CHAPTER 6.

DISTRIBUTED GLOBAL EVENT SYNCHRONIZATION

6.1. Introduction

Initially the system is asynchronous, and each node has its own local transmission clock. The objective of the synchronization procedure is to generate a global synchronous system clock. The method of achieving global synchronization is by fixing the slot duration in all the system's nets. The synchronization procedure proceeds in the following steps: (i) determining two synchronization conditions, (ii) designing two synchronization algorithms for matching the phases of the time slots on orthogonal nets, and (iii) analyzing these algorithms ([OfFa87b]).

The result of maintaining global synchronization among all the system's nets is that this distributed system preserves a **total global ordering** of all the events in the system. This total ordering is achieved with a small synchronization overhead (less than 5%).

Global event synchronization is a fundamental operating principle used for communication management and for concurrency control. It is simpler and more efficient to implement these functions under a synchronous scheme rather than an asynchronous one. The network architecture is a hypergraph which makes the synchronization simpler, and more reliable and efficient than a point–to–point network.

Each node determines time by using a slot counter. The values of the slot counters are used for stamping all computation and communication events. As a result, at any later time, on any node, and for all pairs of events A and B, it is possible to determine whether A was before, after, or simultaneous with B.

Global system synchronization enables a **well-defined global state transition** at the end of each time slot, which simplifies the operation of distributed network control algorithms (e.g., routing, buffer management), and the concurrency control of parallel processing transactions, i.e., time stamps can be used for maintaining the serializability of parallel algorithms. For example, time stamps are used in optimistic concurrency control algorithms for distributed database systems (see [KuRo81]). This algorithm is simple and enables a high degree of parallelism, higher than for techniques using only locks.

Global synchronization of a regular hypergraph and partial hypergraph, is modeled and analyzed in this chapter. The objectives of the analysis are to find the necessary conditions for maintaining global synchronization, and to find the average and worst-case synchronization efficiency for a single net and for the whole network, i.e., the time overhead for maintaining synchronization.

The broadcast to all the net's nodes from one point in space has the inherent advantages of efficient distributed synchronization and fault tolerance (Chapter 8). It will be shown that the distributed synchronization technique leads to a small reduction in the number of bits that may be transmitted in a given time interval. The synchronization efficiency is defined as the ratio of this number to the maximum number of bits that could be transmitted in that interval over the shared medium.

Each time slot has a fixed time duration T_s. The one-way delay of a node i from the star center is Δ_i. Thus, the n nodes of each net may be regarded as lying on the circumference of an imaginary circle with radius R, such that $T_R \geq \max\{\Delta_i$, such that $n \geq i \geq 1\}$, as shown in Figure 5.1. The slots are grouped into frames of duration T_f, with f slots per frame and $T_f = fT_s$, as shown in Figure 5.2. For synchronization purposes, the frame duration is equal to $2T_R$.

The synchronization mechanism is based on the periodic exchange of timing and control (or state) information, which is also used for other network functions. Each slot is divided into *r+1 minislots.* The first r are very short control minislots (CMS) used for network control.

A message sent during a CMS is broadcast over the net from a **known origin** but without a **specific destination**. The use of the CMSs on each net is deterministic. The set of n nodes is partitioned into r **disjoint subsets** of sizes either $\left\lceil \frac{n}{r} \right\rceil$ or $\left\lfloor \frac{n}{r} \right\rfloor$ nodes. The sum of the sizes of all these subsets is exactly n, and each node belongs to exactly one of the subsets. The nodes in a subset use the corresponding CMS in a round-robin fashion.

In the following analysis it is important to note that each node can use its CMS every $l = \left\lceil \frac{n}{r} \right\rceil$ time slots. As a result, each node can broadcast its *timing view* every l time slots. The *timing view* will be discussed later, and consists of the time difference observed by a node over its two orthogonal nets. The *timing views* are sent in order to match the phases of all the nets in the network.

Other synchronization procedures which are found in literature are more general and abstract, and use explicit message--passing protocol, e.g., the Byzantine Agreement which is extensively discussed in literature. A general analysis of the complexity of network synchronization is found in [Awer85]. The following synchronization scheme is based on:

(i) specific optical architecture and its parameters, and

(ii) implicit and explicit message passing.

