A Characterisation of Dynamic Distributed Systems

Félix García-Neiva, Rubén de Juan-Marín and Francesc D. Muñoz-Escoí

Institut Universitari Mixt Tecnològic d'Informàtica
Universitat Politècnica de València
46022 València - SPAIN

{fgarcia,rjuan,fmunyoz}@iti.upv.es

Technical Report ITI-SIDI-2012/01

A Characterisation of Dynamic Distributed Systems

Félix García-Neiva, Rubén de Juan-Marín and Francesc D. Muñoz-Escoí

Institut Universitari Mixt Tecnològic d'Informàtica
Universitat Politècnica de València
46022 València - SPAIN

Technical Report ITI-SIDI-2012/01

e-mail: {fgarcia,rjuan,fmunyoz}@iti.upv.es

Abstract

Highly-available and scalable distributed systems are inherently dynamic, but no clear definition of this dynamism has been widely accepted up to now. The aim of this paper is to survey recent works that have presented new results in the area of dynamic distributed systems, describing their contributions. Additionally, this survey shows that all their assumed dynamic system models can be adequately characterised using the definition and classification given by Baldoni, Bertier, Raynal and Tucci-Piergiovanni (i.e., the BBRTP definition). This classification is based on two main parameters: (a) the number of processes that compose the system, and (b) the diameter of the networking graph that interconnects those processes. Other important characteristics of dynamic systems can be derived from these two parameters.

1 Introduction

High availability has traditionally been one of the objectives in many distributed applications and systems. To this end, processes and data should be replicated, but replication, concurrency and failures should be transparent to the users [38]. One of the aims of many distributed applications is to provide a single-system image to the user; i.e., to achieve *distribution transparency*. Unfortunately system components may fail. This demands that they eventually recover and were re-integrated in the system. As a result, a first level of dynamism is introduced in any distributed algorithm when a recoverable system model is assumed.

The advent of peer-to-peer [5] collaborative applications, where nodes may join and leave the system at will, and the emergence of the cloud computing paradigm [41] that introduces the need of elasticity and scalability [34] in order to manage variable and potentially huge workloads, has augmented the inherent dynamism in those new kinds of distributed applications. So, a definition for dynamic distributed systems seems to be needed, although only a few proposals have appeared and none of them has been clearly accepted.

This paper shows that the definition and classification given in [7] characterises in a precise way all dynamic distributed system variants. Its two key parameters (degree of concurrency and networking graph diameter) condition the characteristics of each possible class of dynamic distributed applications. This is confirmed providing a short survey of a set of recent publications where dynamic distributed systems are described or assumed.

The rest of this paper is structured as follows. Section 2 recalls the definition of dynamic distributed systems given in [7]. Section 3 summarises the BBRTP classification. Section 4 provides a slight refinement to that classification, considering in which classes is assumable a synchronous system model, leading thus an outlook over the set of classes where the consensus problem may be solved. Later, Section 5 presents a brief survey on recent research papers that have assumed a dynamic system, analysing to which class or classes they belong. Finally, Section 6 concludes this paper.

2 Definition of Dynamic Distributed Systems

Quoting *Baldoni et al.* [7], a possible definition of a dynamic distributed system is the following:

Definition 1 (Dynamic System)**.** *"A dynamic system is a continually running system in which an arbitrarily large number of processes are part of the system during each interval of time and, at any time, any process can directly interact with only an arbitrary small part of the system."*

This definition is later formalised in [7] providing a characterisation of dynamic distributed systems that is also a classification of them (presented in Section 3.) Before entering in detail in such classification, several properties may already be extracted from the informal definition given above:

1. Since the number of participating processes could be large, applications developed for dynamic systems do not need that all those processes know each other. As a result, a complete system membership management is not mandatory in dynamic systems, nor possible in many of them.

2. Processes have only a partial view of the system. In the general case, a fully interconnecting graph cannot be assumed.

   Note that in most cases distributed algorithms have assumed that communication between any pair of processes is feasible. This was modelled assuming a fully-connected graph. Such logical topology makes sense when the sender process knows the identity and address of the receiving one. In that case, message routing and delivery reliability can be ensured by the network and transport layers of the protocol stack. On the other hand, in a large dynamic system it is not possible to know the identity of all other system processes. As a result, such logical fully-connected topology is not realistic nor directly usable in any algorithm for dynamic systems.

3 The BBRTP Classification

Since dynamism implies that processes may join and leave the system at will, the concepts of *system run*, *system graph* and *graph sequence* [7] need to be defined.

Definition 2 (System run)**.** *A system run is a total order on the join and leave events issued by processes that respects their real time occurrence order.*

Definition 3 (System graph)**.** *A distributed system can be represented by a graph $G = (P, E)$, where P is the set of processes that compose the system and E is a set of edges (p_i, p_j), representing bidirectional reliable channels connecting processes p_i and p_j.*

Note that system graphs do not need to be fully connected, and that the addition or removal of any node or edge generates a different graph. So, these graph updates define a sequence of graphs in a given system run.

Definition 4 (Graph sequence)**.** *Let $\{G_n\}_{run}$ denote the sequence of graphs through which the system passes in a given run. Each $G_n \in \{G_n\}_{run}$ is a connected graph whose diameter can be greater than one.*

Based on these concepts, the BBRTP classification [7] considers these two complementary dimensions:

- *Number of concurrent entities* (P). Assuming the *infinite arrival model* proposed in [32, 2], these variants can be distinguished[1]:
  - P^b. The number of processes that concurrently belong to the system is bounded by a constant b in all system runs.
  - P^n. The number of processes that concurrently belong to the system is bounded in each system run, but may be unbounded when the union of all system runs is considered.

[1] Although the original notation in [2] uses M^b, M^n and M^∞ to refer to these models, we use P^b, P^n and P^∞ in order to avoid confusion with the BBRTP classification, since the latter also uses an M in its model specification.

- P^{∞}. The number of processes that concurrently belong to the system in a single run may grow to infinity as the time passes.

