PlanetenWachHundNetz: Locality-Aware MapReduce for Peer-to-Peer

RSG
Robust Systems Group

Vitus Lorenz-Meyer, Eric Freudenthal

University of Texas at El Paso

Motivation

- Efficient reduction of data from self-organized (P2P) sources

Problem

- Dynamic membership
  - Static tree (for parallel prefix) inappropriate

Problems w/ Known Techniques

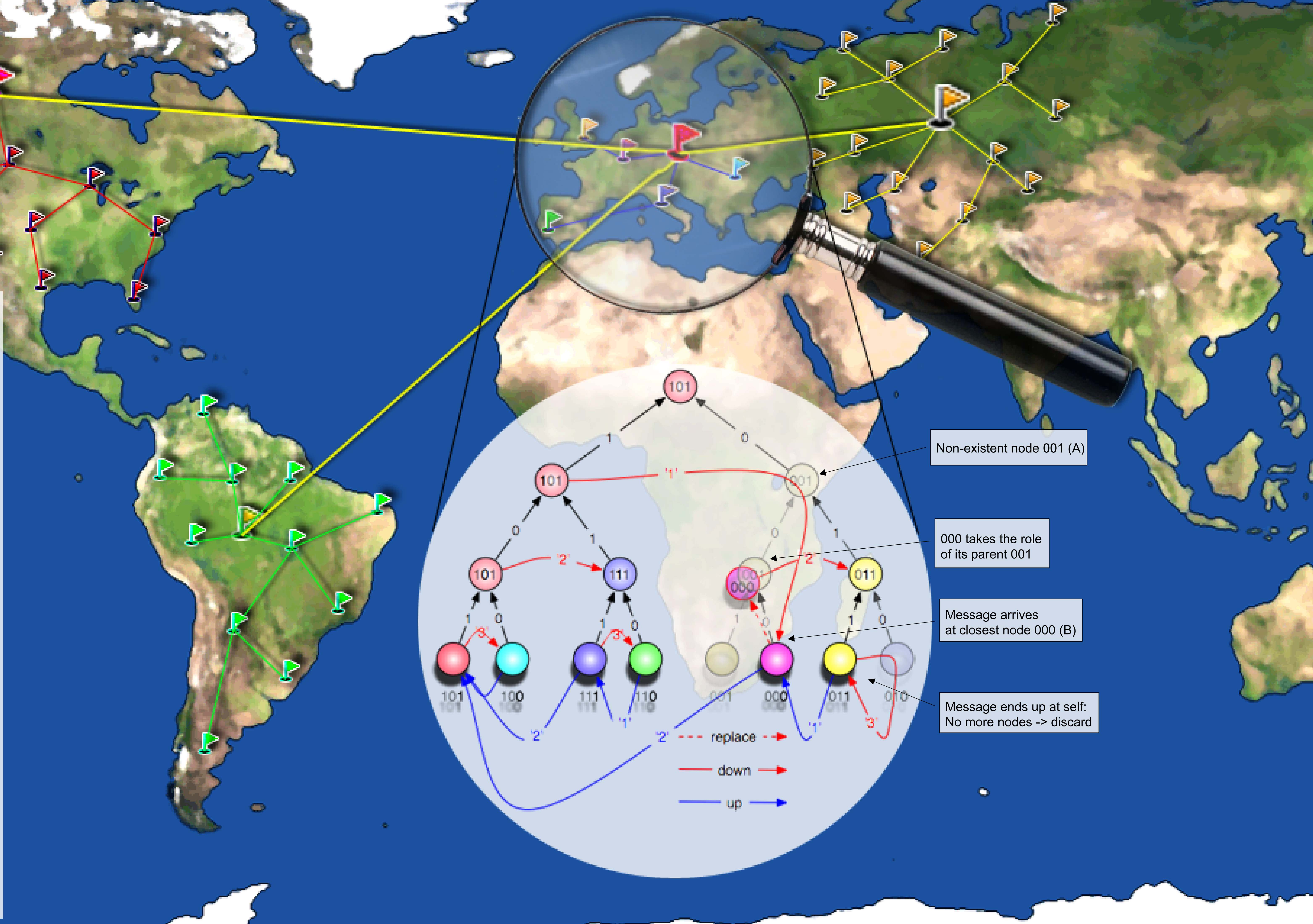
- Do not localize network traffic
- Branching factor not controlled
- State lost when parent nodes fail/leave
- Details on right side of figure

Our Goals

- Exploit locality, minimize remote communication
- Efficient parallelism: Control of branching factor
- Minimal state-loss when nodes enter/leave
- Self-healing continuous queries

Implementation

- Left-tree
  - Node role determined by key
- Locality/clustering: Coral or Oasis