6.2. Distributed Synchronization

6.2.1. The synchronization hierarchy

Global distributed synchronization is achieved in a hierarchical manner, as shown in Figure 6.1. The lowest synchronization level is the net. There is a slot timer at each port, which is used to measure the duration of one time slot (T_s). Knowing its delay from the net center enables each port to determine when the next time slot should begin. The next synchronization level is done in a distributed manner by matching the timing differences between the two ports at each node.

Initialization of all the slot counters is centralized and done only at startup. The concurrency control level is the distributed operating system, which uses the underlying global synchronization. The lower levels are completely transparent to the concurrency level, and can be viewed as network functions. In general, the network functions use the CMSs, while the distributed operating system uses the DMSs.

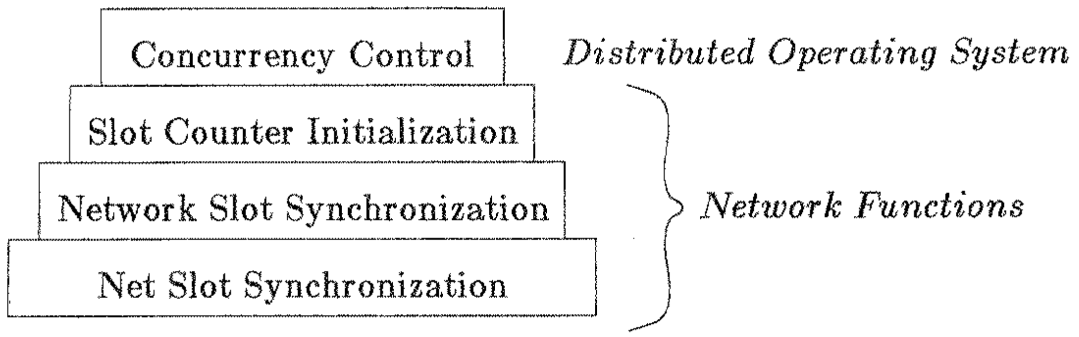
Concurrency Control
Slot Counter Initialization
Network Slot Synchronization
Net Slot Synchronization
Distributed Operating System
Network Functions

Figure 6.1: The Synchronization Hierarchy

6.2.2. Synchronization conditions for 2D hypergraph

In this section the necessary and sufficient synchronization conditions for ensuring a total temporal event ordering in the system are discussed. It will be shown how these conditions can be achieved and what the consequences are for the system's performance efficiency.

The following are the necessary and sufficient conditions for total event ordering:

SC1 – Synchronization Condition 1 (local condition):

At all times the reading of each slot counter (C_i) is not greater than the actual number of slots that have *passed* at each of the node's ports, and can be smaller by one, for durations less than half a time slot. As a result, the slots of the two ports can be **paired**, such that the time difference between the two paired slots, (one from each port), is less than half a time slot, i.e., the two streams of events (or slots) which each node "sees" via its two ports can be **uniquely paired**.

SC2 – Synchronization Condition 2 (global condition):

At all times the difference between any two slot counters C_i and C_j can be **zero** or **one**, i.e., at each time step there is an interval greater than half a time slot in which all the system's slot counters have the same value. Figure 6.2 presents a global view of the system's timing. During time interval d ($d < 0.5T_s$) all the slot counters are incremented.

