Dynamic Hypercube Topology

Stefan Schmid

URAW 2005
Upper Rhine Algorithms Workshop

University of Tübingen, Germany
Static vs. Dynamic Networks (1)

- **Network graph** \( G=(V,E) \)
  - \( V \) = set of vertices (“nodes”, machines, peers, …)
  - \( E \) = set of edges (“connections”, wires, links, pointers, …)

- “Traditional”, **static networks**
  - Fixed set of vertices, fixed set of edges
  - E.g., interconnection network of parallel computers

![Parallel Computer](image1)

![Fat Tree Topology](image2)
Static vs. Dynamic Networks (2)

• Dynamic networks
  – Set of nodes and/or set of edges is dynamic
  – Here: nodes may join and leave
  – E.g., peer-to-peer (P2P) systems (Napster, Gnutella, …)
Dynamic Peer-to-Peer Systems

Peer-to-Peer Systems

- cooperation of many machines (to share files, CPU cycles, etc.)
- usually desktop computers under control of individual users
- user may turn machine on and off at any time
- => Churn

How to maintain desirable properties such as connectivity, network diameter, node degree, ...?
Talk Overview

• Model

• Ingredients: basic algorithms on hypercube graph

• Assembling the components

• Results for the hypercube

• Conclusion, generalization and open problems

• Discussion
Model (1): Network Model

- Typical P2P overlay network
  - Vertices $v \in V$: peers (dynamic: may join and leave)
  - Directed edges $(u,v) \in E$: $u$ knows IP address of $v$ (static)

- Assumption: Overlay network builds upon complete Internet graph
  - Sending a message over an overlay edge => routing in the underlying Internet graph

![Diagram of overlay network]
Model (2): Worst-Case (Adversarial) Dynamics

- Model worst-case faults with an adversary $\text{ADV}(J,L,\lambda)$

- $\text{ADV}(J,L,\lambda)$ has complete visibility of the entire state of the system

- May add at most $J$ and remove at most $L$ peers in any time period of length $\lambda$
Model (3): Communication Rounds

• Our system is synchronous, i.e., our algorithms run in rounds
  – One round: receive messages, local computation, send messages

• However: Real distributed systems are asynchronous!

• But: Notion of time necessary to bound the adversary
Overview of Dynamic Hypercube System

- Idea: Arrange peers into a *simulated hypercube* where each node consists of several (logarithmically many) peers!
  - Gives a certain redundancy and thus time to react to changes.
  - But still guarantees diameter $D = O(\log n)$ and degree $\Delta = O(\log n)$, as in the normal hypercube ($n =$ total number of peers)!
Ingredients for Fault-Tolerant Hypercube System

Simulation: Node consists of several peers!

Basic components:

- Route peers to sparse areas
  Token distribution algorithm!

- Adapt dimension
  Information aggregation algorithm!
Components: Peer Distribution and Information Aggregation

Peer Distribution
- Goal: Distribute peers evenly among all hypercube nodes in order to balance biased adversarial churn
- Basically a token distribution problem

Counting the total number of peers (information aggregation)
- Goal: Estimate the total number of peers in the system and adapt the dimension accordingly
Dynamic Token Distribution Algorithm (1)

Algorithm: Cycle over dimensions and balance!

1: (* algorithm running on node $b_0...b_{d-1}$ *)
2: $my\_id := b_0...b_{d-1}$;
3: $T_{my\_id}$ := tokens at this node;
4: for $i := 0$ to $d - 1$ do
5:     $buddy\_id := b_0...b_{i-1}b_{i+1}...b_{d-1}$;
6:     SEND $|T_{my\_id}|/2$ tokens to node $buddy\_id$;
7:     update $T_{my\_id}$ accordingly;
8:     $T_{buddy\_id}$ := REVC tokens from node $buddy\_id$;
9:     $T_{my\_id} := T_{my\_id} \cup T_{buddy\_id}$;
10: end for

Perfectly balanced after $d$ steps!
Dynamic Token Distribution Algorithm (2)

• Problem 1: Peers are not fractional!

