Today's distributed systems

→ sprawling, chaotic, complex, unmanageable?
How many machines does Google have?

Chaotic
Complex

- **Orbacus**: CORBA middleware (interoperable RPC)
  - Implemented in C++
  - On top of Linux (POSIX)
  - One “Hello world” request processed

- How many C++ classes and C functions involved (server)?)

- How many local invocations (server)?
Complex

Portability

Interoperability

Transparency

... 

one RPC request,
- **2065** individual invocations
- > **50** C-functions
- > **140** C++ classes

Source: [TKF2009]
Unmanageable?

- **Globus** client
  - 1 creation, 4 requests, 1 destruction

- How many local Java invocations?
  - On the client?
  - On the server?
Unmanageable?

Globus client
- 1 creation, 4 requests, 1 destruction

Projection w.r.t.
- stack depth
- package

client: 1,544,734 Java method call (sic)
server: 6,466,652 Java method calls (sic) [+time out]

The Impact of Web Service Integration on Grid Performance. Taïani, Hiltunen, Schlichting, HPDC-14, 2005
Netflix never used its $1 million algorithm due to engineering costs

By Casey Johnston | Published April 13, 2012 4:25 PM

Netflix awarded a $1 million prize to a developer team in 2009 for an algorithm that increased the accuracy of the company's recommendation engine by 10 percent. But today it doesn't use the million-dollar code, and has no plans to implement it in the future, Netflix announced on its blog Friday. The post goes on to explain why: a combination of too much engineering effort for the results, and a shift from movie recommendations to the "next level" of personalization caused by the transition of the business from mailed DVDs to video streaming.
Netflix never used its $1 million algorithm due to engineering costs

By Casey Johnston | Published April 13, 2012 4:25 PM

Netflix awarded a developer team in 2009 for an algorithm that increased the accuracy of the company’s movie recommendation engine by 10 percent. But today it doesn’t use the million-dollar code, and has no plans to implement it in the future, Netflix announced on its blog Friday. The post goes on to explain why a combination of too much engineering effort and the transition of the business from mailed DVDs to video streaming led to an unmanageable solution.
Netflix never used its $1 million algorithm due to engineering costs

By Casey Johnston | Published April 13, 2012 4:25 PM

Netflix awarded a developer team in 2009 for an algorithm that increased the accuracy of the company's movie recommendation engine by 10 percent. But today it doesn't use the million-dollar code, and has no plans to implement it in the future, Netflix announced on its blog Friday. The post goes on to explain why a combination of too much engineering effort and movie recommendations to the "next level" of personalization caused by the transition of the business from mailed DVDs to video streaming.
Why is distribution hard?

- **Information** takes **time** to travel
  - Some DS protocols inspired from general relativity

- **Machines and networks** **fail**
  - If MTTF 4 years: 1M machines \(\rightarrow\) 1 failure every 2 minutes
Impossibility Results

Asynchronous system with crash failures

- **Consensus** impossible [FLP85]
  - even if only one node crashes
  - (and this assumes we know all the nodes beforehand)

- Consistency + Availability + Partition tol. Impossible
  - Brewer's theorem [GL02]

Consequences

- N crash prone machines not Turing complete [HRR12]
Architecting Scale & Distribution

- Scale + Distribution + Openness + Dynamicity
  - Impossible to envisage all runtime scenarios
  - Very hard to stop system for evolution (long running)

- What developers need
  - Abstraction (intent rather than mechanisms)
  - Ability to change system on the fly (dynamic evolution)

- Two properties provided by self-organizing systems
Self-Organization

- Goal: emergence of structure in a distributed systems
  - nodes, software components, policies, ...

- Needs to way to tell system what is "desirable"
  - should often be a function of system's context
  - Ideal specification: high-level, generic, expressive, holistic

- Needs mechanisms to get there
  - Ideal mechanisms: scalable, robust, fault-tolerance

Concrete example with most of these properties: Self-organizing overlays
Self-Organization

- Goal: emergence of structure in a distributed systems
  - nodes, software components, policies, ...
- Needs to way to tell system what is "desirable"
  - should often be a function of system's context
  - Ideal specification: high-level, generic, expressive, holistic
- Needs mechanisms to get there
  - Ideal mechanisms: scalable, robust, fault-tolerance

Concrete example with most of these properties:

Self-organizing overlays
Outline

- Gossip Protocols
- The case of self-organizing overlays
- Self-organization++: collective survival
- Conclusion
Outline

- Gossip Protocols
- The case of self-organizing overlays
- Self-organization++: collective survival
- Conclusion
Traditional Distribution

