



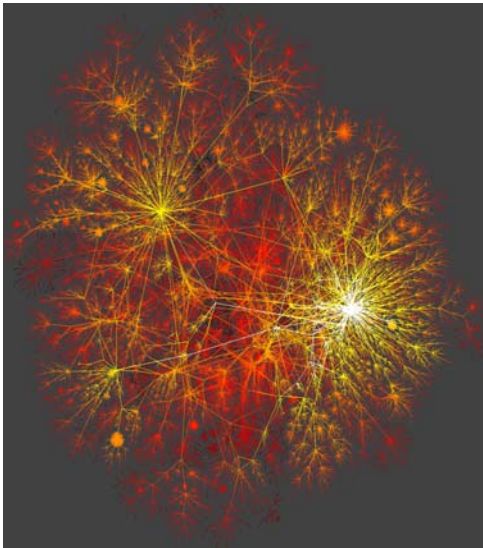
# CS514: Intermediate Course in Computer Systems

Lecture 16: October 23, 2003

Astrolabe (and a few other gossip tricks)

(Astrolabe slides from Ken Birman)

## The Internet



Massive scale.  
Constant flux



Source: Burch and Cheswick





## Demand for more “autonomic”, intelligent behavior

CS514

- Human beings constantly adapt as their environment changes
  - You bike up hill... start to breath hard and sweat. At the top, cool off and catch your breath
  - It gets cold so you put on a sweater
- But computer systems tend to be rigid and easily disabled by minor events



## Typical examples

CS514

- IBM finds that many Web Services systems are perceived as unreliable
  - End-user gets erratic response time
  - Client could be directed to the wrong server site
- But the Web Sphere software isn't at fault!
  - Usually these problems arise from other systems to which WS is connected
  - A snarl of spaghetti sits behind the front end





## A tale of woe

CS514

- Human operators lack tools to see state of system
  - Can't easily figure out what may be going wrong
- In fact operators cause as much as 70% of all Web-related downtime!
- And they directly trigger 35% of crashes



## Sample tale of woe

CS514

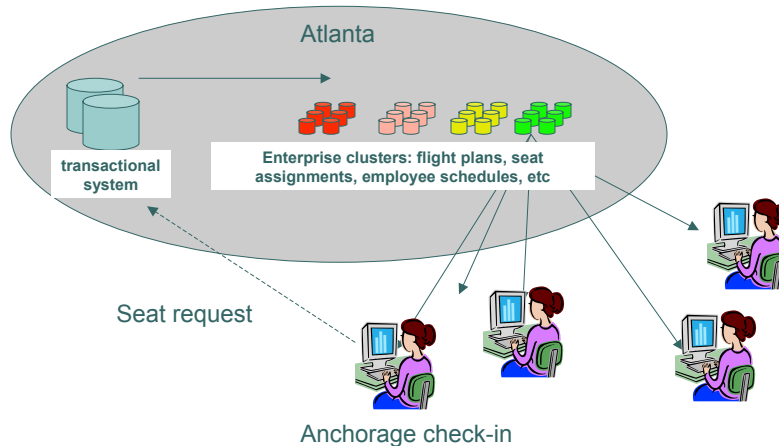
- Delta Airlines maintains the “Delta Nerve Center” in Atlanta
  - It has an old transactional mainframe for most operations
  - Connected to this are a dozen newer relational database applications on clusters
  - These talk to ~250,000 client systems using a publish-subscribe system





