Research Article Open Access

Francesc D. Muñoz-Escoí* and Rubén de Juan-Marín

On synchrony in dynamic distributed systems

https://doi.org/10.1515/comp-2018-0014
Received February 22, 2018; accepted July 10, 2018

Abstract: Many modern distributed services are deployed in dynamic systems. Cloud services are an example. They are expected to provide service to a potentially huge amount of users and may require a wide geographical deployment in multiple data centres. Their service processes vary in volume in accordance with workload variations, showing an adaptive behaviour in order to minimise economical costs.
Dynamic distributed systems may be classified considering two axes: (a) the number of processes that compose the system, and (b) the diameter of the networking graph that interconnects those processes. Other important features of dynamic systems can be derived from these two characteristics, e.g., their attainable synchrony. We analyse the level of synchrony that may be achieved in each dynamic system class and revise the existing techniques for transforming an initially asynchronous large dynamic system into another one with a higher synchrony level. With this, a larger set of problems may be handled in dynamic distributed systems. This facilitates the implementation and provision of additional services in those systems.

Keywords: distributed system, dynamic system, system interconnection, synchrony, failure detector, participant detector

1 Introduction

Service continuity has traditionally been one of the objectives in many distributed applications. To this end, processes and data should be replicated, but replication, concurrency and failures should be transparent to users [1], since another goal of those distributed applications is to provide a single-system image, i.e., to achieve *distribution transparency*. Unfortunately system components may fail. Failed components may eventually recover and be reintegrated in the system. As a result, a first level of dynamism is introduced in distributed algorithms when a recoverable system model is assumed, since the set of participating processes may evolve along the algorithm execution.

Peer-to-peer collaborative applications [2], where nodes may join and leave the system at will, mobile adhoc networks [3] in which the participating elements may temporarily be out of reach, elastic cloud computing services [4] that manage variable and potentially huge workloads, and several kinds of IoT services [5] (e.g., vehicular monitoring and reporting services for driving/routing assistance in smart traffic systems) have shown that multiple types of distributed systems are dynamic. So, a definition for dynamic distributed systems is needed.

Baldoni *et al.* [6] provide that definition with a complementary classification of those systems. Its two key parameters (degree of concurrency and networking graph diameter) condition the characteristics of each possible class of dynamic distributed system. We revise that classification considering in which classes a synchronous or partially synchronous system model may be assumed. This identifies the set of classes where some distributed problems, like consensus, are solvable when the set of participating processes may vary while those dynamic services run. In order to transform an initially asynchronous dynamic system into another one with a higher level of synchrony, two approaches exist: (a) to identify a stable subset of processes, or (b) to organise the system into a hierarchy of interconnectable subsystems. Those approaches make possible the implementation of additional services in dynamic systems, since the applications that provide those services may run in an apparently synchronous environment.

The rest of this paper is structured as follows. Section 2 summarises the existing definition and classification of dynamic distributed systems. Section 3 refines that classification considering synchrony and states some consequences of that study. Section 4 revises some related work. Finally, Section 5 concludes the paper.

***Corresponding Author: Francesc D. Muñoz-Escoí:** Institut Universitari Mixt Tecnològic d'Informàtica, Universitat Politècnica de València, 46022 València, Spain; E-mail: fmunyoz@iti.upv.es
Rubén de Juan-Marín: Institut Universitari Mixt Tecnològic d'Informàtica, Universitat Politècnica de València, 46022 València, Spain; E-mail: rjuan@iti.upv.es

2 Classification of dynamic distributed systems

According to Baldoni *et al.* [6], a dynamic distributed system is:

Definition 1 (Dynamic System). *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 provides a characterisation of dynamic distributed systems that is accompanied with a classification of them. Several considerations arise from Definition 1:

C1: Applications developed for dynamic systems are unable to know the identity of all processes that currently belong to their system.

C2: Because of C1, the communication topology in a system cannot be a fully connected graph. This means that algorithms should use a multi-hop communication mechanism in order to reach all processes when messages should be broadcast to all participants.

Since dynamism implies that processes may join and leave the system at will, the concepts of *system run*, *system graph* and *graph sequence* are defined [6].

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.*

System graphs do not need to be fully connected, and 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 classification given by Baldoni *et al.* [6] considers these two dimensions:

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

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

- *Number of concurrent entities* (P). Assuming systems with infinitely many processes, as they were identified and classified in [7], these variants can be distinguished:
  - 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.
  - P^∞: The number of processes that concurrently belong to the system in a single run may grow to infinity as 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 bounds value is known by the algorithms, i.e.: $\forall D_n \in \{D_n\}_{run}, D_n \leq b$.
  - D^n, *bounded and unknown diameter*: All diameters $\{D_n\}_{run}$ are finite in each run, but the union of all D_n in $\{D_n\}_{run}$ may be unbounded. Therefore, an algorithm has no information on the diameter.
  - D^∞, *unbounded diameter*: The diameter may grow indefinitely in a run.

With those parameters, the $M^{P,D}$ set of models is defined. Both parameters can assume values b, n and ∞ to indicate their three variants. From the nine possible models, $M^{b,n}$, $M^{b,\infty}$ and $M^{n,\infty}$ are not possible since their diameter is bounded by the number of processes, with a maximum value $D_{max} =| P | -1$. As a result, we may only have six models that correspond to the following combinations: $M^{b,b}$, $M^{n,b}$, $M^{\infty,b}$, $M^{n,n}$, $M^{\infty,n}$ and $M^{\infty,\infty}$. Those models are depicted in Table 1.