- Typical distributed protocols:
  - do not rely on ‘chance’
  - provide deterministic guarantees

- But
  - either rely on a central core of servers for coordination
    - e.g. Zookeeper, Map Reduce frameworks (Hadoop, Spark)
  - or not very scalable ($O(n^k)$, $k=2,3$)
Gossip Protocols

- Alternative: probabilistic approach

- Gossip (aka epidemic) Protocols
  - Introduce some ‘chaos’
  - Goal: system to converge to a desirable outcome
  - But some nodes might be left out

- Trading determinism for scalability & robustness
Example 1: Overlay Multicast

- **Overlays**: "logical network" on top of TCP/IP
  - Foundation of many P2P systems

- **Deterministic multicast**
  - Flooding → highly inefficient
  - Spanning tree → very efficient, but complex, costly, brittle

- **Gossip multicast**: for each new broadcast, each node
  - on first reception randomly pick k targets in the rest of the system (k = ‘fanout’), and forward msg to them
  - (discard duplicate reception)
Example 1: Multicast

How to choose $k$ (fanout) to reach everyone?
Which k? Bimodal behaviour

(Kemarrec, Massoulie, Ganesh / Microsoft Research)

Flat gossip: Results in failure free execution for 10,000 node group [KMG03]

Threshold value: ~ 9.21 (= log(10000) )

Proportion of nodes reached in non atomic broadcast  Proportion of atomic broadcast
Gossip: Key Ingredients

State ➔ Stochastic Data exchange ➔ New State

- What state to maintain?
  - msg already seen (multi-cast)
  - sensor value (aggregation, averaging)
  - neighbours list (topology construction, membership)
  - user profile (social networks)

- When to gossip? (periodic vs. event based)

- With which probability and with whom to gossip?

- Which information to exchange?

- How to compute new state?
Ex. 2: Peer Sampling

- “Random Peer Sampling Service” [JGK04]
  - Membership service: who is in the system?

- **State**: list of k neighbours (k = system param)

- Main idea: **Periodically**: each node \( n \)
  - picks one neighbour \( i \)
  - sends own list of neighbours, receives that of \( i \)
  - \( i \) and \( n \) **merge, shuffle, truncate** the 2 lists
Random Peer Sampling (cont.)

Result → (pseudo) random graph

Highly resilient against churn, partition

(1) peer list exchange  (2) merge and truncation

Small diameter (good for multicast)
Other Uses of Gossip

- Self-organisation in WSN: Who should do what?
- Knowledge aggregation: e.g. average temperature
- Information dissemination (failure detection, AWS S3)
- Self-organizing overlays

From [LGKVB06]
Outline

- Gossip Protocols
- The case of self-organizing overlays
- Self-organization++: collective survival
- Conclusion
From Chaos to Self-organization

From gossip protocols → self-organizing overlays

- **Intuition** of scalable self-organization:
  - Local greedy procedure
  - Global emergent properties (= organization)

- **Greedy** → Self-stabilizing, fault-tolerant
- **Local** → Decentralized, scalable
- **Randomized** → Flexible, fault-tolerant
Self-Organizing Overlays

**Vicinity** [VS05], **T-Man** [JMB09], **Gossple** [BFGKL10]

- State: each node has a position (e.g. in $\mathbb{R}^3$)
- Metrics: Euclidian distance between node
- Goal: find $n$ closest neighbours
Self-Organizing Overlays

Vicinity [VS05], T-Man [JMB09], Gossple [BFGKL10]

- **Greedy local optimisation**: in each round
  - each peer tries to improve its local neighbourhood
  - look for potential new peers: (1) from random layer

![Diagram showing peer connections and layer clustering]

- **clustering layer**
- **gossip-based topology clustering**
- **RPS layer providing random sampling**

- **random link**
- **topology link**
- **node**
Self-Organizing Overlays

Vicinity [VS05], T-Man [JMB09], Gossple [BFGKL10]

- **Greedy local optimisation**: in each round
  - each peer tries to **improve** its local neighbourhood
  - look for **potential new peers**: (2) from friends of friends

![Diagram of peer connections](image-url)
one gossip round (~ control loop)

a current neighborhood

b neighbor candidates from (1) & (2)

c distance computation

\[ \text{Dist}(\text{node}, -) = 3 \]  
\[ \text{distances: } 3, 6, 9, 1, 8, 4 \]

d ranking

d selection

f new neighborhood

Greedy Procedure
Self-Organization in Action

(taken from [JMB09], T-Man)