# Delta Architecture

CS514



# Anchorage goes down

CS514

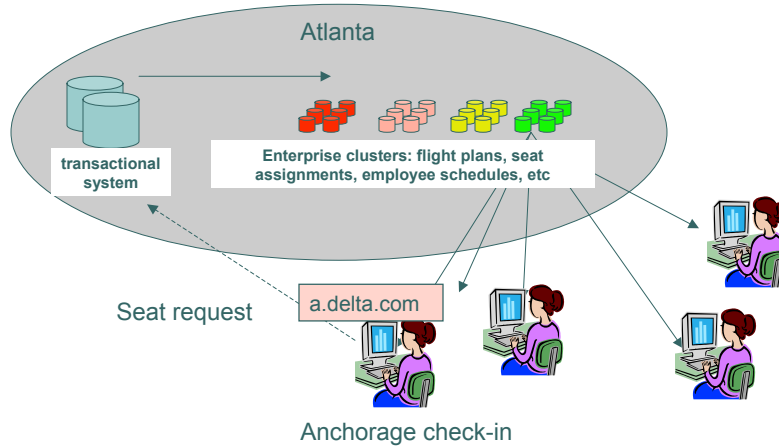
- Problem starts in Atlanta
  - System operator needs to move a key subsystem from computer A to B
  - But the DNS is slow to propagate the change (maybe 48 to 72 hours)
- Anchorage still trying to talk to A
  - So a technician fixes the problem by typing in the IP address of B
  - Gets hired by United with a hefty raise





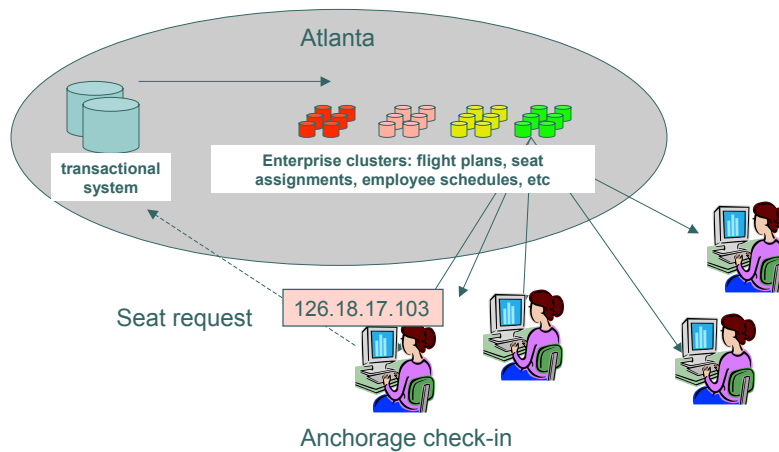
# Delta Architecture

CS514



# Delta Architecture

CS514

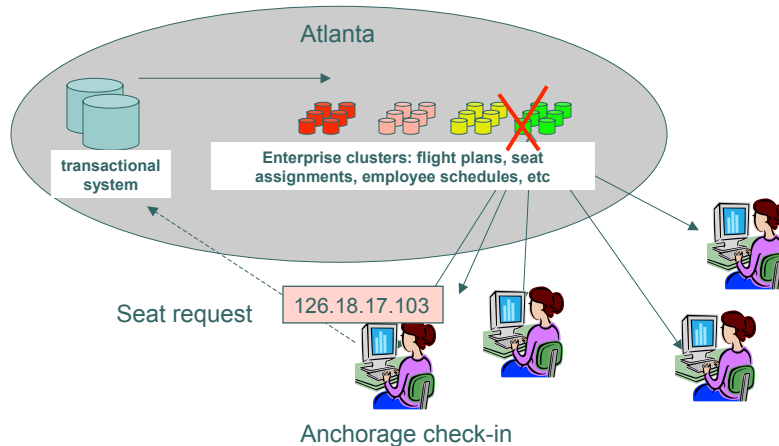






# Delta Architecture

CS514



## Six months later...

CS514

- Time goes by and now we need to move that service again
- Anchorage crashes again...
  - But this time nobody can figure out why!
  - Hunting for references to the service or even to B won't help
  - Need to realize that the actual IP address of B is wired into the application now
  - Nobody remembers what the technician did or why he did it!





