Design of Adaptive Distributed Systems by Protocol Switching

Sushanta Karmakar
Introduction

- Distributed system
  - Multiple processes communicate to achieve a goal
  - Communication by message passing or shared memory

- Some examples
  - Railway reservation system
  - Routing in the Internet
Adaptive Distributed Algorithms

- Performance of a distributed algorithm depends on environment.
  - ex. load, mobility etc

- Environment may change with time

- Need for distributed algorithms that can cope with changing environment
Adaptation Techniques

- Modify runtime parameters
  - Example – adjusting buffer size in routers with load

- Adaptation by nature
  - Adaptive mutual exclusion by Anderson et al. [1]

- Adaptation as a protocol layer
  - Snoop protocol by H. Balakrishnan et al. [3]
Motivation of the Work

Existing approaches not sufficient in many cases, may need to run different algorithms in different conditions

An Example

- Routing in ad-hoc networks
  - AODV
  - DSR
Protocol Switching

- $P_1$ and $P_2$ are two protocols for the same problem, $E_1$ and $E_2$ are two environments, and $M$ is the performance evaluation metric
  - $P_1$ is better than $P_2$ under $E_1$
  - $P_2$ is better than $P_1$ under $E_2$

- Dynamically switch from $P_1$ to $P_2$ as environment changes from $E_1$ to $E_2$ and vice-versa
Additional Criteria

- Maintain desirable properties during switching

- Examples of desired properties
  - Mutual exclusion
    - No more than one process can enter the critical section during switching
  - Routing
    - No loss of packet during switching
Components of Distributed Protocol Switching

- When to switch
  - May require global coordination

- How to switch
  - Switching algorithm
Solution Approaches

- **Centralized switching by two-phase-commit**
  - Simple and easy to implement
  - Large switching overhead
  - Global freeze
  - Not scalable

- **Localized distributed switching**
  - Switching is based on local information
  - Low overhead per node
  - Local freeze
  - Scalable
Overall Motivation

- Proposing localized distributed algorithms for dynamic switching from one protocol to another

- Maintaining some desirable property of the system during switching
Related Work


- Bar-Noy et al. [4] → shifting between different algorithms on the fly to solve byzantine agreement
Related Work (contd.)

- Arora et al. [2] → fault-tolerant method to switch from one state to another without requiring global freeze.

- Liu et al. [15] → adaptation by dynamically mapping the state of a process in one protocol to the state in another.

- Liu et al. [16] → overview of the communication properties for correct functioning of the protocol in [15]

- Mocito and Rodrigues [19] → switching between different total order algorithms.
Objective of the Work

- Design of adaptive algorithm by protocol switching for **single source broadcast** problem
  - Tolerating node failure
    - Crash fault
    - Transient fault

- Design of adaptive algorithm by protocol switching for **token based distributed mutual exclusion** problem
Adaptive Broadcast by Switching from a BFS tree to a DFS tree
BFS to DFS switching

- Non-fault-tolerant algorithm for dynamic switching from a BFS tree to a DFS tree
  - Proof of correctness

- Fault-tolerant algorithm for dynamic switching from a BFS tree to a DFS tree
  - Proof of correctness
System Model

- Asynchronous message passing system
- Reliable and FIFO channels
- Crash fault
- Connected Graph
- The single source $r$ does not fail
Solution Approach

- Non-fault-tolerant switching algorithms

- Local repair of BFS and DFS
  - Faults may happen when no switching is in progress

- Fault-tolerant actions that help tolerate arbitrary crash faults during switching
Switching from a BFS tree to a DFS tree

- $G = (V, E)$ is the graph
- $T$ is a BFS tree of $G$ rooted at $r$
- $T'$ is a DFS tree of $G$ rooted at $r$
- Switch from $T$ to $T'$
Definitions

- Let $G_v = (V', E')$ be some subgraph of graph $G$ such that
  - $V' \subseteq \{v\} \cup N(v)$
  - $E' \subseteq \{(u,v) | u,v \in V'\}$

- If $T_v$ is a DFS spanning tree of $G_v$ rooted at $v$ then $T_v$ is defined as the \textit{local DFS subtree of $G_v$} rooted at $v$
Local DFS Subtree

\[ G = (V,E) \]
Switching Algorithm for BFS to DFS

- TOKEN based local switching from BFS to DFS
- The root of the BFS tree, \( r \), gets the TOKEN first
- For a node \( v \), \( CSet(v) = N(v) - [TSet(v) \cup \{p(v)\}] \)
- On receiving the TOKEN for the first time, a node \( v \) builds a local DFS subtree, rooted at itself, of the graph induced by \( CSet(v) \cup \{v\} \).
Switching Algorithm for BFS to DFS (contd.)

