Systèmes pair à pair de partage de données

And some other things I do

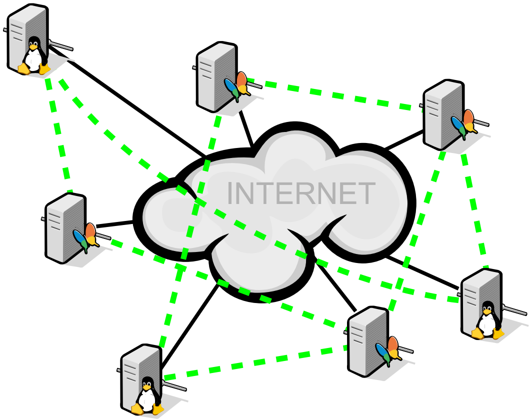
Remigiusz Modrzejewski

May 11, 2012
MASCOTTE Project

I3S(CNRS/UNS)–INRIA, PACA

Outline

Intuition of P2P



Peer to peer networks — end systems creating a virtual overlay

Peer-to-peer storage

With: Frédéric Giroire, Sandeep Kumar Gupta, Julian Monteiro, Stéphane Perennes

Related works:

- Analysis of Failure Correlation Impact on Peer-to-Peer Storage Systems by Dalle et al. looks into whole disk failures, but assumes exponential reconstruction time; 2009
- Simulation analysis of download and recovery processes in P2P storage systems by Dandoush et al. find download/recovery time hypo-exponential, but looks only at single fragment level; 2009
- Availability in Globally Distributed Storage Systems by Ford et al. bases on a large body of data tracing Google storage systems; 2010

Peer-to-peer storage

Considered system:

- Indefinite backup
  - negligible read rate
  - high reliability: 10^{-5} loss probability/100GB $\rightsquigarrow$ 10^{-12} loss probability/5MB
- Cheap and scalable
  - highly distributed
  - unreliable hardware
  - uses consumer connections

Our work: find a better model of the system, thoroughly validate

Peer-to-peer storage

Considered system:

- Indefinite backup
  - negligible read rate
  - high reliability: 10^{-5} loss probability/100GB $\rightsquigarrow$ 10^{-12} loss probability/5MB
- Cheap and scalable
  - highly distributed
  - unreliable hardware
  - uses consumer connections

Our work: find a better model of the system, thoroughly validate

Peer-to-peer storage

Considered system:

- Indefinite backup
  - negligible read rate
  - high reliability: 10^{-5} loss probability/100GB $\rightsquigarrow$ 10^{-12} loss probability/5MB
- Cheap and scalable
  - highly distributed
  - unreliable hardware
  - uses consumer connections

Our work: find a better model of the system, thoroughly validate

Redundancy

- Blocks divided into s fragments
- Additional r fragments of redundancy
- Using **erasure codes**
  - Reed-Solomon
  - Regenerating codes

All data can be **reconstructed** as long as $\geq s$ fragments are available

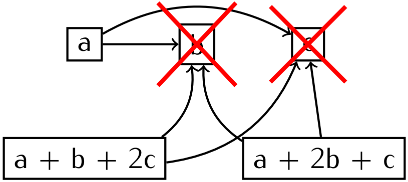
a b c

a + b + 2c a + 2b + c

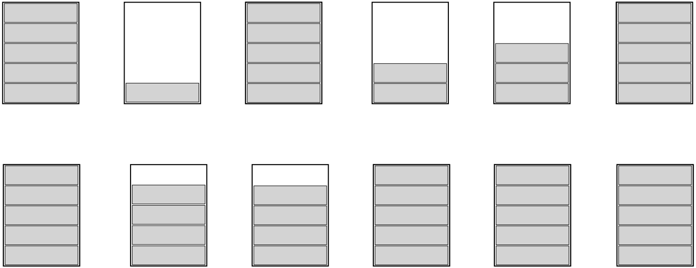
Redundancy

- Blocks divided into s fragments
- Additional r fragments of redundancy
- Using **erasure codes**
  - Reed-Solomon
  - Regenerating codes

All data can be **reconstructed** as long as $\geq s$ fragments are available

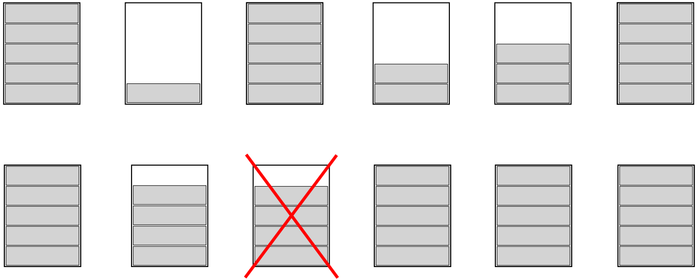


System mechanics



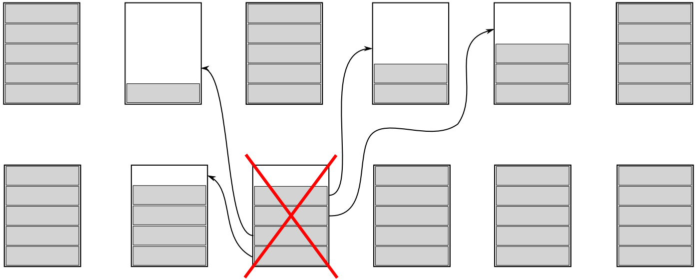
Data redundancy maintained in continuous repair process