- *Diameter of the interconnecting graph* (D). This parameter models the "*geographical*" dynamism of the system. To this end, $\{D_n\}_{run}$ denotes the set of diameters of the graphs in $\{G_n\}_{run}$. The alternatives to be considered regarding the graph diameter are:

  - D^b, *Bounded and known diameter*. The diameter is bounded by b and that bound is known by the algorithms designed for that model. Formally: $\forall D_n \in \{D_n\}_{run}, D_n \leq b$.
  - D^n, *Bounded and unknown diameter*. All the diameters $\{D_n\}_{run}$ are finite en each run, but the union of all D_n in $\{D_n\}_{run}$ may be unbounded. This implies that in this model an algorithm has no information on the diameter.
  - D^{∞}, *Unbounded diameter*. In this model the diameter may grow indefinitely in a run.

Number of processes	Network diameter		
	D^b	D^n	D^{∞}
P^b	$M^{b,b}$	–	–
P^n	$M^{n,b}$	$M^{n,n}$	–
P^{∞}	$M^{\infty,b}$	$M^{\infty,n}$	$M^{\infty,\infty}$

Figure 1: Dynamic models considering the P and D parameters.

As a result, considering both parameters, a composed $M^{P,D}$ set of models is defined. Both parameters can assume the values b, n and ∞ to indicate their three variants, as described above. From the nine possible models, $M^{b,n}$, $M^{b,\infty}$ and $M^{n,\infty}$ models are not possible since their number of active concurrent processes bounds in some way such network diameters. In other words, their diameter is in fact bounded by $D = P - 1$. Therefore, as a result, we have six different models that correspond to the following combinations: $M^{b,b}$, $M^{n,b}$, $M^{\infty,b}$, $M^{n,n}$, $M^{\infty,n}$ and $M^{\infty,\infty}$. The resulting models are depicted in Fig. 1.

4 Classification Refinement

Authors of [7] suggest that other complementary parameters could be considered in order to refine the given classification. Indeed, they suggest *synchrony* as a possible refining characteristic; i.e., at a glance, each resulting dynamic model might be subdivided considering different degrees of synchrony.

4.1 Achievable Synchronicity

Let us now consider synchrony in order to apply such refinement. Regarding that characteristic, several degrees of synchrony (or asynchrony) may be distinguished. To this end, two axes of asynchrony are identified in a distributed system:

1. *Processor asynchrony* allows a processor to remain in the same step in its execution for arbitrary long finite amounts of time while other processors continue to run.

2. *Communication asynchrony* does not allow to bound message delivery time.

Communication asynchrony seems to be the main axis. It is accepted that processor synchrony can be simulated with a reasonable effort in a system that uses *logical buffering* [42]; i.e., an algorithm that assumes synchronous processes can be executed using asynchronous processes if the algorithm steps are appropriately numbered in each processor and messages are buffered until their intended receiver has reached the appropriate algorithm step. Note that communication may be asynchronous to this end. Although there are also general synchronisers (i.e., algorithms able to simulate both synchronous processors and

synchronous communications) they cannot be easily implemented in a real system (e.g., they require unbounded space). As a result, let us revise –in the following sections– all these dynamic models regarding whether synchronous communication may be achieved in them, since this is a key property in order to decide whether the studied system models could be synchronous or not. Besides that, note also that all these systems could trivially be assumed as asynchronous, and that communication is also considered *partially synchronous* when there is a bound on message transmission time but such bound is unknown [15].

To begin with, let us assume the most favourable scenario: each channel is synchronous, being δ its known bound on message transmission time. However, in a dynamic system the full set of processors is unknown. This implies that a given sender is unable to directly reach every other process in the system. If any algorithm step requires that a given process (for instance, a coordinator) sends a message to every other process, such message propagation would require *epidemic communication* [24].

Most classical distributed algorithms had been devised for static systems; i.e., systems where the set of executing processes is known. Moreover, almost all algorithms are designed for managing communications at the application layer (or, at least, at the transport one) considering the regular network protocol stack. This implies that routing is not a problem and that the assumed channels are able to communicate every pair of system processes. Synchronous communication in that context implies that every pair of system processes can exchange messages and that such message transmission time is bounded. In a dynamic system, on the other hand, a process may only know about a small set of neighbour processes. As a result, some kind of application-layer routing is demanded in order to intercommunicate all system processes. This demands a slight variation in the definition of synchronous communication.

Definition 5 (Synchronous communication)**.** *Communication in a dynamic distributed system S is considered synchronous when the message transmission time between every pair of processes (p_1, p_2) in S is bounded and such bound is known.*

Thus, process-to-process channel bounds are not enough in the general case since message forwarding is demanded in many algorithms. So, let us analyse whether processes may assume bounds in such message forwarding or not.

4.1.1 $M^{*,b}$ Models

These dynamic system models assume that the network diameter is bounded and known. In that case, if each channel is synchronous (as said above), the resulting system will be also synchronous. Thus, message transmission time will be bounded in a $M^{*,b}$ system to $b\delta$ time.

Theorem 1 ($M^{*,b}$ synchronous communication)**.** *The message transmission time between every pair of processes p_1 and p_2 in a distributed system S that follows any of the $M^{*,b}$ models (i.e., $M^{b,b}$, $M^{n,b}$ or $M^{\infty,b}$) is bounded by $b\delta$.*

Proof. Immediate, given the bounds on the network diameter (b) and the channel transmission time (δ).

Note that if the network diameter is b, every epidemic broadcast will be able to reach all current system processes in b steps and this requires a bounded time. □

4.2 $M^{*,n}$ Models

These dynamic system models assume that the network diameter is bounded, but the bound is unknown. At a glance, the algorithms may not assume any message transmission bound, although they can be certain that the messages will reach their destinations since reliable channels have been assumed.

Lemma 1 ($M^{*,n}$ non-synchronous communication)**.** *The message transmission time between every pair of processes p_1 and p_2 in a distributed system S that follows any of the $M^{*,n}$ models (i.e., $M^{n,n}$ or $M^{\infty,n}$) cannot have a known bound.*

Proof. In this kind of system there are not any known bounds in the number of concurrent processes nor in the network diameter. This implies that each process only maintains a limited neighbourhood and such known processes vary throughout time.