3 Classification refinement

Let us refine in Section 3.1 dimension P from the classification given in [6]. To this end, we present the concurrency subclasses proposed in [8]. Section 3.2 analyses which level of synchrony may be achieved in each class of dynamic distributed system. That analysis may be used for studying in which classes some distributed system problems (e.g., consensus) may be solved. Section 3.3 describes which techniques may increase that inherent level of synchrony in non-synchronous classes. Later on, Section 3.4 discusses the place of static distributed systems in that classification.

3.1 Concurrency refinement

Let S be a distributed system. Marcos K. Aguilera [8] proposes an alternative classification of distributed systems with infinitely many processes. In it, he focuses on the arrival of processes, since this is another source of dynamism. The identified classes are:

- M_1^n: S has a finite number (n) of processes. This model is called the *n-arrival* model. Algorithms know the n value in this model.
- M_2: S has infinitely many processes, but each run has only finitely many. This model is called the *finite arrival* model. Algorithms do not know how many processes will participate in each run.
- M_3: S has infinitely many processes, runs can have infinitely many processes, but only finitely many processes take steps in each finite time interval. This model is known as the *infinite arrival* model and contains two subsets:
  - M_3^b: An M_3 model where every run has a maximum concurrency bounded by constant b (known by algorithms). In this model, there may be infinitely many processes only when processes depart at the same rate that new processes join S. It is known as the *infinite arrival model with b-bound concurrency*. It is the P^b model in [6].
  - M_3^{finite}: An M_3 model in which every run has a maximum concurrency that is finite. It is known as the *infinite arrival model with bound concurrency*. It is the P^n model in [6].
- M_4: S has infinitely many processes, runs can have infinitely many processes, and a finite time interval may have infinitely many processes. This is the *infinite concurrency* model. This is model P^∞ in [6].

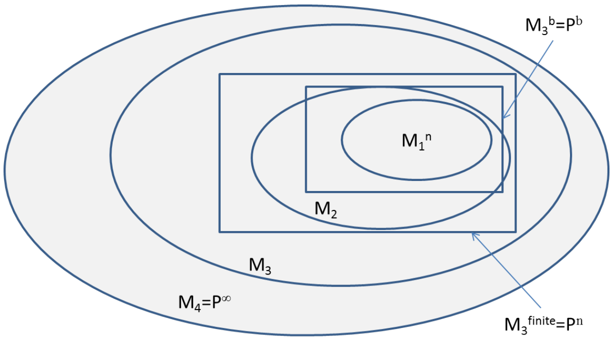

Figure 1: Relations among process system classes.

The classification of infinitely many concurrent processes proposed by Merritt and Taubenfeld [7], assumed in [6], considers similar sets than the one given in [8]. However, Aguilera [8] identifies more classes and carefully describes the inclusion relations among classes, as we summarise hereafter (They are depicted in Figure 1):

- $M_1^n \subset M_2 \subset M_3^{finite} \subset M_3 \subset M_4$;
- $M_3^b \subset M_3^{finite}$;
- $M_2 \not\subset M_3^b$;
- $M_1^n \subset M_3^b$, if $n \le b$.

These inclusion relations also arise in the P^* models identified in [6], since $P^b = M_3^b$, $P^n = M_3^{finite}$ and $P^\infty = M_4$. Therefore, in order to avoid ambiguity, when we refer hereafter to a model P^x, the following consideration should be assumed:

C3: P^∞ refers to its set of runs not included in P^n. P^n refers to its set of runs not included in P^b.

A dynamic system consists of infinitely many processes; i.e., classes M_2, M_3 and M_4 in [8]. Those classes define different process arrival models. Therefore, our classification refinement can be depicted as a third axis that refines the first one: the *arrival model* (A), either *finite* (f) or *infinite* (i). So, the resulting classification generates a set of models $\mathscr{M}^{P,D,A}$. The two arrival variants that define the A axis should be understood as follows:

- A^f (finite arrival): In a model $\mathscr{M}^{p,d,f}$, where $p \in \{b,n\}$ and $d \in \{b,n\}$, that model encompasses the subset of runs with finite processes (i.e., complying with M_2) that belong to the corresponding $M^{p,d}$ model defined in [6].
- A^i (infinite arrival): In a model $\mathscr{M}^{p,d,i}$, where $p \in \{b,n,\infty\}$ and $d \in \{b,n,\infty\}$, that model encompasses the subset of runs with infinite processes (i.e., comply-

Table 2: Dynamic models considering the P, D and A axes.

Number of	Network diameter			Process
processes	D^b	D^n	D^∞	arrival
P^b	$\mathscr{M}^{b,b,f}$	–	–	A^f
	$\mathscr{M}^{b,b,i}$	–	–	A^i
P^n	$\mathscr{M}^{n,b,f}$	$\mathscr{M}^{n,n,f}$	–	A^f
	$\mathscr{M}^{n,b,i}$	$\mathscr{M}^{n,n,i}$	–	A^i
P^∞	$\mathscr{M}^{\infty,b,i}$	$\mathscr{M}^{\infty,n,i}$	$\mathscr{M}^{\infty,\infty,i}$	

ing with M_3 or M_4) that belong to the corresponding $M^{p,d}$ model defined in [6].

In specific cases, we may also refer to the *n-arrival* (A^n) variant (i.e., a subset of runs in the *finite arrival* one) that corresponds to the M_1^n model identified in [8].

