Towards Byzantine Fault-resistance in Workflows: Challenges and Directions

Sofia Vaz[1,2,*], Vitor A. Cunha[1,2] and Rui L. Aguiar[1,2]

[1]*Instituto de Telecomunicações, Universidade de Aveiro, 3810-164 Aveiro, Portugal*
[2]*Departamento de Eletrónica, Telecomunicações e Informática, Universidade de Aveiro, 3810-164 Aveiro, Portugal*
E-mail: sofiateixeiravaz@ua.pt
[*]*Corresponding Author*

Received 01 September 2025; Accepted 16 February 2026

Abstract

As distributed workflows increase in complexity, they become more vulnerable to Byzantine faults, wherein components may demonstrate adversarial behavior. Addressing such faults is critical for ensuring system reliability, especially in fields such as MLOps, Blockchain, and quantum computing. Byzantine Fault Tolerance (BFT) encompasses concepts wherein systems can continue to operate correctly in the presence of faulty nodes. This paper explores core concepts in Byzantine fault-tolerant systems and discusses their relevance to modern distributed environments, such as serverless approaches applied to scientific workloads, IoT, and APIs that may power 6G and the compute continuum. Several defensive strategies and aggregation techniques are analyzed to underscore their role in alleviating the impact of Byzantine faults. Ultimately, this study highlights the continued significance of Byzantine fault tolerance in safeguarding distributed systems' resilience, security, and functionality for a new era facing increasing threats.

Keywords: Byzantine Fault Tolerance, Distributed Workflows.

Journal of Mobile Multimedia, Vol. 22_1, 97–120.
doi: 10.13052/jmm1550-4646.2214

1 Introduction

Workflows are redefining the computing paradigm by enabling automated, scalable orchestration of complex, interdependent tasks across heterogeneous systems. Their ability to integrate diverse tools – such as Machine Learning (ML) models, stream-processing data transformation tools, diverse High Performance Computing (HPC) workloads, various event-driven functions for services in the cloud and the edge – while ensuring reproducibility and fault tolerance makes them indispensable for modern scientific, industrial, and data-driven applications [3, 10]. For example, climate modeling and bioinformatics workflows leverage distributed architectures to process massive datasets efficiently, enabling previously infeasible breakthroughs [8, 9].

Workflows operate as a specialized subset of parallel computer systems, inheriting challenges such as task scheduling, load balancing, and faults [3]. However, they also introduce unique complexities, such as managing data dependencies across distributed environments (e.g., cloud and the compute continuum), ensuring temporal constraints in time-sensitive pipelines, and standardizing interoperability between domain-specific tools [8, 10]. These challenges are particularly evident in multi-facility workflows, where data movement and coordination across geographically dispersed systems introduce additional layers of complexity [9].

Because the workflow may use functions (i.e., tools), data, or infrastructure provided by third parties, ensuring system reliability is further complicated by the potential for Byzantine faults [15]. Byzantine faults occur when system components (e.g., a function) exhibit arbitrary and potentially malicious behavior, such as sending conflicting or false information to different system parts. These faults can severely disrupt the operation of a workflow by causing inconsistencies and making consensus difficult to achieve [19].

The significance of Byzantine Fault Tolerance (BFT) has grown as workflows are increasingly deployed in critical domains such as machine learning, blockchain, and cloud computing. In these environments, Byzantine faults can arise due to hardware malfunctions, software bugs, or even deliberate attacks, posing severe risks to the reliability and security of the system [17]. For instance, in ML systems, Byzantine failures can corrupt training data or model updates, leading to degraded model accuracy or even model failure [4]. Similarly, in blockchain networks, Byzantine nodes can attempt to alter transaction records or disrupt consensus, threatening the integrity of the decentralized ledger [20].

Byzantine Fault Resistant Systems (BFRSs) [7] provide mechanisms to mitigate the impact of such faults by ensuring that the system can continue to function correctly, even when some components behave unpredictably. These systems typically rely on consensus protocols that require agreement among multiple participants, redundancy, and careful information validation to withstand the presence of Byzantine nodes.