System mechanics



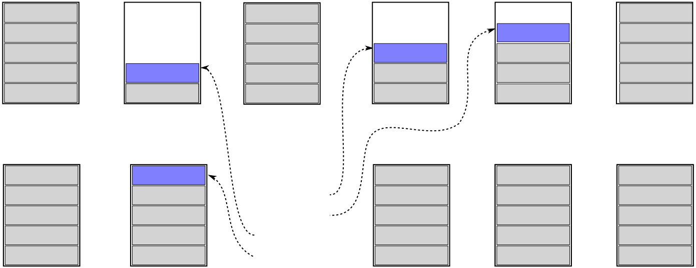
Data redundancy maintained in continuous repair process

System mechanics



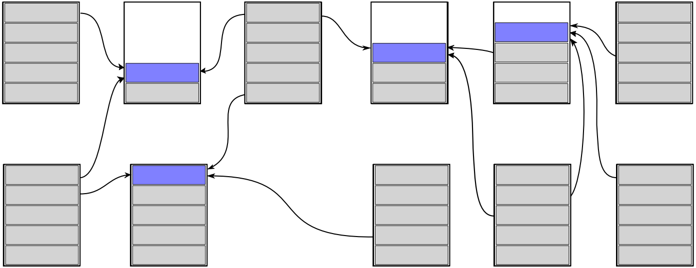
Data redundancy maintained in continuous repair process

System mechanics



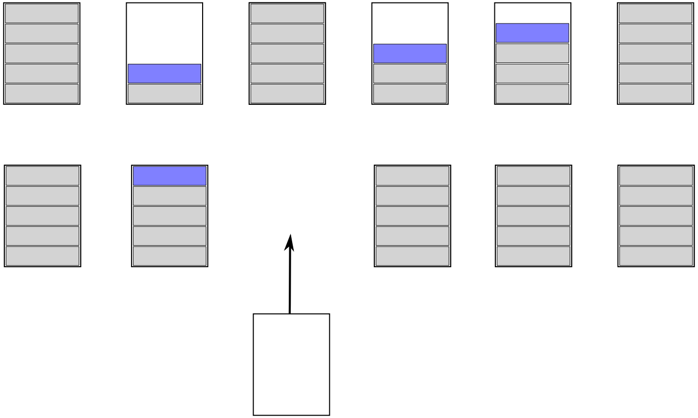
Data redundancy maintained in continuous repair process

System mechanics



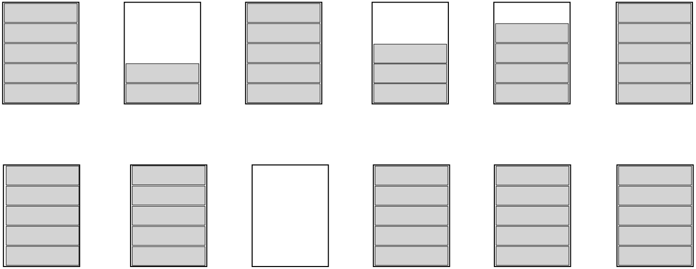
Data redundancy maintained in continuous repair process

System mechanics



Data redundancy maintained in continuous repair process

System mechanics



Data redundancy maintained in continuous repair process

Factor of efficiency

Workload proportional to stored data $\wedge$ new disks have little data $\wedge$ nearly all repairs need a full disk $\Rightarrow$ wasted bandwidth

Let:

- x be the disk **overcapacity** — average capacity / average usage
- ρ be the **factor of efficiency** — total throughput / total bandwidth

We found out that:

$$\rho \approx \frac{1}{x}$$

Factor of efficiency

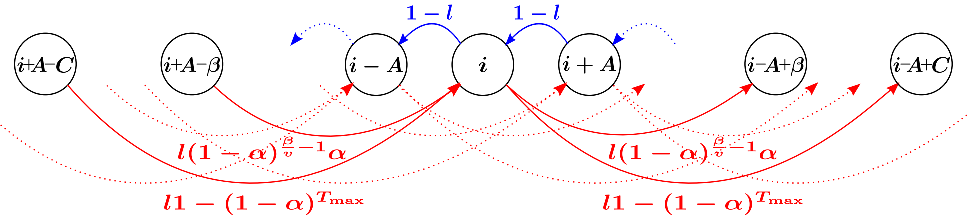
Workload proportional to stored data $\wedge$ new disks have little data $\wedge$ nearly all repairs need a full disk $\Rightarrow$ wasted bandwidth
Let:

- x be the disk **overcapacity** — average capacity / average usage
- ρ be the **factor of efficiency** — total throughput / total bandwidth

We found out that:

$$\rho \approx \frac{1}{x}$$

Markovian queuing model



- **Global** $M^{\beta}/D/1$ queue of all blocks needing repair
- States — number of fragments in queue
- Transitions — reconstructions or failings

Case study

Scenario:

- Users have 1Mbps connections, but allocate 128kbps to repairs
- Users allocate 300GB disk space, insert 100GB data
- Expected lifetime = 1 year, neighbourhood size = 100 peers

Simple intuition:

- Repair time of 1 disk = **17 hours** ($= 100 \cdot 8 \cdot 10^6$kb$/(100 \cdot 128$kbps$)$)
- Probability of data loss per year **(PDLPY) of 10^{-8}**

Our model (more realistic assumptions, validated by simulation and experiments):

- Repair time = **9 days**
- **PDLPY = 0.2**

Case study

Scenario:

- Users have 1Mbps connections, but allocate 128kbps to repairs
- Users allocate 300GB disk space, insert 100GB data
- Expected lifetime = 1 year, neighbourhood size = 100 peers

Simple intuition:

- Repair time of 1 disk = **17 hours** ($= 100 \cdot 8 \cdot 10^6$kb$/(100 \cdot 128$kbps$)$)
- Probability of data loss per year **(PDLPY) of** 10^{-8}

Our model (more realistic assumptions, validated by simulation and experiments):

- Repair time = **9 days**
- PDLPY = 0.2

Case study

Scenario:

- Users have 1Mbps connections, but allocate 128kbps to repairs
- Users allocate 300GB disk space, insert 100GB data
- Expected lifetime = 1 year, neighbourhood size = 100 peers

Simple intuition:

- Repair time of 1 disk = **17 hours** ($= 100 \cdot 8 \cdot 10^6$kb$/(100 \cdot 128$kbps$)$)
- Probability of data loss per year **(PDLPY) of** 10^{-8}

Our model (more realistic assumptions, validated by simulation and experiments):

- Repair time = **9 days**
- PDLPY = 0.2

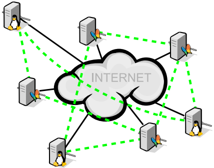
Peer-to-peer streaming

With: Frédéric Giroire, Stéphane Perennes

Related works:

- **Epidemic live streaming: optimal performance trade-offs**, by Bonald, Massoulié, Mathieu, Perino, and Twigg; simple random schemes
- **Prime: Peer-to-peer receiver-driven mesh-based streaming**, by Magharei and Rejaie; an unstructured algorithm which yields temporarily structured flows
- **SplitStream: high-bandwidth multicast in cooperative environments**, by Castro, Druschel, Kermarrec, Nandi, Rowstron and Singh; the best known structured network

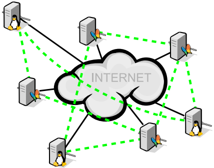
Video distribution



File sharing

Live streaming

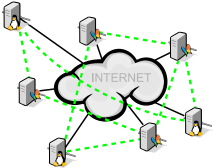
Video distribution



File sharing

Live streaming

Video distribution



File sharing

Live streaming

Problem definition

- Disseminate a stream of data
- Single source
- Multiple recipients
- Recipients contribute to further disseminate
- Finite dissemination deadline
- High bandwidth utilization
- Participants are autonomous – **distributed algorithms**
- Local, delayed view

Churn

- **Node dynamics** shown to be biggest problem of live systems
- When $n \approx 20000$, almost 1000 peers join and leave per minute
- Often overlooked in literature

Churn

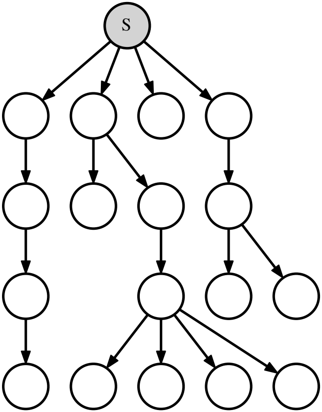
- **Node dynamics** shown to be biggest problem of live systems
- When $n \approx 20000$, almost 1000 peers join and leave per minute
- Often overlooked in literature

Churn

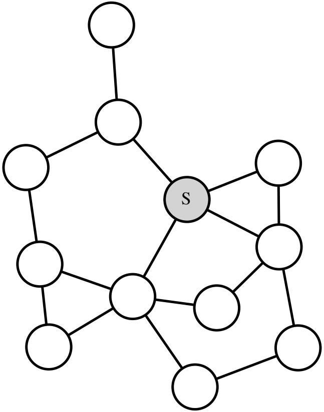
- Node dynamics shown to be biggest problem of live systems
- When $n \approx 20000$, almost 1000 peers join and leave per minute
- Often overlooked in literature

Types of overlays

Structured:



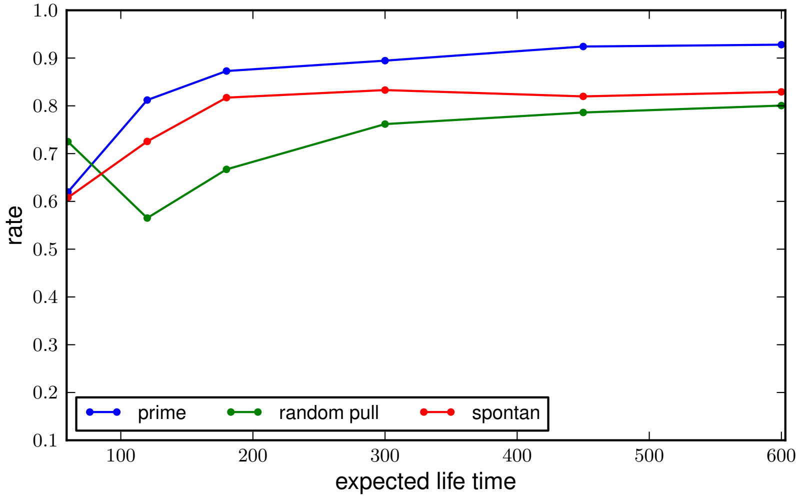
Random:



