DISTRIBUTED SYSTEMS—TOWARDS A FORMAL APPROACH

GÉRARD LE LANN
IRISA—Université de Rennes—BP 25 A
35 031 Rennes Cedex, France

Packet-switching computer communication networks are examples of distributed systems. With the large scale emergence of mini and micro-computers, it is now possible to design special or general purpose distributed systems. However, as new problems have to be solved, new techniques and algorithms must be devised to operate such distributed systems in a satisfactory manner. In this paper, basic characteristics of distributed systems are analysed and fundamental principles and definitions are given. It is shown that distributed systems are not just simple extensions of monolithic systems. Distributed control techniques used in some planned or existing systems are presented. Finally, a formal approach to these problems is illustrated by the study of a mutual exclusion scheme intended for a distributed environment.

1. INTRODUCTION

Computer communication networks using packet-switching technology provide for the interconnection of data-processing equipments of any kind. Such systems, sometimes simply referred to as computer networks, may be viewed as multi-macroprocessors whenever the goals of resource-sharing are achieved. With the large-scale emergence of mini and microcomputers, it is now possible to envision building general or special purpose multimini and multimicrocomputers to be operated in a non-centralized manner. The need for automatic resource-sharing arises here as in a similar way it does for multimacroprocessor systems.

Two kinds of resources must be considered :
- system resources, multi-accessed by users and for which multiplexing is required (hidden sharing)
- user resources, which users agree to share according to some protocol of their own (explicit sharing).

This paper discusses the problems of system resource-sharing in a distributed environment. An example of a user-sharing problem is distributed data-base sharing.

2. DISTRIBUTED SYSTEMS-ELEMENTS FOR A FORMAL APPROACH

Experimental and public packet-switching computer communication networks have been built and operated since 1968 ; examples are Arpanet, [7], Cyclades, [13], EIN, [1], Telenet, [17] and Datapac, [4]. The communication subnet of these networks is an example of a distributed system : all nodes have equal rights and responsabilities and no central "controller" is needed for the subnet to switch packets. Other examples are multicomputers like DCS [6] and Pluribus [8]. Finally, some multimicroprocessors currently planned by manufacturers will include "distributed features", in particular, those designed to have high availability characteristics.

A definition of what is meant by distributed system is needed. Then, analysis of the fundamental properties of computer systems makes it possible to tell whether or not a given system has distributed features.

2.1 Definitions

In a computer system, system resources are not accessed as such by users but through a set of services usually referred to as an operating system. Examples of services which we call operating functions are : communication, user scheduling, resource allocation, hardware resource handling, system data management, ... These functions are run through processes called logical entities.

Let $F = \{f_i, i \varepsilon I\}$ be the set of operating functions and $E_i = \{e_j^i, j \varepsilon J(i)\}$ be the set of entities participating in fuction f_i. At any instant t, it is possible to define $s_t(e_j^i)$ as the instantaneous state of entity e_j^i. It is therefore theoretically possible to define the global state of E_i at instant t as the vector $S_t(E_i) = \{\ldots, s_t(e_j^i), \ldots$ for all $j \varepsilon J(i)\}$.

A system will be said to be f_i-centralized if there exists $k \varepsilon J(i)$ such that $S_t(E_i)$ is known to e_k^i.

An example is a system in which $\dim(E_i) = 1$; another example is a system in which entities run the operating function f_i by using a common "system table".

A system will be said to be totally centralized if it is f_i-centralized for any $i \varepsilon I$.

In contrast, a system will be said to be f_i-distributed if there does not exist $k \varepsilon J(i)$ such that $S_t(E_i)$ is known to e_k^i.

A system will be said to be totally distributed if it is f_i-distributed for any $i \varepsilon I$.

Typical cases of distributed systems are systems in which cooperating entities do not share the same physical space and/or do not have a common time reference. In such systems, an entity may get either a partial and coherent view of the system or a complete but incoherent view of the system, coherence meaning that observations are made at the same moment in the system (absolute time). This absence of uniqueness both in time and space has very important consequences.

