Synchronous Partitioning and its Effects on Fault Tolerance

Raimundo José de Araújo Macêdo
macedo@ufba.br

Distributed Systems Laboratory (LaSiD)
DCI/IC – Federal University of Bahia (UFBA)

Abstract. *This paper investigates the resilience of the Partitioned Synchronous (PaS) model—a hybrid distributed system model in which synchronous subsystems communicate asynchronously. To this end, we evaluate two consensus algorithms: a novel flooding-based protocol and new simulations of a previously developed leader-based protocol. The results show that consensus algorithms in the PaS model achieve resilience comparable to fully synchronous systems, while reducing implementation costs, as only partitions are required to operate under strict synchrony assumptions. The PaS model is particularly suitable for cyber-physical systems, edge computing infrastructures, and modular data centers, where localized synchrony can be enforced within partitions, while global communication remains asynchronous.*

Resumo. *Este artigo investiga a resiliência do modelo Síncrono Particionado (PaS)—um modelo híbrido de sistemas distribuídos no qual subsistemas síncronos se comunicam de forma assíncrona. Para isso, avaliamos dois algoritmos de consenso: um novo protocolo baseado em flooding e novas simulações de um protocolo baseado em líder previamente desenvolvido. Os resultados mostram que algoritmos de consenso no modelo PaS alcançam resiliência comparável à de sistemas totalmente síncronos, ao mesmo tempo em que reduzem os custos de implementação, já que apenas as partições precisam operar sob suposições estritas de sincronismo. O modelo PaS é particularmente adequado para sistemas ciberfísicos, infraestruturas de edge computing e data centers modulares, nos quais o sincronismo localizado pode ser imposto dentro das partições, enquanto a comunicação global permanece assíncrona.*

1. Introducation

The fault-tolerance guarantees of distributed system applications are fundamentally tied to the timing assumptions governing process interactions. In synchronous distributed systems, where known upper bounds exist on communication delays and processing times for all processes, core fault-tolerance mechanisms such as distributed consensus can be implemented with relative ease [Lamport et al. 1982, Cristian 1991, Lynch 1996, Raynal 2018]. These systems offer predictable behavior, enabling processes to coordinate effectively within bounded timeframes. However, the assumption of perfect synchrony is often unrealistic in real-world deployments, particularly when the Internet is involved as part of the communication infrastructure.

In contrast, fully asynchronous systems, lacking such a timing guarantee, present inherent challenges. The seminal work by Fischer, Lynch, and Paterson (FLP) demonstrated the impossibility of achieving deterministic consensus in such environments, even in a single process failure [Fisher et al. 1985]. This result highlights the fundamental trade-off: while asynchrony offers flexibility and robustness against timing uncertainties, it severely limits the solvability of critical distributed problems.

Researchers have extensively explored intermediate system models to bridge the gap between these extreme cases, often called partially synchronous models. These models aim to balance the ease of problem solvability with the practical constraints of implementation costs [Dwork et al. 1988, Chandra and Toueg 1996], offering different fault-tolerance guarantees and trade-offs.

This paper analyzes the resilience of the Partitioned Synchronous (PaS) model—an intermediate distributed system model in which the system is divided into synchronous subsystems that communicate asynchronously across partitions [de Araújo Macêdo and Gorender 2008, de Araújo Macêdo and Gorender 2009]. This model is well-suited to cyberphysical systems, edge computing infrastructures, and modular data centers, where local synchrony can be enforced within partitions, while global communication remains inherently asynchronous.

To evaluate the fault-tolerance capabilities of the model, we study two consensus algorithms: a novel flooding-based protocol introduced in this work and a previously developed leader-based protocol, now explored through new simulations focusing on resilience and execution bounds under different partitioning configurations. The results show that these consensus algorithms in the PaS model achieve resilience and bounded step complexity comparable to fully synchronous systems, while reducing implementation costs, as strict synchrony is required only within partitions.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 reviews the PaS model and the main fault-tolerance problems previously addressed within this model. Section 4 presents a novel flooding-style consensus algorithm along with its correctness proofs, demonstrating the solvability power and efficiency of the PaS model for fail-silent failures. In Section 6, the results of new simulations of a leader-based consensus protocol, previously developed for the PaS model, are presented, analyzing the impact of the number of synchronous partitions on the resilience of the consensus protocol. Finally, Section 7 concludes the paper.