• However, by induction, the integer discrepancy is at most \( d \) larger than the fractional discrepancy.

\[
|v|_{t+1}^{int} \leq \left|\frac{|v|^{int}_t + |u|^{int}_t}{2}\right| \leq \left|\frac{|v|^{frac}_t + \frac{i}{2} + |u|^{frac}_t + \frac{i}{2}}{2}\right|
\]

\[
\leq \frac{|v|^{frac}_t + \frac{i}{2} + |u|^{frac}_t + \frac{i}{2}}{2} + \frac{1}{2}
\]

\[
\leq \frac{|v|^{frac}_t + |u|^{frac}_t + i + 1}{2} = |v|^{frac}_{t+1} + \frac{i + 1}{2}.
\]
Dynamic Token Distribution Algorithm (3)

- Problem 2: An adversary inserts at most $J$ and removes at most $L$ peers per step!

- Fortunately, these dynamic changes are balanced quite fast (geometric series).

$$J_t + \frac{J_{t-1}}{2} + \frac{J_{t-2}}{4} + \ldots + \frac{J_{t-(d-1)}}{2^{d-1}} + \frac{J_{t-d}}{2^d} + \frac{J_{t-(d+1)}}{2^d} + \frac{J_{t-(d+2)}}{2^d} + \ldots$$

  \begin{align*}
  &< 2J \\
  &\text{shared by all nodes}
  \end{align*}

- Thus

Theorem 1: Given adversary $ADV(J,L,1)$, discrepancy never exceeds $2J+2L+d$!
Excursion: Randomized Token Distribution

- Again the static case, but this time assign “dangling” token to one of the edge’s vertices at random

- “Randomized rounding”

- Dangling tokens are binomially distributed => Chernoff lower tail

Theorem 2: The expected discrepancy is constant (~ 3)!
Components: Peer Distribution and Information Aggregation

Peer Distribution

- Goal: Distribute peers evenly among all hypercube nodes in order to balance biased adversarial churn
- Basically a token distribution problem

Counting the total number of peers (information aggregation)

- Goal: Estimate the total number of peers in the system and adapt the dimension accordingly

Tackled next!
Information Aggregation Algorithm (1)

• Goal: Provide the same (and good!) estimation of the total number of peers presently in the system to all nodes
  – Thresholds for expansion and reduction

• Means: Exploit again the recursive structure of the hypercube!
Information Aggregation Algorithm (2)

Algorithm: Count peers in every sub-cube by exchange with corresponding neighbor!

Correct number after $d$ steps!
Information Aggregation Algorithm (3)

- But again, we have a concurrent adversary!

- Solution: Pipelined execution!

Theorem 3: The information aggregation algorithm yields the same estimation to all nodes. Moreover, this number represents the correct state of the system \(d\) steps ago!
Composing the Components

- Our system permanently runs
  - Peer distribution algorithm to balance biased churn
  - Information aggregation algorithm to estimate total number of peers and change dimension accordingly

But: How are peers connected inside a node, and how are the edges of the hypercube represented?
Intra- and Interconnections

• Peers inside the same hypercube vertex are connected \textit{completely} (clique).

• Moreover, there is a matching between the peers of neighboring vertices.
Maintenance Algorithm

- Maintenance algorithm runs in *phases*
  - Phase = 6 rounds

- In phase $i$:
  - *Snapshot* of the state of the system in round 1
  - One exchange to estimate number of peers in sub-cubes (information aggregation)
  - Balances tokens in dimension $i \mod d$
  - Dimension change if necessary

All based on the snapshot made in round 1, ignoring the changes that have happened in-between!
Results for Hypercube Topology

• Given an adversary $ADV(d+1,d+1,6)\ldots$
  => Peer discrepancy at most $5d+4$ (Theorem 1)
  => Total number of peers with delay $d$ (Theorem 3)

• ... we have, in spite of $ADV(O(\log n), O(\log n), 1)$:
  – always at least one peer per node,
  – peer degree bounded by $O(\log n)$ (asymptotically optimal!),
  – network diameter $O(\log n)$. 
A Blueprint for Many Graphs?

• Conclusion: We have achieved an asymptotically optimal fault-tolerance for a $O(\log n)$ degree and $O(\log n)$ diameter topology.

• Generalization? We could apply the same tricks for general graphs $G=(V,E)$, given the ingredients (on $G$):
  – token distribution algorithm
  – information aggregation algorithm

• For instance: Easy for skip graphs ($\Delta = D = O(\log n)$), possible for pancake graphs ($\Delta = D = O(\log n / \log \log n)$).
Open Problems

• Experiences with other graphs?

• Other models for graph dynamics?

• Less messages?

Thank you for your attention!
Discussion

• Questions?
• Inputs?
• Feedback?