This article takes three highly relevant and emerging domains that benefit from workflow approaches – distributed machine learning/MLOps, blockchain systems, and quantum cryptography – to identify the existing challenges and potential solutions, creating a body of knowledge for future directions where Byzantine fault tolerance can be achieved in these workflows. By employing the perspective offered by BFT on these heterogeneous environments, it becomes feasible to determine, with some degree of precision, which methodologies from the respective domains are most efficient for specific workflows, depending on the task, environment, and anticipated data characteristics.

After this chapter, Section 2 gives a theoretical overview of Byzantine faults and some typical resistance methods. Then, Byzantine faults in ML (Section 3) and other areas (Section 4) are explored. Section 5 outlines the future directions for tackling these issues in workflows. Finally, some conclusions and closing remarks are drawn in Section 6.

2 Byzantine Faults

Byzantine faults, as introduced in the pioneering work of Lamport, Shostak, and Pease, first published in 1982, characterize a category of failures within distributed systems wherein components may demonstrate arbitrary and possibly malicious behaviors [15]. Contrary to conventional failure models that presume nodes either operate correctly or fail completely, Byzantine faults encompass more intricate failure modes, including malfunctioning nodes that propagate conflicting or erroneous information to various segments of the system. This unpredictable behavior can significantly compromise the reliability and consistency of distributed systems, thus making BFT a pivotal area of investigation within disciplines such as distributed computing, parallel processing, blockchain, and machine learning.

The term “Byzantine” originates from the Byzantine generals problem, which serves as a metaphorical representation of the difficulties inherent in reaching consensus among distributed agents when confronted with actors exhibiting noncompliant behavior. In this scenario, multiple generals, each

commanding a segment of an army, are required to agree on a unified strategy for the battle. However, some generals may act as traitors, attempting to create confusion by transmitting contradictory messages. The goal is for the loyal generals to achieve consensus notwithstanding the presence of these disloyal actors. This abstraction highlights the inherent challenges of establishing consensus within systems where components may operate erratically. Byzantine faults, moreover, can manifest in various forms; while a Byzantine agent typically engages in outright deceit, it may also disregard communications.

The most general type of attack is the gambler [22], in which attackers change a portion of the data on the communication, such as physical wires or network interfaces. They may also pick up data and maliciously change it. Another attack is the omniscient attack. This is more applicable to distributed ML systems, as the attacker knows the gradients sent by all workers and uses the sum of them, scaled by a large negative value, to replace some of the gradient vectors. The goal is to mislead Stochastic Gradient Descent (SGD), making it go in the opposite direction. Then, there is the Gaussian attack, in which Byzantine actors, instead of sending specific values or vectors, send random data sampled from a Gaussian distribution with large variance. These attacks tend to be weaker; however, attackers do not require any information from workers.

A system is considered fault tolerant if it possesses the capability to detect, tolerate, and recover from errors or crashes induced by faults, thus maintaining continuous operation without the need for manual intervention [5]. Concerning Byzantine fault tolerance, the objective is to ensure the system's continued correct functionality, even when a subset of its components demonstrate Byzantine behavior. To achieve this end, the system must incorporate a consensus mechanism that enables the correct nodes to reach an agreement on the system's state, notwithstanding the existence of faulty nodes.

Nevertheless, a Byzantine fault-tolerant system is constrained to having, at most, one-third of its nodes as faulty, as expressed in Definition 1. Should the proportion of faulty nodes exceed this threshold, it would render resistance unattainable. This principle is established in the foundational 1982 paper and is reaffirmed in subsequent literature.