2. Related Work

Dwork, Lynch, and Stockmeyer, in their seminal 1988 paper on consensus in distributed systems, introduced the concept of Global Stabilization Time (GST) to characterize models of partial synchrony [Dwork et al. 1988]. They emphasized the importance of preserving safety properties (agreement and validity) even when liveness (termination) is not always achievable, allowing consensus termination to depend on system stability concerning timing and failures. GST refers to an unknown but finite point in time after which a distributed system behaves synchronously, meaning message transmission and processing delays become bounded and predictable. Before GST, the system behaves asynchronously, where messages may experience arbitrary delays and no fixed timing guarantees exist.

Subsequently, other partially synchronous models emerged, establishing solvability criteria based on synchronous behavior occurring for sufficiently long periods, even without continuous assurance. These systems, therefore, alternate between time-bounded synchronous behavior and unbounded asynchronous behavior, exhibiting temporal hybridity. For instance, timed asynchronous systems depend on stable periods for fault tolerance services, alternating between synchronous and asynchronous phases [Cristian and Fetzer 1999].

Asynchronous systems augmented with failure detectors, specifically the $\diamond\mathcal{S}$ class, illustrate another partially synchronous approach [Chandra and Toueg 1996]. Identified as the weakest class capable of solving consensus in asynchronous environments, $\diamond\mathcal{S}$ detectors exhibit Strong Completeness (eventual permanent suspicion of crashed processes by correct ones) and Eventual Weak Accuracy (eventual absence of suspicion of correct processes by correct ones). The Global Stabilization Time (GST) assumption, which requires eventual synchronous behavior [Dwork et al. 1988], is crucial for $\diamond\mathcal{S}$-based consensus protocols to function correctly.

Paxos [Lamport 2001] and RAFT [Ongaro and Ousterhout 2014] are known protocols that rely on GST to guarantee consensus termination from a stable leader. However, due to its strong leader election mechanism, RAFT can ensure progress more efficiently: If there is no leader or the leader fails, RAFT guarantees that a new leader will be elected in a bounded number of steps after GST.

In these models, consensus is solvable, but there is no (*upper bound*) on execution time regarding the number of communication rounds. Given that *GST* is the time the system stabilizes, it is only possible to prove an upper bound of $\beta + GST$, for a known β and arbitrary values of *GST*.

In the synchronous model with crash failures, $f + 1$ communication rounds are necessary and sufficient for consensus with up to f faulty processes [Fischer et al. 1981]. Applying the PaS model also allows us to establish an *upper bound* for communication steps to terminate consensus, whether it is flooding-based (Section 4) or leader-based (Section 6.1).

Beyond temporal hybridity, some models simultaneously incorporate synchronous and asynchronous components, creating spatial hybridity. A key example is the Trusted Computing Base (TCB) model, which employs a synchronous wormhole for fault tolerance [Veríssimo and Casimiro 2002][1].

Furthermore, other hybrid models in space and time have been introduced [Gorender et al. 2005, Veríssimo 2006]. Notably, the model Gorender, Macêdo, and Raynal (2005) proposed in [Gorender et al. 2005, Macêdo et al. 2005] allows systems to transition between fully synchronous and fully asynchronous states dynamically. This transition is achieved through self-adaptive fault-tolerant protocols, which are guided by QoS management mechanisms that adjust based on the system's runtime conditions.

The Partitioned Synchronous (PaS) model discussed in this paper is a hybrid model, characterized by the partition of the system into synchronous subsystems or components that are interconnected by asynchronous communication [de Araújo Macêdo and Gorender 2012]. This distinguishes it from the models mentioned above, as synchronous partitions are permanent, each containing at least one cor-

[1]In a synchronous wormhole, every system process has access to synchronous communications.

rect process. Consequently, the system cannot become fully asynchronous or fully synchronous—the synchronous partitions remain interconnected by asynchronous channels at all times.

Because synchronous and asynchronous channels and processes may exist in parallel, the underlying system model is hybrid in space (similar to the Trusted Computing Base (TCB) model). However, unlike TCB, the nature of PaS's hybridity does not require that all processes be interconnected by timely channels (which would characterize a synchronous wormhole).

Perfect failure detectors (class P [Chandra and Toueg 1996]), which never produce false positives or negatives, cannot be implemented in purely asynchronous systems, as this would contradict the FLP result [Fisher et al. 1985]. Additionally, they are not implementable in partially synchronous systems with eventual stability conditions, such as Global Stabilization Time (GST) [Larrea 2002].

Researchers have proposed alternative implementations of failure detectors of class P. Veríssimo and Casimiro utilized a synchronous wormhole [Casimiro and Veríssimo 1999], while Fetzer introduced a hardware watchdog that forces crashes to prevent false suspicions [Fetzer 2003]. However, as shown in [de Araújo Macêdo and Gorender 2009], the PaS Model implements P without requiring a synchronous wormhole, a synchronous spanning tree [Gorender and de Araújo Macêdo 2002], or forced crashes.