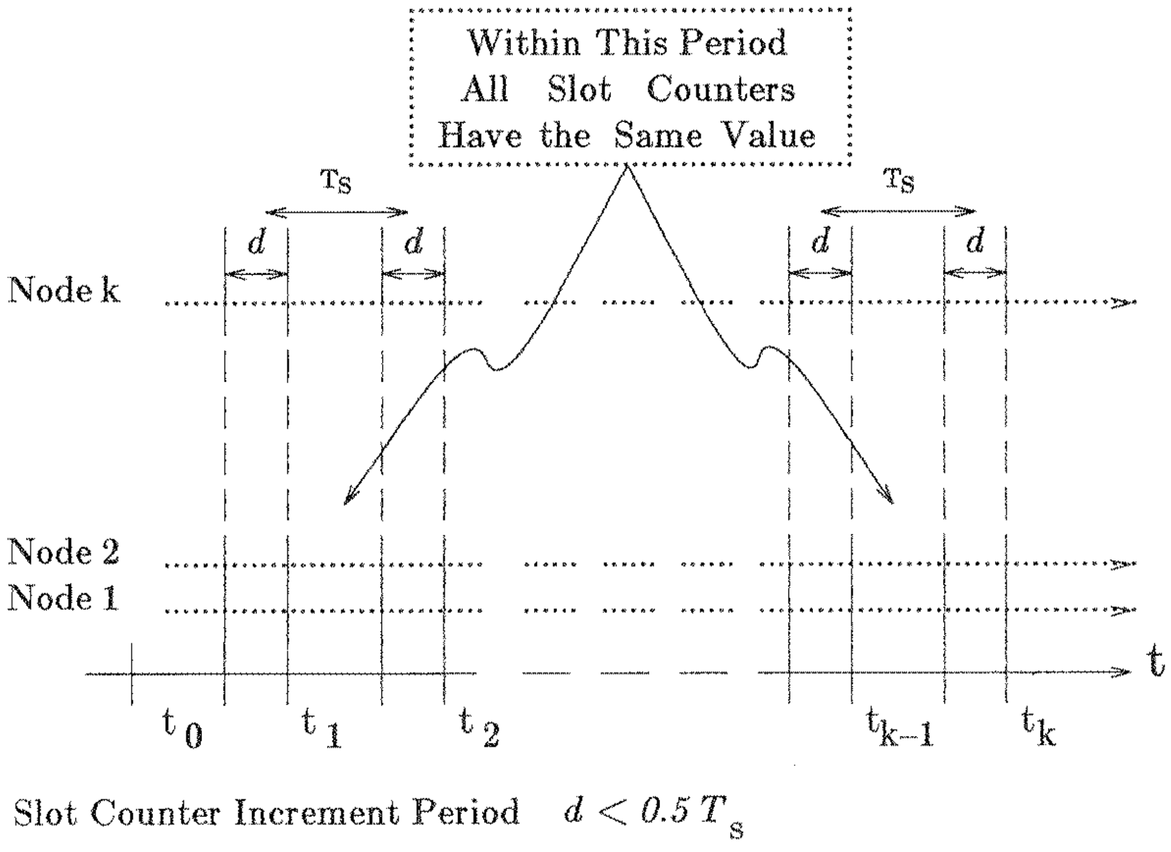
Within This Period All Slot Counters Have the Same Value
T_S
d
Node k
Node 2
Node 1
t
t_0
t_1
t_2
t_{k-1}
t_k
Slot Counter Increment Period $d < 0.5\,T_S$

Figure 6.2: Global Synchronization Condition

In the discussion the following definitions are used:

Slot Phase – of a time slot is the first bit of the first control minislot.

Early Port – of a node is the port in which its slot phase occurs before the other port slot phase.

Later Port – of a node is the port in which its slot phase occurs after the other port slot phase.

Increment – of a slot counter follows the phase of the later port.

Time Stamp – of a packet which is sent via the early port is by next value of the slot counter, and of a packet which is sent via the later port is by the current value of the slot counter. Arriving packets are stamped in the same way: the packet from the early port gets the next value, and the packet from the later port gets the current value of the slot counter.

Net Cross Time – of a packet is the time it takes to arrive to all the nodes of the net. This time is measured by stamping the packet upon its transmission and upon its arriving. Thus, the net cross time is **exactly** f time slots.

Total Ordering Proposition:–

If the synchronization conditions SC1 and SC2 are maintained, and each data packet is time stamped by its originating node, then the global total ordering of packets is preserved.

Proof:–

The proof proceeds in three steps:

Step 1: Unique Time Stamping – following SC1 and the above definitions a packet which travels in the system is stamped uniquely upon transmission and arrival. This is a result of the unique pairing of phases on each node (SC1) and the fact that a packet crosses a net within **exactly** f time slots.

Step 2: Consistent Time Stamping – Assume that all the slot counters have the same initial value, then following SC2 they will continue to have the same value outside of the incrementing period d which is less than half a time slot, as shown in Figure 6.2.

Assume that: (i) a packet p_0 is sent from node i at t_0, (ii) a copy of p_0 is kept at node i, and (iii) this copy is time stamped every time slot by the slot counter of node i; then if p_0 is stamped whenever it reaches an arbitrary node j, then the two copies of p_0 will have the same time stamp. Thus, the two copies of p_0 are time consistent.

Step 3: Total Order – Let A_{t_0} be an arbitrary subset of packets which were originated and sent into the system at time t_0. After some k slots (k is any integer greater than zero) these packets are stamped again by their destination nodes. Clearly, since time stamping is unique and consistent all these packets will be stamped with the same value. Therefore, the relative time difference between any two arbitrary subsets A and B is preserved, and the total ordering is maintained.