Definition 1 (Byzantine fault-tolerance constraint) *A system of n nodes can only be Byzantine-fault tolerant if, considering f faulty nodes,*

$$n \geq 3f + 1.$$

A historic approach to fault tolerance is Practical Byzantine Fault Tolerance (PBFT), which was proposed by [7]. This is seminal work, seen as the first state-machine replication protocol that correctly survives Byzantine faults in asynchronous networks. The service is modeled as a state machine that is replicated across different nodes in a distributed system. Replicas move through a succession of configurations, which are called views. In a view, one replica is the primary, while the others are backups. The primary of a view is a replica p such that $p = v \mod |R|$, where v is the view number and R is the number of replicas. Views change when it appears the primary has failed. The algorithm has the following steps, where f is the maximum allowed Byzantine nodes the system can sustain:

1. A client sends a request to invoke a service operation to the primary.
2. The primary multicasts the request to the backups.
3. Replicas execute the request and send a reply to the client.
4. The client waits for $f + 1$ replies from different replicas with the same result; this is the result of the operation.

This algorithm is visible in Figure 1.

There are, however, two requirements, that are typical of state machine replication techniques. First, they must be deterministic, and they must start in the same state.

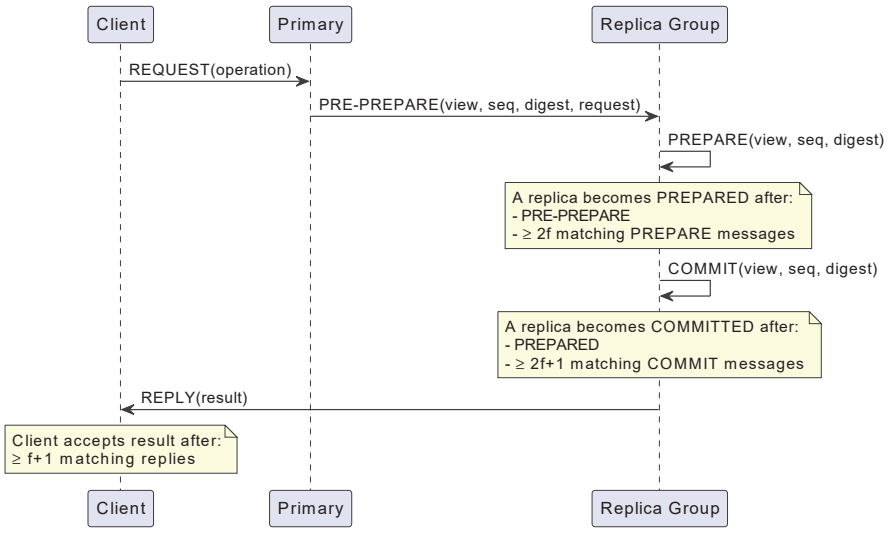

Figure 1 Detailed PBFT flow.

When the primary, p, receives a client request, m, it starts a protocol of its own. It has three phases – pre-prepare, prepare, and commit.

In the pre-prepare phase, the primary assigns a sequence number, n, to the request and multicasts a pre-prepare message, with m piggybacked, to all the backups. This message is also appended to the log. A backup will accept the pre-prepare message if it meets the following conditions: the signatures in both the request and the pre-prepare message are valid, the message belongs to the current view, it has not already accepted a pre-prepare message for the same view and sequence number that contains a different digest, and the sequence number falls between the low water mark, h, and the high water mark, H.

The pre-prepare phase is visible in Figure 2.

If a backup accepts a message, it enters the prepare phase by multicasting a prepare message to all other replicas. This message is also logged. If the backup does not accept the message, nothing happens.

A replica, including the primary, accepts prepare messages and adds them to its log, as long as their signatures are correct, their view number is the current one, and their sequence number is between h and H. It is vital to note,