Communication between each pair of processes could be reached using a given route of intermediate processes. Since the network diameter has no known bound, the length of such routing paths cannot be bounded a priori. Once a route has been found, a given bound exists. Unfortunately, routes are not stable since their composing processes could leave and re-join the system at will. As a result, in an $M^{*,n}$ system, interconnecting routes are dynamic and their bounds (both in number of "hops" at the transport or application layer and in the overall transmission time) cannot be set.

Given this intuitive justification, let us prove that communication cannot have a known bound using *reductio ad absurdum*. To this end, the assumptions are:

1. System S belongs to the $M^{n,n}$ or $M^{\infty,n}$ models; i.e., its network diameter is bounded but unknown.
2. Communication between every pair of processes (p_1, p_2) in S is bounded, being γ (with $\gamma > \delta$) the known bound on message transmission time.

If communication time is bounded by γ between every pair of system processes, a given process may start at time t an epidemic broadcast algorithm able to reach every other system process in γ time. This epidemic algorithm collects the paths needed to intercommunicate its starting process with any other system process and instructs each visited process to start this same algorithm on its own. To this end, the message progressively holds the identifiers and addresses of all processes that it has visited. This information is also useful to avoid cycles and to guarantee that eventually the initial broadcast reaches all system nodes.

Finally, each of the receivers of those epidemic broadcasts will be able to collect all system paths needed to intercommunicate itself with every other process (recall that channels were assumed bidirectional in Def. 3). This algorithm would require 2γ time to complete; i.e., it terminates at time $t+2\gamma$. Note that termination is guaranteed by hypothesis 2. As a result, each system process is able to know the network diameter. This shows that the system network diameter is actually bounded and known, contradicting the initial hypothesis 1. This concludes the proof. □

As a result of this, it is clear that these models may not assume synchronous communication. Despite this, it is still possible to assume a *partially synchronous* [15] communication; i.e., a bound actually exists but its value is unknown to the processes.

Theorem 2 ($M^{*,n}$ partial synchronous communication)**.** *The message transmission time between every pair of processes p_1 and p_2 in a distributed system S that follows any of the $M^{*,n}$ models (i.e., $M^{n,n}$ or $M^{\infty,n}$) may be bounded but its bound is unknown.*

Proof. The definition of $M^{*,n}$ systems assumes that they have a bounded (although unknown) network diameter. We had already assumed that each interconnecting channel is synchronous, with δ as its upper bound on message propagation time. Let us now assume that the unknown upper bound on the network diameter is β. As a result, there is an actual upper bound on the message propagation time between any pair of processes in S: $\beta\delta$. Unfortunately, such upper bound cannot be known as it has been already proven in Lemma 1. □

4.2.1 $M^{\infty,\infty}$ Model

The $M^{\infty,\infty}$ system model assumes that both the network diameter and the degree of concurrency are infinite when long runs are considered. Additionally, dynamic systems assume that processes enter and leave the system at will. As a result, the following theorem can be stated:

Theorem 3 ($M^{\infty,\infty}$ asynchronous communication)**.** *The message transmission time between every pair of processes p_1 and p_2 in a distributed system S that follows the $M^{\infty,\infty}$ model cannot be bounded.*

Proof. This system model, for concrete run intervals, is similar to model $M^{\infty,n}$ (additionally, its network diameter may grow throughout time in a given run). This implies that the proof given in Lemma 1 is also applicable here and, given the $M^{\infty,\infty}$ definition, no unknown bound may exist. As a result, communication cannot be bounded in this kind of systems. □

4.2.2 Consequences

The results given in the previous sections assume the best possible configurations regarding communication synchrony. Such configurations are difficult to implement in an $M^{*,n}$ model. Indeed, none of the surveyed papers belonging to that family of dynamic systems assumes that process-to-process channels have a bounded message propagation time. As a result, although the $M^{*,n}$ model could provide a partially synchronous communication model, algorithms that assume such model only rely on asynchronous communication.

As a result, the classification refinement given in the previous section suggests that some classical problems that are not solvable in "*static*" asynchronous reliable distributed systems where processes may fail, will not be solvable in $M^{*,n}$ and $M^{\infty,\infty}$ dynamic systems. This includes *consensus* [17], for instance. On the other hand, problems that require a synchronous system demand that the resulting dynamic model becomes one of the $M^{*,b}$ classes.

Regarding these synchronous solutions, several basic approaches may be found:

- The first one sets a hierarchical organisation of the system processes, defining multiple subsystems and interconnecting them using inter-system channels. Each subsystem maintains at least a distinguished processor that bridges its contained set of processes with those of other subsystems. As a result, each subsystem is fully-connected and its internal network diameter is 1, whilst communication with other subsystems is managed by the distinguished processor and it also has a bounded and known diameter that depends on the number of layers being defined in such hierarchy. This hierarchical architecture was proposed in the context of interconnectable memory consistency models [9, 16] or interconnectable message broadcast protocols [26, 4], still in a static context, but it also provides a solid basis for dynamic systems. The regular architecture of the current cloud system providers [41] follows a similar pattern (separate interconnected large data-centres) that boosts both scalability and dynamism in the management of the hosted applications.

- The second one [33] defines a *stable subset* of processes able to ensure algorithm progress. This stable set should comply with some constraints: a minimal number of processes (α) that remain in the system long enough (*stability*) –note that α simulates the static-system requirement of maintaining at least $n - f$ correct processes[2]–, and a strong cooperation among those α processes (and this suggests that they assume a fully-interconnecting network among them). Additionally, two complementary communication primitives are provided: a *query-response* that broadcasts a query and waits for α answers, and a *broadcast* operation that is able to propagate information to all system processes. At a glance, this implies that the stable subset conforms to the $M^{b,b}$ system model, whilst the overall system may assume even the $M^{\infty,\infty}$ one. Algorithms are executed in an $M^{b,b}$ subpart, propagating their advancements to the remaining processes that may join the distinguished subset if they are sufficiently stable. To this end, they only need to be one of the first α repliers to the *query-response* primitives being executed in the corresponding algorithm.