**Abstract**

- 114



100



lives in a





## Two options

CS514

- We could ship all the data to the analyst's workstation
  - E.g. ship every image, in real-time
  - If  $N$  machines gather  $I$  images per second, and we have  $B$  bytes per image, the load on the center system grows with  $N \cdot I \cdot B$ . With  $A$  analysts the load on the network grows as  $N \cdot I \cdot B \cdot A$
- Not very practical.



## Two options

CS514

- Or, we could ship the work to the remote sensor systems
  - They do all the work “out there”, so each just searches the images it is capturing
  - Load is thus quite low
- But how could we build such a system?





# Sentient Distributed Systems

CS514

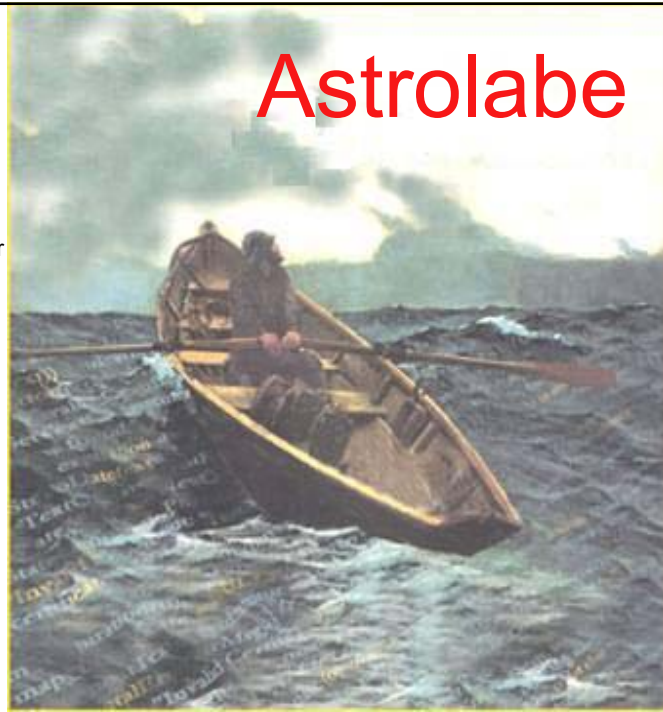
- Can we build a new generation of smart middleware in support of these new intelligent systems?
  - Middleware that *perceives* the state of the network
  - It can *represent this knowledge* in a form smart applications can exploit
  - Although built from large numbers of rather dumb components the *emergent behavior is intelligent*. These applications are more robust, more secure, more responsive than any individual component
  - When something unexpected occurs, they can *diagnose the problem* and trigger a *coordinated distributed response*
  - They *repair themselves* after damage

## Astrolabe

- Intended as help for applications adrift in a sea of information
- Structure emerges from a randomized peer-to-peer protocol
- This approach is robust and scalable even under extreme stress that cripples more traditional approaches

## Developed at Cornell

- By Robbert van Renesse, with many others helping...
- Just an example of the kind of solutions we need
- Astrolabe is a form of knowledge representation for sentient networks





## First, let's take stock of where we are with gossip

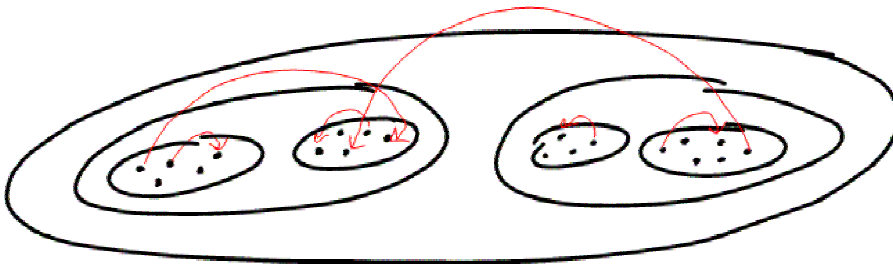
CS514

- We saw “simple” algorithms where each participant selects randomly among other participants
  - But, requires that each node knows all others--- $O(N^2)$  scaling
  - But even if we could fix this, we still have  $O(N^2)$  scaling if all participants contribute a state entry
- We saw that we can exploit locality with a hierarchy of domains
  - But, this slows down the speed of infection