The next section reviews the PaS model and briefly describes the main algorithms developed within this framework. To further demonstrate the model's solvability power, a novel flooding-style consensus algorithm, and its correctness proof are presented in Section 4.

3. The Partitioned Synchronous Model

3.1. System Model and Assumptions

A distributed system consists of a set $\Pi = \{p_1, p_2, ..., p_n\}$ of processes running on networked computers and a set $\chi = \{c_1, c_2, ..., c_m\}$ of communication channels. Processes communicate via a transport-level protocol, where each channel c_i represents a logical "can communicate" relation rather than a direct network link. The system forms a simple complete undirected communication graph $DS(\Pi, \chi)$ with $(n \times (n-1))/2$ edges, meaning every process is linked to all others, and network partitioning is not considered.

Processes execute steps (message reception, message sending, or local state changes) and have local clocks with bounded drift ρ. Processes and channels can be *timely* or *untimely*. A process is timely if its computation steps have a known upper bound ϕ; a channel is timely if it has a bounded transmission delay δ and connects two timely processes. Otherwise, they are untimely, with finite but arbitrary bounds. If a timely network path exists between two processes, their logical communication channel is also timely.

Communication channels are bidirectional and reliable, ensuring messages are neither lost nor altered. The *QoS* function classifies processes and channels as either timely (T) or untimely (U), and their QoS remains static over time. Processes may fail by halting but do not recover within the time interval of interest.

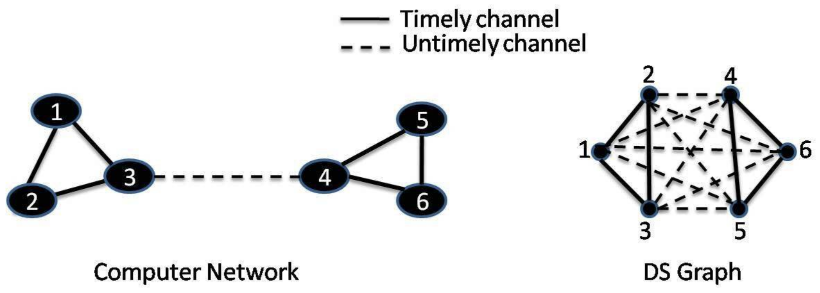

Figure 1. An example of a computer network and its representation in the DS graph.

In a distributed system DS, a synchronous subgraphs is defined as a group of processes that communicate synchronously (i.e., with timely channels). A synchronous partition is a maximal such subgraph, meaning that no additional process can be included while maintaining synchrony. There is at least one correct process in each synchronous partition. Processes in different synchronous partitions communicate asynchronously (i.e., with untimely channels). These concepts are now formally defined.

3.2. Synchronous Subgraphs and Synchronous Partitions

A **synchronous subgraph** of the distributed system DS is an induced subgraph $G_s(\Pi_s, \chi_s)$, $\Pi_s \subseteq \Pi$ and $\chi_s \subseteq \chi$, where every channel $c \in \chi_s$ is timely, and every process $p \in \Pi_s$ is timely.

A **synchronous partition** is a *maximal synchronous subgraph* $G_s(\Pi_s, \chi_s)$, meaning that no additional process from $\Pi \setminus \Pi_s$ can be included while maintaining synchrony.

The set $\mathcal{P} = \{G_s^1, G_s^2, ..., G_s^k\}$ of all synchronous partitions forms a **family of maximal cliques** in DS, where:

- Each G_s^i is a maximal synchronous subgraph,
- For any $i \neq j$, processes in G_s^i and G_s^j communicate asynchronously, by means of untimely channels.

This formulation ensures that synchronous partitions capture the hybrid nature of DS, where *synchronous components* (maximal cliques) are interconnected via *untimely channels*.

In a PaS distributed system, the following property is necessary to implement a perfect failure detector [de Araújo Macêdo and Gorender 2009]:

strong partitioned synchrony:

$$(\forall p_i \in \Pi)(\exists G_s \subset DS) \text{ such that } p_i \in G_s. \tag{1}$$

This property ensures that every process p_i is contained in one of the k synchronous partitions, where $k \geq 2$. If some processes do not belong to a synchronous partition, the following weaker property holds:

weak partitioned synchrony: The non-empty set of processes that belong to synchronous partitions is a proper subset of Π. More formally, assuming that at least one synchronous partition G_s exists: $(\exists p_i \in \Pi)(\forall G_s \subset DS)(p_i \notin G_s)$.