But, fortunately, not all the problems demand a synchronous system in order to be solved. In those cases, each given problem should be carefully studied analysing in which kinds of dynamic system some solution can be found. A sample of this kind is already presented in [7] where the *one-time query* problem [11] is initially solved only in an $M^{*,b}$ system with the *WildFire* algorithm [11], but its specification is later slightly relaxed in order to build the *DepthSearch* algorithm [7] that solves it also in any $M^{*,n}$ model but not in an $M^{\infty,\infty}$ one.

4.3 Refined Model

Another question that arises from the properties outlined above is where to place the frontier between *static* and *dynamic* systems. A traditional static reliable system considers an a priori known set of participating processes that has a logical complete interconnecting network. This corresponds to an $M^{b,1}$ model in our classification, that is a subset of the dynamic $M^{b,b}$ class defined above. Does this mean that all $M^{b,1}$

[2] Being n the initial number of system processes and f the current number of failed processes.

systems are static and that all $M^{b,b}$ systems with network diameter greater than one are dynamic? What does it happen with $M^{n,1}$ and $M^{\infty,1}$ systems? Are they static or dynamic? Let us start with the latter. In $M^{n,1}$ and $M^{\infty,1}$ systems there is no fixed (and known[3]) set of system processes. As a result, these kinds of systems are not static. On the other hand an $M^{b,b}$ system has a known maximal number of processes, but their identity or addresses are unknown a priory. Otherwise, the algorithm would assume a direct channel between every pair of processes, since most algorithms work on top of a tranport layer in the communication stack. This would imply that the resulting logical network diameter would have had a value of 1. This means that in $M^{b,b}$ systems, algorithms do not require that each process knows which is the current set of processes that belong the system; i.e., those algorithms only demand the existence of a given (partial) set of neighbour processes. The key characteristic of these algorithms is decentralisation. With it, management of highly frequent arrivals and departures of processes (i.e., *dynamism*) is not difficult.

From a general point of view, and according to Def. 1, $M^{b,b>1}$ systems are dynamic and only $M^{b,1}$ systems are *static*, although this may allow some divergences considering specific problems. For instance, there is a wide agreement on considering "dynamic" those systems that are able to reconfigure their state when there are any process arrivals or departures, even in the case that there is no membership service in such systems. Examples of $M^{b,1}$ systems of this kind are those managed with group communication services providing view-oriented communication [36], and those supporting atomic consistency in reconfigurable systems [3, 21, 31]. So, there is no clearly-defined border between static and dynamic systems and this demands a refinement of the dynamic characterisation, generating the following concepts:

- *Static distributed systems*: A system that belongs to class $M^{b,1}$ but does not tolerate the arrival of new processes nor the active departure of the existing ones.
- *Reconfiguration-based dynamic systems*: A system that belongs to class $M^{b,1}$ and that is able to manage the arrival or departure of its member processes. Processes in this kind of dynamic systems are able to know and exchange messages with any other process belonging to the same system configuration or system view. Systems of this kind have been described in [3, 21, 30, 31, 36].
- *Strong dynamic systems*: Those that comply with Definition 1 and have a diameter bigger than 1, $D > 1$. Algorithms that assume a strong dynamic system do not require that all their executing processes know each other.

Figure 2 shows how these concepts are related to the original classification given in [7].

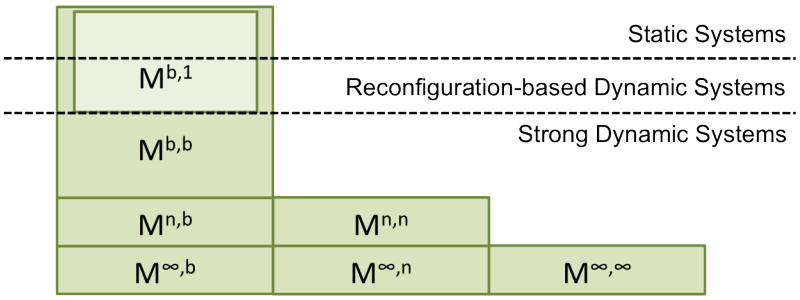

Figure 2: Relationship among the BBRTP classes and the static and dynamic class refinement.

5 A Brief Survey of Related Work

In the last years there have been multiple papers that have presented new solutions to different problems when they are adapted to dynamic environments. Let us revisit such results, associating them to the class families presented in Section 4 and following a chronological order into each family. Recall that Section 4

[3]Note that when the actual number of system processes is unknown the required algorithms should be able to adapt their behaviour to the arrival of an unforeseen number of new processes in a given run.

has analysed which is the maximum level of communication synchrony that can be achieved in each of the dynamic system families, but the actual degree of synchrony may vary from being completely asynchronous to assuming that maximum level. As a result, multiple variants can be found in the sequel.

5.1 $M^{*,b}$ Systems

The older papers found in this family (that encompasses all classes with limited and known network diameter) do not explicitly use the adjective "*dynamic*" in order to describe its assumed system model. One of the characteristics of a dynamic system is that its algorithms should deal with the frequent departure and arrival of processes. This happens with both a large unknown number of processes and with a bounded known one. Let us start with the latter. An example of this kind is the Paxos algorithm [30], designed in 1990. It proposes a consensus algorithm for a system with asynchronous processes and asynchronous communication able to guarantee safety and to reach liveness in most of its executions (but not in all of them), assuming a recoverable failure model. This still respects the known impossibility [17] of reaching consensus in an asynchronous static system where processes may fail, but all the conditions that must appear in order to prevent this algorithm from reaching progress seldom arise at once. Paxos provides a sample of how dynamic solutions may extend the traditional static ones. The assumed system belongs to the $M^{b,1}$ class[4], and shows that the fact of dealing with failures/departures and recoveries/arrivals already introduces some dynamism in a distributed system. Other variants of the Paxos algorithm, as Disk Paxos [19], have enhanced its functionality supporting a greater percentage of process failures/departures, although they still maintain the constraint of not ensuring liveness in all possible executions.

As it has been said above, a research line that introduced another variant of *dynamism* in the system is the one that assumes *unbounded concurrency* [32, 20]. When the number of processes that should be managed in a given algorithm cannot be bounded (i.e. it is infinite), the resulting algorithm easily tolerates departures and arrivals of processes, since it cannot forecast how many processes participate in its execution. This is one of the key characteristics of a dynamic system. On the other hand, the papers that initiated this research line assumed a shared memory model. As a result, they could be translated into a logical network with diameter 1.