## Different kinds of domains...

CS514

- For locality, domains were used only to tweak how often you gossiped to different participants
  - But anybody could still gossip with anybody





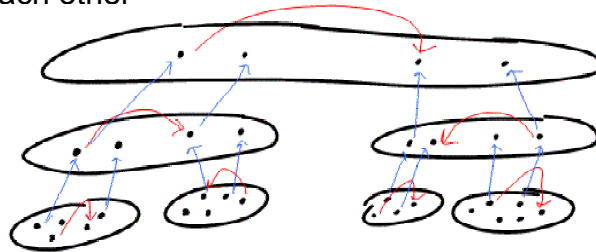


## Different kinds of domains...

CS514

- Astrolabe also has domains, but each domain is a self-contained gossip network
  - Most nodes will never directly gossip with each other

Selected nodes participate in the next higher level of gossip



Astrolabe builds a hierarchy using a P2P protocol that “assembles the puzzle” without any servers

CS514

Dynamically changing query output is visible system-wide

SQL query “summarizes” data

Name	Avg Load	WL contact	SMTP contact
SF	2.6	123.45.61.3	123.45.61.17
NJ	1.8	127.16.77.6	127.16.77.11
Paris	3.1	14.66.71.8	14.66.71.12

Name	Load	Weblog?/	SMTP?	Word Version	
swift	2.0	0	1	6.2	
falcon	1.5	1	0	4.1	
cardinal	4.5	1	0	6.0	

San Francisco

Name	Load	Weblog?/	SMTP?	Word Version	
gazelle	1.7	0	0	4.5	
zebra	3.2	0	1	6.2	
gnu	.5	1	0	6.2	

New Jersey





## Astrolabe in a single domain

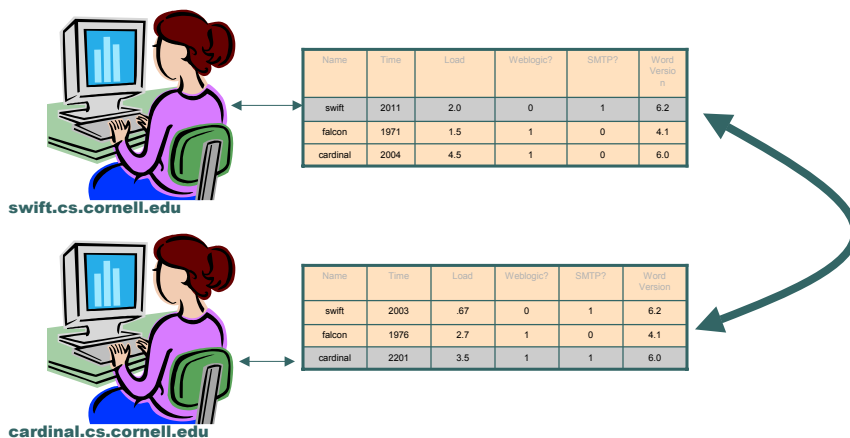
CS514

- Each node owns a single tuple, like the management information base (MIB)
- Nodes discover one-another through a simple broadcast scheme (“anyone out there?”) and gossip about membership
  - Nodes also keep replicas of one-another’s rows
  - Periodically (uniformly at random) merge your state with some else...