Even with weak partitioned synchrony, it is possible to leverage the existing synchrony to enhance the resilience of fault-tolerant protocols [de Araújo Macêdo and Gorender 2012].

In the example of Figure 1, *strong partitioned synchrony* holds because the two synchronous partitions together include all processes in DS.

Because $G_s \subset DS$ in the specification of PaS with *strong partitioned synchrony*, this specification excludes a fully synchronous system with a single synchronous partition containing all processes in Π. Likewise, it does not imply the existence of a synchronous *wormhole* ([Veríssimo and Casimiro 2002]) or a synchronous *spanning tree* ([Gorender and de Araújo Macêdo 2002]), that is, a subgraph G_s where $\Pi = \Pi'$ (note that in the example of Figure 1 there is no such a component G_s that includes all vertices of DS).

3.3. Fault-Tolerant Services under the PaS Model

The concept of Perfect Failure Detectors in the PaS model, introduced in [de Araújo Macêdo and Gorender 2008], has been utilized to enhance the robustness of other fault-tolerant services. Fault-Tolerant k-Mutex Protocols, introduced by Ramos, Macêdo, and Blagojević [Ramos et al. 2012], propose a k-mutex algorithm that ensures liveness without the restrictive assumptions of traditional synchronous solutions. Operating under strong partitioned synchrony, this algorithm relies on quorum formation to grant access to the critical section, with the quorum adjusted using the perfect failure detector. A Byzantine Consensus Algorithm, developed by Silva, Macêdo, and Ramos [da Silva et al. 2018], requires at least $3f+1$ replicas. Given the high redundancy cost, the authors explored an adaptive approach based on the PaS model. The proposed Byzantine failure detector and consensus algorithm dynamically adjust the quorum of operational processes, improving resilience compared to traditional static quorum methods, as confirmed by simulations. Finally, in [Santos and Gorender 2021], the perfect failure detector under the PaS model is used to install new views in the group membership algorithm.

4. A Flooding Consensus Algorithm

The pseudocode presented in Algorithm 1 describes a consensus algorithm for partitioned synchronous distributed systems with crash failures. It is inspired by the flooding algorithm for Byzantine processes introduced by Dolev [Dolev and Strong 1983] and later adapted by Attiya and Welch [Attiya and Welch 1998]. This adaptation modifies the integrity requirement to apply to all processes, regardless of their correctness. If all processes propose the same value, any correct process will decide on that value. The algorithm assumes only process crash failures, meaning that proposed values will not differ due to failures. This assumption allows the use of the minimum (or any deterministic) function to select a decision value from the proposed values. To ensure correctness, a consensus algorithm must satisfy the following specifications [Lynch 1996, Raynal 2018]:

- Termination: A process eventually decides on a value.
- Agreement: Correct processes decide on the same value.
- Integrity: If all processes propose the same value, they must decide on that common value.

Given that the algorithm assumes only crash failures (no Byzantine failures), the integrity property can be replaced by the simpler validity property:

- Validity: The decision value is one of the proposed values.

For synchronous distributed systems, it has been proven that any algorithm solving consensus in a system with up to f crash failures requires at least f + 1 rounds of message exchanges [Dolev and Strong 1983]. This lower bound highlights the inherent cost of achieving consensus in the presence of failures. A similar pattern follows for partitioned distributed systems, as it is shown below.

The flooding algorithm for a synchronous system proceeds in consecutive synchronous rounds until consensus is reached. In each round, correct processes collect proposed values from other processes and broadcast any collected values that have not yet been sent. By assumption, at most f processes may crash, and in the worst case, all f crashes occur over r rounds. Thus, the algorithm requires $f + 1$ rounds to guarantee that all missing messages are recovered.

Now, consider the flooding consensus algorithm for a partitioned synchronous system. Unlike synchronous systems, a round in a partitioned distributed system cannot be defined by the passage of time. Instead, the algorithm proposed here uses sequential round numbers associated with transmitted messages to characterize consecutive rounds and leverages the perfect failure detector presented in [de Araújo Macêdo and Gorender 2012] to determine the end of a round, allowing processes to reach a decision.

Algorithm 1, designed for a process p_i within a partitioned synchronous distributed system, implements consensus among a group of processes denoted by g. The algorithm proceeds in consecutive rounds, represented by the variable r, for a total of $f + 1$ rounds. This means that the algorithm runs until a sent message tagged with round r has been received by every correct process in g.