When this third axis is considered, the resulting set of dynamic system models contains the following elements: $\mathscr{M}^{b,b,f}$, $\mathscr{M}^{b,b,i}$, $\mathscr{M}^{n,b,f}$, $\mathscr{M}^{n,b,i}$, $\mathscr{M}^{n,n,f}$, $\mathscr{M}^{n,n,i}$, $\mathscr{M}^{\infty,b,i}$, $\mathscr{M}^{\infty,n,i}$, $\mathscr{M}^{\infty,\infty,i}$, as depicted in Table 2.

Note that both P^b and P^n, as depicted in Figure 1, contain runs that belong either to the M_2 or the M_3 concurrency models. However, P^∞ only contains runs in M_4 and in $M_3 \setminus P^n$ because of consideration C3. Therefore, P^∞ may only assume an infinite arrival model.

3.2 Achievable synchrony

Several degrees of asynchrony may be distinguished. To this end, Dolev *et al.* [9] (based on [10]) identify three possible axes of asynchrony in a distributed system:

1. *Processor asynchrony* allows a processor to remain in the same execution step for arbitrarily long finite intervals while other processors continue to run.
2. *Communication asynchrony* does not allow message delivery time to be limited.
3. *Message order asynchrony* allows messages to be delivered in an order different from the order in which they were sent.

It can be argued that the message order asynchrony is a consequence of the other two axes; i.e., when both processors and communications are asynchronous it is impossible to order the delivery of messages sent by different processors since there is no way to know their concrete sending order.

From the other two remaining axes, communication asynchrony seems to be the principal one. It is accepted that processor synchrony can be simulated with a reasonable effort in a system that uses *logical buffering* [11]; 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 receiver has reached the appropriate step. Communication may be asynchronous to this end. Although there are also general synchronisers (i.e., algorithms that 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 or not the studied system models could be synchronous. Besides, communication is considered *partially synchronous* when there are bounds on message transmission time but the bounds value is unknown [12], and it is considered *synchronous* when the bounds value is known [9].

Let us assume that each point-to-point communication link is synchronous, and δ is its known bound on message transmission time. This assumption provides the best scenario for ensuring that interprocess communication is synchronous, but we will see that even in this ideal scenario most dynamic system models are asynchronous.

According to consideration C2, if any algorithm step requires that a given process (for instance, a coordinator) sends a message to every other process, such message propagation will need *epidemic broadcasting* [13], i.e., each receiver forwards the message to every neighbour and remembers the message identity. Later on, if the same message is received again, it is not resent. Eventually, those messages become propagated to all system processes.

Let us analyse whether or not processes may assume bounds on message transmission time. To this end, two dimensions may be considered: process concurrency and network diameter. Process concurrency is more restrictive than network diameter in regards to communication synchrony. For instance, $\mathscr{M}^{b,b,i}$ models assume an infinite arrival of processes and such infinite arrival rate may delay without any bounds the propagation of messages. On the other hand, the network diameter may only endanger a finite and known message transmission time when the $\mathscr{M}^{\star,n,\star}$ or $\mathscr{M}^{\infty,\infty,i}$ models are considered, but in those models P^n or P^∞ should be used and both admit longer message propagation delays than P^b. Therefore, let us focus our attention on process arrival models [8] since they refine process concurrency:

– M_1^n (n-arrival) model: There are n processes in the system, and n is known by the algorithms. In this case, the communication paths between those n processes may be found using epidemic broadcasts. Therefore, we may state the following theorem:

  Theorem 1 (M_1^n synchronous communication). *Assuming a δ upper limit on message propagation time through a link, the message transmission time between every pair of processes in a distributed system S that follows the M_1^n model with a bounds value b on the network diameter is bounded and its bounds value is known.*

  Proof. The proof of this theorem is immediate, given the bounds value on the network diameter (b), the known and bounded size of the process set (n) and the link transmission time (δ).

  Note that if the network diameter is b, an epidemic broadcast will be able to reach all current system processes in b steps. Since b and δ are known, the resulting bounds value on message transmission time is also known. It is $b\delta$ in the general case, in which there is at least a stable path between p_1 and p_2. □

– M_2 (finite arrival) model: In this model, algorithms do not know how many processes will participate in each run, but it is guaranteed that there is a time t' after which no new processes are started [8]. In this concurrency model, communication is partially synchronous:

  Theorem 2 (M_2 partial synchronous communic.). *The message transmission time between every pair of processes p_1 and p_2 in a distributed system S in model M_2 is bounded but that bounds value is unknown.*

  Proof. Let p_1 and p_2 be placed on two opposite edges of the interconnecting network. Let p_{i_1} be the initial process that holds a link between $S - \{p_2\}$ and p_2.

  Without loss of generality, let p_k be the single process that directly precedes p_{i_1} in the communication path between p_1 and p_2. (If there were many p_{i_1} processes, what is described hereafter would apply for each one of them, with the same overall result.) Process p_k forwards m to p_{i_1} at time t_0. At time $t_0 + \delta_1$, with $\delta_1 < \delta$, p_{i_1} leaves the system. As a result of this, m is lost. At time $t_0 + \delta_2$, with $0 < \delta_2$ joins the system, defining a path (p_k, p_{i_2}, p_2).

