) rack R
	Number of machines/VMs in a given cluster C
VM Migration	Average utilization of VMs running application X version 1 or version 2
	List of all VMs running application X and are VMWare based
Auditing/Security	Count of all VMs/machines running firewall
	Count of all VMs running ESX server and Sygate firewall
Dashboard	Max response time for Service X
	Count of all machines that are up and running Service X
Patch management	List of version numbers being used for service X
	Count of all machines that are in cluster C and running service X.version Y



PlanetLab slice sizes



HP's Utility Computing

Problem and Approach

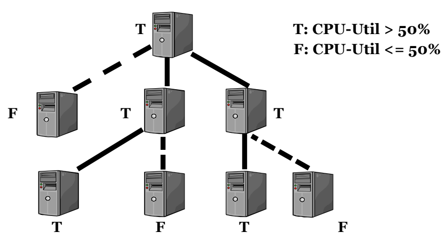
- Query groups of nodes
  - Groups are typically small
  - Groups are dynamic
- Groups are specified implicitly, via a predicate
  - E.g., ((CPU util < 10%) and (rack = R1)) or (Mem util < 500 MB and (rack=R2 or rack=R1))
  - That is, logical expressions: (A and B) or (C and (D or E))
    - Each expression is <attribute op value>
- One approach: flood query. Bad! Especially for repeated queries.
- Moara's approach:
  - Query a small set of servers that would be superset of the group (that is those matching the predicate)
  - For repeated queries, maintain overlay
  - Overlay = collection of trees. One tree per basic term (e.g., CPU util < 10%).
  - Optimize tree management cost vs. query cost

Query Rewriting

- Rewrite query into Conjunctive normal form (CNF)
  - Provably gives lowest number of terms
- Reduce #terms:
  - Use covers based on tree size to reduce terms
    - Cover (A and B) = min ((cover(A), cover(B), cover(A ∪ B))
  - Use semantic information
    - E.g., CPU < 10% is a subset of CPU < 20%

Example Moara Tree

Key idea: Build one tree per term (or reuse tree for that term if it already exists)



Moara's Tree Maintenance

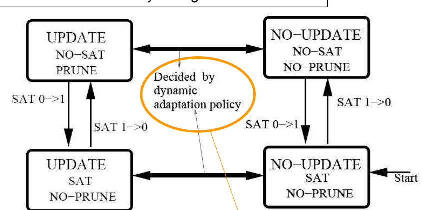
- Two extreme approaches:
  - Never update tree
    - Zero management cost, but high query cost (flood)
    - May be ok if query rate < churn rate
  - Aggressively prune out subtrees that do not satisfy term
    - Low query cost, but management cost high if churn rate high
    - May be ok if churn rate < query rate
- Query rate is user-based, churn rate is system- and term-dependent
- Churn rate different for each node ☹

➔need a decentralized protocol for maintaining each part of a given tree, in order to minimize overall bandwidth utilization

➔So you get best of all worlds in terms of bandwidth

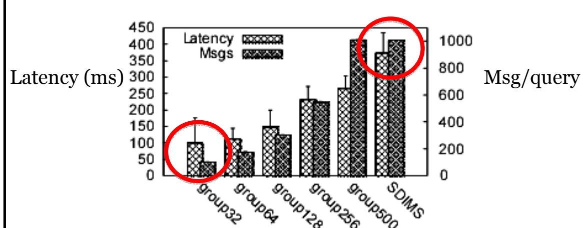
State Machine at Each Moara node

SAT = This subtree satisfies term (some node in it)
PRUNE = tell parent to not forward queries to this subtree
UPDATE = node will update its PRUNE variable at parent as satisfiability changes

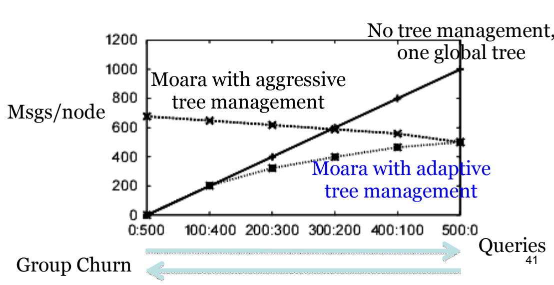


Adaptation policy is local at each node
Compares bandwidth cost based on:
•Local rate of change in SAT, and
•Local rate of queries seen

Emulab (500 instances)



Simulation (10,000 instances)



Discussion

- Tree per term and potentially predicates: too many trees?
  - How do you garbage collect entire trees?
  - What information do you need to maintain the right set of trees?
  - Interesting optimization problem!
- What language *do* sysadmins like to query in?
- Sysadmins often care about how information is visible visually (e.g., CoMON, Zenoss)
  - Automatically inferred queries?
  - Artificial Intelligence/Machine Learning techniques to learn what are the queries sysadmins want most?

Questions?