In the pseudocode below, v_i is the proposed value of process p_i, and V is the set containing all proposed values from the processes in g, updated in each round with retransmitted messages. M is a set of proposed values.

Algorithm 1 A Flooding-Based Consensus. Algorithm for process p_i

```
1: $V \leftarrow \{v_i\}$
2: $r \leftarrow 1$                                              ▷ Initialize round number
3: while $r \leq f + 1$ do
4:     $\forall p_j \in g, j \neq i,$ send $(r, \{v \in V \mid v$ not already sent to $p_j\})$ to $p_j$
5:     whenever receive $(r, M)$ from $p_j$ do
6:         $V \leftarrow V \cup M$
7:     wait until received $(r, \_)$ from all $p_j \in g \setminus faulty_i \cup \{p_i\}$
8:     $r \leftarrow r + 1$                                    ▷ Proceed to the next round
9: end while
10: $d_i \leftarrow \min(V)$
```

Initially, V contains the initial value v_i of process p_i. In each round r, p_i sends the current round number r along with any new messages in V to all other functioning processes in the group g. The process then waits to receive messages tagged with round r from all other functioning processes. Upon receiving a message (r, M) from a process p_j, p_i updates its message set V by adding the received messages M. After all rounds are completed, the process determines the consensus decision value as the minimum value d_i from V. This algorithm ensures that each process continuously broadcasts its messages and receives messages from all other functioning processes in each round, ultimately reaching consensus on the minimum proposed value.

5. Correctness Proof

The correctness of the algorithm is formally established by proving both the safety and the liveness properties.

5.1. Definitions and Assumptions

To be correct, consensus requires agreement, validity, and termination:

- **Agreement**: No two correct processes decide on different values.
- **Validity**: The decision value is one of the proposed values.
- **Termination**: Every correct process eventually decides on some value.

A PaS model with *strong partitioned synchrony*, as presented in Section 3, and the perfect failure detector, as introduced in [de Araújo Macêdo and Gorender 2012] are assumed. Thus, when a process fails, its identification number is promptly added to the $faulty_i$ set of process p_i.

5.2. Safety Proof

Lemma 1 (Agreement). *If two correct processes decide, they decide the same value.*

Let V_i be the set of values received by process i. Because the algorithm (line 10) defines the consensus decision value d_i as the minimum of V_i, proving that $V_i = V_j$ for all processes i, j after round $f + 1$ is sufficient. **Claim:** $\forall i, j$, after round $f + 1$, $V_i = V_j$.

Proof. Suppose, for the sake of contradiction and without loss of generality, that after $f + 1$ rounds, there exists $x \in V_i$ such that $x \notin V_j$.

Observe that $x \notin V_i$ at round f, because otherwise, p_i would have transmitted x to p_j in round $f + 1$. Consequently, some other process p_k that transmitted x to p_i at round $f + 1$ must have failed, since p_i received x, but p_j did not receive x in this round.

Now, consider round f. We have $x \notin V_k$ at round f, because otherwise, p_k would have transmitted x to p_j at round f (note hat p_k only failed in round $f + 1$). Therefore, some other process p_m must have transmitted x to p_k, but failed before transmitting the message to p_j. This means that p_m failed at round f.

The same reasoning applies to the subsequent rounds, down to round 1. Since $x \in V_i$ at round $f + 1$, x was not transmitted in round 1; otherwise, p_j would have received x in round 1. Thus, we conclude that there are $f + 1$ failures, one per round, which contradicts the assumed maximum of f failures. □

Lemma 2 (Validity). *The decision value is one of the proposed values from g.*

Proof. Proving this lemma is straightforward due to the assumed failure models for processes and communication channels. The algorithm for process p_i initializes its V_i set with the proposed value of p_i (line 01) and transmits proposed values in line 4. Since V_i is updated (line 06) only with values transmitted from other processes (line 04), and these values are neither corrupted by channels nor altered by processes, no values other than the proposed values can be decided. □

5.3. Liveness Proof

Lemma 3 (Termination). *Every correct process eventually decides.*

Proof. It is necessary to prove that the while loop (lines 3 to 9) terminates after it begins execution. Since the round counter is initialized to 1 (line 02) and incremented (line 08), it will eventually satisfy its termination condition ($f+1$), unless a process p_i is blocked at the wait command (line 07). This wait condition stipulates that a message of the form (r, _) must be received from every $p_j \in g$, $i \neq j$, excluding faulty processes in the $faulty\_i$ set.