2.2 Time and space

It is well known in Physics that the sentence "event E occured at time t" is meaningless if there is no indication about which time reference is used. Similarly, the statement "I am in location 1" has no meaning as long as a space reference and a time reference have not been defined. Then, to the purpose of describing the behaviour of an entity, we will use a precise time-space reference where internal states, time lengths, names and instants may be defined.

We define the absolute Reference as the ideal reference such that speed of communication between this reference and any other time-space reference is infinite and where every space location may be given a unique name.

(i) System properties

An entity may be viewed as a finite-state automaton; a decision to switch to a new state is possible through the observation of a specific sequence of events received during a time period (t-a, t), measured in the local time reference.

property Q : a is a finite value
property M : there are several time-space references for entities in the system

All systems studied here are assumed to have properties Q and M.

The propagation delay between two entities is the time needed, as measured in the absolute Reference, to transmit an elementary signal from one entity to the other.

property P_1 : for any pair of entities, propagation delays are fixed, finite and known with absolute accuracy ; they may be different for each pair
property P_2 : propagation delays are variable, finite and their values are not known with absolute accuracy
property P_3 : propagation delays are variable, finite and known a posteriori with absolute accuracy
property P_4 : propagation delays are variable but their values are upper bounded.

We should mention that these properties are common to all systems that are spacially distributed with finite propagation delays, including, for example, conventional logic design. Formalization of these properties was felt necessary so as to infer from them, basic principles which should be useful to distributed system designers.

(ii) Classification

- Systems with properties Q, M and P_1 will be called Perfect Multireference Systems (PMS)
- Systems with properties Q, M and P_3 will be called Quasi-Perfect Multireference Systems (QPMS)
- Systems with properties Q, M and P_2 will be called Multireference Systems (MS).

Let V be a sequence of events occuring within a system ; let E be the set of entities observing V. Events may be observed in two ways : referenced within a time-space reference R_i, iεI, with I being the set of the time-space references of E or referenced within the absolute Reference. It may be interesting to consider the following problems :
(a) is it possible to build in R_i, for any iεI, the absolute chronological ordering of events (as observed in the absolute Reference) ?
(b) do all the entities of E observe identically the set of events V ?

Answers to these questions are given in table 1.

Table 1

	PMS	QPMS	MS
(a)	YES	YES	NO
(b)	YES*	NO	NO

* if the value a (property Q) is the same for all

A graphical representation of properties P_2, P_3 and P_4 may help to answer questions (a) and (b) ; an example is given in fig. 1.

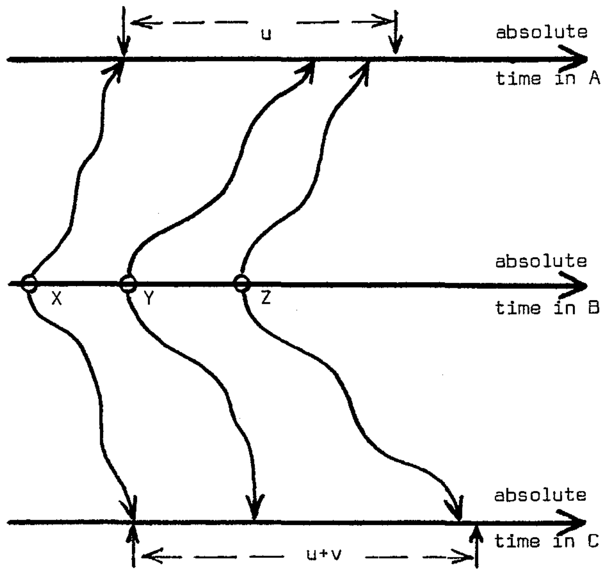


Fig. 1

In this example, three time-space references are represented. Three events X, Y, Z originated in B must be reported to A and C. It is easy to see that there may exist cases for which A, B and C will not be in agreement ; for instance, C may miss the observation of event Z, whatever the finite value of u+v. Thus, if time is the only dimension used to achieve cooperation between entities, systems having property P_4 are identical to Multireference Systems.