6.3. Synchronization Analysis

The main emphasis in this section is on the design and analysis of a two–dimensional regular hypergraph. A partial hypergraph is discussed briefly and shown to operate similarly to the regular hypergraph. The objective of the analysis is to determine the necessary **operating conditions**, such that the synchronization conditions are maintained. The analysis is done first for a single net and then for the network.

In the following analysis the system is modeled as follows:

• Regular hypergraph – each node has two ports, and each net can be accessed via n ports.

• Each frame contains one slot (i.e., $T_f = T_s$). The analysis for $f > 1$ is very similar, and will be discussed in a later section.

• $R_{\max}$ is the maximum of all R's of the system's nets.

• One frame period (T_f) is exactly the time delay for the signal to travel twice the distance $R_{\max}$, i.e., $T_f = T_s = 2T_{R_{\max}}$.

• The time slot (T_s) interval, which is measured by a timer at each port, may have a maximum error of Δ_s. This error is the sum of placement error – the maximum error in measuring the port distance from the center of the optical star and clock error – the error of the node's local clock.

6.3.1. Net synchronization analysis

The Net's Synchronization Algorithm:–

Each node i determines the beginning of the next time slot by adding ($T_s - 2\Delta_i$) to the time it receives the first bit of the first control minislot (CMS). Figure 6.3 illustrates the correctness of this simple algorithm. Each node should delay the beginning of the next time slot by adding its round–trip delay to the circumference of the circle, which is exactly ($T_s - 2\Delta_i$).

The difference in determining the beginning of the next time slot among all the n ports of the net is $\pm\Delta_s$, so that the maximum timing difference is $2\Delta_s$. Since there is

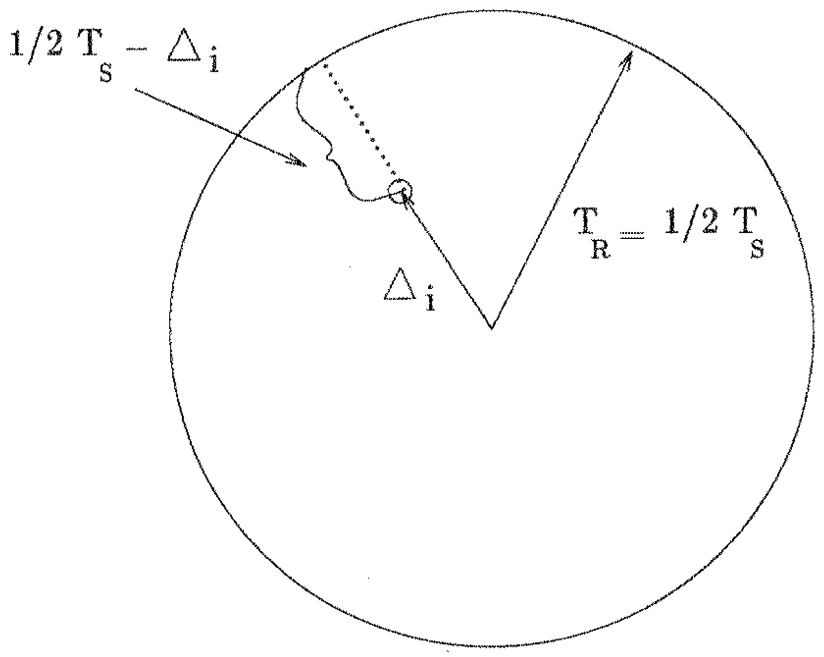

Figure 6.3: Net Synchronization

one slot in each frame, and at the beginning of each time slot all the net's ports are **resynchronized**, there is **no accumulated** timing error during successive time slots; i.e., the beginning of the next time slot is determined by the first bit of the current time slot.

First, the necessary time duration of each time slot is computed. Given T_{CMS} be the duration of each control minislot and T_{DMS} be the duration of the data minislot, what is the total duration of of time slot (T_s). Clearly, due to the error Δ_s there should be a safety margin between the transmission of two successive nodes. During each slot $r+1$ different ports are broadcasting. These ports are numbered according to the order in which they are broadcasting: {Port1, Port2, ... , Port(r+1)}, Port(r+1) sends the data packet during the DMS. Then, with respect to some global time reference $t = 0$, it is possible to determine when each port should transmit:

If	Port1 transmits at	$t = 0,$
then	Port2 transmits at	$t = 2\Delta_s + T_{CMS}$
:	: :	: :
and	Port(r) transmits at	$t = (r-1)(2\Delta_s + T_{CMS})$
and	Port(r+1) transmits at	$t = r(2\Delta_s + T_{CMS})$
and	then the next time slot will start at	

$$t = r(2\Delta_s + T_{CMS}) + 2\Delta_s + T_{DMS}$$

Hence, the slot duration should satisfy the following equality:

$$T_s = (r+1)2\Delta_s + rT_{CMS} + T_{DMS}$$

The above result on the duration of T_s is proven by induction on r.