- After v builds local DFS subtree, it sends the TOKEN to some $u \in \text{Child}(v)$ if $u$ has not already got the TOKEN

- $\forall u \in \text{Child}(v)$, if $u$ has got the TOKEN, $v$ sends the TOKEN to $p(v)$

- If $v$ has already got the TOKEN then it forwards the TOKEN to some $u$ using the same rule
Partial DFS Tree

- The nodes that have received the TOKEN at least once form a DFS tree.
  - partial DFS tree

Tree edges of a local DFS subtree may change with time but that of partial DFS tree will not change.
An Example (BFS to DFS)

- Initially for each node v, \( \text{CSet}(v) = \text{TSet}(v) = \emptyset \)
- ‘a’ has got the TOKEN
- \( \text{CSet}(a) = \{b, c\} \)
- ‘a’ builds a local DFS subtree, rooted at ‘a’, of the graph induced by the set of nodes \{a, b, c\}
Steps

LDFS(a, \{b, c\})
Steps (contd.)

LDFS(a, \{c\})
Steps (contd.)
Steps (contd.)

REMOVE_ACK
Steps (contd.)

LDFS(a, φ)
Steps (contd.)
The topology after local switching at node \(a\)

- Nodes having a green outline belong to the **partial DFS tree** of \(G\).

- Node \(a\) sends the TOKEN to its only child \(b\)
Example (contd.)

- TSet(b) = \{a\}, TSet(c) = \{a\}
- ‘b’ got the TOKEN
- CSet(b) = \{c, d\}
- ‘b’ builds a local DFS subtree, rooted at ‘b’, of the graph induced by the set of nodes \{b, c, d\}
Example (contd.)

- TSet(c) = {a, b}, TSet(d) = {b}
- ‘d’ has got the TOKEN
- CSet(d) = {c, e, f}
- ‘d’ builds a local DFS subtree, rooted at ‘d’, of the graph induced by {c, d, e, f}
Example (contd.)

- $TSet(c) = \{a, b, d\}$, $TSet(e) = \{d\}$, $TSet(f) = \{d\}$
- ‘e’ has got the TOKEN
- $CSet(e) = \{f\}$
- ‘e’ builds a local DFS subtree, rooted at ‘e’, of the graph induced by \{e, f\}

After this there will be no change in the spanning tree
Example (contd.)

- TSet(f) = {d, e}
- 'f' has got the TOKEN
- CSet(f) = {c}
- 'f' builds a local DFS subtree, rooted at 'f', of the graph induced by {c, f}
Example (contd.)

- $TSet(c) = \{a, b, d, f\}$
- $CSet(c) = \emptyset$
- ‘c’ builds a local DFS subtree, rooted at ‘c’, of the graph induced by \{c\}

- Now ‘c’ sends the TOKEN back to ‘f’
Example (contd.)

- Now ‘f’ sends the TOKEN back to ‘e’ and so on.
- Algorithm stops when TOKEN comes to ‘a’
Properties

- Switching eventually completes.

- The algorithm terminates with a DFS tree topology

- The message complexity of the switching algorithm is $O(|E|)$ for no fault case.

- Each broadcast message is eventually correctly delivered in spite of switching provided no failure occurs.
Fault-tolerant Switching from a BFS Tree to a DFS Tree

- When a node fails?
  - No switching in progress
  - Switching in progress
A node $v$ in a tree (BFS/DFS) may crash when no switching is in progress.

The tree must be repaired to continue the broadcast.