Dynamic environments was also the target of [10]. In this case, the dynamic environment referred to systems where nodes can connect and disconnect frequently, network topologies can vary over time and energy consumption is in many cases critical (MANETs). Thus, the author proposed several group membership protocols for this kind of scenarios. The first protocol, HMS, was devoted for systems with a well-known topology and moderate size but partitionable, therefore she considered $M^{b,b}$ systems when proposing this protocol. Moreover, the HMS constituted the basic building block for the others. The second one, HaloMS was intended for large scale systems based on the scenario of client-server architectures, where important interactions are server-server activity and client-server accesses. The assumption of this scenario allowed the author to simplify the large scale system context to: a) the servers, a stable set of nodes where each server knows each other, compounding the core: an $M^{b,1}$ system and b) the clients that only have to know one server which allows them to access other servers, thus composing an $M^{b,2}$ system, the halo. Then, having in mind ad-hoc networks, and more especifically, MANETs, she proposed a membership protocol MODUS (membership on demand) that allowed the activation and deactivation of membership monitoring on demand in order to reduce the energy consumption, a critical factor in MANETs. This protocol was intended for systems where topology can vary, but as the membership was considered in the traditional way, all nodes know all available nodes, so we are again in front of a $M^{b,b}$ system. The last proposal was to use MODUS with Membership Estimation for dynamic networks. In this case, they have a service providing an estimation of the current membership without strict guarantees, and a memerbship service which provides traditional membership semantics. The former service provides an initial view, and basic information for creating new views when required. Thus, these protocols as they are conceived can be used in $M^{n,n}$ systems.

A survey about the management of infinitely many processes in a distributed system was given in [2]. It provides some examples of algorithmic approaches that adapt solutions initially designed for the $M^{n,1}$ model to the $M^{\infty,1}$ one. These approaches are applied to the *group naming*, *atomic snapshot* [1] and

[4]It is a first example of reconfiguration-based dynamic system, as discussed in Section 4.2.2.

other problems. For instance, the mechanisms suggested for solving the *atomic snapshot* problem in an $M^{\infty,1}$ model are also used to solve a relaxed definition of the *group membership* problem. Recall that valid solutions for the group membership problem [13] require consensus and that the latter demands at least partial synchrony in order to be solved when processes may fail (or leave the system). This provides a sample confirming that some $M^{*,b}$ systems require non-asynchronous communication.

Things start to change a bit later, and some papers already refer to the *infinite arrival model* as a synonym for the *dynamic distributed system* concept. One of these first "explicitly dynamic" papers is [18] that assumes an infinite arrival model with finite concurrency for both clients and servers; i.e. each system run may have an infinite number of processes but the amount of concurrent processes in a given time interval can be bounded. This means that its solutions belong to the $M^{\infty,b}$ class assumed in this paper. The contributions of [18] consist in proposing two basic abstractions in order to support atomically (i.e., linearisable [25]) replicated objects in a dynamic system. Those two abstractions are: (a) the usage of operation-related dynamic quorums, and (b) a *persistent reliable broadcast* able to broadcast a message to a sufficiently large number of processes that belong to the system. The latter implies that processes should be reachable in a given interval (i.e. the system has a bounded network diameter) and that processes that join the system whilst a message is being broadcast will be able to receive it. In order to ensure liveness these broadcasts do not need to reach all the system processes that existed at the time the broadcast was initiated but only a stable (sub)set of them. Additionally, some degree of synchrony is also assumed in order to implement this broadcast primitive.

The usage of a stable set of processes and a persistent reliable broadcast primitive is retaken in [33] where these abstractions are proposed as the general basis for developing algorithms for dynamic systems, as we have outlined in Section 4.2.2. Thus, although the overall system complies with the $M^{\infty,n}$ class, progress is ensured by a stable subset able to assume an $M^{n,b}$ or even an $M^{b,b}$ class, simplifying algorithm design.

Despite the availability of the persistent reliable broadcast and a stable subset, an open problem still exists: the actual criteria needed to select such stable subset. This problem is analysed in [22] in the field of P2P applications where *churn* is a difficulty for guaranteeing a correct functionality of those applications. In [22] two parameters are considered for analysing how the stable set is selected: (a) whether the strategy uses information about nodes in order to predict which nodes will be stable (*predictive*) or not (*agnostic*), and (b) whether the strategy replaces a failed node with a new node (*replacement*) or not (*fixed*). Combining these two parameters, four different families of strategies arise and several strategies in each family are described. They are: (a) predictive-fixed (fixed decent, fixed most available and fixed longest lived), (b) agnostic-fixed (fixed random), (c) predictive-replacement (max expectation, longest uptime, optimal), and (d) agnostic-replacement (random replacement, passive preference list, active preference list). The advantages and drawbacks of each strategy are analysed and a set of P2P applications are studied, identifying which is the best strategy in each case. As expected, the best choice generally depends on the application requirements.

Dynamism is also analysed in [36] regarding its implications on group communication systems. To this end, the system model is restricted to $M^{b,b}$ since membership services should be implementable in this context and this demands that the set of participating processes is known by the algorithms or applications that use a group communication system. The paper revises all concepts related to *virtual synchrony* [12] in a context where processes may leave and rejoin the system at will. Although some of the traditional definitions in this area are refined, most of them are inherently correct. This shows that distributed applications that assume a *virtual synchrony* model (i.e., most of the fault-tolerant ones) are implicitly dynamic.

Gramoli [23] studies how to face dynamism and scalability when *atomic consistency* [29] should be guaranteed. His analysis is based on reconfigurable read and write quorums, and is inspired in previous solutions to the same problem [31, 21] in dynamic systems. All the operations should wait for their replies from the intended quorum members. Besides the algorithms proposed in that Ph.D. thesis, one of its contributions is the classification of the quorum systems supporting atomic consistency. Six different classes are distinguished in such taxonomy: (a) *static failure-prone*, (b) *static redundant*, (c) *dynamic replaceable*, (d) *dynamic repairable*, (e) *probabilistic repairable* and (f) *probabilistic structureless*. The two first classes do not tolerate that involved processes leave or join the system. Dynamic replaceable (or reconfigurable) systems are able to tolerate failures or disconnections but are still unable to scale. On the other hand, the dynamic repairable systems are the first class with moderate scalability. These two last classes assume an $M^{n,b}$ or $M^{\infty,b}$ model. In order to mix dynamism and scalability, probabilistic solutions are needed, as

those proposed in [23], tolerating also an $M^{*,n}$ model.

