Self-Stabilization in Tree-Structured P2P Service Discovery Systems

Cédric Tedeschi\textsuperscript{1}
Eddy Caron\textsuperscript{2}, Ajoy K. Datta\textsuperscript{3}, and Franck Petit\textsuperscript{4}

(1) INRIA Sophia Antipolis Méditerranée
(2) LIP Lab., University of Lyon — CNRS — ENS Lyon — UCB Lyon — INRIA
(3) University of Nevada Las Vegas
(4) MIS Lab., University of Picardie Jules Verne

Workshop APRETAF
Jan. 22nd
Context

Computing Needs

Cosmology
Climate prediction
Genomics
Nuclear security

Computing Power

Computational Grids
Context
Context
Context
Context
Context

Service Discovery

DGEMM
Linux*
Mem > 1 GB

Cédric Tedeschi
Context

Service Discovery

DGEMM
Linux*
Mem > 1 GB

140.77.26.11
Context

Service Discovery

Cédric Tedeschi
Self-Stabilization in Tree-Structured P2P Service Discovery Systems
Context
Context
Context

- Service discovery in GRID Computing
  - Services (binary file, library) installed on servers
  - Servers declare their services, client discovers them
- Target platforms: **Peer-to-Peer Platform**
  - Decentralized algorithms (no central infrastructure)
  - Distributed data structure for service retrieval
  - Large scale systems
  - Dynamic (joins and leaves of nodes)
  - Fault-tolerance
Context

- Service discovery in GRID Computing
  - Services (binary file, library) installed on servers
  - Servers declare their services, client discovers them
- Target platforms: **Peer-to-Peer Platform**
  - Decentralized algorithms (no central infrastructure)
  - Distributed data structure for service retrieval
  - Large scale systems
  - Dynamic (joins and leaves of nodes)
  - Fault-tolerance
Context

• Service discovery in GRID Computing
  • Services (binary file, library) installed on servers
  • Servers declare their services, client discovers them
• Target platforms: **Peer-to-Peer Platform**
  • Decentralized algorithms (no central infrastructure)
  • Distributed data structure for service retrieval
  • Large scale systems
  • Dynamic (joins and leaves of nodes)
  • Fault-tolerance
Different Approaches

1. **Unstructured** *(Gnutella)*
   - Flooding:

2. **Structured**
   - Distributed Hashing Table (DHT)
     - Routing
     - Full Search
     - Scalability (logarithmic state and path)
   - Tries (or Prefix Trees)
Different Approaches

### 1 Unstructured (*Gnutella*)
- Flooding: **Costly** and **Partial Search**

### 2 Structured
- Distributed Hashing Table (DHT)
  - Routing
  - Full Search
  - Scalability (logarithmic state and path)
- **Tries** (or **Prefix Trees**)
Different Approaches

1. **Unstructured** *(Gnutella)*
   - Flooding: *Costly and Partial Search*

2. **Structured**
   - Distributed Hashing Table (DHT)
     - Routing
     - Full Search
     - Scalability (logarithmic state and path)
   - **Tries** (or **Prefix Trees**)
Different Approaches

1. **Unstructured** (*Gnutella*)
   - Flooding: **Costly** and **Partial Search**

2. **Structured**
   - Distributed Hashing Table (DHT) **Exact Queries Only**
     - Routing
     - Full Search
     - Scalability (logarithmic state and path)
   - **Tries** (or **Prefix Trees**)
Different Approaches

1. **Unstructured (Gnutella)**
   - Flooding: Costly and Partial Search

2. **Structured**
   - Distributed Hashing Table (DHT) Exacting Queries Only
     - Routing
     - Full Search
     - Scalability (logarithmic state and path)
   - **Tries** (or Prefix Trees)
Trie-based Overlays

- **Advantages**
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries

- **Related Works**
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
  - DLP-Tables (Caron, Desprez, Tedeschi – 2005)

- **Fault-tolerance**: either ignored or based on replication
  - Replication: Costly. What can be done if \( k \) is reached?
  - Does not recover after arbitrary failures (e.g., memory corruption)
Trie-based Overlays

- **Advantages**
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries

- **Related Works**
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
  - DLP-Tables (Caron, Desprez, Tedeschi – 2005)

- **Fault-tolerance**: either ignored or based on replication
  - Replication: Costly. What can be done if $k$ is reached?
  - Does not recover after arbitrary failures (e.g., memory corruption)
Trie-based Overlays

• Advantages
  • Efficient range queries
  • Automatic completion of partial strings
  • Easy extension to multi-dimensional queries

• Related Works
  • Skip Graphs (Aspnes and Shah – 2003)
  • P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  • PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  • Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
  • DLP-Tables (Caron, Desprez, Tedeschi – 2005)

• Fault-tolerance: either ignored or based on replication
  • Replication: Costly. What can be done if \( k \) is reached?
  • Does not recover after arbitrary failures
    (e.g., memory corruption)
Trie-based Overlays