Let (k, _), where $k > 0$, denote a message of round k from process p_j. Because the channels (timely or untimely) guarantee reliable message delivery and in FIFO order, and because round numbers initiate with the value 1 and are increased monotonically (line 08), messages of the form (k, _) from p_j will eventually be received by process p_i unless process p_j crashes. As process crashes are eventually detected by the failure detector, leading to the inclusion of p_j in $faulty\_i$, such a wait condition will eventually be satisfied for all p_j. □

Theorem 1. *Algorithm 1 satisfies the consensus property under the assumed model. The proof follows from the lemmas above.*

6. Impact of Synchronous Partitions on Consensus

To evaluate the impact of the number of synchronous partitions on the resilience of consensus, the algorithm presented in [Gorender and de Araújo Macêdo 2011] was simulated. Unlike the flooding-based algorithm discussed in Section 4, this algorithm employs an asymmetric strategy with round leaders for executing consensus. Experiments were conducted to measure the average consensus time under different numbers of partitions and failures.

6.1. The Leader-based Consensus

The algorithm has its structure inspired by the PAXOS algorithm designed by Leslie Lamport [Lamport 2001]. However, for its termination, the proposed algorithm does not rely on the GST property, which is characteristic of partially synchronous models [Dwork et al. 1988]; therefore, it does not depend on the stability of the environment. The algorithm can tolerate up to $n-k$ process failures, where k is the number of synchronous partitions and n is the total number of processes in the system. In typical configurations, n tends to be significantly larger than k, meaning that $n-k$ is generally greater than $n/2$. This represents an advantage compared to solutions for partially synchronous systems, which can tolerate failures of at most a minority of processes [Dwork et al. 1988].

Moreover, in this solution, similarly to the algorithm presented in Section 4, there is an upper bound on the number of rounds required for consensus termination. Assuming that the system has s processes in k synchronous partitions, the upper bound on the number of rounds for consensus termination is $s - k + 1$, which is advantageous compared to partially synchronous systems, where no such upper bound can be defined. If we consider all processes as part of synchronous partitions ($n = s$), we find that the upper bound of our algorithm approaches that of fully synchronous systems. For example, for $k = 2$ and a non-trivial synchronous model with $n > 1$ and $f = n - 2$, the two upper bounds on the number of rounds are identical: $n - 1$.

The proposed consensus algorithm utilizes a failure detector and a leader election mechanism, both designed for the given model. The perfect failure detector presented in [Gorender and de Araújo Macêdo 2011] was adapted to allow for process recovery. A process with a crash-recovery failure model may fail due to a silent stop and later recover.

Leader election is based on process identification numbers. When the current leader fails, the election mechanism selects as the new leader the next process with the smallest identification number (greater than that of the failed leader), provided that it belongs to a synchronous partition and is not in $faulty_i$. This ensures that unstable processes that have already failed will not be re-elected as leaders.

6.2. Evaluation

The leader-based consensus algorithm was implemented and evaluated in [Blagojevic and de Araújo Macêdo 2012] using the HDDSS Simulator [Freitas and de Araújo Macêdo 2014]. However, the performance data presented in this paper were not included in previous publications and have been extracted from [Blagojevic 2012]. Crash faults were injected both randomly and in a fixed sequence to evaluate their impact on the algorithm's performance. In the experiments, crashed processes did not recover. Each scenario was simulated 10 times, with results averaged and standard deviations computed, yielding a coefficient of variation below 10%, which demonstrates measurement consistency under controlled simulation conditions.

Fixed failures were simulated by forcing processes to fail sequentially, based on their identification numbers, until consensus was reached. To simulate the assumption of one correct process per partition, faults caused all processes in the first partition to fail, except for the last remaining process. The $n - k$ faults (the maximum allowed) were randomly injected into the k partitions in the random failure scenario.

Time was measured in terms of the simulator's time units. The monitoring interval for failure detection was set to 40 time units, and the maximum transmission delay (δ) was 10 units. Synchronous channels had maximum and minimum delays of 10 and 5 time units, respectively, while asynchronous channels had an average delay of 10 units.

6.2.1. Variation in the Number of Partitions

To analyze the influence of the number of partitions on consensus performance, nine scenarios were configured with 50 processes distributed in 2 to 10 partitions. Evaluations were conducted in environments with no failures, fixed failures (inserted every 20 time units), and random failures.

Given the assumption of at least one correct process per partition, the number of tolerated failures decreases as the number of partitions increases, as shown in Table 1.

Table 1. Number of Partitions and Tolerated Failures per Scenario

Scenarios	1	2	3	4	5	6	7	8	9
Partitions	2	3	4	5	6	7	8	9	10
Tolerated Failures	48	47	46	45	44	43	42	41	40

Table 2 shows the distribution of processes among partitions for each simulated scenario.