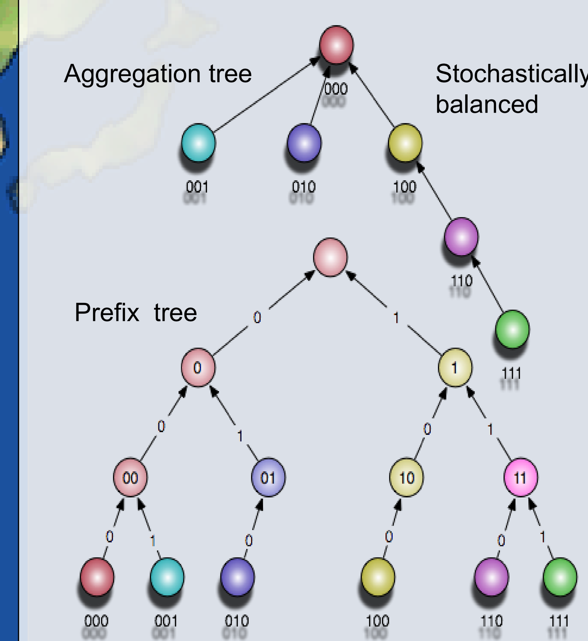
Other Approaches

From distributed databases

Key-based Tree

Routing based on node keys

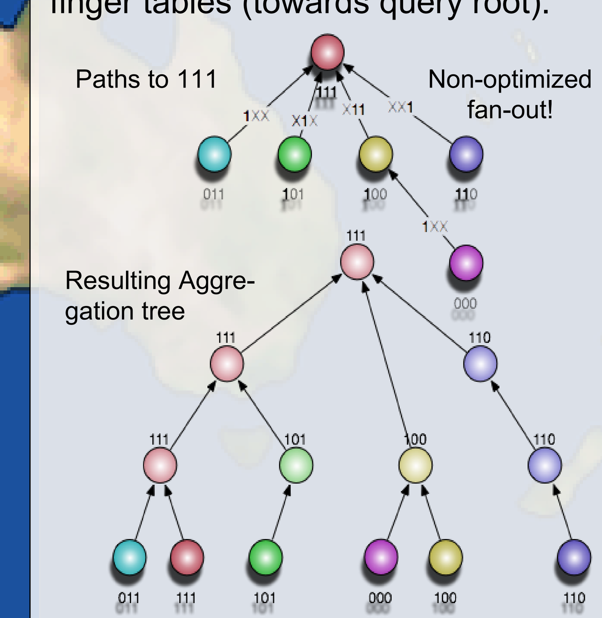
- Children of node *n* with key *k* are nodes with keys closest to *k* with *n-1* prefix bits in common with *k*
- *Node n is its own child!*



•Not locality aware!
•Only 1 global tree.

Finger-table based tree

- Tree edges selected from nodes finger tables (towards query root).



•Unique tree for every query
•Locality awareness from DHT
•but does not force locality
•Challenge: what to do if nodes enter/leave

MapReduce (Google)

Generalized Parallel Prefix

- Associative & communitive operations mapped to an aggregation tree
- Google's map-reduce framework
- Data is *mapped* to low-dimensional tuples
- Tuples are *recursively combined* using an associative *reduction* algorithm that emits a (summary) tuple

Example:

- Counting occurrence of a particular word in a document:
  - Provide *map* algorithm that emits the tuple '(1)' for every occurrence
  - Provide a *reduce* algorithm that sums these tuples

Locality-aware Clustering

Coral (locality-aware DsHT)

- Coral partitions its members in clusters based on connection latency

Aggregation tree at each cluster

- Build *cluster* aggregation trees
- A single member of a cluster-tree serves as aggregation representative

Challenges

- Choosing sub-cluster representatives
- Constructing aggregation tree for sub-cluster
- Avoiding inefficient setup broadcast

Goal: Efficient Aggregation Tree

Our Solution: Key-based MapReduce (KMR)

Overview

- Associate aggregation tree with a operation-specific *root key*.

Characteristics

- Rooted at host whose key is *closest* to root key
  - One tree per query, distributes load over all nodes for multiple queries at a time
  - Nodes that are roots of sub-trees are their own left-children all the way down to their appropriate place among leaves.
  - Each node is leaf and root of the sub-tree at depth *d* if it has the *d*'th bit of root-ID flipped; node may also substitute for missing parents.
- Set of nodes and MR-specific root-key fully defines a unique tree (complete knowledge not required: lookup parent, sibling, children)

Building the Tree

Send build-message to all siblings down the tree
Each node forwards build-message to all its siblings all the way down

- Recall that node is its own child: siblings include its children!

Non-existing parent

- Closest node fulfills its role
- Therefore, search for parent will discover this node

Non-existing child

- If child not in DHT, then node is a leaf

Finding children

- Use DHT

Finding nodes

n = key of some node
t = root-id of aggregation tree
π = n's position among leaves (from left)
$\pi = n \oplus t$
key of n's parent *at height h*: $\pi / 2^h \oplus t$

To find parent, *n* searches for smallest height such that parent is in the tree.

Collecting data

- Each node reduces messages from children, sends result(s) to parent

Challenges	Approach
• Find children	DHT lookup
• Find parent	DHT lookup
• Establishing new tree at exit/departure	DHT lookup (key-space determines topology)

Continuous Queries

Maintaining tree consistency under churn

- Each node periodically checks roles of self and immediate relatives
- Churn (membership change) must result in tree-node role change
  - Change propagated by DHT finger tables

Challenges	Approach
• What roles to check when	periodically check (lookup) self, children, parent only when its assuming role
• Who notifies who	parent → child, old → new