AWERBUCH'S SYNCHRONIZERS (ALPHA, BETA, GAMMA)

Ersel Hengirmen
ehengirmen@hotmail.com

Wireless Systems, Networks and Cybersecurity Laboratory
Department of Computer Engineering
Middle East Technical University
Ankara Turkey

March 28, 2024

Outline of the Presentation

Agenda

The problem

Synchronization

Synchronization in distributed systems is crucial for maintaining coherence and ensuring consistent behavior across processes, especially in transitioning from synchronous to asynchronous environments.

Agenda

What is the solution/contribution

- Implementation of Alpha, Beta, Gamma Algorithms on the AHCv2 platform.
- Anlysis of the algorithms within different topologies

Agenda

Motivation/Importance-1

These synchronizers play a pivotal role in distributed systems by enabling coordination and consistency among processes operating in asynchronous environments. Their importance lies in their ability to bridge the gap between synchronous and asynchronous systems, allowing distributed applications to maintain coherence and achieve desired outcomes despite the lack of a global clock or fixed time intervals.

Motivation/Importance-2

Without effective synchronizers, distributed systems would struggle to ensure consistent behavior, leading to potential issues such as data inconsistencies, race conditions, and overall system instability. Therefore, these synchronizers serve as fundamental building blocks for the reliable and efficient operation of distributed systems in various domains, including cloud computing, networking, and parallel computing.

Agenda

Model, Definitions

- Synchronization: The process of coordinating actions or events across distributed entities to maintain coherence and consistency.
- Asynchronous Systems: Distributed systems where processes operate independently without a global clock or fixed time intervals.

Background and this slide can be combined....

Background

The 1985 paper Awerbuch ”Complexity of network synchronization” proposes the method for alpha beta gamma synchronizers(without naming them as alpha beta gamma) and analyzes the trade-offs between them in this paper. So in essence its a paper that introduces simple methodologies for designing efficient distributed algorithms in asynchronous networks.

Agenda

Alpha

- In the alpha synchronizer, every node sends a message to every neighbor in every round(so that neighboor can synchronize with it)
  - If no message needs to be sent to that neighboor, node sends a dummy message to that neighboor.
- The receiver waits until it receives a message from every neighbor for a particular round before proceeding to the next round.

Alpha Algorithm

Listing 1: Awerbuch's alpha algorithm.

```
Implements:
Uses:
    set_neighboors_not_received,  # sets every value in adjnd as not received or False
    create_current_round_messages,  # creates messages for current round
    do_round_process,
    set_received_rmd_messages_to_none,
    send_messages_to_neighboors,
    set_timer,  # set timer that after given time calls the given function with given parameters. repeats itself until reset_timer call
    reset_timer,
Events:
    Init,
    NewRoundEvent,
    OnMessageReceive,
Needs:
    adjacent_nodes_dict: adjnd
    received_messages_dict: rmd  # for storing this round's messages

OnInit: () do
    round = 0
    NewRoundEvent()

NewRoundEvent () do
    round += 1
    unreceived_count = len(adjnd)
    set_neighboors_not_received()
    do_round_process()  # doing the algorithm event since it received every needed mesage
    set_received_rmd_messages_to_none()
    current_round_messages = create_current_round_messages()
    send_messages_to_neighboors(round, current_round_messages)
    set_timer(send_messages_to_neighboors, round, current_round_messages)

OnMessageReceive: ( neighboor_round, neighboor_node, message ) do
    If neighboor_round == round and adjnd[neighboor_node] == False:
        adjnd[neighboor_node] = true
        unreceived_count -= 1
        rmd[neighboor_node] = message
        If unreceived_count == 0:
            reset_timer()
            NewRoundEvent()
```

Beta

- In the beta synchronizer, messages sent by processes are acknowledged by their receivers.
- Senders wait until they receive acknowledgments (ACKs) from all receivers for the messages they sent in a particular round.
  - That is why since its a reliable connection we act like receive happens only ones(think of TCP)
- Once all ACKs are received and all OKs from child nodes are received, the node sends ok to parent node.
- Once root receives OK from its parent nodes it broadcasts GO signifying the start of new round