The average efficiency ($\mu_{net-ave}$) of a single net is the fraction of each time slot used for transferring useful bits of information (data or control):

$$\mu_{net-ave} = \frac{rT_{CMS}+T_{DMS}}{(r+1)2\Delta_s+rT_{CMS}+T_{DMS}} = \frac{T_s - (r+1)2\Delta_s}{T_s}$$

The actual slot duration is the time interval between the first bit of the first CMSs in two successive slots. The actual length of T_s depends on the relative error between these two ports which is $\pm\Delta_s$. Hence, the time slot duration varies between $T_s - \Delta_s$ and $T_s + \Delta_s$. Therefore, the worst–case efficiency ($\mu_{net-min}$) is when all the net's ports have an error of $+\Delta_s$, and the efficiency is then

$$\mu_{net-min} = \frac{rT_{CMS}+T_{DMS}}{T_s+\Delta_s}$$

6.3.2. Global synchronization analysis

The basic mechanism for global synchronization is that a net which completes its time slot before other nets will delay the beginning of its next time slot, in order to match the other nets' time phases. Via the CMSs, the nodes indicate to their orthogonal nets how to match the phases.

The Distributed Global Synchronization Algorithm:–

(i) each node increments its slot counter according to the phase of the later port; (ii) each port indicates in its message the amount of time needed to match the phases of its two ports (only the port in which the time slot ended earlier will indicate a positive time delay); (iii) the additional delay information is incorporated only when it becomes **common knowledge** *by all the net's ports, and (iv) there are r different time interval corrections (opinions) for adjusting the beginning of the next time slot, and the ports use the* **longest** *one among these r corrections.*

The principle for achieving synchronization by taking the latest (or maximum) among different time phases is also used in [Lamp78]. The above global synchronization algorithm eliminates the resynchronization uncertainty, which could occur if a node would resynchronize before the timing information becomes **common knowledge** on the net. The global synchronization algorithm is used explicitly in the computation of the maximum timing difference; as a result the validity of this algorithm becomes apparent.

Each net is **resynchronized** by the first bit of the first control minislot. Then, for global synchronization analysis, the possible timing error in determining the beginning of the next time slot is $\pm\Delta_s$. For the **worst–case**, assuming that all the nodes of the system have the maximum timing error of $+\Delta_s$, the minimum global efficiency is

$$\mu_{global-min} = \frac{rT_{CMS} + T_{DMS}}{(r+1)2\Delta_s + rT_{CMS} + T_{DMS} + \Delta_s} = \frac{T_s - (r+1)2\Delta_s}{T_s + \Delta_s}$$

If a uniform error distribution is assumed, the average efficiency will still be close to the worst–case, since the global synchronization algorithm **forces** the nets to maximize their timing error.

The actual reduction in the system's efficiency in the worst–case is very small; in most practical situations it will be a few percent. For example, if $\Delta_s = 100\, bits$, $T_s = 8000\, bytes$, and r = 4, then $\mu_{global-min} = 98.28\%$. Note also that the efficiency does not depend on the number of nodes in the system, and that the timing error Δ_s can be much smaller.

Note that the part of the CMS which indicates how to match the phases of the two orthogonal nets causes an additional decrease in the network efficiency. Since this is very small, it is not included in the above expression.

6.3.3. Global timing differences

The maximum time difference should satisfy the local and global synchronization conditions. In the following analysis the worst phase difference between any two nets in the system is computed. Clearly, if this phase difference is less than half a time slot then SC1 holds (by its definition), and also SC2 holds since the phases of orthogonal nets are paired uniquely (SC1), so that the increment of all the system's slot counters cannot differ by more than half a time slot.

Worst–case scenarios – might bring about the worst timing difference in the system. There are two equivalent worst–case scenarios; the first one is shown in Figure 6.4. Under this scenario, (i) all the nodes of one net (say, **maxnet**) have the maximum timing error of $+\Delta_s$, (ii) all the other nodes have the minimum timing error $-\Delta_s$, and (iii) none of the nodes of **maxnet** transmit on the other nets for the maximum possible period of time, which is $l = \left\lceil \frac{n}{r} \right\rceil$ slots. In other words, **maxnet** does not tell the other nodes its differences for l time slots.