  Eventually, p_k detects that m has been lost and it retries at time t_1 to forward m to p_{i_2}. However, in the interval $[t_1 + \delta_1, t_1 + \delta_2]$, p_{i_2} is replaced by p_{i_3} and m is lost again. Indeed, those message sending reattempts and process replacements may still occur multiple times from now on. In spite of this, since the process arrival is finite in this model, there will be a time t' after which no other process arrival will happen. At that moment, the links between p_k and p_2 stabilise, and m is finally delivered to p_2. However, we cannot forecast how many reattempts will be done. Thus, the delivery time of m to p_2 is bounded, but its bounds value cannot be known. □

– M_3^b (infinite arrival with b-bound concurrency) model: Processes depart from the system at an infinite rate and each time a process leaves the system, another new process replaces that leaving one. Communication is asynchronous in that kind of model, as proven in this theorem that assumes a system network diameter greater than 1:

  Theorem 3 (M_3^b asynchronous communication). *The message transmission time between processes p_1 and p_2 has no bounds in a distributed system S that follows the M_3^b model.*

  Proof. Let us look for a case where the arrival of new processes infinitely extends the communication time between p_1 and p_2. Without loss of generality, let us assume that p_1 and p_2 are placed on two opposite edges of the interconnecting network and that a single communication link connects p_2 to the remaining processes in S. Let p_{i_1} be the initial process that holds the unique link between $S - \{p_2\}$ and p_2. If there were other links, the same would happen to them.

  Let us imagine that another process p_k in the communication path between p_1 and p_2 is forwarding m to p_{i_1} at time t_0. At time $t_0 + \delta_1$, with $\delta_1 < \delta$, p_{i_1} leaves the system and m is lost. At time $t_0 + \delta_2$, with $0 < \delta_2 \le \delta_1 < \delta$, p_{i_2} joins the system such that it is able to receive the messages sent by p_k and propagate those messages to p_2.

  Eventually, p_k detects that m has been lost and it retries at time t_1 to forward m to p_{i_2}. However, at times $t_1 + \delta_1$ and $t_1 + \delta_2$, p_{i_2} is replaced by p_{i_3} and m is lost again. Indeed, those message sending reattempts and process replacements may still occur infinitely often from now on (in general, $\forall j > 0$, at times $t_j + \delta_1$, $t_j + \delta_2$, being $p_{i_{j+1}}$ replaced by $p_{i_{j+2}}$). Thus, the delivery of m to p_2 is delayed infinitely often. This means that message propagation time has no bounds in this scenario. So, the resulting communication model is asynchronous. □

– M_3^{finite} (infinite arrival with bound concurrency) model: Infinite arrival model in which each run has a maximum concurrency that is finite. Its communication is also asynchronous, as proven in this corollary:

Corollary 1 (M_3^{finite} asynchronous communication). *The message transmission time between processes p_1 and p_2 has no bounds in a distributed system S that follows the M_3^{finite} model.*

Proof. Immediate from Theorem 3. The scenario described in its proof is directly applicable in this model, too, since $M_3^b \subset M_3^{finite}$. □

– M_4 (infinite concurrency) model: Infinite arrival model in which each run has no bounds on its concurrency. Its communication is also asynchronous, as proven in this corollary:

Corollary 2 (M_4 asynchronous communication). *The message transmission time between processes p_1 and p_2 has no bounds in a distributed system S that follows the M_4 model.*

Proof. Immediate from Theorem 3. The scenario described in its proof is directly applicable in this model, too, since $M_3^b \subset M_4$. □

Once those concurrency models have been analysed, let us translate those results to our $\mathscr{M}^{P,D,A}$ refined set of models:

Corollary 3. *The $\mathscr{M}^{b,b,n}$ model may assume synchronous communication.*

Proof. Since an $\mathscr{M}^{b,b,n}$ system assumes an M_1^n process model, then Theorem 1 directly implies that $\mathscr{M}^{b,b,n}$ may assume synchronous communication. □

Corollary 4. *The $\mathscr{M}^{b,b,f}$, $\mathscr{M}^{n,b,f}$ and $\mathscr{M}^{n,n,f}$ models may assume partial synchronous communication.*

Proof. Since all finite arrival systems assume an M_2 process concurrency model, then Theorem 2 determines that all those models may assume partial synchronous communication. Note that Theorem 2 has made no assumptions on the network diameter value. Therefore, its results may be applied in all those models. □

Corollary 5. *The $\mathscr{M}^{b,b,i}$, $\mathscr{M}^{n,b,i}$, $\mathscr{M}^{\infty,b,i}$, $\mathscr{M}^{n,n,i}$, $\mathscr{M}^{\infty,n,i}$ and $\mathscr{M}^{\infty,\infty,i}$ models should assume asynchronous communication.*

Proof. Since all cited systems assume either an M_3 or M_4 process model, then Theorem 3 (or its corollaries 1 or 2) is applicable in each system. All those cases generate the same result: the asynchronous communication model should be assumed.

Note that Theorem 3 has made no assumptions on the network diameter value. Therefore, its results may be applied in all those models. □

Figure 2 shows graphically which level or levels of communication synchrony correspond to each process concurrency class.

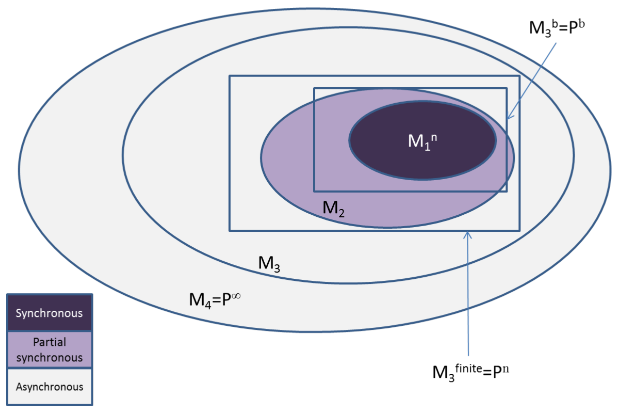

Figure 2: Maximum level of synchrony for each system class.