We do local repair of trees as it is attractive for limited failures in terms of time and message complexity.
Local Repair of BFS

- Let node \( v \) crash

- Each of \( u_1, u_2, \ldots, u_n \) and node \( x \) executes \( \text{BfsCrashAction}(v) \)

```
\text{BfsCrashAction}(v)
N(u) = N(u) - \{v\}
\text{if } p(u) = v \text{ then }
\text{ResetLevelAction}(v)
```
Local Repair of DFS

- Let node $v$ crash
- Each of $u_1, u_2, \ldots, u_n$ and node $x$ executes $\text{DfsCrashAction}(v)$

$$\text{DfsCrashAction}(v)$$

$N(u) = N(u) - \{v\}$

if $p(u) = v$ then

$\text{ChangePathAction}(v)$

Diagram:

- Node $v$ connected to $u_1$, $u_2$, $\ldots$, $u_n$.
- Node $x$ connected to $v$.
ResetLevelAction(v) and ChangePathAction(v)

 Upon receiving ResetLevel(v) ∧ τ(u) = T_u
 01 GetParamBFS()
 02 N_u = \{ x : x \in N(u) ∧ v \notin P_{x \sim r} \}
 03 if N_u = \emptyset then
 04 \forall w \in N(u) : p(w) = u, send ResetLevel(v) to w
 05 else
 06 P_{w \sim r}' = P_{w \sim r}
 07 \exists y \in N_u : L_y = \min_{z \in N_u} \{ L_z \}
 08 if v \in P_{w \sim r} then
 09 L_u = L_y + 1
 10 send REMOVE to p(u)
 11 p(u) = y
 12 send ADD to p(u)
 13 P_{w \sim r} = P_{w \sim r} \cup u
 14 else if L_u > L_y + 1 then
 15 L_u = L_y + 1
 16 send REMOVE to p(u)
 17 p(u) = y
 18 send ADD to p(u)
 19 P_{w \sim r} = P_{w \sim r} \cup u
 20 endif
 21 if (P_{w \sim r} \neq P_{w \sim r}') then
 22 \forall w \in N(u) - \{ p(u) \}, send ResetLevel(v) to w
 23 endif
 24 endif

 Upon receiving ChangePath(v) ∧ τ(u) = T_u
 25 GetParamDFS()
 26 N_u = \{ x : x \in N(u) ∧ v \notin P_{x \sim r} \}
 27 if N_u = \emptyset then
 28 \forall w \in N(u) : p(w) = u, send ChangePath(v) to w
 29 else
 30 P_{w \sim r}' = P_{w \sim r}
 31 path_u = \min_{x \in N_u} \{ path_x \cup \beta_x(u) : x \in N_u \}
 32 send REMOVE to p(u)
 33 p(u) = f(path_u)
 34 send ADD to p(u)
 35 P_{w \sim r} = P_{w \sim r} \cup u
 36 if (P_{w \sim r} \neq P_{w \sim r}') then
 37 \forall w \in N(u) - \{ p(u) \}, send ChangePath(v) to w
 38 endif
 39 endif

 Upon receiving REMOVE from w →
 40 Child(u) = Child(u) - \{ w \}

 Upon receiving ADD from w →
 41 Child(u) = Child(u) ∪ \{ w \}
Fault during Switching

- At any intermediate state during switching, there is a **partial DFS tree** and a **partial BFS tree** of the graph $G$.

- **TOKEN** holding node may belong to either **partial DFS** or **partial BFS**.

- A fault may occur in
  - **partial BFS tree**
  - **partial DFS tree**
Case Study

- Suppose a node $b$ belonging to partial DFS tree crashes.
Case Study (contd.)

- Resultant structure after the crash of b
- Note that node a, c, d have detected the crash of b
- TOKEN is at e
- Node a, c, d remove b from theirs’ neighborhood
Case Study (contd.)

- Node d execute `DfsCrashAction(b)`
- Node a may generate another TOKEN at a to restart switching at a
- Another switching due to TOKEN at e
- Node c, e, f may execute `ChangePathAction(b)`
Node c eventually changes its parent to a either by `ChangePathAction(b)` or due to fresh switching from a.
Eventually d, e, f reassign theirs’ parents as shown in figure due to \texttt{ChangePath(b)} messages.