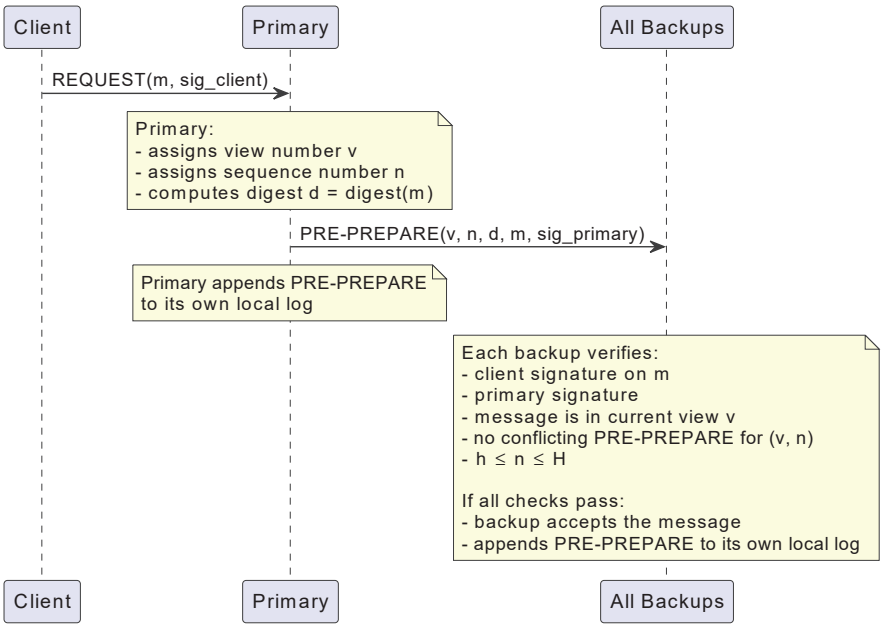

Figure 2 PBFT pre-prepare phase.

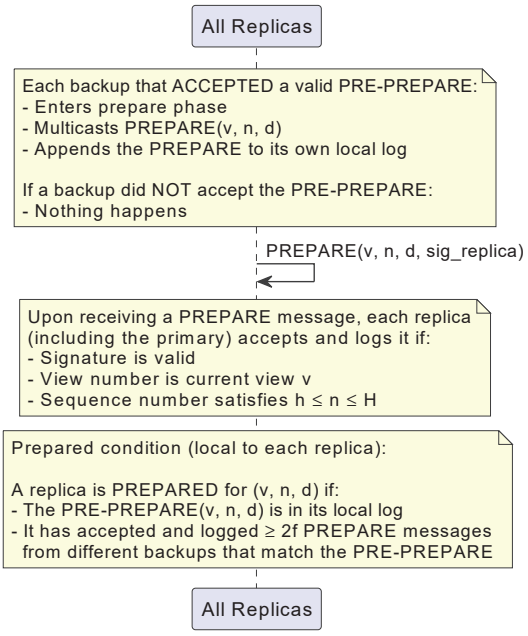

Figure 3 PBFT prepare phase.

however, that a replica is only prepared once it accepts $2f$ prepare messages from different backups that match the pre-prepare.

A diagram of the prepare phase is visible in Figure 3.

Once a replica becomes prepared, it enters the commit stage. Once again, every replica will multicast a commit message, and replicas will only accept commit messages and insert them in their logs if they are properly signed, the view number in the message is equal to the replica's current view, and the sequence number is between h and H.

A replica commits locally once it is prepared and has received $2f + 1$ commits from different replicas that match the pre-prepare for m. Once a replica commits locally, it will finally execute the operation requested by m and send its reply to the client.

A diagram of the commit phase is visible in Figure 4.

An overview of this algorithm is shown in Figure 5. In this case, C is the client, 0 is the primary, and replica 3 is faulty. As visible, in the request phase, the client simply sends a packet to the primary. Then, in the pre-prepare phase, the primary broadcasts its pre-prepare packet to every replica. In the prepare phase, every available replica will broadcast their prepare packets, as