3.3 Consequences

Our classification refinement implies that some classical problems that are not solvable [14] in traditional asynchronous reliable distributed systems where processes may fail, will not be solvable in $\mathscr{M}^{\infty,\infty,i}$ or in any $\mathscr{M}^{\star,n,i}$ or $\mathscr{M}^{\star,b,i}$ dynamic systems. This includes *consensus* [10] and many others [14]. On the other hand, problems solvable in (partially) synchronous systems require that the resulting dynamic model becomes one of the $\mathscr{M}^{n,n,f}$, $\mathscr{M}^{n,b,f}$ or $\mathscr{M}^{b,b,f}$ subclasses.

Therefore, we need a useful framework for structuring any general (and potentially unbounded) dynamic distributed system into a set of smaller and synchronous dynamic subsystems. In this way, traditional algorithms that assume a known number of processes or that need some degree of synchrony could be used in a large dynamic system. In this regard, several basic approaches may be found:

– The first one sets a hierarchical organisation of system processes, defining multiple subsystems and interconnecting them using inter-subsystem channels. Each subsystem uses its own subnetwork and connects with other subsystems using bridges. Therefore, the algorithms in each subsystem see a fully connected network with a logical network graph diameter of length 1. On the other hand, when a process sends a message to another process in a different subsystem, it needs at least one forwarding step driven by some "bridge" process or a path along several bridges. Thus, the global network graph diameter is greater than 1 in that scenario.

  Rodrigues and Veríssimo used those principles in their *causal separators* [15] proposal. Causal separators are a scalability mechanism for causal message multicasting. With them, each subsystem may use its own internal (and different) causal multicast algorithm. Each internally delivered message is forwarded to other subsystems by a specialised bridge process that knows the addresses of the remaining bridges. In order to reach a global causal ordering, such message inter-subsystem forwarding needs FIFO (i.e., first in, first out) order when there are only two subsystems [15] or causal order in other cases [16]. This provides a first example of how to use a classical algorithm intended for a $\mathscr{M}^{b,1,f}$ system (those to be used in each subsystem) combined with another one of the same kind (the interconnecting algorithm, that is also for $\mathscr{M}^{b,1,f}$ systems) in order to manage an $\mathscr{M}^{n,b,f}$ system, since the resulting global system has a large and potentially bounded but unknown set of processes. This first kind of hierarchical architectures was used in the context of interconnectable message broadcast protocols [15–19] and interconnectable memory consistency models [20]. A hierarchical organisation provides a solid basis for building large dynamic systems in order to solve problems with a decomposable domain (e.g., support of fast consistency models [21] in case of using replicated data elements, implementation of FIFO or causal message broadcast algorithms [17]...).

  In this first example, the resulting overall system belongs to the $\mathscr{M}^{n,b,f}$ class. Let us revise whether any $\mathscr{M}^{\star,n,i}$ system may be handled with this same strategy. To this end, each started process is compelled to be integrated in any of the already existing subsystems. That was the principle that drove partially centralised P2P systems [2]. In those systems, a *supernode* (or *super-peer*) is the only process known by all processes in a given area. Supernodes index all resources in that system subset and they forward the requests that cannot be answered in that subset to other supernodes in order to get an answer. Thus, all system resources may be accessed by every process. The population in each subset varies with time and there are no upper bounds on the global amount of participating processes. In this second example, the resulting system belongs to the $\mathscr{M}^{\infty,n,i}$ class since all supernodes do not know each other: each supernode only needs to know a few other supernodes and their communication is driven by an epidemic broadcast. Besides, each supernode does not need to know the addresses of all the nodes that may use its services. That set of user nodes varies dynamically and cannot be bounded nor known with precision. The supernode only knows which resources have been published or downloaded by those processes, i.e., it only knows the addresses of a subset of processes that use its services.

  Another example of global $\mathscr{M}^{\star,n,i}$ system that may rely on simpler $\mathscr{M}^{\star,1,i}$ algorithms in each subsystem is an interconnectable FIFO multicast algorithm [22]. FIFO multicasting has more relaxed requirements than causal multicasting. Thus, FIFO multicasting only needs FIFO propagation through the inter-bridge channels. If messages are tagged with their initial bridge forwarder identifier in order to avoid repeated delivery, inter-bridge propagation can be achieved using an epidemic propagation. In that case, the bounds on the amount of subsystems that compose the distributed system may be unknown by the participating processes.

– The second approach was proposed by Mostéfaoui *et al.* [23]. It 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*), and a strong cooperation among those α processes (and this suggests that they assume a fully-interconnecting network among them). Note that α simulates the static-system requirement of maintaining at least $n - f$ correct processes, where n is the initial number of system processes and f is the current number of failed processes. 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 $\mathscr{M}^{b,b,f}$ system model, whilst the overall system may assume even the $\mathscr{M}^{\infty,\infty,i}$ one. Algorithms are executed in an $\mathscr{M}^{b,b,f}$ 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.

Fortunately, not all problems demand a synchronous system in order to be solved. So, each problem should be carefully studied to analyse in which kinds of dynamic system it is solvable. An example is presented in [6] where the *one-time query* problem [24] is initially solved in an $M^{\star,b}$ system with the *WildFire* algorithm [24], but its specification is later slightly relaxed in order to build the *Depth-Search* algorithm [6] that solves it also in any $M^{\star,n}$ model but not in an $M^{\infty,\infty}$ one.