Table 2. Distribution of Processes Among Partitions

Partitions	1	2	3	4	5	6	7	8	9	10
2 partitions	25	25	-	-	-	-	-	-	-	-
3 partitions	16	16	18	-	-	-	-	-	-	-
4 partitions	12	12	12	14	-	-	-	-	-	-
5 partitions	10	10	10	10	10	-	-	-	-	-
6 partitions	8	8	8	8	8	10	-	-	-	-
7 partitions	7	7	7	7	7	7	8	-	-	-
8 partitions	6	6	6	6	6	6	6	8	-	-
9 partitions	5	5	5	5	5	5	5	5	10	-
10 partitions	5	5	5	5	5	5	5	5	5	5

6.2.2. Results

The results obtained validated the expected behavior of the algorithms. With respect to the variation in the number of partitions, it does not affect the performance of consensus in a failure-free scenario.

Performance is negatively impacted by leader failures. Given the assumption that each partition contains at least one correct process, process within the same partition will eventually assume leadership. As a result, consensus is achieved without the need to elect processes from other partitions. Successive leader failures increase the latency of consensus, with more rounds required for consensus termination.

From the graphs in Figure 2, it can be verified that, in the environment without failures, the duration of consensus remains almost constant with an increase in the number of partitions. The small variations are due to the asynchrony of channels between distinct partitions.

For fixed failure scenarios, the duration of consensus is reduced as the number of partitions increases. When the number of partitions is kept constant, increasing the number of processes leads to more faulty processes becoming leader candidates, which in turn increases the number of rounds required for consensus termination. Conversely, when the number of processes is kept constant, increasing the number of partitions reduces the

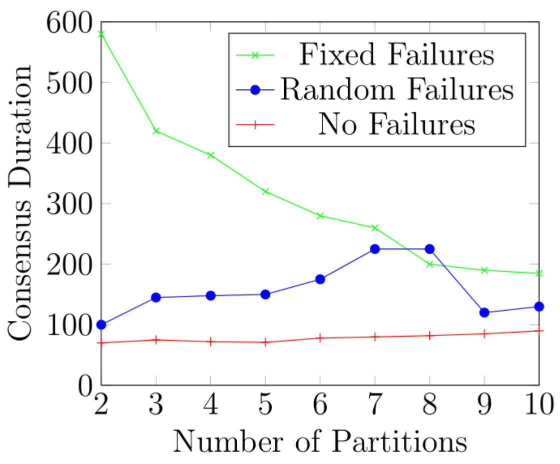

Figure 2. Impact of Synchronous Partitions on Average Time for Consensus

number of faulty processes that can become leader candidates, resulting in a decrease in the number of rounds. In scenarios with random failures, regular behavior in consensus duration cannot be observed, as depending on the moment of failure insertion, there may be more or fewer rounds during execution.

7. Final Remarks

This paper explored the Partitioned Synchronous (PaS) model, a hybrid model designed for systems partitioned into synchronous subsystems or components with asynchronous inter-partition communication [de Araújo Macêdo and Gorender 2009]. The PaS model is highly relevant for modern cyber-physical distributed systems, such as Smart Cities, Smart Agriculture, and Industrial Automation (Industry 4.0), where synchronous subsystems are connected through asynchronous networks like the Internet.

In addition to reviewing the PaS model, this paper introduced a novel flooding-based consensus algorithm, demonstrating its advantages within the PaS framework. It also presented new simulations of a previously developed leader-based consensus protocol, analyzing how different partitioning configurations impact its resilience. The analysis showed that, for fixed failure scenarios and a constant number of processes, increasing the number of partitions reduces the number of faulty processes that can become leader candidates, resulting in fewer rounds. This finding can be leveraged to optimize leader-based consensus by balancing the trade-off between consensus speed (achieved with smaller partitions) and resilience (improved with larger partitions).

Both consensus algorithms require a limited number of communication steps to reach consensus and tolerate up to $n - k$ failures.

Thanks to its ability to implement perfect failure detection, the PaS model can adapt process quorums and limit communication steps, enhancing the resilience of core fault-tolerance mechanisms such as consensus algorithms, k-mutex protocols, and group membership.

Future work will focus on applying the PaS model to further enhance applications' resilience in areas like smart agriculture and environmental management.

References

Attiya, H. and Welch, J. (1998). *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. McGraw-Hill.

Blagojevic, A. and de Araújo Macêdo, R. (2012). Simulação de um algoritmo de consenso distribuído no modelo síncrono particionado utilizando o $hddss$. In *Anais da ERBASE 2012 - WTICG 2012*. SBC.