In [40] a first algorithm implementing the Ω failure detector in an infinite arrival model is presented. To this end, the failure detectors manage lists of active processes, instead of suspected ones since the aim of an Ω failure detector is to solve the leader election problem. The algorithm needs a multicast primitive and this forces to assume a fully connected network with IP-multicast capability. Process identifiers depend on the IP address of their respective host machines. A crash-recovery failure model is assumed. As a result, process identifiers only bound the concurrent number of alive processes but not the overall amount of them in a given algorithm run. These assumptions imply an $M^{n,b}$ model according to our refined classification.

A bit later, Baldoni et al. [8] proposed an implementation of *regular registers* [29] in the same $M^{n,b}$ model, explicitly qualifying such configuration as dynamic. Regular registers have the same computing power than atomic registers (i.e., an atomic register can be implemented with an algorithm that uses regular registers), but allow easier implementations. Additionally, they can be used as a basis to solve consensus in asynchronous systems prone to failures when the latter are complemented by some leader election service, as that outlined in [40]. The first regular register implementation discussed in [8] assumes a synchronous system (confirming that $M^{n,b}$ systems may be synchronous, as proven in Section 4.1.1), and provides a fast (i.e., non-blocking and local) read operation and a write operation based on multicasting the new value and its associated version number. It is complemented with a more elaborated algorithm able to implement the regular register in an eventually synchronous system and a proof on the impossibility of implementing a regular register in a dynamic asynchronous system.

Resource directory services in structured P2P systems [5] provide another sample of the selection of a given subset of processes in order to provide a basic service for a dynamic distributed application. Distributed hash tables (DHT) or similar structured subsystems [35, 37] have been designed in order to implement a directory/location service of this kind. They manage a distributed directory service for a potentially huge domain of resources, easily tolerating the join and leave of serving processes. Efficient solutions to this problem had already been published in 2001 [35, 37]; i.e., before most of the papers discussed in this section were written. These solutions rely on a known function that translates resource names/identifiers into some kind of coordinates in a given space. Each serving process has also an identifier that marks it as the server for a part of the resource identifiers space, and a structure is built arranging those space fragments. As a result, each serving process has a subset of known serving processes that interact with it in order to route the client requests to the appropriate serving process that stores the requested resource addresses. Dynamism is tolerated using some degree of replication and a fast redistribution of the directory information being held by each leaving server. The algorithms being used for setting the directory structure bound the logical diameter of the interconnecting network. Such bound depends on the dimensions of the space being assumed for the resource identifiers (e.g., number of bits in the resource identifiers of Chord [37]). As a result, these systems may be included in an $M^{n,b}$ model.

A related problem is analysed in [27, 28] where the set of processes that belong to a large-scale application are able to self-organise themselves, defining a logical topology where they receive some virtual coordinates. This is interesting for self-organising sensor networks. The proposed algorithm is based on local information. It only requires that each participating node has a different identifier. This implies an $M^{b,b}$ model, since the length of the process identifiers limits the maximal degree of concurrency and network diameter. As a result, the data types required for storing those values are known in advance. The system is organised in two different layers. The first one (VINCOS) is able to assign virtual coordinates to system nodes. The other (NetGeoS) achieves a geometric structure on top of VINCOS. VINCOS is based on finding the "external belt" of the system (its borderline), and dividing such belt in a given number of portions with (approximately) the same amount of nodes. Each of such portions is taken as the origin for a different coordinate axis. The coordinates of a node are calculated as the length of the minimal path from such node to each one of the belt portions, measured in "number of hops".

A recent contribution in the domain of reconfiguration-based dynamic distributed systems is the implementability of atomic consistency without requiring consensus [3]. Aguilera et al. [3] prove that their *DynaStore* algorithm implements atomic registers in an asynchronous dynamic environment, whilst consensus cannot be solved in that environment. This shows that the implementation of atomic registers is a lighter problem than consensus, both in static (as already stated and proven in [6]) and dynamic contexts.

5.2 $M^{*,n}$ Systems

The second family of system classes assumes an interconnecting network with bounded but unknown network diameter. It is composed of only two classes: $M^{n,n}$ and $M^{\infty,n}$.

As proven in Theorem 2, this family of dynamic systems may still assume partially synchronous communication, but the dynamism of the set of processes and the potentially large amount of them leads to ignore such possibility, assuming an asynchronous communication. This has led to the adoption of a stable subset of processes (with a known interconnecting network diameter, usually with value 1 at the application layer) in order to accelerate any decision steps needed by the algorithms as recommended in [33], thus ensuring progress.

Another application in this family is the querying algorithms for unstructured P2P file-sharing systems [5], as Gnutella. These applications commonly use a flooding query (similar to an epidemic broadcast) that is answered by the nodes that hold any resource that matches such query. None of the "peers" needs to know the full set of nodes that execute the file-sharing application and each of them easily tolerates the arrival of new members or the departure of the existing ones in its known set of "neighbours". The actual network diameter is bounded (since the number of processes that use the application is finite although very large), but the bound is unknown. So, this justifies its inclusion in the $M^{n,n}$ class. Similarly to what has been described above, regarding stable subsets, flooding queries in unstructured P2P systems where also limited and "translated" into other variants that could match the $M^{*,b}$ family properties. For instance, since these queries would overload the network, they were "pruned" setting a maximum number of hops in their system traversal. Another variant consists in setting a partial structure using super-peers that define a two-layer hierarchy. Super-peers centralise the directory information of their immediate neighbourhood, and this reduces a bit the cost of the flooding queries. Other complementary approaches (usage of super-peers, dynamic topology adaptation for queries, active flow control in order to avoid node overloading, one-hop replication of directory contents in all nodes) were suggested [14] in order to increase the scalability of these querying techniques, preserving their best characteristics (usage of keywords, success rate on systems with high churn, etc.), but still maintaining a pruned scanning of the interconnecting graph. The best contribution of [14] consists in proving experimentally that none of those complementary approaches is able to individually improve the query performance (regarding the rate of query hits per query message), but when they are properly combined such performance is greatly boosted. All those complementary approaches demand only some knowledge about the state of the immediate neighbour nodes, but not about the overall system topology nor network diameter nor system population. As a result, they can be used in every dynamic distributed system family.