Table 3: Models in the refined $M^{P,D,A}$ taxonomy.

#	Network diameter				Proc
Proc	D^1	D^b	D^n	D^∞	arriv
P^b	$\mathscr{M}^{b,1,n}$	$\mathscr{M}^{b,b,n}$	–	–	A^n
	$\mathscr{M}^{b,1,f}$	$\mathscr{M}^{b,b,f}$	–	–	A^f
	$\mathscr{M}^{b,1,i}$	$\mathscr{M}^{b,b,i}$	–	–	A^i
P^n	$\mathscr{M}^{n,1,f}$	$\mathscr{M}^{n,b,f}$	$\mathscr{M}^{n,n,f}$	–	A^f
	$\mathscr{M}^{n,1,i}$	$\mathscr{M}^{n,b,i}$	$\mathscr{M}^{n,n,i}$	–	A^i
P^∞	$\mathscr{M}^{\infty,1,i}$	$\mathscr{M}^{\infty,b,i}$	$\mathscr{M}^{\infty,n,i}$	$\mathscr{M}^{\infty,\infty,i}$	

3.4 Static and dynamic frontier

A question that arises from the properties outlined above is where to place the frontier between *static* and *dynamic* systems. According to Definition 1, each process in a dynamic system is unable to know the identity and location of all other processes since that system may have a potentially very large number of processes that is dynamically changing. Because of this, the resulting communication graph has a diameter greater than one, since each process is usually unable to directly communicate with every other. Thus, a static system should break both conditions. This means that: (S1) the identity and address of all the remaining processes are known by each process (i.e., the amount of processes in the system is bounded and the bounds value is known); and (S2) this allows a logically fully connected network among all system processes (i.e., a network graph diameter of length 1). Those two characteristics define an $\mathscr{M}^{b,1,n}$ submodel of static distributed systems in the $\mathscr{M}^{b,b,f}$ model from our classification refinement. The n-arrival concurrency model (i.e., model M_1^n in [8]) enables processes to know each other, and a value 1 for the network diameter matches its assumed logical fully connected network.

Therefore, that hypothetical frontier cannot be drawn on a grid of distributed system classes as that depicted in Table 2, since static distributed systems are only a subpart of the minimal set that can be defined in that $M^{P,D,A}$ taxonomy. Thus, in order to represent it, we need to explicitly consider the A^n arrival model that does not comply with condition C1 and we should also define a D^1 class (i.e., the assumed system network graph has diameter 1) that implicitly breaks condition C2. This means that a new refinement step is needed for considering static distributed systems. That new A^n arrival model only makes sense in the P^b process model, since it stipulates that the overall number of processes in a run is bounded and known by the algorithms, and that requirement is only met in the P^b model.

Table 3 shows the resulting grid of distributed system models with this refined $M^{P,D,A}$ taxonomy.

All models in Table 3 that are also in Table 2 correspond to dynamic distributed systems. On the other hand, model $\mathscr{M}^{b,1,n}$ corresponds to static distributed systems. But there are several other cells in the grid that are not in any of those two sets. They may be considered as *partially dynamic* distributed systems, because:

- Those models placed in the first row and not in the first column (i.e., model $\mathscr{M}^{b,b,n}$) comply with condition C2 but do not comply with C1. This means that they are not entirely dynamic. However, they comply with condition S1 but not with S2. So, they are not entirely static.
- Those models placed in the first column and not in the first row (i.e., models $\mathscr{M}^{b,1,f}$, $\mathscr{M}^{b,1,i}$, $\mathscr{M}^{n,1,f}$, $\mathscr{M}^{n,1,i}$, $\mathscr{M}^{\infty,1,i}$) do not comply with C2. So, they are not entirely dynamic. However, they comply with condition S2 but not always with S1 (those runs with A^f will eventually comply with S1 once no other processes arrive to the system). Therefore, they are not entirely static, either.

There have been several examples of those partially dynamic distributed systems. For instance, the system described in [25] belongs to the $\mathscr{M}^{b,1,i}$ class, since it assumes a known upper value in the amount of system processes (P^b), an infinite arrival of them (A^i) and IP-multicasting in order to intercommunicate the currently running processes (D^1).

On the other hand, the $\mathscr{M}^{\infty,1,i}$ class cannot be implemented in a real system. Infinite concurrent processes in a finite interval of time cannot exist. If they were, they would neither have an interconnecting network of diameter 1. It is only a theoretical model.

4 Related work

As it is explained in Consideration C1 and Definition 1, not all processes in a dynamic distributed system know each other. We have analysed the level of synchrony that can be guaranteed in each class of dynamic system. Synchrony may be needed for solving some problems in a distributed system and consensus is an example of those problems. There have been multiple previous papers that have solved consensus with unknown participants. They also studied the needed level of synchrony, although implicitly, since the failure detectors [26] used in each paper were different, and each one required a given level of synchrony in order to be implemented. For instance, according to Larrea *et al.* [27], $\mathscr{W}$, $\mathscr{Q}$, $\mathscr{S}$ and $\mathscr{P}$ detectors demand a synchronous model, while $\Diamond\mathscr{W}$, $\Diamond\mathscr{Q}$, $\Diamond\mathscr{S}$ and $\Diamond\mathscr{P}$ demand at least a partial synchronous model when processes may fail.