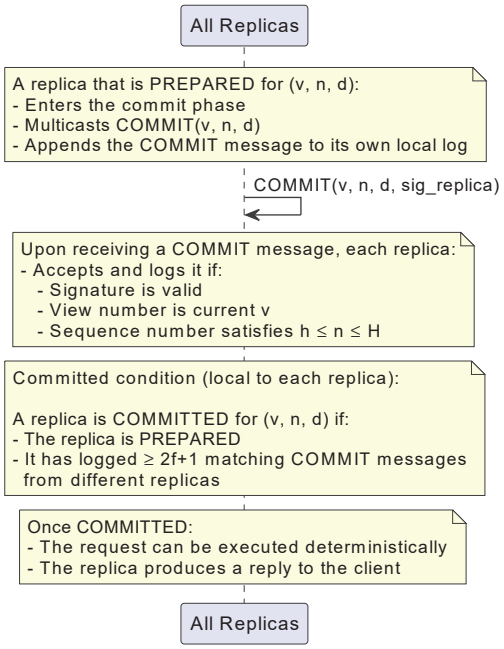

Figure 4 PBFT commit phase.

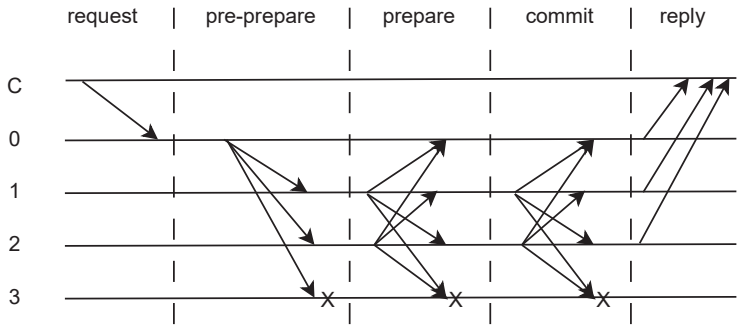

Figure 5 Example of a PBFT flow.

in the commit phase. Once all of them are locally committed, they all reply to the client.

Although PBFT is an old approach, many blockchain frameworks still use it [1].

Another vital work is DLS [12]. The DLS algorithm relies on partial synchrony, and a decision is made per round. The protocol operates by

choosing a leader, who proposes a value to all participating nodes. The nodes exchange messages during multiple rounds of communication to verify the leader's proposal and detect potential inconsistencies. Byzantine nodes may attempt to send conflicting information, but through the exchange of messages among the nodes, the protocol ensures that non-faulty nodes can detect and reject faulty proposals. The DLS algorithm guarantees that as long as fewer than one-third of the nodes are faulty, the remaining nodes will eventually agree on a common value after sufficient communication rounds, even under initially asynchronous conditions.

However, these approaches face challenges when applied to other modern distributed systems, such as distributed ML, where malicious participants may send arbitrarily harmful updates. Recent developments in Byzantine resilient aggregation techniques, such as median-based and trimmed-mean-based defenses, have been proposed to mitigate the influence of Byzantine clients in these domains.

In the median-based defense, the central server aggregates client updates by calculating the element-wise median of the values received from all clients. If we denote the update vector from client i as u_i, and assume n clients are sending updates for each model parameter j, the server computes the median for each parameter j as expressed in Equation 1.

$$Median(u_1[j], u_2[j], \ldots, u_n[j]) \tag{1}$$

where
$u_1, u_2, \ldots u_n$: update vectors
j: parameter.

This elements-wise median is computed across all n clients for each j in the model. By selecting the median instead of the mean, this approach ensures that extreme values, which might result from Byzantine behavior, do not disproportionately affect the global model. For example, if there are three clients, and for a value, they submit 3, 4 and 100, the median-based method would select the value 4, effectively ignoring the outlier.

In the trimmed-mean-based defense, the server aggregates client updates by first sorting the values for each parameter and then trimming a fixed proportion of the highest and lowest values before computing the mean. Specifically, for each parameter j, the server receives update vectors from n clients and trims a fraction α of the largest and smallest values. The remaining values are then averaged to compute the trimmed mean for each parameter. Mathematically, let $u_1[j], u_2[j], \ldots, u_n[j]$ represent the updates

for parameter j, sorted in ascending order. After trimming the top and bottom αn values, the trimmed mean is given by the expression in Equation 2

$$TrimmedMean(j) = \frac{1}{(1-2\alpha)n} \sum_{i=\alpha n+1}^{(1-\alpha)n} u_i[j] \tag{2}$$

where
n : number of clients
u_i : update vector
j : parameter
α : pre-defined trim.