Simulator

- Test overlays under bandwidth constraints and churn
- Custom discrete event simulator
- Implemented 6 overlay algorithms
- Total 9889 lines of Python code
- 572 automated tests

Simulation results for varying churn



Dynamic online tree balancing

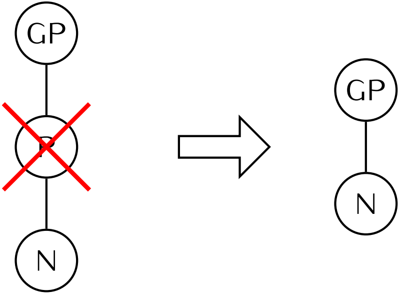
- k-ary tree (simple case: binary)
- Preserve a balanced state
- Random node arrivals and departures
- Local, **distributed** algorithm

Subtree size

- Number of nodes in subtree rooted at a given node
- “Size of the node”
- **Periodically reported** by the node to its parent

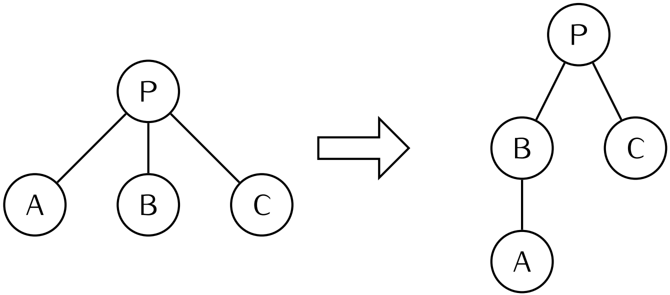
The algorithm

1. Parent left $\rightsquigarrow$ **reattach** to grandparent
2. Overloaded $\rightsquigarrow$ **push** a child to become a grandchild
3. Underloaded $\rightsquigarrow$ **pull** a grandchild to become a child
4. Children imbalanced $\rightsquigarrow$ **balance** the children
   - Smallest underloaded $\rightsquigarrow$ move a child of the biggest to the smallest
   - Smallest saturated $\rightsquigarrow$ swap 2 children between smallest and biggest



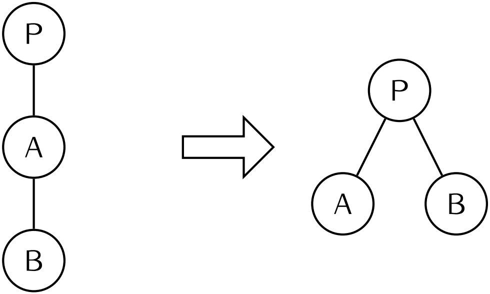
The algorithm

1. Parent left $\rightsquigarrow$ **reattach** to grandparent
2. Overloaded $\rightsquigarrow$ **push** a child to become a grandchild
3. Underloaded $\rightsquigarrow$ **pull** a grandchild to become a child
4. Children imbalanced $\rightsquigarrow$ **balance** the children
    - Smallest underloaded $\rightsquigarrow$ move a child of the biggest to the smallest
    - Smallest saturated $\rightsquigarrow$ swap 2 children between smallest and biggest



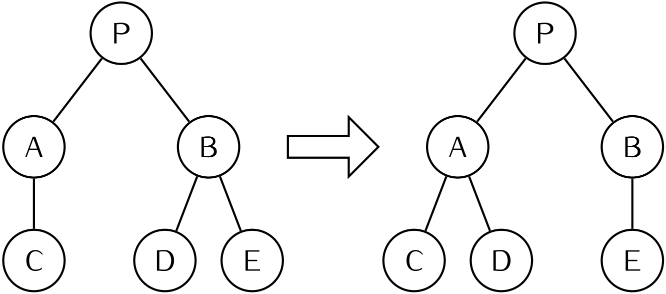
The algorithm

1. Parent left $\rightsquigarrow$ **reattach** to grandparent
2. Overloaded $\rightsquigarrow$ **push** a child to become a grandchild
3. Underloaded $\rightsquigarrow$ **pull** a grandchild to become a child
4. Children imbalanced $\rightsquigarrow$ **balance** the children
    - Smallest underloaded $\rightsquigarrow$ move a child of the biggest to the smallest
    - Smallest saturated $\rightsquigarrow$ swap 2 children between smallest and biggest



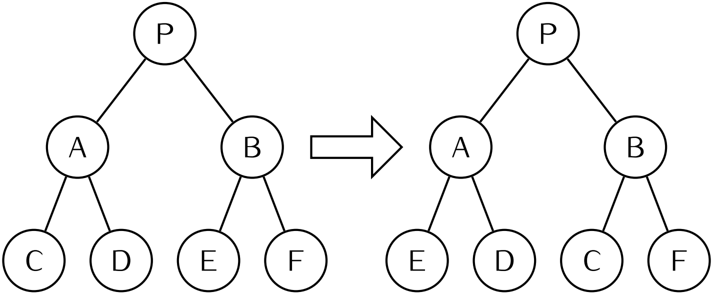
The algorithm

1. Parent left $\rightsquigarrow$ **reattach** to grandparent
2. Overloaded $\rightsquigarrow$ **push** a child to become a grandchild
3. Underloaded $\rightsquigarrow$ **pull** a grandchild to become a child
4. Children imbalanced $\rightsquigarrow$ **balance** the children
   - Smallest underloaded $\rightsquigarrow$ move a child of the biggest to the smallest
   - Smallest saturated $\rightsquigarrow$ swap 2 children between smallest and biggest



