Chapter 12

Synchronizers

So far, we have mainly studied synchronous algorithms because generally, asynchronous algorithms are often more difficult to obtain and it is substantially harder to reason about asynchronous algorithms than about synchronous ones. For instance, computing a BFS tree (cf. Chapter 3) efficiently requires much more work in an asynchronous system. However, many real systems are not synchronous and we therefore have to design asynchronous algorithms. In this chapter, we will look at general simulation techniques, called *synchronizers*, that allow to run a synchronous algorithm in an asynchronous environment.

12.1 Basics

A synchronizer generates sequences of *clock pulses* at each node of the network satisfying the condition given by the following definition.

Definition 12.1 (valid clock pulse)**.** *We call a clock pulse generated at a node v* valid *if it is generated after v received all the messages of the synchronous algorithm sent to v by its neighbors in the previous pulses.*

Given a mechanism that generates the clock pulses, a synchronous algorithm is turned into an asynchronous algorithm in an obvious way: As soon as the i^{th} clock pulse is generated at node v, v performs all the actions (local computations and sending of messages) of round i of the synchronous algorithm.

Theorem 12.2. *If all generated clock pulses are valid according to Definition 12.1, the above method provides an asynchronous algorithm that behaves exactly the same way as the given synchronous algorithm.*

Proof. When the i^{th} pulse is generated at a node v, v has sent and received exactly the same messages and performed the same local computations as in the first $i-1$ rounds of the synchronous algorithm. □

The main problem when generating the clock pulses at a node v is that v cannot know what messages its neighbors are sending to it in a given synchronous round. Because there are no bounds on link delays, v cannot simply wait "long enough" before generating the next pulse. In order satisfy Definition 12.1, nodes have to send additional messages for the purpose of synchronization. The total complexity of the resulting asynchronous algorithm depends on the overhead introduced by the synchronizer. For a synchronizer $\mathcal{S}$, let $T(\mathcal{S})$ and $M(\mathcal{S})$ be the time and message complexities of $\mathcal{S}$ for each generated clock pulse. As we will see, some of the synchronizers need an initialization phase. We denote the time and message complexities of the initialization by $T_{\text{init}}(\mathcal{S})$ and $M_{\text{init}}(\mathcal{S})$, respectively. If $T(\mathcal{A})$ and $M(\mathcal{A})$ are the time and message complexities of the given synchronous algorithm $\mathcal{A}$, the total time and message complexities T_{tot} and M_{tot} of the resulting asynchronous algorithm then become

$$T_{tot} = T_{\text{init}}(\mathcal{S})+T(\mathcal{A})\cdot(1+T(\mathcal{S})) \text{ and } M_{tot} = M_{\text{init}}(\mathcal{S})+M(\mathcal{A})+T(\mathcal{A})\cdot M(\mathcal{S}),$$

respectively.

Remarks:

- Because the initialization only needs to be done once for each network, we will mostly be interested in the overheads $T(\mathcal{S})$ and $M(\mathcal{S})$ per round of the synchronous algorithm.

Definition 12.3 (Safe Node)**.** *A node v is* safe *with respect to a certain clock pulse if all messages of the synchronous algorithm sent by v in that pulse have already arrived at their destinations.*

Lemma 12.4. *If all neighbors of a node v are safe with respect to the current clock pulse of v, the next pulse can be generated for v.*

Proof. If all neighbors of v are safe with respect to a certain pulse, v has received all messages of the given pulse. Node v therefore satisfies the condition of Definition 12.1 for generating a valid next pulse. □

Remarks:

- In order to detect safety, we require that all algorithms send acknowledgements for all received messages. As soon as a node v has received an acknowledgement for each message that it has sent in a certain pulse, it knows that it is safe with respect to that pulse. Note that sending acknowledgements does not increase the asymptotic time and message complexities.

12.2 Synchronizer α

Algorithm 40 Synchronizer α (at node v)

1: **wait** until v is safe
2: **send** SAFE to all neighbors
3: **wait** until v receives SAFE messages from all neighbors
4: start new pulse

Synchronizer α is very simple. It does not need an initialization. Using acknowledgements, each node eventually detects that it is safe. It then reports this fact directly to all its neighbors. Whenever a node learns that all its neighbors are safe, a new pulse is generated. Algorithm 40 formally describes synchronizer α.