(iii) Principles for non-perfect systems (QPMS and MS)

(a) Principle of time nondeterminancy

For any given sequence of events, it is impossible to prove that two different entities will observe identically this sequence of events. An example of an event is a change of internal state ; as these changes are observable from the outside only through communication, one may state another principle ;

(b) Principle of relativistic observation

In a non-perfect system, it is impossible to prove that any two entities will have the same global view of a given subset of the system.

As a consequence, the environment of any entity cannot perceive the pair (t, e(t)) with an absolute certainty, t being the entity local time and e(t) the current internal state of the entity. It will be possible either to know e(t) and to state that the entity will reach that state in a predefined time interval Δt with some probability or to pick up an instant t and not being able to associate with certainty a given state e(t). Then we have the following principle :

(c) Principle of state nondeterminancy

The instantaneous state of an entity may only be expressed in terms of possible values associated with some probabilities.

This means that the global state of a non-perfect system does not exist. As a consequence, according buted systems.

It should be pointed out that if we agree to consider infinite values for propagation delays, then the above principles are valid a fortiori.

In fact, reunion of properties P_2 and Q imposes that entities sometimes have to consider a propagation delay as being infinite. This may, or may not be in accordance with reality. A propagation delay is truly infinite when the signal never reaches the destination entity; this may happen for two reasons :
- entities in charge of communication handling failed
- the destination entity failed (was excluded from the system) before receiving the signal.

In some cases, failure-tolerant techniques are mandatory. There is no essential difference with techniques intended for distributed systems. An example of this approach is given in 4.

As was pointed out before, system resources are accessed through operating functions. For every function, the decision about how and when to run the related entities is what is called control. Obviously, in a centralized system, control is performed by the "central" entity for that function (e_k^1 in the definition). According to the basic principles, problems to be solved with distributed systems are :

- control must be achieved without knowledge of the global state. Therefore, what is needed is that each entity behaves according to some algorithm working on an approximation of this state. In spite of this uncertainty, these algorithms must be such that entities are kept in a legitimate state and that the global system behaviour is a convergent process

- there is no entity which is, a priori, in charge of performing the control. Then, one is left with the problem of designing a common and secure protocol providing for the distribution of control among entities. It must also be proved that algorithms and protocols do not lead to inconsistancies and deadlocks.

According to the basic principles, time should not be used to achieve synchronization between entities.

It it now possible to refine our terminology : what is called a distributed system in this paper is any non-perfect multireference system with distributed control for some of the operating functions.

Examples of solutions to the above problems will now be presented.

3. EXAMPLES OF DISTRIBUTED SYSTEM TECHNIQUES

3.1 Congestion control and end-to-end flow control in computer communication networks

End-to-end flow control allows for monitoring the transmission of data between a source entity and a destination entity, a specific sub-system being in charge of performing the real transmission. The purpose of congestion control is to guarantee that the resources of the transmission sub-system are always used efficiently, i.e. that fair sharing of the sub-system between several source-to-destination flows is possible and that deadlocks may be either avoided or suppressed. Referring to the introduction and thinking in terms of hierarchical systems, end-to-end flow control is a problem of user resource-sharing at a given level, whereas congestion control is a problem of system resource-sharing at the next level, this being true for any level in the hierarchy. Similar views are discussed in detail in [15].

(i) Congestion control

Among the various existing schemes, isarithmic congestion control is one for which extensive analysis has been performed, [16]. A constant amount of permits is maintained over the network ; the algorithm for the entities is the following one : a new input packet is accepted only if the local permit value is greater than zero ; when a packet reaches its destination, the entity may either store the corresponding permit or ship that credit to another entity depending on whether or not a given threshold value has been reached for the amount of local permits.