The second equivalent worst–case scenario has two orthogonal **maxnets**, and the rest of the nodes have minimum error. The nodes of the two **maxnets** do not transmit on their orthogonal nets for a maximum interval of l slots. These two scenarios are easily proved to be the worst, by systematically checking all possible scenarios.

Under this scenario for f =1, the maximum timing difference accumulated is

$$\Delta_{\max} = (\left\lceil \frac{n}{r} \right\rceil + 2)2\Delta_s .$$

In order to satisfy the synchronization conditions, the following inequality should be satisfied:

$$\Delta_{\max} < \frac{1}{2} T_s .$$

For example, if the net size is n=100 nodes (the system size is 10,000 nodes), and the other parameters have the same values as in the previous example, then $\Delta_{\max} = 5400\ bits$, which is much smaller than half the time slot size of 32,000 bits.

Hence, synchronization conditions for achieving total event ordering are feasible, even for very large systems.

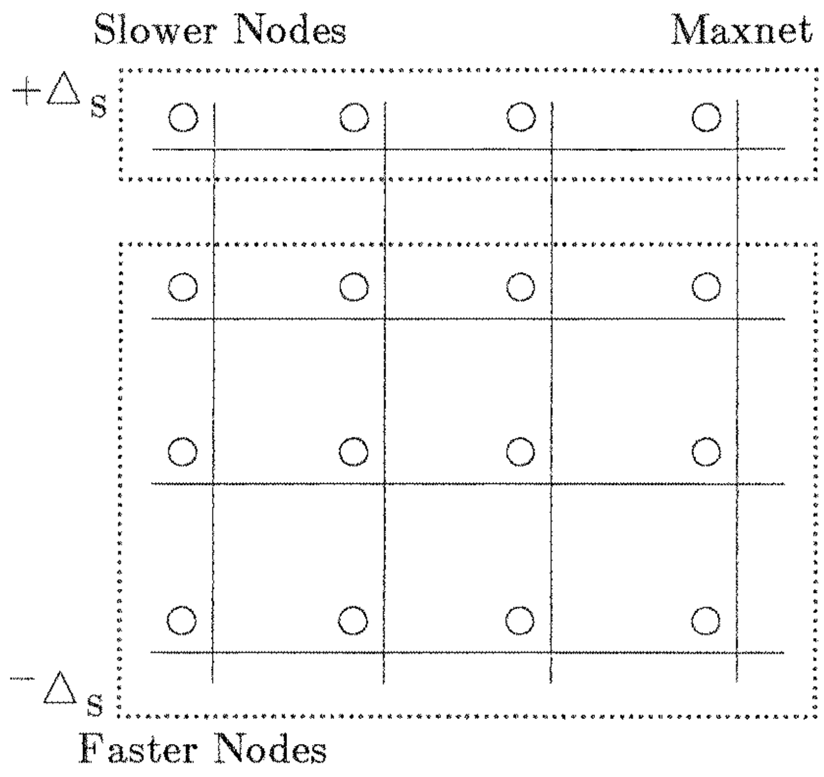

Figure 6.4: Maximum Timing Difference

6.3.4. Global initialization of the slot counters

The following are procedures for loading all the system's slot counters with the same value. These global initialization procedures can be done in a distributed or centralized manner. The following algorithms are for a 2D–R hypergraph.

6.3.4.1. Distributed initialization

This procedure is similar to the global synchronization algorithm. Each node selects at random an initial value for its slot counter. This value is transmitted by the control message during the CMS. When a node receives a slot counter value of another node, it adjusts this value by adding the appropriate propagation delay

(f +1 time slots), and then compares this value with the current value of its own slot counter. If the slot counter value is smaller, then it is replaced by the received value. Clearly, after $\left\lceil \frac{n}{r} \right\rceil + (f+1) + \left\lceil \frac{n}{r} \right\rceil + (f+1)$ time slots the largest among all the randomly selected slot counter values will reach all the nodes of the 2D–R hypergraph.

The above procedure is very fast (fraction of a millisecond); the problem is that the start–up phase of the system is much longer, i.e., the distributed initialization procedure is completed long before all the system's nodes become active. Therefore, it is reasonable to use a centralized algorithm when the system is initialized.

