Mapping a Dynamic Prefix Tree on a P2P Network

Eddy Caron, Frédéric Desprez, Cédric Tedeschi

GRAAL WG - October 26, 2006
Outline

1. Introduction
2. Related Work
3. DLPT architecture
4. Mapping
5. Conclusion
Context

- Resource discovery in grid context
- New needs facing the development of grids
  - large scale
  - no central infrastructure
  - dynamic joins and leaves of nodes
- Adopt peer-to-peer technologies
  - Pure decentralized algorithms
  - Scalable algorithms to retrieve objects
  - Fault-tolerance
Context

- Resource discovery in grid context
- New needs facing the development of grids
  - large scale
  - no central infrastructure
  - dynamic joins and leaves of nodes
- Adopt peer-to-peer technologies
  - Pure decentralized algorithms
  - Scalable algorithms to retrieve objects
  - Fault-tolerance
Context

- Resource discovery in grid context
- New needs facing the development of grids
  - large scale
  - no central infrastructure
  - dynamic joins and leaves of nodes
- Adopt peer-to-peer technologies
  - Pure decentralized algorithms
  - Scalable algorithms to retrieve objects
  - Fault-tolerance
Outline

1. Introduction
2. Related Work
3. DLPT architecture
4. Mapping
5. Conclusion
P2P technologies

- Unstructured P2P approaches
  - flooding based
  - non-exhaustive researches

- Distributed Hash Tables
  - key-based routing
  - exhaustive search
  - scalable:
    - logarithmic local state
    - logarithmic number of hops
  - fault-tolerance
    - periodic scanning
    - replication
  - drawbacks
    - no locality awareness
    - assumptions of homogeneity
    - only exact match queries
P2P technologies

- Unstructured P2P approaches
  - flooding based
  - non-exhaustive researches

- Distributed Hash Tables
  - key-based routing
  - exhaustive search
  - scalable:
    - logarithmic local state
    - logarithmic number of hops
  - fault-tolerance
    - periodic scanning
    - replication
  - drawbacks
    - no locality awareness
    - assumptions of homogeneity
    - only exact match queries
Range queries
- automatic completion
- logarithmic Latency

Approaches
- Skip Graphs (complexities)
- Nodewiz (no fault-tolerance)
- Prefix Hash Tree (static trie)
- P-Grid (static trie)
- locality awareness issue
- DLPT (load balancing)
Trie Based Lookup (2/2)

- Range queries
  - automatic completion
  - logarithmic Latency

- Approaches
  - Skip Graphs (complexities)
  - Nodewiz (no fault-tolerance)
  - Prefix Hash Tree (static trie)
  - P-Grid (static trie)
  - locality awareness issue
  - DLPT (load balancing)
Trie Based Lookup (2/2)

- Range queries
  - automatic completion
  - logarithmic Latency

- Approaches
  - Skip Graphs (complexities)
  - Nodewiz (no fault-tolerance)
  - Prefix Hash Tree (static trie)
  - P-Grid (static trie)
  - locality awareness issue
  - DLPT (load balancing)
A two layer architecture

- Logical indexing structure
- On-line building of a Greatest Common Prefix (GCP) Tree
- Distributed traversal algorithms
- **Mapping on a dynamic network**
  - random DHT-based mapping (no load balancing)
  - each physical node maintains one or more logical nodes
- Replication based fault-tolerance
- Greedy locality awareness
GCP Tree - Preliminaries

- Alphabet $A$ finite set of letters
- $\prec$ an order on $A$
- Word $w$ finite set of letters of $A$, $w = a_1, \ldots, a_i, \ldots, a_l$, $l > 0$
- $u, v$ two words, $uv$ concatenation of $u$ and $v$
- $|w|$ length of $w$
- $\epsilon$ the empty word, $|\epsilon| = 0$
GCP Tree - Definition

- $u = \text{prefix}(v)$ if $\exists w$ s.t. $v = uw$
- $GCP(w_1, w_2, \ldots, w_i, \ldots, w_n)$ is the longest prefix shared by $w_1, w_2, \ldots, w_i, \ldots, w_n$
- Example:
  - $\text{DTR} = \text{prefix}(\text{DTRSM})$
  - $GCP(\text{DTRSM}, \text{DTRMM}) = \text{DTR}$
- GCP Tree labeled rooted tree s.t.
  - The node label is a proper prefix of any label in its subtree
  - The node label is the Proper Greatest Common Prefix of all its son labels
GCP Tree - Dynamic construction

- "real" key - attribute declared
- "virtual" key - created by construction

Contact
Routing
Inserting
GCP Tree - worst case complexities

- Number of hops bounded by twice the depth of the tree
- Depth of the tree bounded by the size of the words
- Local state bounded by the number of characters
- Constant time local decision of routing
- Range query, replication/locality process
  - latency bounded by the depth of the tree
  - linear number of messages
Current mapping