Regarding P2P systems, Tucci Piergiovanni [39] analyses existing system connectivity algorithms. These algorithms are needed to drive the diffusion of information in a P2P system (i.e., to manage an epidemic reliable broadcast). A model with infinite arrival of processes and unbound concurrency (i.e. a P^{∞} model), with asynchronous communication is assumed. She proposes a new algorithm (*DET*) that is able to ensure both scalability and system connectivity tolerating some degree of churn, improving the connectivity maintenance of previous algorithms. Since knowledge of the system diameter is not required by *DET*, it belongs to the $M^{\infty,n}$ model. Since communication is assumed asynchronous, connectivity cannot be always fully ensured. In case of churn, a connectivity restoring algorithm is used and in the worst case at least a star topology is maintained.

5.3 $M^{*,\infty}$ Systems

This third family (with infinite network diameter) consists of only one class: $M^{\infty,\infty}$. It has only theoretical sense. No implementable applications belong to this class.

6 Conclusions

Quoting [7] a dynamic system is "*a continually running system in which an arbitrarily large number of processes are part of it during each interval of time and, at any time, any process can directly interact with only an arbitrary small part of the system*". Six different classes of dynamic systems (specified as $M^{P,D}$)

are identified in [7], crossing two complementary axes: the degree of concurrency (P) and the network diameter (D). The resulting classes are: $M^{b,b}$, $M^{n,b}$, $M^{\infty,b}$, $M^{n,n}$, $M^{\infty,n}$ and $M^{\infty,\infty}$. We analyse the highest degree of system synchrony achievable in each of the identified classes in that taxonomy.

Not all published papers follow the definitions given in [7] regarding dynamic systems. As a result, there are some variations on the dynamic systems concept and this suggests a refinement of the dynamism characterisation. In this context, traditional *static systems* correspond to a subset of the $M^{b,b}$ class: the $M^{b,1}$ model. Such subset manages a bounded number of processes assuming a fully connected network. Sharing this same model, other systems qualified as dynamic do exist. They have been renamed as *reconfiguration-based dynamic systems* and despite allowing the arrival and departure of processes, the set of processes that execute a given algorithm are able to know each other in this dynamic variant. As such, these algorithms and systems do not follow strictly the definition given above. Finally, those systems that strictly comply with such definition are known as *strong dynamic systems* in our classification refinement.

There are multiple sources of dynamism in modern distributed applications. One of them is the management of an unbounded concurrency due to an infinite arrival of processes. In this regard, Aguilera [2] provides a basic classification of those systems (inherited as the concurrency axis in [7]) and a thorough survey on how to adapt P^n algorithms in order to use them in a P^∞ context.

A brief summary of recent work in the dynamic distributed systems area has been presented, briefly describing the contributions of each paper in its corresponding class, illustrating which problems have been solved or are still open in each of the identified classes.

Acknowledgements

This work has been supported by EU FEDER and Spanish MICINN under research grants TIN2009-14460-C03 and TIN2010-17193.

References

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *J. ACM*, 40(4):873–890, 1993.

[2] Marcos Kawazoe Aguilera. A pleasant stroll through the land of infinitely many creatures. *SIGACT News*, 35(2):36–59, 2004.

[3] Marcos Kawazoe Aguilera, Idit Keidar, Dahlia Malkhi, and Alexander Shraer. Dynamic atomic storage without consensus. *J. ACM*, 58(2):7, 2011.

[4] A. Álvarez, Sergio Arévalo, Vicent Cholvi, Antonio Fernández, and Ernesto Jiménez. On the interconnection of message passing systems. *Inform. Process. Lett.*, 105(6):249–254, 2008.

[5] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[6] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *J. ACM*, 42(1):124–142, 1995.

[7] Roberto Baldoni, Marin Bertier, Michel Raynal, and Sara Tucci-Piergiovanni. Looking for a definition of dynamic distributed systems. In *9th Intnl. Conf. Paral. Comput. Tech. (PaCT)*, volume 4671 of *Lect. Notes Comput. Sc.*, pages 1–14, Pereslavl-Zalessky, Russia, September 2007. Springer.

[8] Roberto Baldoni, Silvia Bonomi, Anne-Marie Kermarrec, and Michel Raynal. Implementing a register in a dynamic distributed system. In *29th Intnl. Conf. Distrib. Comput. Sys. (ICDCS)*, pages 639–647, Montreal, Québec, Canada, June 2009. IEEE-CS Press.

[9] Roberto Baldoni, Roy Friedman, and Robbert van Renesse. The hierarchical daisy architecture for causal delivery. In *17th Intnl. Conf. on Distrib. Comput. Syst. (ICDCS)*, pages 570–577, Baltimore, Maryland, USA, May 1997.

[10] Mari Carmen Bañuls. *Group Membership Protocols for Dynamic Environments*. PhD thesis, Departamento de Sistemas Informáticos y Computación, Valencia, Spain, March 2006.

[11] Mayank Bawa, Aristides Gionis, Hector Garcia-Molina, and Rajeev Motwani. The price of validity in dynamic networks. *J. Comput. Syst. Sci.*, 73(3):245–264, 2007.

[12] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM T. Comput. Syst.*, 5(1):47–76, 1987.

[13] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. On the impossibility of group membership. In *15th Symp. Princ. Distrib. Comput. (PODC)*, pages 322–330, Philadelphia, Pennsylvania, USA, May 1996.

[14] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making Gnutella-like P2P systems scalable. In *Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 407–418, Karlsruhe, Germany, 2003. ACM Press.