Theorem 12.5. *The time and message complexities of synchronizer α per synchronous round are*

$$T(\alpha) \;=\; O(1) \quad \textit{and} \quad M(\alpha) \;=\; O(m).$$

Proof. Communication is only between neighbors. As soon as all neighbors of a node v become safe, v knows of this fact after one additional time unit. For every clock pulse, synchronizer α sends at most four additional messages over every edge: Each of the nodes may have to acknowledge a message and reports safety. □

Remarks:

- Synchronizer α was presented in a framework, mostly set up to have a common standard to discuss different synchronizers. Without the framework, synchronizer α can be explained more easily:
    1. Send message to all neighbors, include round information i and actual data of round i (if any).
    2. Wait for message of round i from all neighbors, and go to next round.
- Although synchronizer α allows for simple and fast synchronization, it produces awfully many messages. Can we do better? Yes.

12.3 Synchronizer β

Algorithm 41 Synchronizer β (at node v)

wait until v is safe
wait until v receives SAFE messages from all its children in T
if $v \neq \ell$ **then**
    send SAFE message to parent in T
    wait until PULSE message received from parent in T
end if
send PULSE message to children in T
start new pulse

Synchronizer β needs an initialization that computes a leader node ℓ and a spanning tree T that is rooted at ℓ. As soon as all nodes are safe, this information is propagated to ℓ by means of a convergecast. The leader then broadcasts this information to all nodes. The details of synchronizer β are given in Algorithm 41.

Theorem 12.6. *The time and message complexities of synchronizer β per synchronous round are*

$$T(\beta) \;=\; O(\text{diameter}(T)) \;\leq\; O(n) \quad \textit{and} \quad M(\beta) \;=\; O(n).$$

The time and message complexities for the initialization are

$$T_{\text{init}}(\beta) \;=\; O(n) \quad \textit{and} \quad M_{\text{init}}(\beta) \;=\; O(m + n\log n).$$

Proof. Because the diameter of T is at most $n-1$, the convergecast and the broadcast together take at most $2n-2$ time units. Per clock pulse, the synchronizer sends at most $2n-2$ synchronization messages (one in each direction over each edge of T).

With an improvement (due to Awerbuch) of the GHS algorithm (Algorithm 15) you saw in Chapter 3, it is possible to construct an MST in time $O(n)$ with $O(m+n\log n)$ messages in an asynchronous environment. Once the tree is computed, the tree can be made rooted in time $O(n)$ with $O(n)$ messages. □

Remarks:

- We now got a time-efficient synchronizer (α) and a message-efficient synchronizer (β), it is only natural to ask whether we can have the best of both worlds. And, indeed, we can. How is that synchronizer called? Quite obviously: γ .

12.4 Synchronizer γ

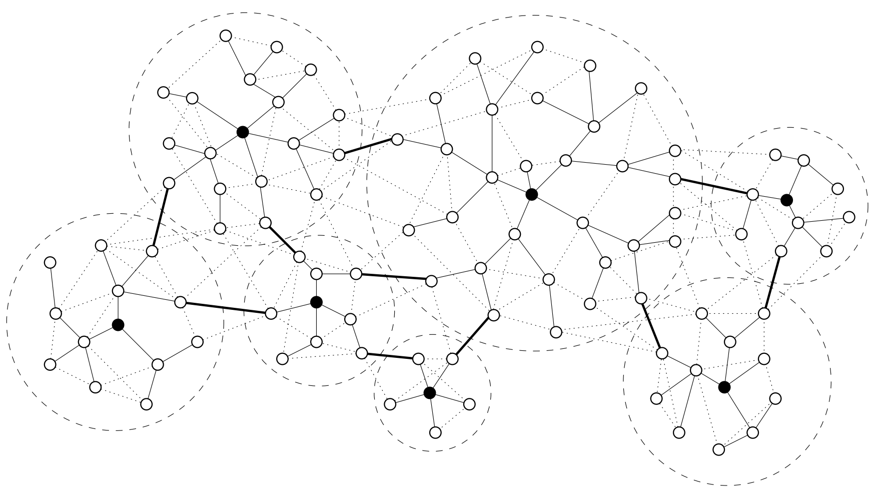

Figure 12.1: A cluster partition of a network: The dashed cycles specify the clusters, cluster leaders are black, the solid edges are the edges of the intracluster trees, and the bold solid edges are the intercluster edges