To begin with, Jiménez *et al.* proved in [28] that none of the eight failure detector classes proposed in [26] may be implemented in a system with unknown membership, but the Ω [29] failure detector can be implemented [30] there. In order to circumvent that impossibility, Cavin *et al.* [31] introduced the concept of *participant detectors*. In that scope, each process calls its local participant detector in order to obtain an approximation on the current set of participating processes. Participant detectors have two properties:

– Information inclusion: The information returned by each detector is non-decreasing over time.
– Information accuracy: A detector never returns a process that does not belong to the system.

In the system assumed in [31] processes cannot fail. That paper proves that consensus may be solved in that system with a *one sink reducibility* (OSR) participant detector. OSR is a participant detector that requires that the detected network graph is connected and that its directed acyclic graph obtained by reducing the original directed interconnection graph to its strongly connected components has only one sink.

In practice, this means that the interconnecting network should be stable and the set of processes respects an M_1^n model [8]. Since failures are not tolerated in that paper, nothing is mentioned about the failure detector being needed for supporting consensus in that scenario or about the actual level of synchrony required for implementing consensus in that system.

In a subsequent technical report [32], the same authors extend their results to a crash-prone model and provide a solution for the consensus problem with unknown participants. In that case, they explicitly recognise that the OSR participant detector should be complemented with a perfect ($\mathscr{P}$) failure detector. According to Larrea *et al.* [27], $\mathscr{P}$ may only be implemented in a synchronous system. This matches what we have outlined in our previous sections, since the system being assumed in [32] is an example of an $\mathscr{M}^{b,b,n}$ system and those systems may assume synchronous communication.

Later, Greve and Tixeuil [33] explicitly combine an Ω failure detector with a k-OSR participant detector in order to solve consensus on an asynchronous system with unknown membership. In that case, a finite process set is assumed and processes may fail by crashing. The k-OSR participant detector ensures a k-connected network graph and the system tolerates up to f concurrent failures, with $f < k$. With our results, this corresponds to a $\mathscr{M}^{b,b,f}$ model that admits up to a partial synchronous communication model. The usage of an Ω failure detector matches that level of synchrony, according to [27, 28].

Recently, Alchieri *et al.* [34, 35] have addressed the problem of Byzantine consensus with unknown participants for systems with n processes where the n value is unknown (i.e., these systems follow the M_2 model) and up to f concurrent failures. The interconnecting network is k-strongly connected, and the algorithm requires a k-OSR participant detector where $f < \frac{k}{2} < n$ and the sink graph component should have at least $3f + 1$ processes. When all those requirements are met, Byzantine consensus may be solved like in any distributed system with known participants. This means that any of the Byzantine-specific failure detectors proposed in [36] may be assumed. Those failure detectors require a partial synchronous system in order to be implemented, according to [36].

5 Conclusions

Baldoni et al. [6] identify six different classes of dynamic systems (specified as $M^{P,D}$), crossing two 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 have analysed the highest degree of system synchrony achievable in each class. To this end, a new A (i.e., arrival model) axis has been added to that classification ($\mathscr{M}^{P,D,A}$).

Our analysis of the attainable degree of synchrony in every dynamic system class has shown that six classes are inherently asynchronous, even when synchronous links are assumed: $\mathscr{M}^{\infty,b,i}$, $\mathscr{M}^{\infty,n,i}$, $\mathscr{M}^{\infty,\infty,i}$, $\mathscr{M}^{n,b,i}$, $\mathscr{M}^{n,n,i}$ and $\mathscr{M}^{b,b,i}$. The algorithms that require at least a partially synchronous model cannot be run in those classes, but there are some system structuring techniques that allow the transformation of asynchronous classes into others that admit a synchronous model. Those techniques are based on two principles: (1) a division of the initial system in multiple interconnectable subsystems with a bounded network graph diameter and a global hierarchical structure, imposing a finite arrival model in each subsystem, and (2) the definition of a stable group of processes that sets the needed level of synchrony for allowing algorithm progress. With those techniques, a larger set of services may be implemented in dynamic systems.

The degree of synchrony that may be attained in different classes of dynamic systems has been implicitly studied in previous works focused on solving consensus with unknown participants. In that scope, synchrony is embedded in the implementation requirements of the failure detectors being needed for solving consensus. Our results provide an alternative way of discussing synchrony in these systems, being more direct than determining synchrony based on failure detectors.

References

[1] Traiger I. L., Gray J., Galtieri C. A., Lindsay B. G., Transactions and consistency in distributed database systems, ACM Transactions on Database Systems, 1982, 7(3), 323–342

[2] Androutsellis-Theotokis S., Spinellis D., A survey of peer-to-peer content distribution technologies, ACM Computing Surveys, 2004, 36(4), 335–371

[3] Ruiz P., Bouvry P., Survey on broadcast algorithms for mobile ad hoc networks, ACM Computing Surveys, 2015, 48(1), 8:1–8:35

[4] Muñoz-Escoí F. D., Bernabéu-Aubán J. M., A survey on elasticity management in PaaS systems, Computing, 2017, 99(7), 617–656

[5] Gubbi J., Buyya R., Marusic S., Palaniswami M., Internet of things (IoT): A vision, architectural elements, and future directions, Future Generation Comp. Syst., 2013, 29(7), 1645–1660

[6] Baldoni R., Bertier M., Raynal M., Tucci-Piergiovanni S., Looking for a definition of dynamic distributed systems, In: 9th International Conference on Parallel Computing Technologies (PaCT), LNCS, Springer, 2007, 4671, 1–14

