iSocial – February 5th, 2014

K-Leader Election with Rank Slicing

Giovanni Simoni, Peerialism AB

The problem

- **Goal:** Within a Distributed Network of *N* nodes, select K leaders basing on some appliction-specific metric.
- The leaders will be assigned to a special role (e.g running a service for non-leaders).
- Broad spectrum of applications:
  - **Distributed storage:** maintain K redundant copies of a file;
  - **Distributed Streaming:** peers acting as source for the chunks of a streamed video;
  - **Decentralized Social Network:** top K neighbors from which content or data should be fetched, preferably.

The problem

- Selection driven by node capabilities
- Examples of relevant metrics
  - **Distributed storage:** available space in hard drive, average network throughput...
  - **Distributed streaming:** bandwidth, network latency, locality...
  - **Decentralized Social Networks:** shared interests, level of trust...

Existing Solutions (1)

Indirect solution based on *Probabilistic Quorum*, for distributed storage

- An instance of Probabilistic Quorum for every stored content;
- Every instance decides K nodes which will hold a replica.

Solutions for routing in Wireless Ad-Hoc Networks

- Local election of a leader among reachable neighbors
- Second election of K local leaders as global leaders

I. R. A. Ferreira, M. K. Ramanathan, A. Grama, and S. Jagannathan: *Randomized protocols for duplicate elimination in peer-to-peer storage systems*

II.Raychoudhury, J. Cao, and W. Wu: Top k-leader election in wireless ad hoc networks.

Existing Solutions (2)

Absolute Slicing

- A regular PSS for normal topology construction
- Inner topology of candidate super-nodes
- Inner-inner topology of size K (using distributed aggregation)

A member of the *Distributed Slicing* family

General purpose

Close to our target

III. A. Montresor and R. Zandonati. Absolute slicing in peer-to-peer systems. In Parallel and Distributed Processing,

Missing requirements:

We need to address real-world issues

- Link construction in the real world is expensive (time)
- Routing in overlays is time expensive
- We don't want to construct additional topologies

We need a self-stabilizing algorithm

- Quick adaptation to changes, without epochs or restarts.

Concepts (1)

Eligibility

- An eligible node is capable of substaining the additional burden of being a Leader
  - E.g. Enough space in hard drive to store a certain content
- Let $\mathbf{E_t}$ be the subset of *eligible* nodes in the network at time **t**.
  - Eligibility changes in time

Goal: Consistency

If no variations occur in the eligible set for a sufficiently long time, each of the leader sets must eventually converge to the same set.

Goal: Adaptiveness

If no variations occur in the eligible set for a sufficiently long time, each of the leader sets must eventually be contained in $\mathbf{E_t}$. In other words, nodes that lose their eligible status must eventually leave the leader set.

Concepts (2)

Goal: Stability

The leader sets must be maintained as stable as possible; i.e., even in the presence of variation of the eligibility set (with new nodes joining the system or nodes outside the leader set leaving it), the leaders set should not vary excessively over time.

Point: minimizing the disruption of the applicative logic which is using the k-leader election service.

E.g. A good node is joining the system.

Goal: Local Reliability

The application must be able to know whether the result is reliable or not. This information must be as up-to-date as possible and should be obtained in a decentralized way.

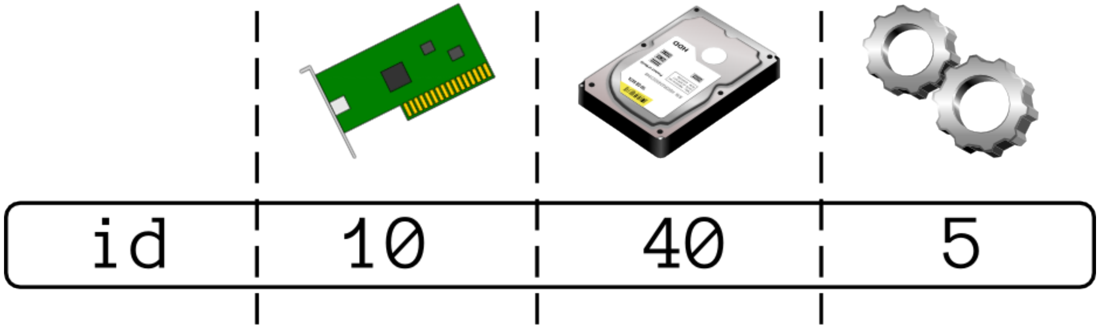
Measure of uniformity of choice of leaders

Point: Each node wants to know if the computation converged, and the result is ready to be used.

