Code-Partitioning Gossip

Lonnie Princehouse
Ken Birman
Cornell University

PLOS '09
Oct. 11 2009
Thinking About Distributed Systems

...from two perspectives

Local

Global

Local

Global
Code-Partitioning Gossip (CPG)

- Describe interaction between a pair of nodes with a single atomic function
- Partition into two functions, one for each node
- Pairs of nodes cooperate to evaluate original function
Gossip Protocols

- What is a gossip protocol?
  - Characteristic behavior
    - Periodic peer-to-peer exchanges
    - Fixed (or small) bandwidth for each exchange
    - Randomness of peer selection
  - Typical Properties
    - Decentralized, scalable
    - Well-behaved in the presence of congestion
    - Fault tolerant
    - Probabilistic guarantees
Gossip Protocols

- **Gossip applications**
  - **Rumor mongering**
    - Nodes "gossip" with their neighbors to disseminate rumors
    - Eventually, all nodes will know the rumor with probability 1
  - **Anti-Entropy**
    - Nodes collect the latest versions of objects
    - Gossip to keep their neighbors current
  - **Overlay Maintenance**
    - Gossip about nodes being added or removed from the system
    - Adjust overlay routing
  - **Peer Sampling**
    - E.g., random walks to sample a population of nodes
Gossip Protocols: Our Formulation

- Two threads per node
  - Active thread periodically initiates gossip
  - Passive thread listens and responds

```plaintext
Active thread @ node a

do forever:
  sleep(interval)
  b ← selectPeer(σa)
  push σa to b
  pull σb from b
  σa ← updatea(σa, σb)

Passive thread @ node b

do forever:
  a = listen()
  receive σa
  send σb
  σb ← updateb(σa, σb)
```
Example: Anti-Entropy

- Nodes collect versioned objects
- Pairs of nodes cooperate to get the latest versions

President=("Obama",44)
President=("Hoover",31)
Example: Anti-Entropy

- Nodes collect versioned objects
- Pairs of nodes cooperate to get the latest versions

President=("Obama",44)
President=("Hoover",31)
President=("Obama",44)
CPG Protocol Definition

CPG update function (imperative)

```java
public void update(Node a, Node b){
    if(a.president.version > b.president.version){
        b.president.name = a.president.name;
        b.president.version = a.president.version;
    } else if(a.president.version < b.president.version) {
        a.president.name = b.president.name;
        a.president.version = b.president.version;
    }
}
```

Peer Selection

```java
@SelectPeerUniform
public Set<Address> view;
```
Implementation

- **Uses Java**
  - A pragmatic decision
  - Custom annotations mark gossip elements
  - Partitioning done at the bytecode level

- **Peer Selection**
  - Uniform random
  - Weighted random
  - Arbitrary function

- **Update Function**
  - Programmer's responsibility: Deterministic
  - Halts

---

```java
public class Node {
    private String name;
    private int version;

    public Node(String name, int version, Set<Address> view) {
        this.name = name;
        this.version = version;
        this.view = view;
    }

    @GossipSelectPeerUniform
    private Set<Address> view;

    @GossipExchangeUpdate
    public void update(Node b) {
        if (version < b.version) {
            name = b.name;
            version = b.version;
        } else if (version > b.version) {
            b.name = name;
            b.version = version;
        }
    }
}
```
Two-phase automatic partitioning

Program slicing
Remove computation not relevant to local state updates

```java
if(a.president.version > b.president.version){
    b.president.name = a.president.name;
    b.president.version = a.president.version;
} else if(a.president.version < b.president.version) {
    a.president.name = b.president.name;
    a.president.version = b.president.version;
}
```

update\textsubscript{a}

```java
if(a.president.version > b.president.version){
    b.president.name = a.president.name;
    b.president.version = a.president.version;
} else if(a.president.version < b.president.version) {
    a.president.name = b.president.name;
    a.president.version = b.president.version;
}
```

update\textsubscript{b}
Code-Partitioning Gossip (CPG)

- Two-phase automatic partitioning
- Program slicing
  Remove computation not relevant to local state updates

```java
if(a.president.version > b.president.version){
    a.president.name = b.president.name;
    a.president.version = b.president.version;
}
else if(a.president.version < b.president.version) {
    a.president.name = b.president.name;
    a.president.version = b.president.version;
}

if(a.president.version > b.president.version){
    b.president.name = a.president.name;
    b.president.version = a.president.version;
}
```
Two-phase automatic partitioning

1. Program slicing
   Remove computation not pertinent to local state updates

2. Proxy remote objects
   Determine what state to fetch from the remote node

```java
if(a.president.version > b.president.version){
    b.president.name = a.president.name;
    b.president.version = a.president.version;
}
```
Two-phase automatic partitioning

1. Program slicing
   Remove computation not pertinent to local state updates

2. Proxy remote objects
   Determine what state to fetch from the remote node

```java
temp_version = fetch(a, "president.version");
temp_name = fetch(a, "president.name");
if(temp_version > b.president.version){
    b.president.name = temp_name;
    b.president.version = temp_version;
}
```
Protocol Composition

Self-Stabilization Example

Layer 1: Builds a tree overlay, given a root node
Layer 2: Count nodes in subtrees
Layer 3: Assigns numbers to nodes in depth first search order

- Composite protocol includes sub-protocols as members
- Update function calls sub-update functions
Future Work

- Lots of work left to do!
  - Dynamic slicing and proxying
  - Further study of composition and OO paradigm: Inheritance, encapsulation, interfaces
  - Benchmarks
  - Static analysis tools as proof assistants