Blagojevic, A. P. (2012). Algoritmos de consenso em sistemas distribuídos: Uma análise de desempenho em modelos parcialmente síncrono e síncrono particionado. Monografia de Conclusão, Bacharelado em Ciência da Computação. Orientador: Raimundo Macêdo. IM/DCC/UFBA, 2012.

Casimiro, A. and Veríssimo, P. (1999). Timing failure detection with a timely computing base. In *Proceedings of the Third European Research Seminar on Advances in Distributed Systems (ERSADS'99), Madeira Island, Portugal.*

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267.

Cristian, F. (1991). Reaching agreement on processor group membership in synchronous distributed systems. *Distributed Computing*, 4(4):175–187.

Cristian, F. and Fetzer, C. (1999). The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657.

da Silva, W. L. S., Ramos, M. A. D., and Macedo, R. J. d. A. (2018). Byzantine fault tolerance in the partitioned synchronous system model. In *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 106–113.

de Araújo Macêdo, R. J. and Gorender, S. (2008). Detectores perfeitos em sistemas distribuídos não síncronos. In *IX Workshop de Teste e Tolerância a Falhas (WTF 2008), Rio de Janeiro, Brazil.*

de Araújo Macêdo, R. J. and Gorender, S. (2009). Perfect failure detection in the partitioned synchronous distributed system model. In *Proceedings of the Fourth International Conference on Availability, Reliability and Security (ARES 2009)*, pages 273–280.

de Araújo Macêdo, R. J. and Gorender, S. (2012). Exploiting partitioned synchrony to implement accurate failure detectors. *International Journal of Critical Computer-Based Systems*, 3(3):168–186.

Dolev, D. and Strong, H. (1983). Authenticated algorithms for byzantine agreement. *SIAM Journal of Computing*, 12(4):656–66.

Dwork, C., Lynch, N., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323.

Fetzer, C. (2003). Perfect failure detection in timed asynchronous systems. *IEEE Transactions on Computers*, 52(2):99–112.

Fischer, M. J., Lynch, N. A., and Lynch, N. A. (1981). A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14:183–186.

Fisher, M. J., Lynch, N., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382.

Freitas, A. E. and de Araújo Macêdo, R. J. (2014). A performance evaluation tool for hybrid and dynamic distributed systems. In *ACM SIGOPS - Operating Systems Review, Vol. 4, Issue 1, pp. 11-18*.

Gorender, S. and de Araújo Macêdo, R. J. (2002). A dynamically qos adaptable consensus and failure detector. In *Proceedings of The IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2002 - Fast Abstract Track*, pages B80–B81.

Gorender, S. and de Araújo Macêdo, R. J. (2011). Um algoritmo eficiente de consenso distribuído para o modelo síncrono particionado. In *Anais do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Gorender, S., Macêdo, R., and Raynal, M. (2005). A hybrid and adaptive model for fault-tolerant distributed computing. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN2005)*, pages 412–421.

Lamport, L. (2001). Paxos made simple. *Distributed Computing Column of ACM SIGACT News*, 32(4):51–58.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401.

Larrea, M. (2002). Understanding perfect failure detectors. In *PODC '02: Proceedings of the twenty-first Annual Symposium on Principles of Distributed Computing*, pages 257–257, New York, NY, USA. ACM.

Lynch, N. A. (1996). *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc.

Macêdo, R. J. A., Gorender, S., and Cunha, P. (2005). The implementation of a qos-based adaptive distributed system model for fault tolerance. In *Anais do Simposio Brasileiro de Redes de Computadores (SBRC 2005)*, pages 827–840.

Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA. USENIX Association.

Ramos, M. A. D., Macêdo, R. J. d. A., and Blagojevic, A. (2012). An efficient mutual exclusion algorithm for redundant resources in distributed operating systems. In *2012 Brazilian Symposium on Computing System Engineering*, pages 208–213.

Raynal, M. (2018). *Fault-Tolerant Message-Passing Distributed Systems: An Algorithmic Approach*. Springer.

Santos, M. C. and Gorender, S. (2021). Um algoritmo de membership para o modelo síncrono particionado (spa) de sistemas distribuídos com particionamento forte. In *Anais do XXII Workshop de Testes e Tolerância a Falhas*, pages 43–56, Porto Alegre, RS, Brasil. SBC.

Veríssimo, P. E. (2006). Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1):66–81.

Veríssimo, P. and Casimiro, A. (2002). The timely computing base model and architecture. *IEEE Transactions on Computers*, 51(8):916–930.