[7] Merritt M., Taubenfeld G., Computing with infinitely many processes, In: 14th International Conference on Distributed Computing (DISC), LNCS, Springer, 2000, 1914, 164–178

[8] Aguilera M. K., A pleasant stroll through the land of infinitely many creatures, SIGACT News, 2004, 35(2), 36–59

[9] Dolev D., Dwork C., Stockmeyer L. J., On the minimal synchronism needed for distributed consensus, Journal of the ACM, 1987, 34(1), 77–97

[10] Fischer M. J., Lynch N. A., Paterson M., Impossibility of distributed consensus with one faulty process, Journal of the ACM, 1985, 32(2), 374–382

[11] Welch J. L., Simulating synchronous processors, Information and Computation, 1987, 74(2), 159–170

[12] Dwork C., Lynch N. A., Stockmeyer L. J., Consensus in the presence of partial synchrony, Journal of the ACM, 1988, 35(2), 288–323

[13] Gupta I., Kermarrec A. M., Ganesh A. J., Efficient epidemic-style protocols for reliable and scalable multicast, In: Proceedings of 21st IEEE Symposium on Reliable Distributed Systems (SRDS), IEEE-CS Press, 2002, 180–189

[14] Fich F., Ruppert E., Hundreds of impossibility results for distributed computing, Distributed Computing, 2003, 16(2-3), 121–163

[15] Rodrigues L., Veríssimo P., Causal separators for large-scale multicast communication, In: Proceedings of 15th International Conference on Distributed Computing Systems (ICDCS), 1995, 83–91

[16] Baldoni R., Friedman R., van Renesse R., The hierarchical daisy architecture for causal delivery, In: Proceedings of 17th International Conference on Distributed Computing Systems (ICDCS), 1997, 570–577

[17] Johnson S., Jahanian F., Shah J., The inter-group router approach to scalable group composition, In: Proceedings 19th IEEE International Conference on Distributed Computing Systems (ICDCS), 1999, 4–14

[18] de Juan-Marín R., Cholvi V., Jiménez E., Muñoz-Escoí F. D., Parallel interconnection of broadcast systems with multiple FIFO channels, In: On the Move to Meaningful Internet Systems (OTM 2009), LNCS, Springer, 2009, 5870, 449–466

[19] de Juan-Marín R., Decker H., Armendáriz-Íñigo J. E., Bernabéu-Aubán J. M., Muñoz-Escoí F. D., Scalability approaches for causal multicast: a survey, Computing, 2016, 98(9), 923–947

[20] Fernández A., Jiménez E., Cholvi V., On the interconnection of causal memory systems, In: Proceedings of the 19th annual ACM Symposium on Principles of Distributed Computing (PODC), ACM Press, 2000, 163–170

[21] Attiya H., Friedman R., Limitations of fast consistency conditions for distributed shared memories, Information Processing Letters, 1996, 57(5), 243–248

[22] Álvarez Á., Arévalo S., Cholvi V., Fernández A., Jiménez E., On the interconnection of message passing systems, Information Processing Letters, 2008, 105(6), 249–254

[23] Mostéfaoui A., Raynal M., Travers C., Patterson S., Agrawal D., El Abbadi A., From static distributed systems to dynamic systems, In: 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05), IEEE-CS Press, 2005, 109–118
[24] Bawa M., Gionis A., Garcia-Molina H., Motwani R., The price of validity in dynamic networks, Journal of Computer and System Sciences, 2007, 73(3), 245–264
[25] Tucci-Piergiovanni S., Baldoni R., Eventual leader election in infinite arrival message-passing system model with bounded concurrency, In: 8th European Dependable Computing Conference (EDCC), 2010, 127–134
[26] Chandra T. D., Toueg S., Unreliable failure detectors for reliable distributed systems, Journal of the ACM, 1996, 43(2), 225–267
[27] Larrea M., Fernández A., Arévalo S., On the implementation of unreliable failure detectors in partially synchronous systems, IEEE Transactions on Computers, 2004, 53(7), 815–828
[28] Jiménez E., Arévalo S., Fernández A., Implementing unreliable failure detectors with unknown membership, Information Processing Letters, 2006, 100(2), 60–63
[29] Chandra T. D., Hadzilacos V., Toueg S., The weakest failure detector for solving consensus, Journal of the ACM, 1996, 43(4), 685–722
[30] Aguilera M. K., Delporte-Gallet C., Fauconnier H., Toueg S., On implementing omega with weak reliability and synchrony assumptions, In: Proceedings of the 22nd annual Symposium on Principles of Distributed Computing (PODC'03), ACM Press, 2003, 306–314
[31] Cavin D., Sasson Y., Schiper A., Consensus with unknown participants or fundamental self-organization, In: 3rd International Conference on Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW), 2004, 135–148
[32] Cavin D., Sasson Y., Schiper A., Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes, Technical report IC/2005/026, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2005
[33] Greve F., Tixeuil S., Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks, In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), 2007, 82–91
[34] Alchieri E. A. P., Bessani A. N., da Silva Fraga J., Greve F., Byzantine consensus with unknown participants, In: 12th International Conference on Principles of Distributed Systems (OPODIS), 2008, 22–40
[35] Alchieri E. A. P., Bessani A., Greve F., da Silva Fraga J., Knowledge connectivity requirements for solving Byzantine consensus with unknown participants, IEEE Transactions on Dependable and Secure Computing, 2018, 15(2), 246–259
[36] Kihlstrom K. P., Moser L. E., Melliar-Smith P. M., Byzantine fault detectors for solving consensus, The Computer Journal, 2003, 46(1), 16–35