The algorithm

1. Parent left $\rightsquigarrow$ **reattach** to grandparent
2. Overloaded $\rightsquigarrow$ **push** a child to become a grandchild
3. Underloaded $\rightsquigarrow$ **pull** a grandchild to become a child
4. Children imbalanced $\rightsquigarrow$ **balance** the children
   - Smallest underloaded $\rightsquigarrow$ move a child of the biggest to the smallest
   - Smallest saturated $\rightsquigarrow$ swap 2 children between smallest and biggest



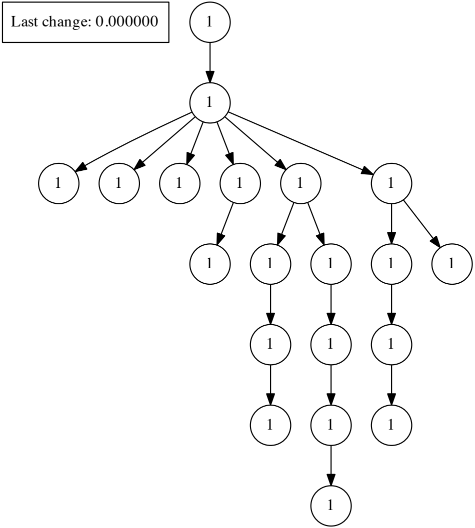
Model

1. Each node performs operations according to a **Poisson process**

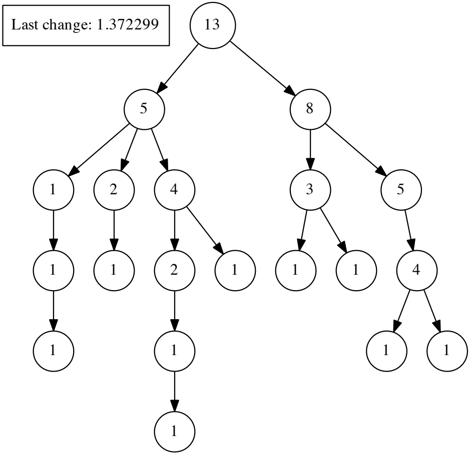
or

2. In each **round** each node performs one operation, random order

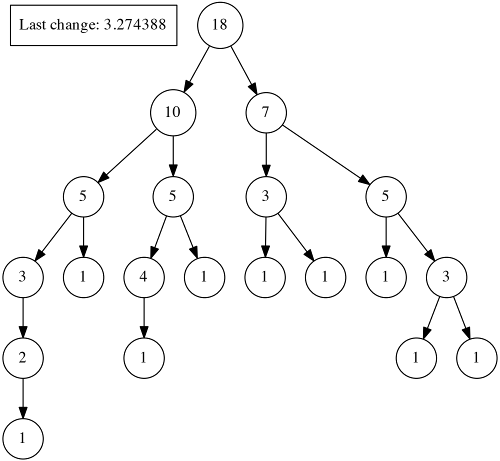
Example



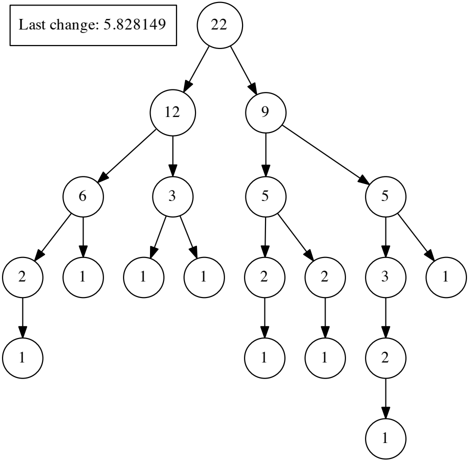
Example



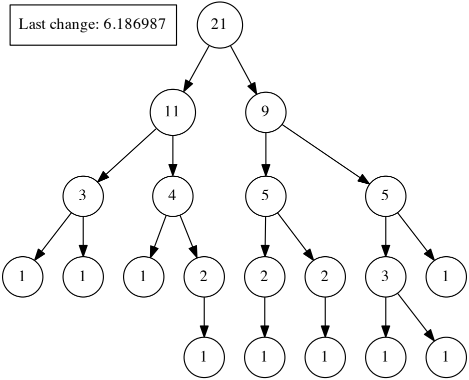
Example



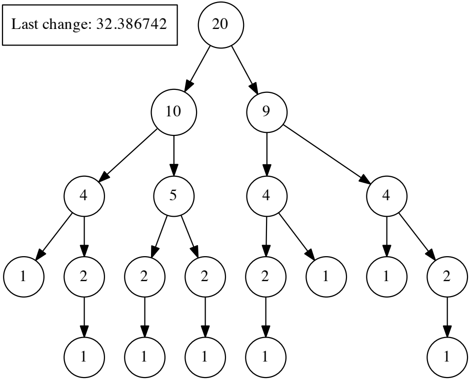
Example



Example



Example



Work in progress

Already done:

- Guaranteed stop in tree of optimal height
- Expected time to fix tree after single failure $= k - 1 + \log_k \log_k n$
- Convergent in $O(n^3)$ rounds

Still working on:

- Proof of convergence in $O(n^2)$ rounds or $O(n^2)$ operations
- Worst cases: convergent in $O(n)$ rounds or $O(n \log n)$ operations

More practical model:

- Extend to heterogenous out-degree
- Extend to multi-trees

Work in progress

Already done:

- Guaranteed stop in tree of optimal height
- Expected time to fix tree after single failure $= k - 1 + \log_k \log_k n$
- Convergent in $O(n^3)$ rounds

Still working on:

- Proof of convergence in $O(n^2)$ rounds or $O(n^2)$ operations
- Worst cases: convergent in $O(n)$ rounds or $O(n \log n)$ operations

More practical model:

- Extend to heterogenous out-degree
- Extend to multi-trees

Work in progress

Already done:

- Guaranteed stop in tree of optimal height
- Expected time to fix tree after single failure $= k - 1 + \log_k \log_k n$
- Convergent in $O(n^3)$ rounds

Still working on:

- Proof of convergence in $O(n^2)$ rounds or $O(n^2)$ operations
- Worst cases: convergent in $O(n)$ rounds or $O(n \log n)$ operations

More practical model:

- Extend to heterogenous out-degree
- Extend to multi-trees

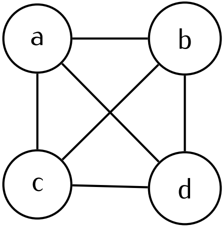
Energy Efficient Content Distribution

With: Julio Araujo, Frédéric Giroire, Yaning Liu, Joanna Moulierac

Related works:

- **Minimizing Routing Energy Consumption: from Theoretical to Practical Results** by Giroire, Mazauric, Moulierac and Onfroy; turning off links with traditional demands
- **Energy-Aware Network Management and Content Distribution** by Chiaraviglio and Matta; turning off links and servers in CDN

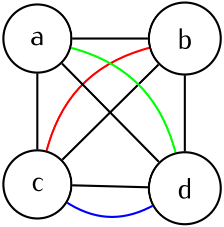
Energy Efficient Content Distribution



- We want to shut down links in a network
- Traditional demands
- Overlay demands
  - Demands can be served by any one of replicated servers
  - Servers have limited capacity
- Content caches
  - Caches located at routers
  - Can be on/off, consume energy
  - Can be selective in what to cache
- Routing is **fractional**, links are undirected

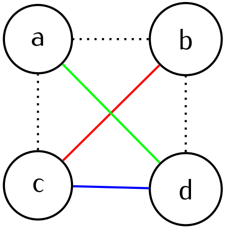
Energy Efficient Content Distribution

- We want to shut down links in a network
- Traditional demands
- Overlay demands
  - Demands can be served by any one of **replicated servers**
  - Servers have limited capacity
- Content caches
  - Caches located at routers
  - Can be on/off, consume energy
  - Can be selective in what to cache
- Routing is **fractional**, links are undirected



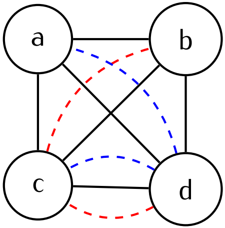
Energy Efficient Content Distribution

- We want to shut down links in a network
- Traditional demands
- Overlay demands
  - Demands can be served by any one of **replicated servers**
  - Servers have limited capacity
- Content caches
  - Caches located at routers
  - Can be on/off, consume energy
  - Can be selective in what to cache
- Routing is **fractional**, links are undirected

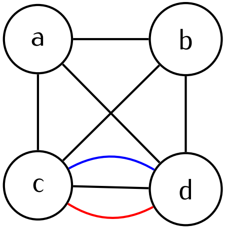


Energy Efficient Content Distribution

- We want to shut down links in a network
- Traditional demands
- Overlay demands
  - Demands can be served by any one of **replicated servers**
  - Servers have limited capacity
- Content caches
  - Caches located at routers
  - Can be on/off, consume energy
  - Can be selective in what to cache
- Routing is **fractional**, links are undirected



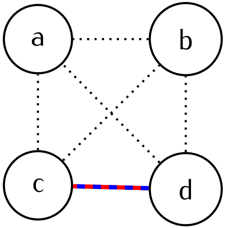
Energy Efficient Content Distribution



- We want to shut down links in a network
- Traditional demands
- Overlay demands
  - Demands can be served by any one of **replicated servers**
  - Servers have limited capacity
- Content caches
  - Caches located at routers
  - Can be on/off, consume energy
  - Can be selective in what to cache
- Routing is **fractional**, links are undirected

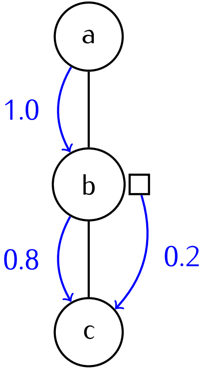
Energy Efficient Content Distribution

- We want to shut down links in a network
- Traditional demands
- Overlay demands
  - Demands can be served by any one of **replicated servers**
  - Servers have limited capacity
- Content caches
  - Caches located at routers
  - Can be on/off, consume energy
  - Can be selective in what to cache
- Routing is **fractional**, links are undirected