Synchronizer γ can be seen as a combination of synchronizers α and β. In the initialization phase, the network is partitioned into clusters of small diameter. In each cluster, a leader node is chosen and a BFS tree rooted at this leader node is computed. These trees are called the *intracluster trees*. Two clusters C_1 and C_2 are called neighboring if there are nodes $u \in C_1$ and $v \in C_2$ for which $(u, v) \in E$. For every two neighboring clusters, an *intercluster edge* is chosen, which will serve for communication between these clusters. Figure 12.1 illustrates this partitioning into clusters. We will discuss the details of how to construct such a partition in the next section. We say that a cluster is safe if all its nodes are safe.

Synchronizer γ works in two phases. In a first phase, synchronizer β is applied separately in each cluster by using the intracluster trees. Whenever the leader of a cluster learns that its cluster is safe, it reports this fact to all the nodes in the clusters as well as to the leaders of the neighboring clusters. Now, the nodes of the cluster enter the second phase where they wait until all the neighboring clusters are known to be safe and then generate the next pulse. Hence, we essentially apply synchronizer α between clusters. A detailed description is given by Algorithm 42.

Algorithm 42 Synchronizer γ (at node v)

wait until v is safe
wait until v receives SAFE messages from all children in intracluster tree
if v is not cluster leader **then**
    send SAFE message to parent in intracluster tree
    wait until CLUSTERSAFE message received from parent
end if
send CLUSTERSAFE message to all children in intracluster tree
send NEIGHBORSAFE message over all intercluster edges of v
wait until v receives NEIGHBORSAFE messages from all adjacent intercluster edges and all children in intracluster tree
if v is not cluster leader **then**
    send NEIGHBORSAFE message to parent in intracluster tree
    wait until PULSE message received from parent
end if
send PULSE message to children in intracluster tree
start new pulse

Theorem 12.7. *Let m_C be the number of intercluster edges and let k be the maximum cluster radius (i.e., the maximum distance of a leaf to its cluster leader). The time and message complexities of synchronizer γ are*

$$T(\gamma) \;=\; O(k) \quad \textit{and} \quad M(\gamma) \;=\; O(n+m_C).$$

Proof. We ignore acknowledgements, as they do not affect the asymptotic complexities. Let us first look at the number of messages. Over every intracluster tree edge, exactly one SAFE message, one CLUSTERSAFE message, one NEIGHBORSAFE message, and one PULSE message is sent. Further, one NEIGHBORSAFE message is sent over every intercluster edge. Because there are less than n intracluster tree edges, the total message complexity therefore is at most $4n + 2m_C = O(n + m_C)$.

For the time complexity, note that the depth of each intracluster tree is at most k. On each intracluster tree, two convergecasts (the SAFE and NEIGHBORSAFE messages) and two broadcasts (the CLUSTERSAFE and PULSE messages) are performed. The time complexity for this is at most $4k$. There is one more time unit needed to send the NEIGHBORSAFE messages over the intercluster edges. The total time complexity therefore is at most $4k + 1 = O(k)$. □

12.5 Network Partition

We will now look at the initialization phase of synchronizer γ. Algorithm 43 describes how to construct a partition into clusters that can be used for synchronizer γ. In Algorithm 43, $B(v,r)$ denotes the ball of radius r around v, i.e., $B(v,r) = \{u \in V : d(u,v) \leq r\}$ where $d(u,v)$ is the distance between u and v. The algorithm has a parameter $\rho > 1$. The clusters are constructed sequentially. Each cluster is started at an arbitrary node that has not been included in a cluster. Then the cluster radius is grown as long as the cluster grows by a factor more than ρ.

Algorithm 43 Cluster construction

while unprocessed nodes **do**
    select an arbitrary unprocessed node v;
    $r := 0$;
    while $|B(v,r+1)| > \rho|B(v,r)|$ **do**
        $r := r+1$
    end while
    makeCluster($B(v,r)$) //all nodes in $B(v,r)$ are now processed
end while

Remarks:

- The algorithm allows a trade-off between the cluster diameter k (and thus the time complexity) and the number of intercluster edges m_C (and thus the message complexity). We will quantify the possibilities in the next section.