Algorithm Description (1)

Idea: decentralized ranking of nodes

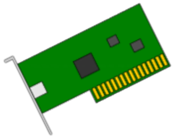
- Every node, periodically, contacts a neighbor and exchanges the owned descriptors (**gossip**)
  - Eligible nodes emit descriptors
  - Descriptors contain the result of a ranking function
- Assumption: a topology management layer provides us with an established connection to a neighbor



Ranking Array

Algorithm Description (2)

- Sorting accordingly to positional significance;
- Keep descriptors only for eligible nodes;
- Keep only the first K entries.

			
05	40	20	4
03	40	19	8
15	30	25	2
07	30	20	2
02	20	55	2

Result: Gossip view

Algorithm Description (3)

Pursuing **Consistency**

- Through **Gossip** descriptors reach all nodes
- Keeping only K entries in the view:
  - Eligible nodes emit their descriptors;
  - Only the best K are maintained after merging

If no variations occur in the eligible set for a sufficiently long time, each of the leader sets must eventually converge to the same set.

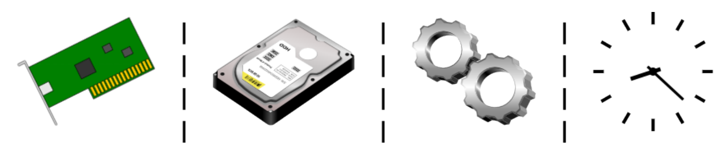
Algorithm Description (4)

Pursuing **Adaptiveness**

- Descriptors are tagged with a timestamp;
  - Incremental integer value
- Recent descriptors override old ones

Note: Issuing a fresh descriptor implies the re-computation of the ranking array

If no variations occur in the eligible set for a sufficiently long time, each of the leader sets must eventually be contained in $\boldsymbol{E}_t$*. In other words, nodes that lose their eligible status must eventually leave the leader set.*



05	40	20	4	5
05	40	5	4	6
03	40	19	8	6
15	30	25	2	2
07	30	20	2	4
02	20	55	2	5

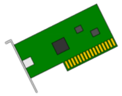
Algorithm Description (5)

Pursuing **Stability**

- Descriptors are enriched with an age counter (milliseconds)
  - Propagation Age Limit (PAL)
  - Basically a TTL
- Unavailable nodes cannot propagate new descriptors
  - The old one will eventually disappear from the network.

Stability can change dramaticaly depending on the ranking function, which is application-dependent

The leader sets must be maintained as stable as possible; i.e., even in the presence of variation of the eligibility set (with new nodes joining the system or nodes outside the leader set leaving it), the leaders set should not vary excessively over time.