Here, α represents the proportion of values to be trimmed from both ends, so $(1-2\alpha)n$ is the number of remaining values after trimming. By discarding the extreme values at both ends of the sorted list, the trimmed-mean approach reduces the influence of outliers caused by Byzantine clients, while still using more data points than the median-based method. For example, if five clients submit updates -50, 3, 4, 5, 100 for a parameter and the system trims the top and bottom 20% of the values, the updates -50 and 100 are removed, and the mean is only calculated using the apparently valid values, which are 3, 4 and 5, resulting in 4.

Both methods offer robust solutions to the Byzantine fault problem in distributed systems. The median-based defense provides strong protection against outliers when Byzantine clients form a significant fraction of the system, as long as the number of Byzantine clients is less than half of the total participants. This guarantees that the median reflects the majority of honest client contributions, ensuring the model's integrity. Any approach may discard useful information when there is natural variability in the updates from honest clients. However, the trimmed-mean defense strikes a balance by retaining more information from the honest clients while still filtering out extreme updates. It is particularly effective when the number of Byzantine clients is smaller and the updates from honest clients have some inherent variance. However, it is vital to note that these defenses are only applicable in systems in which the Byzantine fault can be triggered due to numeric input.

This is the case of federated learning and other distributed ML systems, wherein these defenses are vital for maintaining the integrity of the global model. Byzantine clients can attempt to poison the learning process by submitting corrupted gradients or model updates, which, if aggregated using a simple mean, could cause significant model degradation or even divergence. The median-based and trimmed-mean-based defenses protect against these

attacks by ensuring that the influence of any malicious or extreme updates is minimized.

3 Byzantine Faults in Machine Learning

With the growth of large-scale neural networks, it was found that training these models on a single machine posed significant challenges [5]. This led to the adoption of distributed learning, which improves the efficiency of optimization algorithms in terms of computational and communication costs while maintaining accuracy.

In distributed ML systems, however, it is necessary to coordinate multiple agents to build learning models, dealing with gradient aggregation, data partitioning and ensuring that the model converges to a global solution [18].

However, there is a chance that there may be Byzantine participants during the training phase, injecting erroneous or malicious data samples, making it harder to ensure convergence [5]. A Byzantine participant in the training process can choose arbitrary inputs, increasing the risk of inference attacks that extract sensitive model information [16].

A vulnerable distributed learning framework is that of a Parameter Server (PS) [23]. In this architecture, there are two types of nodes: server nodes and worker nodes. The server nodes maintain a global copy of the model, aggregate gradients from workers, apply the said gradients to the current version of the model, and broadcast the model to the workers. The workers, on the other hand, pull the latest model, compute the gradients from their local portion of the training data, and send the gradients back. However, it is vital to note that gradients are assumed to be calculated honestly, instead of guessed. A diagram of this framework is present in Figure 6.

But these are, of course, only assumptions. There is a risk a node could be Byzantine, and instead of calculating their gradients, is taking part in a Gaussian attack. Some works focus on this question.

For example, [6] proposes a system that will help ensure the correct gradient calculation by regularly obtaining small, clean datasets and setting them as the ground truth. The authors provide both mathematical and practical proofs of their method's efficiency.