Energy Efficient Content Distribution

- We want to shut down links in a network
- Traditional demands
- Overlay demands
  - Demands can be served by any one of **replicated servers**
  - Servers have limited capacity
- Content caches
  - Caches located at routers
  - Can be on/off, consume energy
  - Can be selective in what to cache
- Routing is **fractional**, links are undirected



Work in progress

Already done:

- NP-Complete, APX-har
- ILP formulations and 5 heuristics (without caches)

Still working on:

- Find a good model for caches
- Extend the algorithms for caches
- Find good instance set

Work in progress

Already done:

- NP-Complete, APX-har
- ILP formulations and 5 heuristics (without caches)

Still working on:

- Find a good model for caches
- Extend the algorithms for caches
- Find good instance set

Weighted Improper Colouring

With: Julio Araujo, Jean-Claude Bermond, Frédéric Giroire, Frédéric Havet, Dorian Mazauric

Related works:

- **Models and solution techniques for frequency assignment problems** by Aardal, van Hoesel, Koster, Mannino and Sassano; survey on FAP
- **An enumerative algorithm for the frequency assignment problem** by Mannino and Sassano; similar model, different results
- **Frequency assignment in mobile radio systems using branch-and-cut techniques** by Fischetti, Lepschy, Minerva, Romanin-Jacur and Toto; similar model, different results

Weighted Improper Colouring

- We assign colours to nodes of a graph
- Nodes of the same colour interfere with each other
  - Interference is function of distance
  - In general case $f(a, b) \rightarrow \mathbb{R}_+$
- A certain amount of interference can be tolerated at each node

Simple distance function

We consider a simple interference function:

$$f(d) = \begin{cases} 1, & \text{if } d = 1 \\ \frac{1}{2}, & \text{if } d = 2 \\ 0, & \text{otherwise} \end{cases}$$

In other words: given a graph $G = (V, E)$ and its square $G^2 = (V, E^2)$, we study *from now on* the function $w : E \to \{1, 0.5\}$ such that $w(e) = 0.5$ if, and only if, $e \in E^2 \setminus E$.

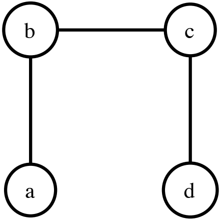
Simple distance function

We consider a simple interference function:

$$f(d) = \begin{cases} 1, & \text{if } d = 1 \\ \frac{1}{2}, & \text{if } d = 2 \\ 0, & \text{otherwise} \end{cases}$$

In other words: given a graph $G = (V, E)$ and its square $G^2 = (V, E^2)$, we study *from now on* the function $w : E \to \{1, 0.5\}$ such that $w(e) = 0.5$ if, and only if, $e \in E^2 \setminus E$.

Simple distance function



We consider a simple interference function:

$$f(d) = \begin{cases} 1, & \text{if } d = 1 \\ \frac{1}{2}, & \text{if } d = 2 \\ 0, & \text{otherwise} \end{cases}$$

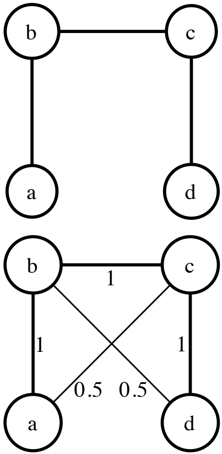
In other words: given a graph $G = (V, E)$ and its square $G^2 = (V, E^2)$, we study *from now on* the function $w : E \to \{1, 0.5\}$ such that $w(e) = 0.5$ if, and only if, $e \in E^2 \setminus E$.

Simple distance function

We consider a simple interference function:

$$f(d) = \begin{cases} 1, & \text{if } d = 1 \\ \frac{1}{2}, & \text{if } d = 2 \\ 0, & \text{otherwise} \end{cases}$$

In other words: given a graph $G = (V, E)$ and its square $G^2 = (V, E^2)$, we study *from now on* the function $w : E \to \{1, 0.5\}$ such that $w(e) = 0.5$ if, and only if, $e \in E^2 \backslash E$.



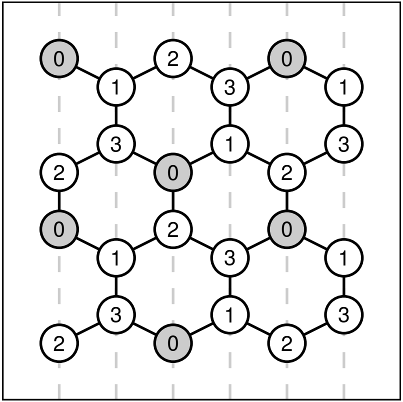
Results

- NP-Complete, APX-hard
- General bounds (e.g. $\chi_t(G, w) \leq \left\lceil \frac{\Delta(G,w)+\text{gcd}(w)}{t+\text{gcd}(w)} \right\rceil$)
- Optimal solutions for infinite grids
- IP, greedy heuristic and branch-and-bound algorithm