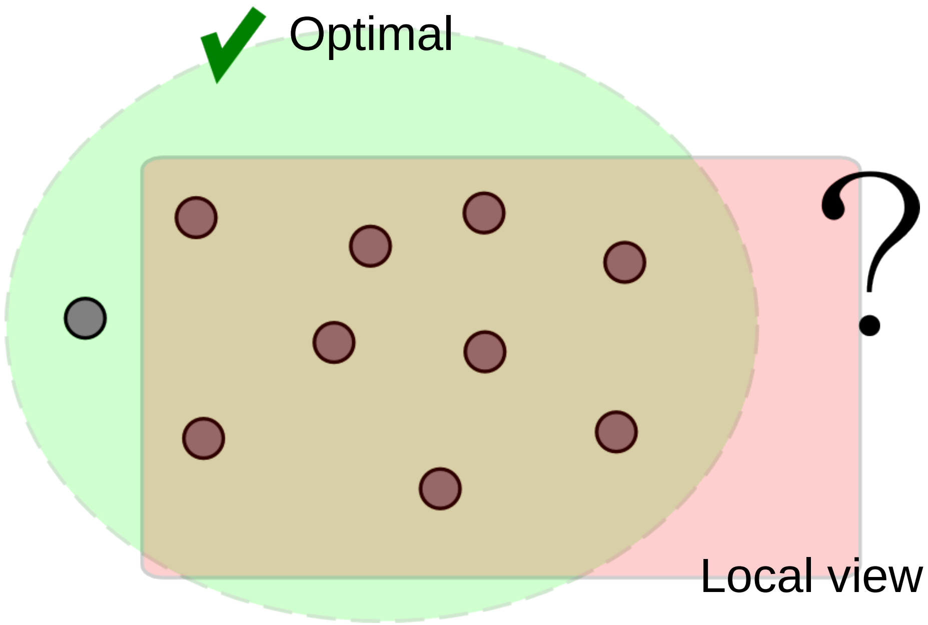
					
05	40	20	4	5	✔
03	40	19	8	6	✔
15	30	25	2	2	✘
07	30	20	2	4	✔
02	20	55	2	5	✔

Algorithm Description (6)

The application must be able to know whether the result is reliable or not. This information must be as up-to-date as possible and should be obtained in a decentralized way

Pursuing **Local Reliability**

- Through a **Quality Measure**
  - 0: Starting point: no knowledge
  - 1: All nodes sharing the same leaders set
  - The value gets averaged over nodes.
- Approximation: obtained through decentralized computation
  - Reliability information directly available for the application, along with result.



K=10, Quality: (K-1)/K = 0.9

Quality measure

- We lack of the optimal leaders set!
  - Quality is improving at each gossip cycle, closer and closer to optimal result
  - Use the next-step improvement as it was the optimal
  - The resulting quality is noisy and over-optimistic:
    - depends on local information.
    - A moving average can be used to smooth it

$$q_{i,0} = \frac{|V_i \cap V_i'|}{k}$$

$$q_{i,1}^{(0)} = 0$$
$$q_{i,1}^{(n)} = \alpha \cdot q_{i,1}^{(n-1)} + (1-\alpha) \cdot q_{i,0}$$

$$\alpha \in [0,1]$$

- **Perceived**, compared with **actual quality**

Implementation

- Using real-world tools
  - Mesmerizer
  - Peerialism testing network
- No interference with the topology
  - Requires a peer sampling system underneath, but completely decoupled from it.
    - PSS as service (layer)
    - Asking for a neighbor to gossip with
    - Using WPSS, a NAT aware protocol.
      - **Credits:** Roberto Roverso

Improvements

- ✓ Three-way gossip session, based on descriptors freshness
  1. Blind send
  2. Savvy response
  3. Savvy termination
- ✓ Open-Internet override
  - Optimization for WPSS, where public nodes converge quickly
  - Overriding procedure as quality gets closer to 1
  - **Credits:** *Alberto Montresor*