Beta Algorithm

Listing 2: Awerbuch's Beta algorithm.

```
Implements:
Uses:
    create_current_round_messages,  # creates messages for current round
    do_round_process,
    set_received_rmd_messages_to_none,
    send_messages_to_neighboors,
    is_root,
    broadcast_go,
Events:
    Init,
    NewRoundEvent,
    OnMessageReceive,
    OnOKReceive,
    NodeRoundEndCheck,
Needs:
    parent_node,        # if parent node is none it is root
    adjacent_node_count: adj_count,
    children_node_count: child_count,
    received_messages_dict: rmd  # for storing this round's messages

OnInit: () do
    NewRoundEvent()

NewRoundEvent () do
    unreceived_count = adj_count
    unreceived_OK_count = child_count
    do_round_process()  # doing the algorithm event since it received every needed mesage
    set_received_rmd_messages_to_none()
    current_round_messages = create_current_round_messages()
    send_messages_to_neighboors(round, current_round_messages)

OnMessageReceive: ( neighboor_node, message ) do
    unreceived_count -= 1
    rmd[neighboor_node] = message
    NodeRoundEndCheck()

OnOKReceive: ( child_node ) do
    unreceived_OK_count -= 1
    NodeRoundEndCheck()

NodeRoundEndCheck: () do
    If unreceived_count == 0 and unreceived_OK_count == 0:
        If is_root():
            broadcast_go()  # every node will activate newroundevent when they receive it
            NewRoundEvent()
        Else:
            SendOK(parent_node)
```

Figure:

Gamma

- The gamma synchronizer is the merge of both alpha and beta synchronizers. It does this by having multiple roots which all have their own spanning trees where:
  - The beta algorithm runs within each tree
  - The alpha algorithm runs between trees
- In here in addition to beta when the root of a tree gets all acks and OKs, it sends ready to the roots of all adjacent trees (and itself).
  - Two trees are considered to be adjacent when any of their members are adjacent.
- When the root is READY(which means every one of its descendants are OK). And all their adjacent nodes are READY(which it knows by receiving their message). It broadcasts go down its tree

Agenda

Main Result Alpha

Alpha: guarantees local synchronization, ensuring that no process proceeds to the next round since it waits for all its neighboors which means it knows every neighboor finished its previous message neighbors have completed the current round.

Main Result Beta

guarantees global synchronization, ensuring that no process proceeds to the next round since everyone waits for root and root waits for OKs from its children And since its children being OK would recursively prove that every other node is OK this algorithm is correct.

Main Result Gamma-1

As in the alpha synchronizer, we can show that no root process to the next round unless it and all its neighbors are in ready state, which happens only after both all nodes in the root's tree and all their neighbors have received acks for all messages. This proves that within nodes there is local syncronization.

Main Result Gamma-2

And since every tree uses beta syncronizer in itself we can see that within that tree networkwise syncronization has been achieved.

Main Result Gamma-3

Since every connection of a node inside 1 tree to another implies connection between roots the syncronization of roots will be achieved. While this sentence does not prove its correctnes, the idea of thinking every tree as a giant node will since it will make the situation same as a normal alpha synchronizer.

Agenda

Conclusions

Hindsight is Clearer than Foresight

Advices come from [?].

- You can now make observations that would have been confusing if they were introduced earlier. Use this opportunity to refer to statements that you have made in the previous three sections and weave them into a coherent synopsis. You will regain the attention of the non- experts, who probably didn't follow all of the Technicalities section. Leave them feeling that they have learned something nonetheless.
- Give Open Problems It is traditional to end with a list of open problems that arise from your paper. Mention weaknesses of your paper, possible generalizations, and indications of whether they will be fruitful or not. This way you may defuse antagonistic questions during question time.
- Indicate that your Talk is Over An acceptable way to do this is to say "Thank-you. Are there any questions?"[?]

References

How to prepare the talk?

Please read `http://larc.unt.edu/ian/pubs/speaker.pdf`

- The Introduction: Define the Problem, Motivate the Audience, Introduce Terminology, Discuss Earlier Work, Emphasize the Contributions of your Paper, Provide a Road-map.
- The Body: Abstract the Major Results, Explain the Significance of the Results, Sketch a Proof of the Crucial Results
- Technicalities: Present a Key Lemma, Present it Carefully
- The Conclusion: Hindsight is Clearer than Foresight, Give Open Problems, Indicate that your Talk is Over

Questions

THANK YOU

AWERBUCH'S SYNCHRONIZERS (ALPHA, BETA, GAMMA)

presented by Ersel Hengirmen
ehengirmen@hotmail.com

March 28, 2024