## State Merge: Core of Astrolabe epidemic

CS514

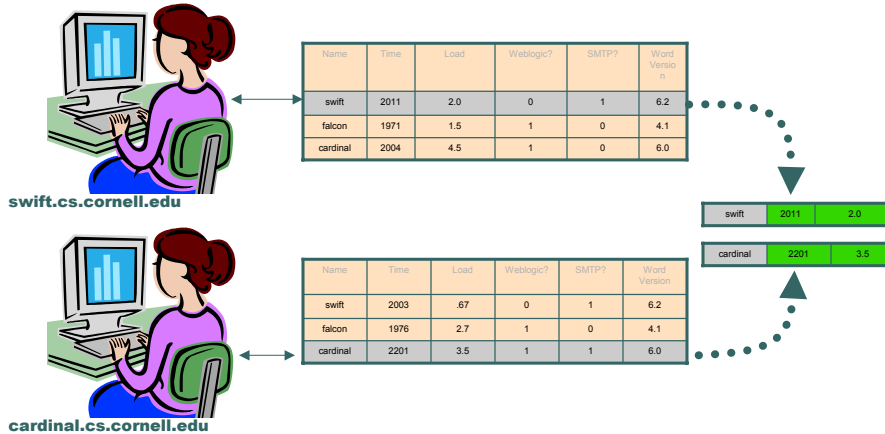






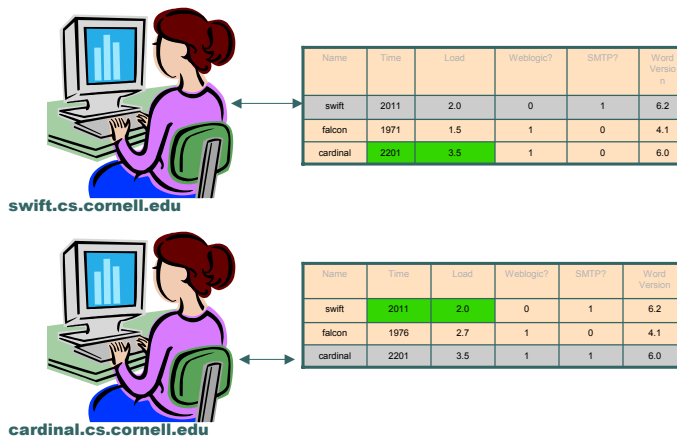
## State Merge: Core of Astrolabe epidemic

CS514



## State Merge: Core of Astrolabe epidemic

CS514







## Observations

CS514

- Merge protocol has constant cost
  - One message sent, received (on avg) per unit time.
  - The data changes slowly, so no need to run it quickly – we usually run it every five seconds or so
  - Information spreads in  $O(\log N)$  time
- But this assumes bounded region size
  - In Astrolabe, we limit them to 50-100 rows



## Big system will have many regions

CS514

- Astrolabe usually configured by a manager who places each node in some region, but we are also playing with ways to discover structure automatically
  - For example could use Paul's ID Maps or the new CMU GNP concept
- A big system could have *many* regions
  - Looks like a pile of spreadsheets
  - A node only replicates data from its neighbors within its own region





## Scaling up... and up...

CS514

- With a stack of domains, we don't want every system to "see" every domain
  - Cost would be huge
- So instead, we'll see a summary



cardinal.cs.cornell.edu

Name	Time	Load	Weblog?	SMTP?	Word
swift	2011	2.0	0	1	6.2
falcon	1976	2.7	1	0	4.1
cardinal	2201	3.5	1	1	6.0



Astrolabe builds a hierarchy using a P2P protocol that "assembles the puzzle" without any servers

CS514

Dynamically changing  
query output is visible  
system-wide

Name	Avg Load	WL contact	SMTP contact
SF	2.6	123.45.61.3	123.45.61.17
NJ	1.8	127.16.77.6	127.16.77.11
Paris	3.1	14.66.71.8	14.66.71.12

SQL query  
"summarizes"  
data

Name	Load	Weblog?	SMTP?	Word Version
swift	2.0	0	1	6.2
falcon	1.5	1	0	4.1
cardinal	4.5	1	0	6.0

San Francisco

Name	Load	Weblog?	SMTP?	Word Version
gazelle	1.7	0	0	4.5
zebra	3.2	0	1	6.2
gnu	.5	1	0	6.2