Simulation & Results

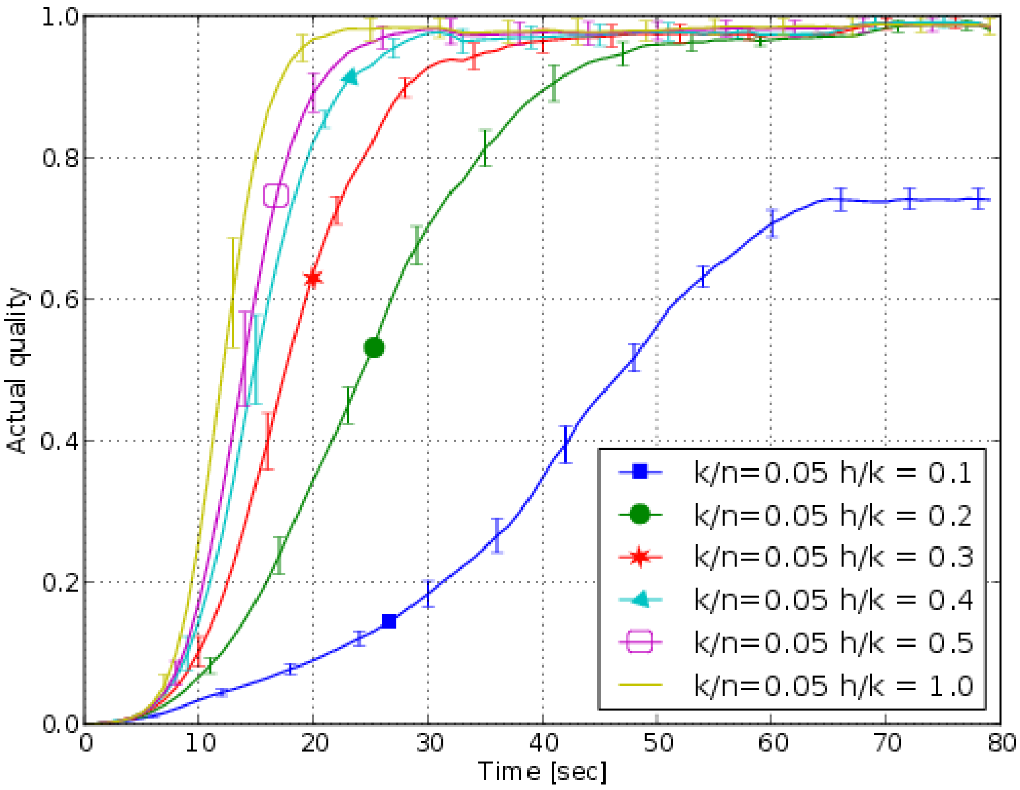
Algorithm Parameters

- K and T (period)
- h (number of shared descriptors per gossip session)
- α (smoothing factor)
- PAL (Propagation Age Limit)
- OQT (Override Quality Threshold)
- Simple ranking function: for each node, choose a random value.

Parameters were studied in Simulation

- N = 1000
- Different classes of churn, X% of nodes joining/leaving the network within 10 seconds
  - $X \in \{0.3, 0.5, 1\}$
- Behavior with different ratios K/N and h/K.
- Best value for α
- Best PAL

