03_02_Synchronizer

1. Introduction

1.1. Background

QUESTION

Is it possible to run synchronous DAs on asynchronous systems?

Answer

The general answer is **no**: synchronous systems/DAs are more powerful than asynchronous systems/DAs

1.2. Purpose

- to make an asynchronous system **look synchronous**
- **local** simulations
- to run **non-fault-tolerant synchronous distributed algorithms** in asynchronous systems

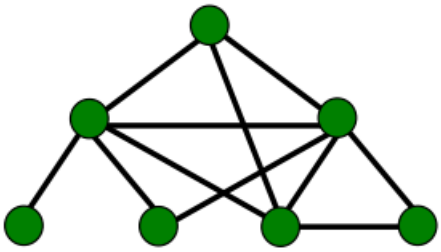
1.3. Basic Idea

- let the processes proceed in **(simulated) rounds**
- check whether all messages of a round have been received
- then let a synchronizer generate a "**clock pulse":** means the next round is prepared

1.4. Types of Synchronizers

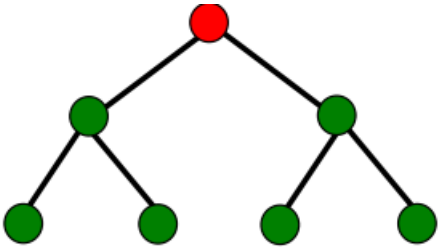
α-synchronizers

- communication on **all point-to-point link**s
- communication-inefficient, time-efficient



β-synchronizers

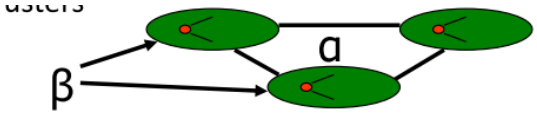
- first elect a **leader** and create a **spanning tree**
- communication along branches of this tree o communication
- efficient, time-inefficient



γ-synchronizers

- combine α-synchronizers and β-synchronizers

- first cluster the nodes and connect the clusters
- **intracluster: β-synchronizers**
- **intercluster: α-synchronizers**



2.Generating a pulse

2.1. Method 1: use acknowledgements

- a node **is safe** with respect to a certain pulse :
  - if each of its messages has been received
- this can be **determined from** the reception of an **ACK** from every neighbor to which the node sent a message
- a **safe node sends a SAFE** message to its neighbors
- a node knows that **it has received all its messages** in a pulse:
  - when all its neighbors are safe

2.2. Method 2: single-message method

- if a node **does not send a message** to some neighbor in some pulse in the synchronous algorithm, let it send an **"empty"** message
- if a node **sends multiple messages** to a neighbor in some pulse in the synchronous algorithm, **pack them into a single message**
- so now, a node should in every pulse **receive a single message** from **every** neighbor
- a node knows that it has **received all its messages** in a pulse:
  - when it has received one message from each of its neighbors

3. General description of alph-type: Use acks

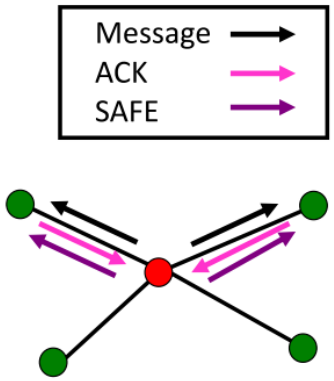
Process

Every node:

- eventually finds that it is **safe (=gets all ACKs)**
  - sends a **SAFE** message to all its neighbors
  - receives a **SAFE** message from all its neighbors
  - generates **a new local pulse**

Complexity

- V set of nodes
- communication: $C(\alpha) = O(|V|^2)$
- time: $T(\alpha) = O(1)$

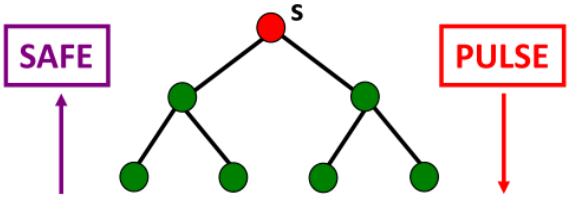


4. General description of beta-type

Process

Initialzation

- elect a **leader s**
- create a s**panning tree** rooted at s



Synchronizer

- when a node **finds it is safe** and all **its descendants are safe**, it sends a **SAFE** message to its **parent**
- when the root gets a **SAFE** message from all its descendants and is safe, it sends a new **PULSE message down** the tree

Complexity

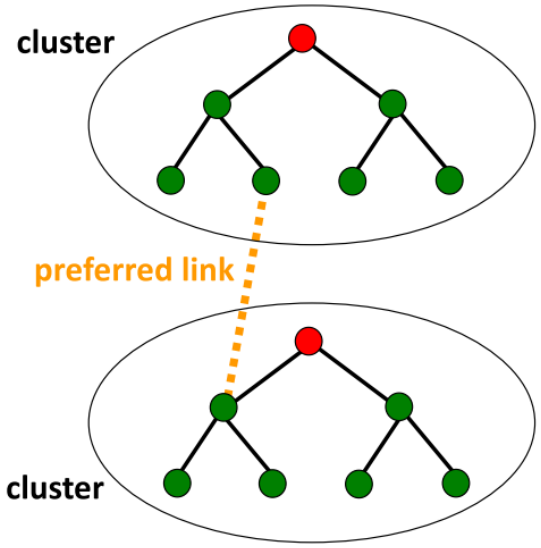
- Communication: $C(\beta) = O(|V|)$ (number of links in tree)
- Time: $T(\beta) = O(|V|)$ (depth of the tree)