New Jersey





## Large scale: “fake” regions

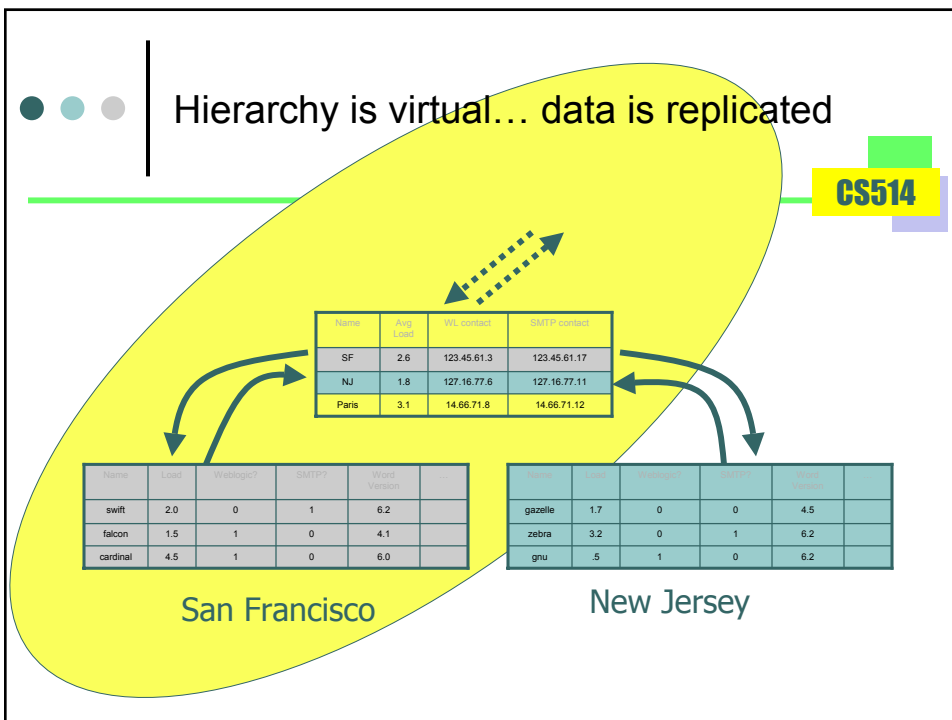
CS514

- These are
  - Computed by queries that summarize a whole region as a single row
  - Gossiped in a read-only manner within a leaf region
- But who runs the gossip?
  - Each region elects “k” members to run gossip at the next level up.
  - Can play with selection criteria and “k”



## Hierarchy is virtual... data is replicated

CS514







Hierarchy is virtual... data is replicated

CS514

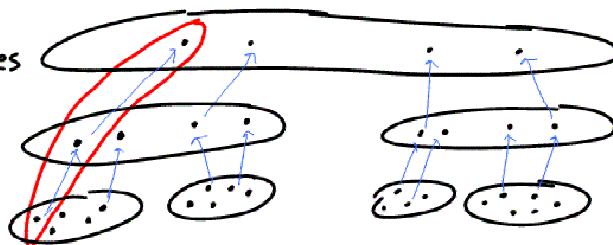


Worst case load?

CS514

- A small number of nodes end up participating in  $O(\log_{\text{fanout}} N)$  epidemics
  - Here the fanout is something like 50

This selected node participates in three gossip domains. (There are  $\log(u)$  levels.)







## Worst case load?

CS514

- A small number of nodes end up participating in  $O(\log_{\text{fanout}} N)$  epidemics
  - Here the fanout is something like 50
  - In each epidemic, a message is sent and received roughly every 5 seconds
- We limit message size so even during periods of turbulence, no message can become huge.
  - Instead, data would just propagate slowly
  - Haven't really looked hard at this case



## What makes Astrolabe a good fit

CS514

- Notice how hierarchical database abstraction “emerges” without ever being physically represented on any single machine
  - Moreover, this abstraction is very robust
  - It scales well... localized disruptions won't disrupt the system state... consistent in eyes of varied beholders
  - Yet individual participant runs a nearly trivial peer-to-peer protocol
- Supports distributed data aggregation, data mining. Adaptive and self-repairing...