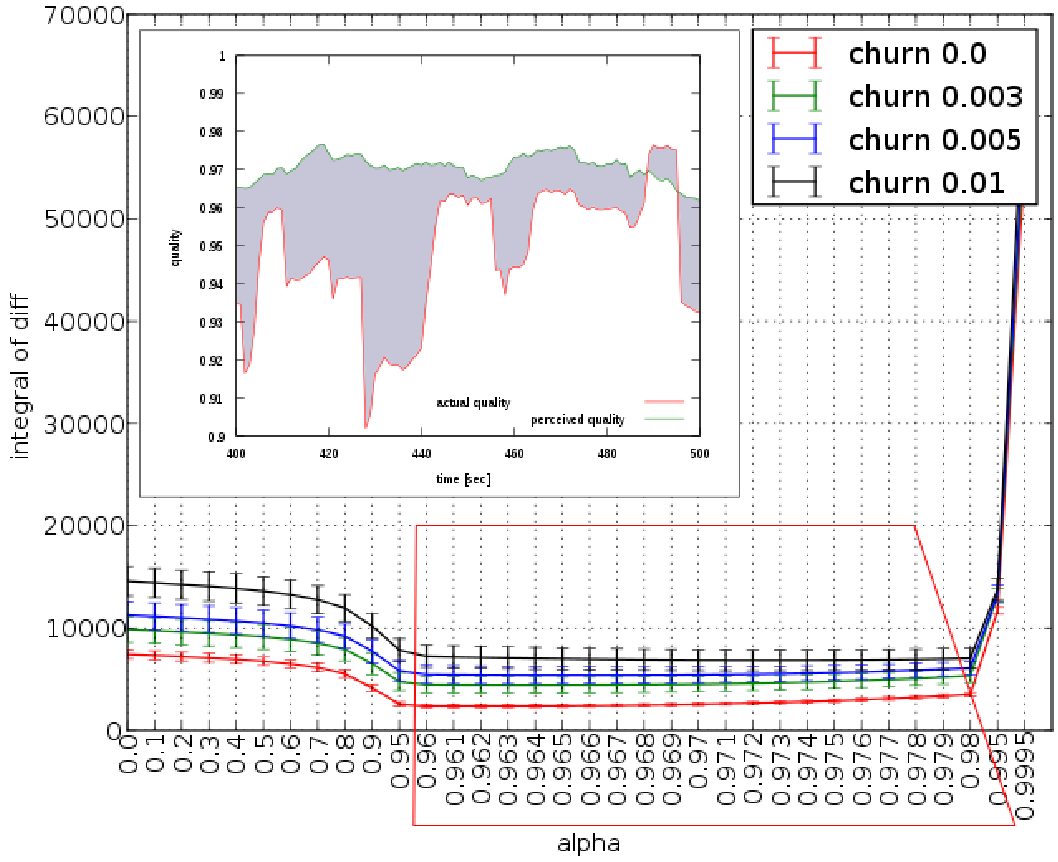
Convergence time



Time for reaching a good result quality (**actual quality**)

- In simulation, different values of K and h
- Estimated convergence time, around 20 seconds

Parameter study for Alpha



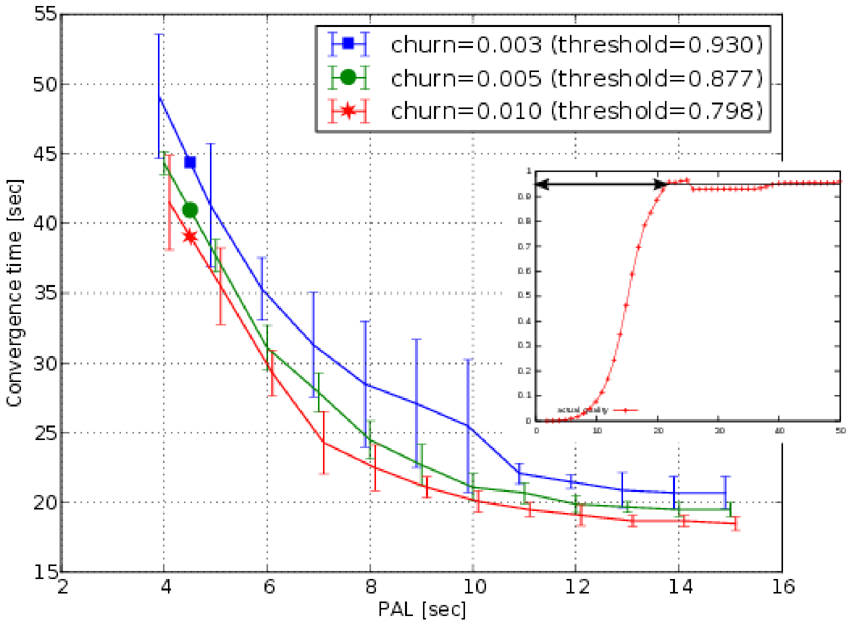
Smoothing factor: tuning for obtaining an accurate quality estimation

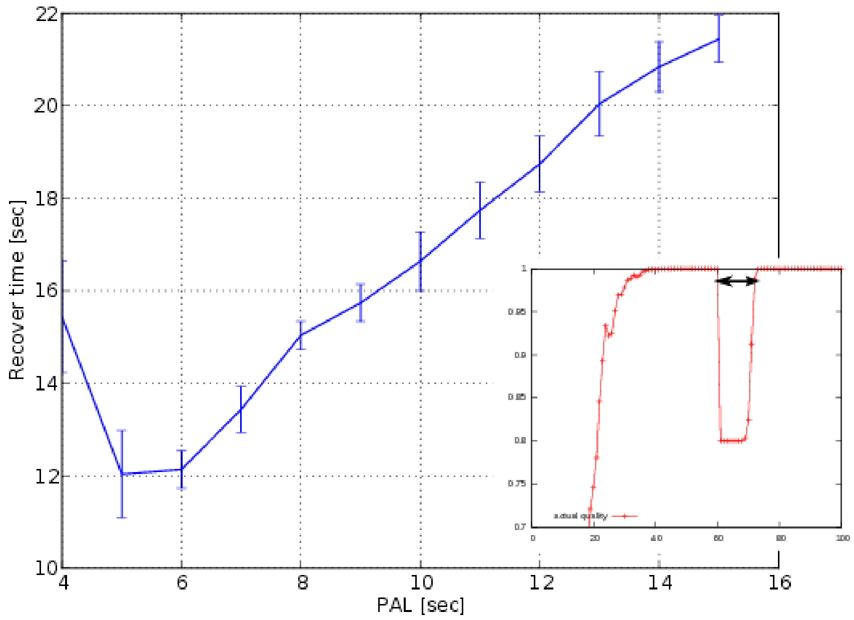
- Difference function between **perceived** and **actual quality**
- Minimization of the integral, with different classes of churn.
- A good range for α is the 0.95-0.98 interval