Here, control is completely and permanently distributed over the entities. Isarithmic control appears as an efficient mechanism for avoiding global congestion. Nevertheless, this technique does not fulfill the requirement of resistance to entity failures and no solution has been proposed as yet for controlling losses of permits.

Most of congestion control mechanisms are uniform, in the sense that they do not discriminate between traffic sources. Some mechanisms, for example CLL in Cigale, [14], attempt to intervene only on those traffic sources which contribute to the load.

(ii) End-to-end flow control

The purpose of end-to-end flow control is to monitor data transmission between two entities such that the resources of these entities are properly utilized and the entities are kept synchronized. Because of different physical spaces and time references, existence of errors and duplication in transmission, traditionnal mechanisms like the producer-consumer scheme are not practicable ; this view is consistent with the conclusions of a recent analysis described in [18].

A basic scheme which is now widely accepted and was first introduced in Cyclades [3] is the Window mechanism, [2, 19]. Messages to be transmitted are sequentially numbered by the sender. This allows the receiver to detect loss and duplication of messages. The flow of data is acknowledged up to the first missing message. Together with the acknowledgment number (a), a credit value (c) is returned to the sender meaning that transmission of messages is allowed up to number (a+c). It can easily be shown that the Window mechanism is resistant to erroneous, lost or duplicated messages.

3.2 Load-sharing in multicomputer systems

The operating function considered here is processor-assignment in a system where processors are logically equivalent, i.e. they all have the same capabilities. Entities are processes responsible for computing processor load and running the load-sharing algorithm. System-wide throughput and response time are optimal when the load is equally distributed over the processors.

Automatic load sharing should only be performed by the entities themselves and not by the external requestors. This provides for the necessary independance between different logical levels in the system. As a consequence, processor failures, extension/reduction of the configuration and changes in the hardware topology are totally irrelevant to the other system levels.

Two methods for achieving automatic load-sharing are now presented.

(i) Diffusion technique

This technique is similar to adaptive routing mechanisms. For every entity, the notion of a neighbour is defined ; regularly, entities exchange a load-vector V = {l, i} with their neighbours, l being the minimum value of the loads most recently received by the entity and i being the identity of the corresponding processor. Then, at any moment, upon receiving an external request, an entity is able to route it immediately to the less loaded processor. Stability conditions must be computed according to hardware performances and processing requirements.

(ii) Circulating vector technique

A successor is defined for each entity, such that all entities are located on a virtual ring. On this ring, one or several load vectors circulate, the dimension of the vectors being equal to the number of entities. The individual algorithm simply consists of each entity updating its own component with the current load value upon receiving a vector, recording a copy of it and transmitting this vector on to the ring. Notice that loss of a vector or creation of several vectors is not catastrophic to the system.

The efficiency of these techniques has been evaluated by means of a simulation model. Some results may be found in [10]. These mechanisms may be used as they stand to distribute the load evenly in a system or they may be used in connection with some other mechanisms when it is necessary to take locality constraints into account.

3.3 Distributed allocation of resources

The problem is the following one : U-entities (users) must be allocated R-entities (resources) ; specific entities are in charge of multiplexing several U-entities and performing the resource allocation (in a system described somewhere else [9], these entities are called controllers). A communication subsystem is used by the controllers to send their requests directly to R-entities ; how should deadlocks be avoided ? From several techniques, the circulating control token scheme is now presented.

For every controller, a successor is defined such that controllers are located on a virtual ring. One representation of a n-controller ring may be {i→i+1, modulo n, i∈(0,n-1)}, each integer i being the identity of a controller. Asynchronous and natural time division is achieved by the means of a unique control token circulating on this virtual ring ; a controller is allowed to send allocation requests only when it owns the control token ; R-entities are provided with waiting files in which requests are recorded, up to a pre-defined limit (congestion control). When all requests have been answered, the control token is transmitted to the successor on the ring ; later, the U-entity will receive a message from each requested R-entity indicating that it has now moved to the first position in the file and that utilization of the resource is allowed. The average number of R-entities to be requested at a time is not independant of the circulating speed of the control token and it influences directly the system-wide job throughput. Extensive simulation described in [12] has been performed to evaluate the performances of this technique.