6.3.4.2. Centralized initialization

The following algorithm is centralized but can be performed by any of the system's nodes (the node can be selected at random). It has two phases (for a 2D–R hypergraph): phase 1 – a specific node broadcasts on one of its nets the message *"be ready to load t_0 into the slot counter"*. Then in phase 2 – the nodes which received this message broadcast the following message via their other port: *"load $t_0 + i$ into the slot counter"*. The number i indicates the number of slots passed between the original message (phase 1) and the time this message is actually transmitted. If n is the number of nodes on a net, then after at most $(n+1)$ time slots all the system's slot counters have the same value. Note that during this initialization procedure the synchronization conditions hold, and therefore, the value of i is consistent for all the nodes in the system.

6.3.5. Synchronization of a 2D partial hypergraph

In the following discussion the system topology is changed, such that each net has k nodes with 2 ports, and a nodes with one port $(k+a=n)$, as shown in Figure 2.3 and discussed in Section 2.2.

The efficiency results for the net and network synchronization remain the same as obtained before. The scenario for the worst–case timing difference is the same as for the regular hypergraph, but the consequences are slightly different. The nodes of **maxnet** cannot update the rest of the system's nodes within one time slot, i.e., the nodes with one port on nets which are parallel to **maxnet** have a distance of two nets from the nodes of **maxnet**. Now, if none of the k nodes (with two ports) of one of the nets, which is parallel to **maxnet**, will transmit before all the a nodes transmitted, then the worst case timing difference between **maxnet** and these a nodes is

$$\Delta_{max-part} = (\left\lceil \frac{n}{r} \right\rceil + \left\lceil \frac{a}{r} \right\rceil + 3)2\Delta_s .$$

Hence, in order to satisfy the synchronization conditions for the partial hypergraph, the following inequality should be satisfied

$$\Delta_{max-part} < \frac{1}{2} T_s .$$

6.4. Synchronization Analysis of a Larger Net

In the previous analysis it was assumed that each frame has exactly one slot. As the bandwidth increases, the physical length of the slot decreases, and the net can have more nodes in a larger area. If the number of bits in a slot remains the same, then every frame will have several slots, i.e., $T_f = 2T_R = fT_s$.

The synchronization conditions and algorithms remain the same. The consequences of this change can be on the synchronization efficiency and the global time difference. It is assumed that Δ_s in bit periods remains the same, even when the bandwidth increases.

6.4.1. Synchronization efficiency

The net and network synchronization efficiency is basically the same as in the previous analysis. This can be shown by assuming that each frame has $f(r+1)$ minislots, which implies that, for safety, $f(r+1)$ [illegible] Δ_s is added to the time frame. This safety margin is f times larger than before, but also the frame duration (in bit periods) is f times larger than before. Thus, the synchronization efficiency remains the same.

6.4.2. Global time difference

The worst–case scenarios remain the same, but there is additional time delay in order to match the phases of orthogonal nets. This additional time difference consists of: (i) f slots due to the delay of the node view on how to match the phases, and (ii) f slots due the delay until the phase match information becomes **common knowledge**. Thus, the maximum timing difference accumulated is

$$\Delta_{\max}(f > 1) = (\left\lceil \frac{n}{r} \right\rceil + 2f)2\Delta_s .$$

In order to satisfy the synchronization conditions, the following inequality should be satisfied:

$$\Delta_{\max}(f > 1) < \frac{1}{2}T_s .$$

6.5. Discussion and Conclusions

The foregoing analysis exhibits the feasibility of achieving global event synchronization of a large area network with very high efficiency. This is due, first, to the use of a centralized, passive, optical star which can be simply synchronized; second, to the high bandwidth optical medium which can be time–shared by many ports, and third, to the two–dimensional hypergraph topology, which minimizes the distances between nodes. The limitation of global synchronization is that all events should be of the same size. The event size is one time slot, and for efficiency purposes should have a duration of several kilobytes.

Global synchronization can have an important impact on the overall complexity of the system's operation and computation control (see the following example on global mutual exclusion). Parallel distributed algorithms can use **common knowledge** as their basic principle. Since each event has its own time signature and the system is synchronized, these algorithms can be performed in an **open loop mode**, i.e., without the need for **acknowledgements**. This has two important consequences: (i) the execution of these distributed algorithms is efficient, and (ii) the system operation is simpler. There appears to be a tradeoff between a somewhat complex network operation with synchronization and the simplicity of the higher levels of system operation (communication and computation management).