- **Advantages**
  - Efficient range queries
  - Automatic completion of partial strings
  - Easy extension to multi-dimensional queries

- **Related Works**
  - Skip Graphs (Aspnes and Shah – 2003)
  - P-Grid (Datta, Hauswirth, John, Schmidt, Aberer – 2003)
  - PHT (Ramabhadran, Ratnasamy, Hellerstein, Shenker – 2004)
  - Nodewiz (Basu, Banerjee, Sharma, Lee – 2005)
  - DLP-Tables (Caron, Desprez, Tedeschi – 2005)

- **Fault-tolerance**: either ignored or based on replication
  - Replication: Costly. What can be done if $k$ is reached?
  - Does not recover after arbitrary failures (e.g., memory corruption)

**Best-Effort → Self-Stabilization**
Self-Stabilization

- General technique to tolerate transient faults
- Guaranteed to converge to the intended behavior
Self-Stabilizing Overlays for Peer-to-Peer Networks

- Self-Stabilizing Tree (Herault et al. – 2007)
- Self-Stabilizing Hypertree (Dolev and Kat – 2007)
- Snap-Stabilizing Tries (Caron et al. – 2006)
  - Assumes a rooted connected tree
  - Written in a coarse grain communication model
Self-Stabilizing Overlays for Peer-to-Peer Networks

- Self-Stabilizing Tree (Herault et al. – 2007)
- Self-Stabilizing Hypertree (Dolev and Kat – 2007)
- Snap-Stabilizing Tries (Caron et al. – 2006)
  - Assumes a rooted connected tree
  - Written in a coarse grain communication model
Contributions

Self-Stabilizing Trie Maintenance

- Built in the \textit{P2P} model
- Message-Passing
- Arbitrary initial topology
- Comprehensive proof of stabilization
- Simulation results
Outline

1. P2P Network
2. Proper Greatest Common Prefix Tree
3. Self-Stabilizing PGCP Tree
4. Conclusion
Outline

1. P2P Network
2. Proper Greatest Common Prefix Tree
3. Self-Stabilizing PGCP Tree
4. Conclusion
P2P Network

Physical Network

Basic entity: \textbf{peer (processor)}
- Message-Passing
- $P_1$ can communicate with $P_2$ if and only if $P_1$ “knows” $P_2$.
- Runs logical \textit{nodes}

Logical Tree

Basic entity: (tree) \textbf{node}
- Executes the protocol
- Distributed among peers
P2P Network

Physical Network

Basic entity: peer (processor)

- Message-Passing
- $P_1$ can communicate with $P_2$ if and only if $P_1$ “knows” $P_2$.
- Runs logical nodes

Logical Tree

Basic entity: (tree) node

- Executes the protocol
- Distributed among peers
Outline

1. P2P Network
2. Proper Greatest Common Prefix Tree
3. Self-Stabilizing PGCP Tree
4. Conclusion
Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure
- Dynamically constructed
- Bounded degree and height

Definition
Each node is the Proper Greatest Common Prefix of any pair of its children
Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure

- Dynamically constructed
- Bounded degree and height

Self-Stabilization in Tree-Structured P2P Service Discovery Systems
Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure

- Dynamically constructed
- Bounded degree and height
Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure

- Dynamically constructed
- Bounded degree and height
Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure
- Dynamically constructed
- Bounded degree and height
Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure
- Dynamically constructed
- Bounded degree and height

Lookup
- Exact match
- Autocompletion
- Range queries
Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure
- Dynamically constructed
- Bounded degree and height

Lookup
- Exact match
- Autocompletion
- Range queries
Proper Greatest Common Prefix Tree (PGCP Tree)

Distributed Logical Structure
- Dynamically constructed
- Bounded degree and height

Lookup
- Exact match
- Autocompletion
- Range queries
Arbitrary Initial Configuration

Expected PGCP Tree

- D
- DSWAP
- S
- SSWAP

- DROT
- DROTM
- DROTMG
- DROTG

- SROT
- SROTM
- SROTMG
- SROTG
An Arbitrary Labeled Tree

DROTM

S

BWAP

DSWAP  DROTMG  J  BOSWAP

DROTMG  J  SROTMG

SROT  DROT  SROTG
Arbitrary Initial Configuration

An Arbitrary Labeled Forest

- DROTM
- S
- BWAP
- DSWAP
- DROTMG
- J
- BOSWAP
- SROTMG
- DROTG
- SROT
- DROT
- SROTG
Outline

1. P2P Network
2. Proper Greatest Common Prefix Tree
3. Self-Stabilizing PGCP Tree
4. Conclusion
Protocol

Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots

Children processing
Protocol

**Infinite Cycle (Timeout)**

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots

2. Children processing
   - Merge
   - Prefix Relation
   - GCP

![Diagram of tree structure](image-url)
Protocol

Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots

2. Children processing
   - Merge
   - Prefix Relation
   - GCP

```plaintext
Protocol

Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots

2. Children processing
   - Merge
   - Prefix Relation
   - GCP
```