[15] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[16] Antonio Fernández, Ernesto Jiménez, and Vicent Cholvi. On the interconnection of causal memory systems. In *19th Annual ACM Symp. on Princ. of Distrib. Comp. (PODC)*, pages 163–170, Portland, Oregon, USA, July 2000.

[17] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[18] Roy Friedman, Michel Raynal, and Corentin Travers. Two abstractions for implementing atomic objects in dynamic systems. In *9th Intnl. Conf. Princ. Distrib. Sys. (OPODIS)*, volume 3974 of *Lect. Notes Comput. Sc.*, pages 73–87, Pisa, Italy, December 2005. Springer.

[19] Eli Gafni and Leslie Lamport. Disk Paxos. In *14th Intnl. Conf. Distrib. Comput. (DISC)*, volume 1914 of *Lect. Notes Comput. Sc.*, pages 330–344, Toledo, Spain, October 2000. Springer.

[20] Eli Gafni, Michael Merritt, and Gadi Taubenfeld. The concurrency hierarchy, and algorithms for unbounded concurrency. In *20th Annual Symp. on Princ. of Distrib. Comp. (PODC)*, pages 161–169, Newport, Rhode Island, USA, August 2001. ACM Press.

[21] Seth Gilbert, Nancy A. Lynch, and Alexander A. Shvartsman. RAMBO II: Rapidly reconfigurable atomic memory for dynamic networks. In *Intnl. Conf. on Depend. Sys. and Netw. (DSN)*, pages 259–268, San Francisco, CA, USA, June 2003. IEEE-CS Press.

[22] P. Brighten Godfrey, Scott Shenker, and Ion Stoica. Minimizing churn in distributed systems. In *Conf. Appl. Techn. Arch. Protoc. Comput. Comm. (SIGCOMM)*, pages 147–158, Pisa, Italy, September 2006. ACM Press.

[23] Vicent Gramoli. *Distributed Shared Memory for Large-Scale Dynamic Systems*. PhD thesis, Université Rennes 1, November 2007.

[24] Indranil Gupta, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *21st Symp. Reliab. Distrib. Syst. (SRDS)*, pages 180–189, Osaka, Japan, October 2002. IEEE-CS Press.

[25] Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM T. Progr. Lang. Sys.*, 12(3):463–492, 1990.

[26] Scott Johnson, Farnam Jahanian, and Jigney Shah. The inter-group router approach to scalable group composition. In *19th Intnl. Conf. on Distrib. Comput. Syst. (ICDCS)*, pages 4–14, Austin, TX, USA, June 1999.

[27] Anne-Marie Kermarrec, Achour Mostéfaoui, Michel Raynal, Gilles Trédan, and Aline Carneiro Viana. From anarchy to geometric structuring: the power of virtual coordinates. In *27th Annual Symp. Princ. Distrib. Comput. (PODC)*, page 435, Toronto, Canada, August 2008. ACM Press.

[28] Anne-Marie Kermarrec, Achour Mostéfaoui, Michel Raynal, Gilles Trédan, and Aline Carneiro Viana. Large-scale networked systems: From anarchy to geometric self-structuring. In *10th Intnl. Conf. Distrib. Comput. Netw. (ICDCN)*, volume 5408 of *Lect. Notes Comput. Sc.*, pages 25–36, Hyderabad, India, January 2009. Springer.

[29] Leslie Lamport. On interprocess communication. *Distrib. Comput.*, 1(2):77–101, 1986.

[30] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[31] Nancy A. Lynch and Alexander A. Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In *16th Intnl. Conf. on Distrib. Comput. (DISC)*, volume 2508 of *Lect. Notes Comput. Sc.*, pages 173–190, Toulouse, France, October 2002.

[32] Michael Merritt and Gadi Taubenfeld. Computing with infinitely many processes. In *14th Intnl. Conf. Distrib. Comput. (DISC)*, volume 1914 of *Lect. Notes Comput. Sc.*, pages 164–178, Toledo, Spain, October 2000. Springer.

[33] Achour Mostéfaoui, Michel Raynal, Corentin Travers, Stacy Patterson, Divyakant Agrawal, and Amr El Abbadi. From static distributed systems to dynamic systems. In *24th Symp. on Reliab. Distrib. Syst. (SRDS)*, pages 109–118, Orlando, FL, USA, October 2005. IEEE-CS Press.

[34] M. Remedios Pallardó-Lozoya, Javier Esparza-Peidro, José-Ramón García-Escrivá, Hendrik Decker, and Francesc D. Muñoz-Escoí. Scalable data management in distributed information systems. In *Wshop. Inform. Syst. Distrib. Env. (ISDE)*, volume 7046 of *Lect. Notes Comput. Sc.*, pages 208–217, Hersonissos, Crete, Greece, October 2011. Springer.

[35] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *ACM SIGCOMM Conf.*, pages 161–172, San Diego, CA, USA, August 2001. ACM Press.

[36] André Schiper. Dynamic group communication. *Distrib. Comput.*, 18(5):359–374, 2006.

[37] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Conf.*, pages 149–160, San Diego, CA, USA, August 2001. ACM Press.

[38] Irving L. Traiger, Jim Gray, Cesare A. Galtieri, and Bruce G. Lindsay. Transactions and consistency in distributed database systems. *ACM Trans. Database Syst.*, 7(3):323–342, 1982.

[39] Sara Tucci-Piergiovanni. *Concurrent Connectivity Maintenance with Infinitely Many Processes*. PhD thesis, University of Rome "La Sapienza", Rome, Italy, November 2005.

[40] Sara Tucci Piergiovanni and Roberto Baldoni. Eventual leader election in the infinite arrival message-passing system model. In *22nd Intnl. Symp. Distrib. Comput. (DISC)*, volume 5218 of *Lect. Notes Comput. Sc.*, pages 518–519, Arcachon, France, September 2008. Springer.

[41] Luis Miguel Vaquero, Luis Rodero-Merino, Juan Cáceres, and Maik A. Lindner. A break in the clouds: towards a cloud definition. *Comput. Comm. Rev.*, 39(1):50–55, 2009.

[42] Jennifer L. Welch. Simulating synchronous processors. *Inf. Comput.*, 74(2):159–170, 1987.