## Solutions that share properties

CS514

- Scalable
- Robust against localized disruption
- Have emergent behavior we can reason about, exploit in the application layer
- Think of the way a hive of insects organizes itself or reacts to stimuli. There are many similarities



## Revisit our goals

CS514

- Are these potential components for sentient systems?
  - ✓ Middleware that *perceives* the state of the network
  - ✓ It *represent this knowledge* in a form smart applications can exploit
  - ✓ Although built from large numbers of rather dumb components the *emergent behavior is intelligent*. These applications are more robust, more secure, more responsive than any individual component
  - ✓ When something unexpected occurs, they can *diagnose the problem* and trigger a *coordinated distributed response*
  - ✓ They *repair themselves* after damage
- We seem to have the basis from which to work!





## Brings us full circle

CS514

- Our goal should be a new form of very stable “sentient middleware”
- Have we accomplished this goal?
  - Probabilistically reliable, scalable primitives
  - They solve many problems
  - Gaining much attention now from industry, academic research community
- Fundamental issue is skepticism about peer-to-peer as a computing model




## Conclusions?

CS514

- We're at the verge of a breakthrough – networks that behave like sentient infrastructure on behalf of smart applications
- Could open the door to big advances
- But industry has yet to deploy these ideas and may think about them for a long time before doing so





## Ok, problems/overheads with Astrolabe?

CS514

- You have to configure this “information summarization” hierarchy
  - Sometimes natural fit for problem domain, sometimes not
  - Sometimes useful to force administrator to think about this, but not always
- Q: Is there a way to avoid domains, and yet still avoid  $O(N^2)$  scaling???



## One answer: SCAMP

CS514

- Anne-Marie Kermarrec et.al.
  - Microsoft Cambridge
- Basic insight is this:
  - If each node knows of a small number of random nodes, and limits its gossip to these nodes, then you still get the spreading effect of gossip
  - Difference is only that the “gossip path” for a state element is always the same





## In other words...

CS514

- Instead of knowing all  $N$  nodes, and picking  $K$  randomly (during  $K$  rounds)
- You know  $K$  random nodes, and you pick each of the  $K$  during  $K$  rounds
- Result is the same:  $K$  “random” nodes picked



## How SCAMP works

CS514

- Each node has a “local view”
  - This is the set of random nodes it knows about
  - Local view converges to size  $(c+1)\log(N)$





## How SCAMP works

CS514

- How nodes join:
  - Joining node J selects arbitrary member M
  - M tells all nodes in its view about J
    - They add J to their view
  - M also selects C random nodes in its view and forwards a “subscription” request



## How SCAMP works

CS514

- Handling a subscription request for node J received at node R
  - Node R puts J in its view with a probability in inverse relation to the size of its view
    - In other words, the larger R's view, the less likely J will be added
  - If R does not add J, then it forwards the subscription request to a random member in its view
    - Which does the same thing, etc. . .