- Random
- No Load balancing
- Cost of maintaining both physical and logical level
  - \( \Rightarrow \) Reduction the total communication cost
  - \( \Rightarrow \) Load balancing heuristics
Current mapping

- Random
- No Load balancing
- Cost of maintaining both physical and logical level
  - \(\Rightarrow\) Reduction the total communication cost
  - \(\Rightarrow\) Load balancing heuristics
General design (1/2)

- **The physical layer**
  - Structured as a ring dynamically growing
  - Each *peer* is placed by a *lexicographic* hash function
  - Each peer maintains a *predecessor* and a *successor*

- **The logical layer**
  - Dynamic GCP Tree (built as objects are declared)
  - Each node is mapped on its *successor peer*
General design (2/2)
Inserting a physical node - principle

- Finding the successor peer \(\equiv\) finding the target node (labeled by the greatest ID smaller than the new peer ID)

- 3 phases
  - 0. Not in the right branch: going up
  - 1. In the right branch: routing down
  - 2. Inserting: the successor searched is
    - the peer hosting the target node
    - the successor of the peer hosting the target node
Inserting an object

- Routing according to the object’s key
- Potential creation of new nodes
- Finding peers to host new nodes
Load balancing heuristics - related work

- Karger and Ruhl
  - periodic random item balancing
  - homogeneity of peer capacities
- Godfrey et al.
  - periodic item redistribution
  - semi-centralized
- Ledlie and Seltzer
  - K-choices
  - Chooses the best location for a new peer among $k$
Load balancing - Max Local Throughput

- $C_p$ capacity of the peer $p$
- $l_n$ load of the node $n$
- considering two adjacent peers $s$ and $p$
- $I_s$ set of nodes currently managed by $s$
- $I_p$ set of nodes currently managed by $p$
- $T = T_p + T_s$
- Considering $n$ nodes, $n = |I_s| + |I_p|$
- Find $k$ such that

$$
\min\left(\sum_{i \in \{0,...,k\}} l_i, C_p\right) + \min\left(\sum_{i \in \{k+1,...,n\}} l_i, C_s\right)
$$

is maximum (algorithm linear in the number of nodes)
Load balancing - Max Local Throughput

- $C_p$ capacity of the peer $p$
- $l_n$ load of the node $n$
- considering two adjacent peers $s$ and $p$
- $I_s$ set of nodes currently managed by $s$
- $I_p$ set of nodes currently managed by $p$
- $T = T_p + T_s$
- Considering $n$ nodes, $n = |I_s| + |I_p|$
- Find $k$ such that

$$
\min\left(\sum_{i \in \{0, \ldots, k\}} l_i, C_p\right) + \min\left(\sum_{i \in \{k+1, \ldots, n\}} l_i, C_s\right)
$$

is maximum (algorithm linear in the number of nodes)
Load balancing - Max Local Throughput

- $C_p$ capacity of the peer $p$
- $l_n$ load of the node $n$
- Considering two adjacent peers $s$ and $p$
- $I_s$ set of nodes currently managed by $s$
- $I_p$ set of nodes currently managed by $p$
- $T = T_p + T_s$
- Considering $n$ nodes, $n = |I_s| + |I_p|$
- Find $k$ such that

$$\min\left(\sum_{i \in \{0, \ldots, k\}} l_i, C_p\right) + \min\left(\sum_{i \in \{k+1, \ldots, n\}} l_i, C_s\right)$$

is maximum (algorithm linear in the number of nodes)
Load balancing - Max Local Throughput

- $C_p$ capacity of the peer $p$
- $l_n$ load of the node $n$
- considering two adjacent peers $s$ and $p$
- $\mathcal{I}_s$ set of nodes currently managed by $s$
- $\mathcal{I}_p$ set of nodes currently managed by $p$
- $T = T_p + T_s$
- Considering $n$ nodes, $n = |\mathcal{I}_s| + |\mathcal{I}_p|$
- Find $k$ such that

$$\min\left( \sum_{i \in \{0,\ldots,k\}} l_i, C_p \right) + \min\left( \sum_{i \in \{k+1,\ldots,n\}} l_i, C_s \right)$$

is maximum (algorithm linear in the number of nodes)
Load balancing - Max Local Throughput

- $C_p$ capacity of the peer $p$
- $l_n$ load of the node $n$
- considering two adjacent peers $s$ and $p$
- $I_s$ set of nodes currently managed by $s$
- $I_p$ set of nodes currently managed by $p$

\[
T_p = \min\left(\sum_{i \in I_p} l_i, C_p\right), \quad T_s = \min\left(\sum_{i \in I_s} l_i, C_s\right)
\]