Cédric Tedeschi
Self-Stabilization in Tree-Structured P2P Service Discovery Systems
15/22
Protocol

Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots

2. Children processing
   - Merge
   - Prefix Relation
   - GCP
Protocol

Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots

2. Children processing
   - Merge
   - Prefix Relation
   - GCP
Protocol

Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots

2. Children processing
   - Merge
     - Prefix Relation
     - GCP
Protocol

Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots

2. Children processing
   - Merge
   - Prefix Relation
   - GCP

Diagram:

```
D  
/ 
DR  DRA  DZ  DV
```

```
D  
/ 
DR  
   / 
   DRA
```

```
D  
/ 
DR  DZ  DV
```

```
D  
/ 
DR  
   / 
   DRA
```

```
D  
/ 
DR  DZ  DV
```

Protocol

Infinite Cycle (Timeout)

1. Parent processing
   - Bad Prefix Relation
   - Reduction of root number
   - Creation of roots

2. Children processing
   - Merge
   - Prefix Relation
   - GCP
Protocol
Protocol

1-parent processing

Cédric Tedeschi

Self-Stabilization in Tree-Structured P2P Service Discovery Systems
Protocol

Cédric Tedeschi
Protocol

2-children processing
Protocol

2-children processing
2.1-merge
Protocol

2-children processing
2.2-prefix relation

ABA ABC ABDC ABDD ABE ABEA
Protocol

2-children processing
2.2-prefix relation

<UpdateParent, ABE>
Protocol

2-children processing
2.2-prefix relation
Protocol

2-children processing
2.3-GCP

ABA
ABC
ABDC
ABDD
ABE
ABEA
Protocol

```
Protocol

Cedric Tedeschi

Self-Stabilization in Tree-Structured P2P Service Discovery Systems

16/22
```
Protocol

Cédric Tedeschi

Self-Stabilization in Tree-Structured P2P Service Discovery Systems
Protocol
Protocol
Protocol

1-parent processing

C
B

ABA
ABC

ABD

ABDC
ABDD
ABEA

AB

ABA
ABC

ABD

ABDC
ABDD
ABEA
Protocol

1-parent processing

GetEpsilon()
Protocol

Cédric Tedeschi

Self-Stabilization in Tree-Structured P2P Service Discovery Systems
Protocol

2-children processing

Cédric Tedeschi
Protocol

2-children processing
2.1-merge

C

B

AB

ABA

ABC

ABD

ABE

ABDC

ABDD

ABEA
Sketch of proof

1. Stabilized communications (Awerbuch et al. 1996)
2. Only present processes
3. Parent’s copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition
Sketch of proof

1. Stabilized communications (Awerbuch et al. 1996)
2. Only present processes
3. Parent’s copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition
Sketch of proof

1. Stabilized communications (Awerbuch et al. 1996)
2. Only present processes
3. Parent’s copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition
Sketch of proof

1. Stabilized communications (Awerbuch et al. 1996)

2. Only present processes

3. Parent’s copies of child labels are correct

4. Prefix relations are correct

5. Number of roots = 1

6. The tree satisfies the PGCP tree definition
Sketch of proof

1. Stabilized communications (Awerbuch et al. 1996)
2. Only present processes
3. Parent’s copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition
Sketch of proof

1. Stabilized communications (Awerbuch et al. 1996)
2. Only present processes
3. Parent’s copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition
Sketch of proof

1. Stabilized communications (Awerbuch et al. 1996)
2. Only present processes
3. Parent’s copies of child labels are correct
4. Prefix relations are correct
5. Number of roots = 1
6. The tree satisfies the PGCP tree definition
Simulation: Convergence Time
Simulation: Convergence Time

Graph size

Convergence time

[avg]
Simulation: Communication Amount

Graph size

Number of messages (per node, per time step)
Simulation: Communication Amount

Graph size versus average communications per node per time step.

- X-axis: Graph size
- Y-axis: Communications (per node, per time step)

The graph shows the trend of communication amount as the graph size increases.
Simulation: Performance
Outline

1. P2P Network
2. Proper Greatest Common Prefix Tree
3. Self-Stabilizing PGCP Tree
4. Conclusion
Conclusion

• Contributions
  • Practical P2P prefix tree maintenance
  • Comprehensive correctness proof
  • Promising simulations for scalability

• On-going and future work
  • Better metrics of performance
  • Optimizations (cache, shortcuts)
  • Combination with redundancy
  • Implementation and deployment over a real platform
Conclusion

• Contributions
  • *Practical* P2P prefix tree maintenance
  • Comprehensive correctness proof
  • Promising simulations for scalability

• On-going and future work
  • Better metrics of performance
  • Optimizations (cache, shortcuts)
  • Combination with redundancy
  • Implementation and deployment over a real platform