## The intuition

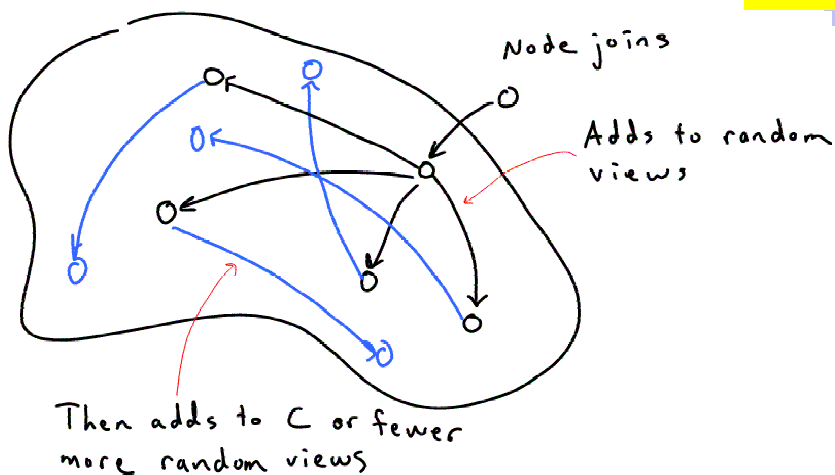
CS514

- If at the time a new node joins, the local view consists of random nodes, then after a new node joins, the modified views will consist of random nodes
- Because subscription essentially takes a random walk through a random network
  - Tending to stop at nodes with smaller views, which results in homogeneous view sizes



## The intuition

CS514







## SCAMP pros and cons

CS514

- Much more efficient for large networks
- Extra complexity to create views
  - But, the algorithm is still simple
- Less robust
  - What if all the nodes in a view crash...a node can be isolated
  - But in practice this is not much of an issue
- No scaling benefit is there is per node state to gossip



## The other $O(N^2)$ problem

CS514

- SCAMP solves the problem of having to gossip about all  $N$  nodes
- But, what if all  $N$  nodes have state to contribute?
- Can we deal with this without the need for hierarchies?
- The answer here is a partial yes...






## Gossip-based Computation of Aggregate Information

CS514

- Kempe, Dobra, Gehrke (Cornell)
- Idea is that you can calculate an aggregate over a collection of nodes, by gossiping partial calculations of the aggregate
  - Aggregates are averages, sums, random samples, and quantiles
- (got that???)



## How it works, for averages

CS514

- Each node has a value, you want the average of the values over all nodes
- Each node maintains a running sum and a running weight
  - $\text{Average} = \text{sum} / \text{weight}$
- The initial values are:
  - $\text{Sum} = \text{local value}$
  - $\text{Weight} = 1$





## How it works, for averages

CS514

- In each round  $t$ , do:
  - $\text{sum}_t = \text{sum}_{t-1}/2$ ,  $\text{weight}_t = \text{weight}_{t-1}/2$
  - Send  $\text{sum}_{t-1}/2$  and  $\text{weight}_{t-1}/2$  to another node
  - Receive  $\text{sum}_{t-1,i}$  and  $\text{weight}_{t-1,i}$  from multiple other nodes  $i$
  - Set  $\text{sum}_t = \text{sum}_{t-1}/2 + \sum_i \text{sum}_{t-1,i}$  and
  - $\text{weight}_t = \text{weight}_{t-1}/2 + \sum_i \text{weight}_{t-1,i}$
- Do this about  $\log(N)$  times



## What just happened?

CS514

- In each round, each node took  $\frac{1}{2}$  its sum and weight, and gave it to another node
  - This preserves the “mass” of the sums and weights over the whole system
- After the first round, each node has the average value of two nodes
- After the second round, each node has the average value of four nodes
- Etc.





## This works for sums too

CS514

- Compute sum of all values over all nodes
- Can anyone guess how?



## This works for sums too

CS514

- Same as averages, except:
- Initially set one weight to 1, all others to 0
  - Instead of  $\sum \text{values} / \sum \text{weights}$ , you get
  - $\sum \text{values} / 1$ , which is the sum
- Analogous approaches exist for calculating random samples, quantiles, max, and min





## Pros and cons

CS514

- Very neat and elegant trick!
- Since not all members are known,  $N$  isn't known, and therefore don't know how many rounds ( $\log N$ ) to gossip
  - Have to gossip until values seem to stabilize, and then some...



## What about Astrolabe then?

CS514

- Does SCAMP and aggregate computation mean we don't need Astrolabe?
- No, because:
  - Sometimes you want the domains, so that you can know what is happening in specific...well...domains!
  - With full knowledge, you can compute more complex computations, and more quickly