Result ➔ structured overlay

Highly resilient against churn, partition (RPS)

Fast convergence (Swap)

after 2 cycles

after 7 cycles
Not limited to Euclidian space

- E.g. with user profiles (here metric = overlap)

Perform decentralised KNN graph construction

→ Intuition: similar users can learn from each other
Outline

- Gossip Protocols
- The case of self-organizing overlays
- **Self-organization++: collective survival**
- Conclusion
Towards Collective Survival

Catastrophic Failure on Self-Organizing Overlay

- The topology **heals**
- But the overall **shape** is **lost**

How to recreate whole shape from surviving nodes?
Polystyrene’s Architecture

Polystyrene

Node position

Topology Construction
(T-Man, Vicinity, Gossple)

Peer Sampling Service
(RPS, Cyclon, SCAMP)

Neighbours
Polystyrene Protocol

1. backup (incoming)
2. recovery
3. projection

Node position

Topology Construction

1'. backup (outgoing)
3'. Neighbours
4. migration

ghosts

guests

FD

1' backup (incoming)
The Migration Process

$p.guests_t$  
$p.pos_t$  
$q.pos_t$  
$q.guests_t$
The Migration Process

Bi-clustering of guest points

Heuristics: diameter
The Migration Process

- Bi-clustering of guest points
  ➔ Heuristics: diameter
The Migration Process

Bi-clustering of guest points

- Heuristics: diameter + minimum move
Evaluation

- Shape: 3D torus (replication K = 4)
- Round 20: 50% correlated node crashes
- Round 100: 50% node reinjection (repair)

Polystyrene recreates shape with surviving nodes
Evaluation: Node Reinjection

Polystyrene, $r = 125$

T-Man, $r = 125$

Polystyrene recreates uniform shape after repair
Eval: Quality of Neighborhoods

\[ \text{proximity} = \mathbb{E}_{n_i \in \text{nodes}} \left( \mathbb{E}_{n_j \in n_i.\text{neighbors}} d(n_i.\text{pos}, n_j.\text{pos}) \right) \]

Polystyrene maintains good neighborhoods
Eval: Quality of Shape

$$\text{homogeneity} = \mathbb{E}_{x \in \text{data-points}} \left( \min_{n \in \hat{\text{guests}}^{-1}(x)} \{d(x, n.\text{pos})\} \right)$$

And the torus gets restored!
Eval: Overheads

- Message + Memory

Reasonable Cost
Eval: Scalability (1)

- Time (rounds) until homeogeneity less than

\[ H_{|A|}^{|N|} = \frac{1}{2} \sqrt{\frac{|A|}{|N|}} \]

Logarithmic convergence!
Outline

- Gossip Protocols
- The case of self-organizing overlays
- Self-organization++: collective survival
- Conclusion
Conclusion

- Growth in scale / complexity / openness / dynamicity
  - Automation of structuring tasks required

- Self-organisation is possible in
  - Large scale systems
  - Under very harsh conditions (catastrophic failure)
  - With very good performance (log(n))

- Price to pay: No deterministic guarantees (usually)

- Next challenge: holistic programmability of overlays
  - Link with "self-awareness"
Thank you

Questions?
(Some) Refs: Computability


(Some) References: Protocols


(Some) References: Protocols


(Some) References: Protocols

(Some) Refs: Frameworks


- [LTB08] Lin, S., Taiani, F., and Blair, G. S. (2008). Facilitating gossip programming with the gossipkit framework, DAIS’08, pages 238–252


Ideal software artefact

→ structured, predictable, open, evolvable
A Distributed System Today ...

- Facebook
- Twitter
- JSON
- External services
- External developers
- Geosocial app, est. 2009
- MongoDB
- Amazon Web Services
- 45M Users
T-Man: Swap Mechanism

- To speed convergence each node $n$
  - picks one node $i$ in clustering neighbours
  - sends own list of neighbours, receives that of $i$
  - $i$ and $n$ merge, sort, keep $k$ closest neighbours

1. peer list exchange
2. merge and truncation
Eval: Scalability (2)

- Strong influence of splitting mechanism

![Graph showing the relationship between reshaping time and size of network for different splitting mechanisms. The graph includes four lines representing Split_Basic, Split_Advanced (MD+PD), and Split_MD. The x-axis represents the size of the network in terms of nodes, ranging from 100 to 100,000. The y-axis represents the reshaping time in terms of rounds, ranging from 0 to 30. The Split_Basic line has the steepest slope, followed by Split_Advanced (MD+PD), and then Split_MD with the least steep slope.]