5. General description of Gamma-type

Process

Initialization Phase

- partition nodes into **clusters**
- elect **leaders** in clusters
- create **spanning trees** in clusters
- create a **preferred link** between each pair of clusters



Synchronizer

- first phase: β-synchronizer **in clusters** separately
- second phase: report safeness **among clusters**

beta-synchronizer: PULSEs and SAFEs

The β-synchronizer is executed **in every tree**:

1. every **root** broadcasts a **PULSE** message down its own tree
2. the nodes of the tree convergecast **SAFE** messages upwards

alpha-synchronizer: CLUSTER_SAFEs and READYs

1. The α-synchronizer is executed **among the trees:**

- it broadcasts a **CLUSTER_SAFE** message down its tree
- and across the **preferred links** to the neighboring trees

2. the nodes convergecast **READY** messages upwards

- when they have received
  - a **READY** message from each of its descendants
  - and a **CLUSTER_SAFE** message along all its preferred links

Preparation (Structure)

Messages:

- **ACK**: **acknowledge** a message of the synchronous algorithm

- **PULSE**: message sent by a node to its descendants to indicate **new round**
- **SAFE:** message sent by a node to its parent when all its descendants and the node itself are safe
- **CLUSTER_SAFE**: message sent by a node to its descendants and over the preferred links it is connected to
- **READY**: message sent by a node to its parent when all its descendants are ready and when all clusters connected by preferred links are safe

Variables:

- neighbors(N): set of all neighbors
- parent: parent in the intracluster tree (root is own parent)
- descendents(D): descendants in the intracluster tree
- preferred (P): set of preferred links to other clusters that the node is part of
- **dif(j): counter**: the number of messages sent to j minus the number of ACKs received from j
- **safe(j): boolean**: SAFE received from descendent j
- **cluster_safe(j): boolean:** CLUSTER_SAFE received across preferred link j or from parent
- **ready(j): boolean:** READY received from descendent j

Implementation

```
I.   Receiving a PULSE message                    /* PULSEs down the tree */
       upon receipt of (PULSE) do
           for all (j in descendants) do
               safe(j) := 0                       /* expect a safe message */
               send(PULSE) to j                   /* propagate pulse down */
           for all (j in neighbors) do dif(j):=0  /* balance of messages */
           for all (j in preferred) do cluster_safe(j):=0
                /* await CLUSTER_SAFE messages across preferred links */
           execute next round of synchronous algorithm
```

```
II.  Sending a message of the synchronous algorithm
       send(message) to j
           dif(j):=dif(j)+1                   /* keep track of number of messages sent */
III. Receiving a message of the synchronous algorithm
       upon receipt of (message) from j do
           send(ACK) to j                     /* ack message */
IV.  Receiving an ACK
       upon receipt of (ACK) from j do
           dif(j):=dif(j)-1                   /* keep track of number of ACKs received */
V.   Round of the synchronous algorithm has been completed
       when (all actions at this pulse have been completed) then
           safe_propagation()                 /* check if SAFE can be propagated up */
```

```
VI.  safe_propagation()                        /* SAFEs up the tree */
       if ((dif(j)=0 for all neighbors j)      /* if all ACKs received */
          and
          (safe(j)=1 for all descendants j))   /*and all descendents safe */
       then
           if (i≠root) then send(SAFE) to parent   /* propagate SAFE up */
           else send(CLUSTER_SAFE) to node itself
                                  /* or let root start broadcasting CLUSTER_SAFE */
VII. Receiving a SAFE message from a descendant
       upon receipt of (SAFE) from j do
           safe(j):=1                         /* record descendent safe */
           safe_propagation()                 /* check if safe */
```

```
VIII. Receiving a CLUSTER_SAFE message
      upon reception of (CLUSTER_SAFE) from j do
          if (j in preferred) then cluster_safe(j):=1   /* from other cluster */
          if (j = parent) then                          /* from parent, so own */
              for all (k in descendants) do             /* cluster is safe, */
                  send(CLUSTER_SAFE) to k               /* so propagate safe */
                  ready(k):=0                           /* message down */
              for all (k in preferred) do               /* and also report safe */
                  send(CLUSTER_SAFE) to k               /* to other clusters */
          ready_propagation()                           /* next slide */
```

Each time receive a **SAFE** message, it will chekck whether the safe is able to be propagated

```
IX.   Receiving a READY message
        upon reception of (READY) from j do      /* from a descendant */
            ready(j):=1                          /* record ready */
            ready_propagation()
X.    ready_propagation()
      if (cluster_safe(j)=1 for all j in preferred)   /* all neighboring */
          and                                         /* clusters are safe and */
        (ready(j)=1 for all j in descendants)         /* all descendants ready */
      then
        if (i≠root) then                         /* if not root */
            send(READY) to parent                /* propagate READY up */
        else
            send(PULSE) to node itself           /* else root triggers */
                                                 /* new PULSE */
```

TU Delft

Now, start new round (new **PULSE**)

Complexity

Let **E** be the set of **all branches** in the cluster trees and the preferred links

- At most four message are sent along each link in E So **communication complexity** $C(\gamma) = O(|E|)$

Let **H** be the **maximal height** of the cluster trees

- time complexity is $T(\gamma) = O(|H|)$

Big problem

How to find suitable partitioning