Parameter study for PAL

Propagation Age Limit, remove old information from the system

- Higher convergence time if too short
- Long-lived outdated information if too long
- Finding some good trade-off: around 12 seconds

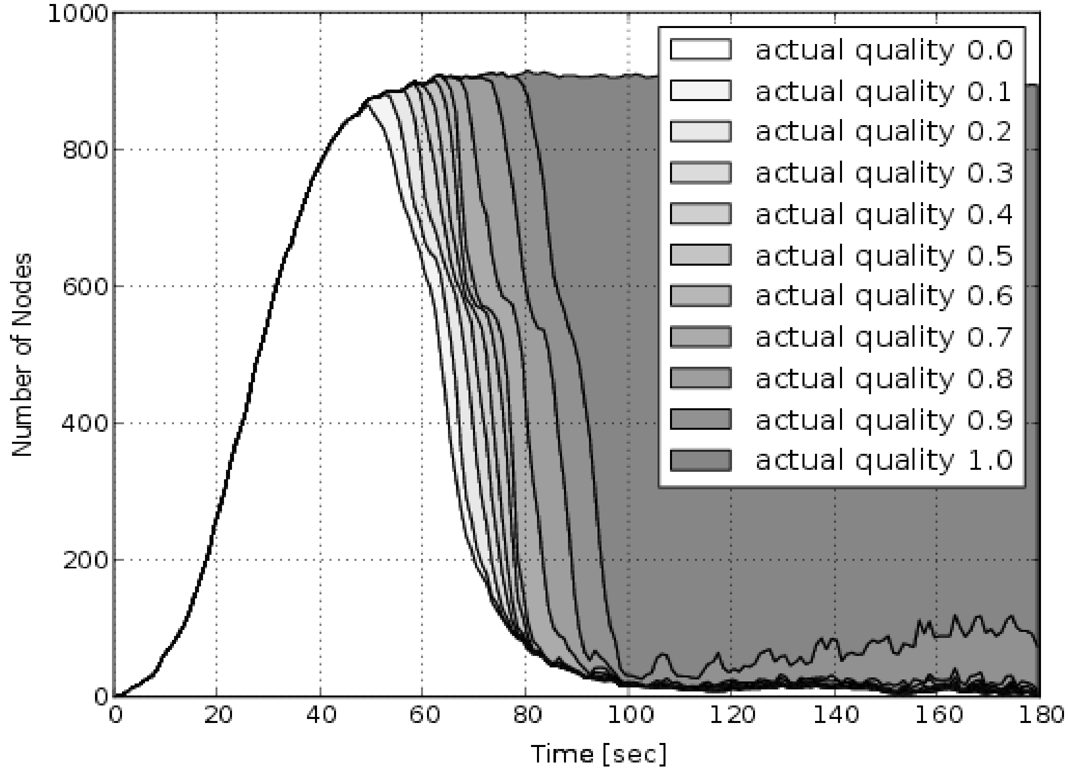




Deployment & Results

Deployed on testing facility in Peerialism

- Experiments up to N=1000, 20% of which Open Internet
- Selection of parameters:
  - K=10, h=K, T=1sec
  - PAL=12sec
  - A=0.95
- **Note: different starting time**



Here OQT (Override Quality Threshold) was added as parameter

Quality Threshold Percent	Quality Threshold Value	Convergence Time (seconds) μ	σ
90%	0.874442513738	56.4706454741	11.4602132189
97.5%	0.947312723216	62.2236587216	12.1045159194

Conclusions

- Strength Points
  - Working on Real-world scenario
  - Resilient to local/global dynamics
  - Self-evaluating
  - Self-stabilizing
- Weak points
  - Needs additional esperimentation, with actual application logic
  - Room for improvement

Submitted to ICDCS 2014, waiting for feedback.