Hexagonal grid

If G is an infinite hexagonal grid, then

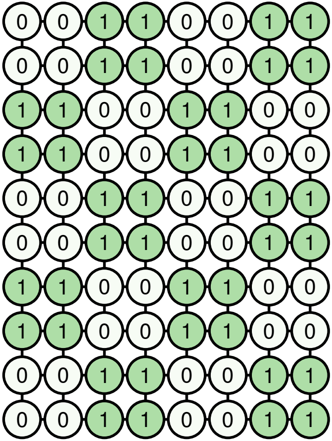
$$\chi_t^w(G^2) = \begin{cases} 4, & \text{if } 0 \leq t < 1; \\ 3, & \text{if } 1 \leq t < 2; \\ 2, & \text{if } 2 \leq t < 6; \\ 1, & \text{if } 6 \leq t. \end{cases}$$



Square grid

If G is an infinite square grid, then

$$\chi_t^w(G^2) = \begin{cases} 5, & \text{if } 0 \leq t < 0.5; \\ 4, & \text{if } 0.5 \leq t < 1; \\ 3, & \text{if } 1 \leq t < 3; \\ 2, & \text{if } 3 \leq t < 8; \\ 1, & \text{if } 8 \leq t. \end{cases}$$



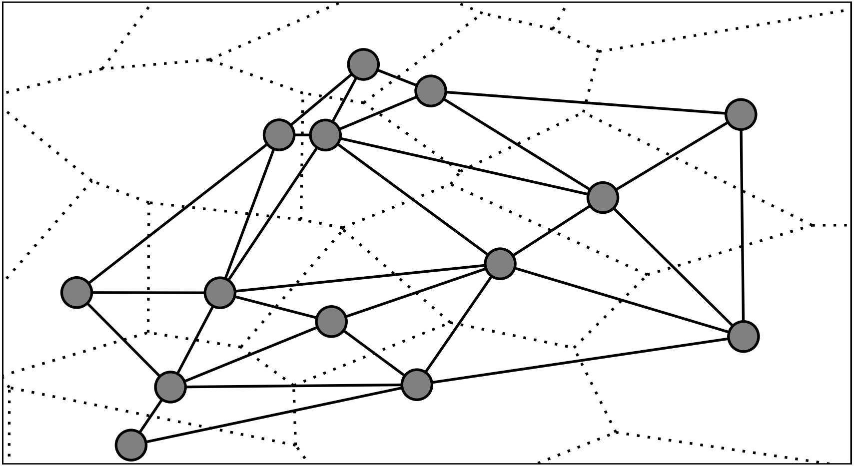
6-regular grid

Theorem

If G is an infinite triangular grid, then

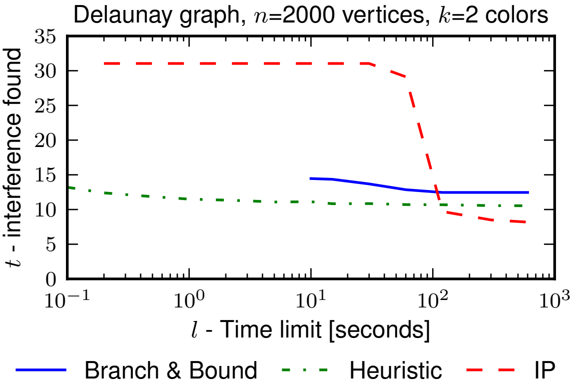
$$\chi_t^w(G^2) = \begin{cases} \leq 7, & \text{if } t = 0; \\ \leq 6, & \text{if } t = 0.5; \\ \leq 5, & \text{if } t = 1; \\ \leq 4, & \text{if } 1.5 \leq t < 3; \\ \leq 3, & \text{if } 3 \leq t < 5; \\ \leq 2, & \text{if } 5 \leq t < 12; \\ 1, & \text{if } 12 \leq t. \end{cases}$$

Delaunay graph

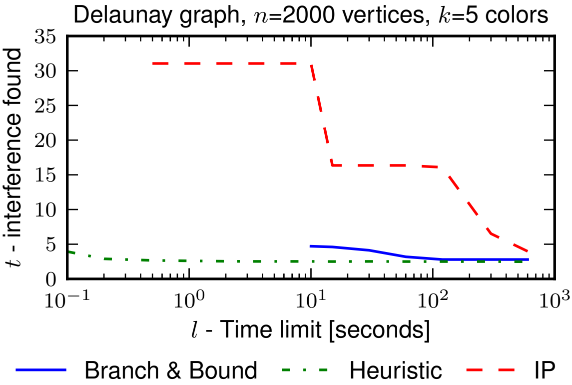


Effect of Delaunay tesselation for a set of random points.
Dual of Voronoi diagram.

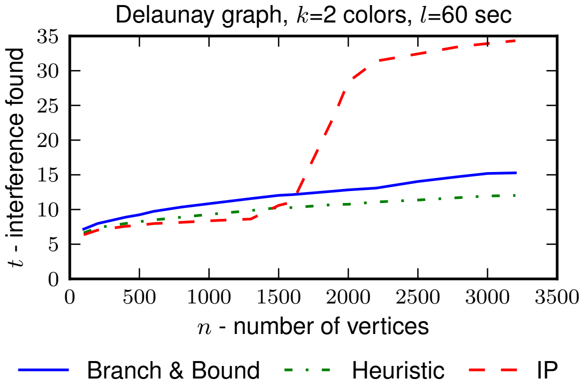
Algorithms performance



Algorithms performance



Algorithms performance



Questions