- Two very simple partitions would be to make a cluster out of every single node or to make one big cluster that contains the whole graph. We then get synchronizers α and β as special cases of synchronizer γ.

Theorem 12.8. *Algorithm 43 computes a partition of the network graph into clusters of radius at most $\log_\rho n$. The number of intercluster edges is at most $(\rho - 1) \cdot n$.*

Proof. The radius of a cluster is initially 0 and does only grow as long as it grows by a factor larger than ρ. Since there are only n nodes in the graph, this can happen at most $\log_\rho n$ times.

To count the number of intercluster edges, observe that an edge can only become an intercluster edge if it connects a node at the boundary of a cluster with a node outside a cluster. Consider a cluster C of size $|C|$. We know that $C = B(v,r)$ for some $v \in V$ and $r \geq 0$. Further, we know that $|B(v,r+1)| \leq \rho \cdot |B(v,r)|$. The number of nodes adjacent to cluster C is therefore at most $|B(v,r+1) \setminus B(v,r)| \leq \rho \cdot |C| - |C|$. Hence, the number of intercluster edges adjacent to C is at most $(\rho - 1) \cdot |C|$. Summing over all clusters, we get that the total number of intercluster edges is at most $(\rho - 1) \cdot n$. $\square$

Corollary 12.9. *Using $\rho = 2$, Algorithm 43 computes a clustering with cluster radius at most $\log_2 n$ and with at most n intercluster edges.*

Corollary 12.10. *Using $\rho = n^{1/k}$, Algorithm 43 computes a clustering with cluster radius at most k and at most $O(n^{1+1/k})$ intercluster edges.*

Remarks:

- Algorithm 43 describes a centralized construction of the partitioning of the graph. For $\rho \geq 2$, the clustering can be computed by an asynchronous distributed algorithm in time $O(n)$ with $O(m+n\log n)$ (reasonably sized) messages (showing this will be part of the exercises).

- It can be shown that the trade-off between cluster radius and number of intercluster edges of Algorithm 43 is asymptotically optimal. There are graphs for which every clustering into clusters of radius at most k requires $n^{1+c/k}$ intercluster edges for some constant c.

The above remarks lead to a complete characterization of the complexity of synchronizer γ.

Corollary 12.11. *The time and message complexities of synchronizer γ per synchronous round are*

$$T(\gamma) \;=\; O(k) \quad \textit{and} \quad M(\gamma) \;=\; O(n^{1+1/k}).$$

The time and message complexities for the initialization are

$$T_{\text{init}}(\gamma) \;=\; O(n) \quad \textit{and} \quad M_{\text{init}}(\gamma) \;=\; O(m + n\log n).$$

Remarks:

- The synchronizer idea and the synchronizers discussed in this chapter are due to Baruch Awerbuch.

- In Chapter 3, you have seen that by using flooding, there is a very simple synchronous algorithm to compute a BFS tree in time $O(D)$ with message complexity $O(m)$. If we use synchronizer γ to make this algorithm asynchronous, we get an algorithm with time complexity $O(n + D\log n)$ and message complexity $O(m+n\log n+D\cdot n)$ (including the initialization phase).

- The synchronizers α, β, and γ achieve global synchronization, i.e., every node generates every clock pulse. The disadvantage of this is that nodes that do not participate in a computation also have to participate in the synchronization. In many computations (e.g. in a BFS construction), many nodes only participate for a few synchronous rounds. An improved synchronizer due to Awerbuch and Peleg can exploit such a scenario and achieves time and message complexity $O(\log^3 n)$ per synchronous round (without initialization).

- It can be shown that if all nodes in the network need to generate all pulses, the trade-off of synchronizer γ is asymptotically optimal.

- Partitions of networks into clusters of small diameter and coverings of networks with clusters of small diameters come in many variations and have various applications in distributed computations. In particular, apart from synchronizers, algorithms for routing, the construction of sparse spanning subgraphs, distributed data structures, and even computations of local structures such as a MIS or a dominating set are based on some kind of network partitions or covers.