- $T = T_p + T_s$
- Considering $n$ nodes, $n = |I_s| + |I_p|$
- Find $k$ such that

\[
\min\left(\sum_{i \in \{0,\ldots,k\}} l_i, C_p\right) + \min\left(\sum_{i \in \{k+1,\ldots,n\}} l_i, C_s\right)
\]

is maximum (algorithm linear in the number of nodes)
Load balancing - Max Local Throughput

- \( C_p \) capacity of the peer \( p \)
- \( l_n \) load of the node \( n \)
- considering two adjacent peers \( s \) and \( p \)
- \( \mathcal{I}_s \) set of nodes currently managed by \( s \)
- \( \mathcal{I}_p \) set of nodes currently managed by \( p \)

\[
T_p = \min\left(\sum_{i \in \mathcal{I}_p} l_i, C_p\right), \quad T_s = \min\left(\sum_{i \in \mathcal{I}_s} l_i, C_s\right)
\]

\( T = T_p + T_s \)

- Considering \( n \) nodes, \( n = |\mathcal{I}_s| + |\mathcal{I}_p| \)
- Find \( k \) such that

\[
\min\left(\sum_{i \in \{0, \ldots, k\}} l_i, C_p\right) + \min\left(\sum_{i \in \{k+1, \ldots, n\}} l_i, C_s\right)
\]

is maximum (algorithm linear in the number of nodes).
Load balancing - Max Local Throughput

- $C_p$ capacity of the peer $p$
- $l_n$ load of the node $n$
- considering two adjacent peers $s$ and $p$
- $I_s$ set of nodes currently managed by $s$
- $I_p$ set of nodes currently managed by $p$

\[
T_p = \min\left(\sum_{i \in I_p} l_i, C_p\right), \quad T_s = \min\left(\sum_{i \in I_s} l_i, C_s\right)
\]

- $T = T_p + T_s$
- Considering $n$ nodes, $n = |I_s| + |I_p|$
- Find $k$ such that

\[
\min\left(\sum_{i \in \{0, \ldots, k\}} l_i, C_p\right) + \min\left(\sum_{i \in \{k+1, \ldots, n\}} l_i, C_s\right)
\]

is maximum (algorithm linear in the number of nodes)
Load balancing - Max Local Throughput

- $C_p$ capacity of the peer $p$
- $l_n$ load of the node $n$
- considering two adjacent peers $s$ and $p$
- $\mathcal{I}_s$ set of nodes currently managed by $s$
- $\mathcal{I}_p$ set of nodes currently managed by $p$

$$T_p = \min\left(\sum_{i \in \mathcal{I}_p} l_i, C_p\right), \quad T_s = \min\left(\sum_{i \in \mathcal{I}_s} l_i, C_s\right)$$

$$T = T_p + T_s$$

- Considering $n$ nodes, $n = |\mathcal{I}_s| + |\mathcal{I}_p|$
- Find $k$ such that

$$\min\left(\sum_{i \in \{0,\ldots,k\}} l_i, C_p\right) + \min\left(\sum_{i \in \{k+1,\ldots,n\}} l_i, C_s\right)$$

is maximum (algorithm linear in the number of nodes)
Simulation results

Load balancing – stable network

Rate of satisfied requests vs. Time

- Max local throughput [30 run]
- K-choices [30 run]
- No LB [30 run]
Simulation results

Load balancing – stable network – high load

Max local throughput [30 run]
K-choices [30 run]
No LB [30 run]
Simulation results

Load balancing – dynamic network – low load

Max local throughput [30 run]
K-choices [30 run]
No LB [30 run]
Simulation results

Load balancing – dynamic network – high load

Max local throughput [30 run]
K-choices [30 run]
No LB [30 run]
## Simulation results

<table>
<thead>
<tr>
<th>Load</th>
<th>Stable network</th>
<th>Dynamic network</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max local th.</td>
<td>K-choices</td>
</tr>
<tr>
<td>5%</td>
<td>39,62%</td>
<td>38,58%</td>
</tr>
<tr>
<td>10%</td>
<td>103,41%</td>
<td>58,95%</td>
</tr>
<tr>
<td>16%</td>
<td>147,07%</td>
<td>64,97%</td>
</tr>
<tr>
<td>24%</td>
<td>165,25%</td>
<td>59,27%</td>
</tr>
<tr>
<td>40%</td>
<td>206,90%</td>
<td>68,16%</td>
</tr>
<tr>
<td>80%</td>
<td>230,51%</td>
<td>76,99%</td>
</tr>
</tbody>
</table>
Simulation results

Load balancing – dynamic network – dynamic load

Max Local Throughput [50 run]
K-choices [50 run]
No LB [50 run]
Conclusion

- Algorithms to map a Prefix tree on a P2P network
- Reduction of maintenance cost of trie-based P2P systems
- New heuristic for load balancing