This is an example of a technique which must be shown to survive failures ; of major concern is the guarantee that the control token (CT) is never lost and that there is only one token on the ring. A protocol fulfilling this requirement exists and is now presented.

4. A SECURE PROTOCOL TO ACHIEVE MUTUAL EXCLUSION IN A DISTRIBUTED SYSTEM

We will discuss problems related mainly to controller failures such as what we should do when the controller which owns the CT goes down and thus removes the CT from the ring ?

4.1 Ring failures

First, assume that an error control mechanism based on "life messages", [7, 9], is provided at the hardware level which allows for the virtual ring reconfiguration. Then, controller i may be temporarily excluded from the ring, the successor of controller i-1 being controller i+1.

Second, let us briefly discuss problems related to failures of the interconnection structure (unibus, multibus, digital loop, multidrop telephone line, radio/satellite communication channel, matrix switch, store-and-forward networks,...). Transmission errors are easily recovered by using a simple mechanism like the Window technique. If the structure does not provide for more than one physical path between any pair of controllers, then it is of no help to design a failure tolerant distributed protocol ; it the structure does so, then failure of a structure subset can be controlled and recovered by using well known techniques like adaptive routing or alternate fixed routing.

4.2 The protocol

The protocol consists of a precedence rule and an election phase algorithm.

(i) Hypothesis

- controllers identities are integer values : (0, n-1) for n controllers (H1)
- controllers may access the header of messages circulating on the ring
- the CT and other tokens are empty messages (only the header)
- each controller owns a timer ; this timer is reset at each CT occurence
- the system is asynchronous ; timer values are not necessarilly identical ; if they are, no consequence can be drawn from this (principle of time nondeterminancy) ; each controller is provided with its own time clock (H2)
- the sequence of messages on the ring is unchanged, i.e. messages received by a controller are retransmitted FIFO
- creation of a CT is an instantaneous action
- when timer awakes, the controller generates instantaneously a token carrying its own identity ; this token is candidate for being the new CT

(ii) Precedence rule

A controller which has generated a token and which receives the CT before its own token has completed a period must remove this token from the ring.

Indeed, "early" generation of a token may be due to large round trip times for the CT, short timer values and so forth. In any case, as a CT is circulating on the ring, there is no need for local action.

(iii) Election phase algorithm

The problem is to design an algorithm such that one can prove that, when the control token (CT) is lost, there is a unique controller to be elected as responsible for regenerating a new CT (constraint A), in a finite time delay (constraint B).

Let I be the set of controllers participating in the election i.e. controllers for which timers awake between the loss of CT and regeneration of a new CT; let S(i), i∈I, be the set of tokens identities as recorded by controller i after a complete rotation of token i ; obviously, one of these identities will be i itself.

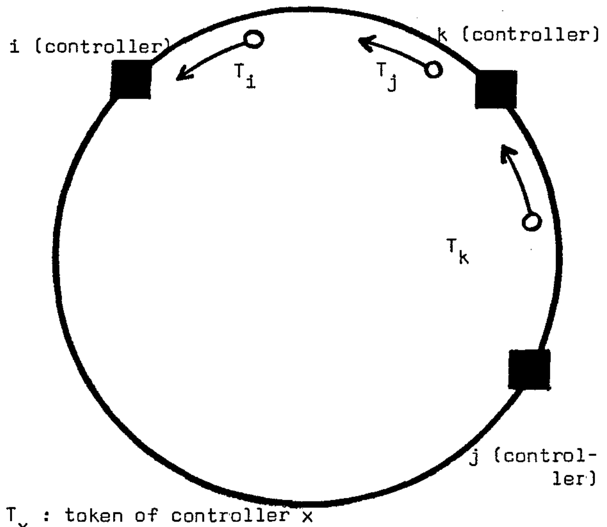
Uniqueness in the choice for several controllers is

condition (a) : the algorithm is unique for all controllers
condition (b) : value of S(i), $i \in I$ is the same for all controllers.