6.5.1. Maximum clock synchronization

The synchronization procedure described above may be viewed as generating a slower synchronous clock from multiple asynchronous clocks. The maximum timing difference on a 2D–R hypergraph (with $f=1$) is $\Delta_{\max} = (\left\lceil \frac{n}{r} \right\rceil + 2)2\Delta_s$. This timing difference constitutes the maximum synchronous rate of the system's slot counters, which is the reciprocal value of the minimum slot period (basic event) in the system

$$SLOT\text{–}CNT_{\max} = \frac{1}{T_{s_{\min}}} = \frac{1}{4\,(\left\lceil \frac{n}{r} \right\rceil + 2)\,\Delta_s}.$$

Let $w = 4\left\lceil \frac{n}{r} \right\rceil$. The error Δ_s is a function of (i) the frequency differences among the high–speed transmitting clocks – $\pm b$ (typically less than 0.01%), and (ii) the physical measurement error $\pm d$.

$$\Delta_s = d + bT_s$$

Therefore,

$$T_{s_{\min}} = w(d + bT_{s_{\min}})$$

Thus,

$$T_{s_{\min}} = \frac{wd}{1 - wb}$$

Finally, the maximum synchronous clock is

$$SLOT\text{–}CNT_{\max} = \frac{1}{T_{s_{\min}}} = \frac{1 - wb}{wd} \approx \frac{1}{wd.}$$

Thus, the maximum system clock does not depend on the high speed clock differences but only on the physical measurement (d). The synchronization depends only on the number of nodes per net and not on the total number of nodes in the system. If the number of CMSs is increased, then the maximum system clock bandwidth is proportionally increased.

For example, let n=32, r=4, d=25 nanoseconds, w=40, and the maximum bandwidth is

$$SLOT\text{--}CNT_{\max} \approx \frac{1}{wd} = \frac{1}{1000} = 1 \ megahertz$$

6.5.2. Synchronous and asynchronous operation

A more general view of the synchronization procedure is possible. Each event may have multiple asynchronous subevents; therefore, the synchronization procedure can be viewed as placing an upper bound on the maximum timing difference among the asynchronous subevents.

One possible conclusion is that in order to have a more effective synchronization, the duration of each subevent should be longer, i.e., in the case of distributed computation, a larger subevent means a larger process granularity.

6.5.3. Distributed mutual exclusion

In order to illustrate the potential of global event synchronization and time stamping, a distributed mutual exclusion algorithm is presented in this section. They are designed for a regular hypergraph and use the CMSs or DMS for sending control messages. The mechanism discussed here allows the sharing of *independent, replicated items* among collections of distributed, cooperating processes. An *item* is a portion of a process (usually a data structure) that must be accessed atomically. For more details see [McOf87].

Net (or local) distributed mutual exclusion algorithm :– the node, wishing to acquire the control of a common resource or a shared item, broadcasts its request to all nodes of the net. A time stamp is attached to this message which is sent via a DMS. Since only one node can use the DMS during each time slot, no two messages on a net can have the same time stamp. Hence, the message with the earlier time stamp receives the resource control rights. Note that, since all messages are broadcast over the net, the mutual exclusion resolution is computed independently by each node, and no acknowledgement is needed.

Globally distributed mutual exclusion algorithm :– this algorithm uses the CMSs for sending control messages to all the system nodes and is performed in two steps.

First, the requesting node broadcasts the message *"node i wants x at time t_0"* (t_0 is the time stamp) to all its adjacent nodes via one port. Then these nodes rebroadcast the message via their adjacent nets. If these messages have the highest priority, then after $\left\lceil \frac{n}{r} \right\rceil + f + 1$ time slots, at most, the resource request message has reached all the network nodes. The requesting node can determine the validity of its mutual exclusion request by comparing the time stamp of its original request message with all other messages requesting this resource. In the case of two requests of the same resource originating at the same time, the procedure repeats after a random delay.

The case of having too many mutual exclusion requests at the same time is discussed in [McOf87]. One possible solution is a distributed protocol which limits the maximum number of requests on a net within a period of time of $l = \left\lceil \frac{n}{r} \right\rceil$ time slots.

In this case the first step lasts l slots and each node may try to reserve a space (by broadcast over the net) for its request which will be broadcast on all the orthogonal nets during the second step. If no space is available the node will request it again after l slots. The second step also lasts l slots, so the node should wait $2l$ time slots in order to determine its exclusive right.