- TOKENs at e may perish or may result in switching

- Overall, a correct DFS tree of G rooted at a results
Crash of a Node in Partial BFS Tree

- If $f$ crashes then each neighbor belonging to **partial BFS tree** should execute $BfsCrashAction(f)$

- Each neighbor belonging to **partial BFS tree** should execute $ResetLevelAction(f)$ on receiving a $ResetLevel(f)$ message
Crash of TOKEN Holder

What happens if TOKEN holding node e crashes?
Crash of TOKEN Holder (contd.)

- Suppose the TOKEN holding node e crashes.
- TOKEN can be generated at any of a, b, d to continue the switching
- TOKEN is actually generated at the nearest ancestor d
Crash of TOKEN Holder (contd.)

- A fresh **local DFS subtree** formation starts at **d**
- Eventually a DFS tree rooted at **a** results.
Suppose d is currently doing the local switching.
- d crashes, already covered
- Some member of Cset(d) crashes

Cset(d) = \{c,e,f\}

In this case the local switching is just restarted at d
Fault-tolerant Actions for Switching from BFS to DFS

\[(S_1) \text{tokenVisited}(u) \land \text{Crash}(v) \rightarrow \text{DfsCrashAction}(v)\]

\[(S_2) \text{tokenVisited}(u) \land \text{received ChangePath}(v) \rightarrow \text{ChangePathAction}(v)\]

\[(S_3) \neg\text{tokenVisited}(u) \land \text{Crash}(v) \rightarrow \text{BfsCrashAction}(v)\]

\[(S_4) \neg\text{tokenVisited}(u) \land \text{received ResetLevel}(v) \rightarrow \text{ResetLevelAction}(v)\]

\[(S_5) \neg\text{tokenVisited}(u) \land \text{received ChangePath}(v) \rightarrow \text{ChangePathFlag}(u) = 1; \text{ID}(u) = v\]

\[(S_6) \text{tokenVisited}(u) \land \text{ChangePathFlag}(u) = 1 \rightarrow \text{ChangePathFlag}(u) = 0; \text{ChangePathAction}(\text{ID}(u))\]

\[(S_7) \text{tokenVisited}(u) \land \neg\text{tokenHolder}(u) \land \text{Crash}(v) \land \text{tdir}(u) = v \land \text{tdir}(u) \neq p(u) \rightarrow \text{tokenHolder}(u) = \text{true} \quad \text{tokenVisited}(u) = \text{false}\]
\[\forall w \in \text{CSet, Reset}(w)\]

\[(S_8) \text{tokenVisited}(u) \land \text{tokenHolder}(u) \land \text{Crash}(v) \land v \in \text{CSet}(u) \rightarrow \text{tokenVisited}(u) = \text{false}\]
\[\forall w \in \text{CSet, Reset}(w)\]
Properties

- Under arbitrary crash failures, the BFS to DFS switching algorithm eventually terminates with a DFS tree as the broadcast topology. No specific broadcast delivery guarantee in this case.
Broadcast Properties under Single Crash Fault

- Under single crash fault, each broadcast message having timestamp less than or equal to $\gamma$ is eventually correctly delivered to all the non-faulty nodes where $\gamma = \min\{\tau_{u_1}, \tau_{u_2}, \ldots, \tau_{u_m}\}$ and $\tau_{u_i}$ is the timestamp of the last message received by $u_i$ before it detects the crash of $v$. 
Broadcast Properties under Single Crash Fault (contd.)

- Under single crash failure, each message broadcast by the single source r after the system reaches a state of Z is eventually correctly delivered to all the nodes where Z is the set of states of the system where any node \( w \in V \) does not change \( p(w) \) anymore due to receipt of \( \text{ChangePath}(v) \) or \( \text{ResetLevel}(v) \) message, but \( w \) may change \( p(w) \) due to the receipt of an \( \text{LDFS} \) messages.
Properties

- Under single transient failure, each broadcast message $m$ read by each child of the faulty node $i$ before time $T_s$ is eventually correctly read by all the non-faulty nodes.

- Under single transient failure, each broadcast message $m$ that has not yet been read by the faulty node $i$ before time $T_{ss}$ is eventually correctly read by all the nodes.
Thank You