Unfortunately, there are cases for which condition (b) is not true, see fig. 2 ; T_j being generated after T_i crossed controller j and before T_k crossed the same controller, one is left with a situation where S(i) = {i,k} and S(k) = {i,j,k}.

One may be tempted to solve the problem by using one of these solutions :
Solution 1 : for each token crossing a controller, the local timer is reset to its initial value ; a new token is generated locally only when the timer awakes.
Solution 2 : solution 1 plus : each controller must remove from the ring any token circulating after the first token so that only that one will be left on the ring.

It should be noted that in this case, all controllers, even those not participating in the election (which means permanently), are required to record crossing of any token. It is not difficult to conclude that these solutions are not acceptable, because of H2 and condition (b).



T_x : token of controller x
T_i and T_k have completed a revolution
Fig. 2

As the set of controllers is strictly ordered (H1), a simple algorithm may be proposed :
Ω : if i = min S(i), then entity i immediately generates the new CT

In order to demonstrate that Ω satisfies constraint A, let us describe first the state-transition table of entity i on the ring (table 2).

Table 2

external events / state (i)	0	1	2	3	4
α	β	α	α	α	α
β	β	α	γ	β	α*
γ	β	α	γ	γ	

Comments :
0 : awaking of the timer
1 : reception of the control token
2 : reception of a candidate token, the identity of which is smaller than i
3 : reception of a candidate token, the identity of which is larger than i
4 : reception of the candidate token i (after one complete revolution)
α : idle, control token timer is set
β : candidate token timer is set and i is prepared to regenerate a new control token
γ : candidate token timer is set and i is not responsible for the control token regeneration
α* : generation of the new control token and immediate switching to state α

We will use the following notation :
I(CT,x) = instant of control token reception by entity x
I(t(x),y) = instant of reception of the candidate token x by entity y
I(x,o) = instant of generation of a candidate token by entity x
I(x,x) = occurence of event 4.

Let us imagine that two entities x and y generate "simultaneously" (during the same revolution on the ring) a control token, thus violating constraint A and we will show that this situation is impossible. Let us assume, for example, that identity (x) < identity (y).

Entity y will generate a new token if and only if state (y) at time I(y,y) is β ; this implies : $\overline{1}$ and $\overline{2}$ between I(y,o) and I(y,y) where $\overline{n}$ means non occurence of event n.

Identically, assuming that x will generate a new token implies (x < y) : $\overline{1}$ between I(x,o) and I(x,x).

It is easy to show that a subset of these conditions leads to a contradiction.

$\overline{2}$ between I(y,o) and I(y,y)   I(t(x),y) > I(y,y) for entity y ; $\overline{1}$ between I(x,o) and I(x,x)   I(CT,x) > I(x,x) with the CT received by x being the token generated by y. This constraint and the FIFO hypothesis imply that for entity y : I(t(x),y) < I(y,y).

It has thus been demonstrated that the CT cannot be generated by two different entities during one revolution on the ring.

Constraint B is obviously fulfilled by Ω and it is possible to compute bound values for the time T needed to regenerate a new CT. If x is the identity of the first controller to initiate the election phase after loss of the CT and if θ is the maximum value of the time required for a controller to process a token and to hand it down to its neighbour on the ring, then we have :

$$R \leq T \leq x(R - \theta) + \theta$$

T being counted from the instant of timeout for controller x.

(iv) Failures during the election phase

When considering the failure of a controller participating in the election phase, two problems must be tackled. Failure of the controller which is precisely the one being elected by the other controllers as responsible for generating the new CT is not catastrophic ; protection against infinite waiting is provided by timers ; the election phase will only be longer than for a failure-free situation.

The other problem is what to do with tokens generated by controllers which have failed before tokens have

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase®
Smarter legal research.