introduction

Churn: an important factor for most distributed systems

Turnover causes dropped requests, increased bandwidth, ...

Would like to optimize for stability

Select which nodes to use
Past work uses heuristics for specific systems

Our goal: a general study of minimizing churn

How can we select nodes to minimize churn?

Can we characterize how existing systems select nodes and the impact on their performance?
contents

• an example system
• evaluation of node selection strategies
  (how can we minimize churn?)
• applications
  (how do existing systems select nodes?)
• conclusions
example: overlay multicast

Join:

- Consider $m$ random nodes with # children $<$ max
- Pick one as parent to minimize latency to root
example: overlay multicast
example: overlay multicast

Latency to root (ms)

Interruptions per node per day

Nodes considered when picking parent (m)

+86%
example: overlay multicast

In terms of interruption rate,

Random Replacement of parent (m=1) better than Preference List selection (large m)

Why?
contents

• an example system

• **evaluation of node selection strategies**
  
  *(how can we minimize churn?)*

• applications
  
  *(how do existing systems select nodes?)*

• conclusions
the core problem

Node selection task

- *n* nodes available
- pick *k* to be “in use”
- when one fails, pick a replacement

Minimize **churn**: rate of change in set of in-use nodes
For each node:

Intuition: when a node joins or leaves a DHT, $\frac{1}{k}$ of stored objects change ownership

...then divide by runtime

$\text{churn} += \frac{1}{k}$

$k = \# \text{ of nodes in use}$
node selection strategies

- **Predictive**
  - Longest uptime
  - Most available
  - Max expectation
  - ...

- **Agnostic**
  - Random Replacement
  - Preference List
agnostic selection strategies

- **Random Replacement**
  - Select random available node to replace failed node

- **Passive Preference List**
  - Rank nodes (e.g. by latency);
  - Select most preferred as replacement;
  - ...and switch to more preferred nodes when they join

- **Active Preference List**
  - Pref List is:
    - (1) essentially static across time
    - (2) essentially unrelated to churn
    - ...and switch to more preferred nodes when they join
evaluation

churn

passive PL

1.2-3 ×

random replacement

1.2-2.2 ×

longest uptime, max expectation

active PL

2.5-8 ×

Why such a difference?

...even though neither uses history?
evaluation

5 traces of node availability

- PlanetLab [Stribling 2004-05]
- Web sites [Bakkaloglu et al 2002]
- Microsoft PCs [Bolosky et al 2000]
- Skype superpeers [Guha et al 2006]
- Gnutella peers [Saroiu et al 2002]

Main conclusions held in all cases
evaluation: PlanetLab trace

![Graph showing churn and fraction of nodes in use for different strategies.](image)

- Active PL
- Passive PL
- RR
- Max Exp

Churn (turnover per day)
- Fraction of nodes in use
intuition: PL

- uses the top $k$ nodes in the preference list
- preference list unrelated to stability
- failure rate is about mean node failure rate

<--- becomes more and more true for Passive as $k$ increases
**intuition: RR**

- RR like picking a node at a **random time**

  ![Diagram](image)

  - **selected**
  - **TTF**
  - **Time**

- Long sessions occupy more time (trivially)

- So, RR biased towards landing in longer sessions

- Failure rate can be **arbitrarily lower than mean**

An example of the classic "inspection paradox"
RR vs. PL: analysis

\[ E[C] = \frac{2}{\alpha d} \sum_{i=1}^{d} \frac{1}{\mu_i} \left( 1 - E \left[ \exp \left\{ -\frac{\alpha}{2(1-\alpha)} E[C] \cdot L_i \right\} \right] \right) \]

- Churn of RR decreases as session time distributions become “more skewed” (\(\Rightarrow\) higher variance)
- RR can never have more than 2x the churn of PL strategies
contents

• an example system

• evaluation of node selection strategies
  \( \textit{how can we minimize churn?} \)

• \textbf{applications}
  \( \textit{how do existing systems select nodes?} \)

• conclusions
applications of RR & PL

- anycast
- DHT replica placement
- overlay multicast
- DHT neighbor selection
overlay multicast

![Graph showing latency to root (ms) and interruptions per node per day against nodes considered when picking parent (m).]

Two separate effects of increasing m:

1. Tree becomes more balanced (small decrease in interruptions)
2. Move from RR- to PL-like strategy (big increase)
a peek inside the tree

![Graph showing failures per day vs. depth in tree with two lines:
- m = 1 (random selection)
- m = n (latency-optimized)]
overlay multicast notes

- Basic framework from [Sripanidkulchai et al SIGCOMM’04]
- Found random parent selection surprisingly good
- Tested 2 other heuristics to minimize interruptions
  - Both can perform better with some randomization!
DHT neighbor selection

- Active PL strategy for selecting each finger
- Preference List arises accidentally

Standard Chord topology
DHT neighbor selection

Divide keyspace into 1/2, 1/4, 1/8, ...

Finger points to random key within each interval
DHT neighbor selection

 Datagram-level simulation, i3 Chord codebase, Gnutella trace

![Graph showing fraction of requests failed vs. average number of nodes in DHT]

- Standard Chord topology
- Randomized Chord topology

Easy 29% reduction at $n = 850$
contents

• an example system

• evaluation of node selection strategies
  (how can we minimize churn?)

• applications
  (how do existing systems select nodes?)

• conclusions
conclusions

A guide to minimizing churn

- RR is pretty good; PL much worse
- RR and PL arise in many systems

Design insights

- watch out for (implicit) PL strategies
- easy way to reduce churn: add some randomness

doing less work may improve performance!
backup slides
Why use RR?

- Simplicity: no need to monitor and disseminate failure data
- Robustness to self-interested peers
- Legacy systems