Another approach focusing on gradients, this time on distributed implementations of SGD, is proposed by [4]. Their approach, also known as Krum, is argued to be the first provably Byzantine-resilient algorithm for distributed SGD. The paper proposes that some values should be discarded, similarly to the median-based and trimmed-mean defenses. However, no means nor

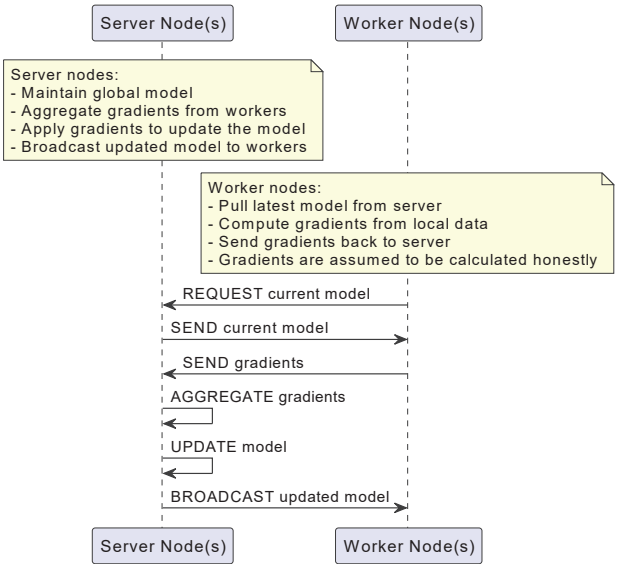

Figure 6 PS diagram.

medians are used as the authors prove that no linear combination of the vectors can tolerate a single Byzantine worker. Krum is defined in Definition 2. The value selected for Krum is the one used to calculate the new gradient.

Definition 2 (Krum) *Taking into account that the system has n workers, f of which are Byzantine, and that the system is, in fact, solvable, which means that $n \geq 3f + 1$.*

For any $i \neq j$, denote by $i \rightarrow j$ the fact that V_j belongs to the $n - f - 2$ vectors closest to V_i.

Then, each worker i has score $s(i) = \sum_{i \rightarrow j} ||V_i - V_j||^2$ where the sums run over the $n - f - 2$ closest vectors to V_i.

Then $KR(V_1, V_2, \ldots, V_n) = V_{i_*}$, where i_* refers to the worker minimizing the score, $s(i_*) \leq s(i)$ for all i.

However, Krum has some limitations. When there are no Byzantine workers, the usage of Krum slows learning, as valid vectors are discarded. However, when there are Byzantine workers, models are still able to converge, even when averaging is unable to.

The authors also propose Multi-Krum, which is an extension of Krum. In this version, distances are still calculated; however, the lowest scores of

m workers are selected and averaged, $m \in \{1, 2, \ldots, n\}$. If $m = 1$, regular Krum is applied and if $m = n$, then averaging is applied. The parameter m can be used as a trade-off between convergence speed and resilience to Byzantine workers, as a higher m makes the system less resilient.

However, [22] argues that Krum is not dimensional Byzantine resilient. They argue this from Theorem 1 and Theorem 2.

Theorem 1 *Any aggregation rule Aggr*$(\{\tilde{v}_i : i \in [n]\})$ *that outputs Aggr* $\in \{\tilde{v}_i : i \in [n]\}$ *is not dimensionally Byzantine resilient.*

Theorem 2 *Averaging is not dimensional Byzantine resilient.*

Theorem 1 makes Krum not dimensional Byzantine resilient. The theorem may seem bizarre, but the authors provide mathematical proof. It is vital to note that dimensional Byzantine resilient is also a subtype of Byzantine resilience. Theorem 2 makes Multi-Krum not dimensional Byzantine resilient. The theorem is empirically deducible.

Instead, the authors propose three aggregation methods and mathematically prove their Byzantine resilience under certain conditions. These approaches are the use of geometric median, marginal median, and mean-around-median.

The geometric median of $\{v_i : i \in [n]\}$, denoted by $GeoMed(\{v_i : i \in [n]\})$, is defined in Equation 3.

$$GeoMed(\{v_i : i \in [n]\}) = argmin_{v \in \mathbb{R}^d} \sum_{i=1}^{n} ||v - v_i|| \tag{3}$$

The marginal median aggregation is defined in Definition 3.

Definition 3 $MarMed(\{v_i : i \in [n]\})$*, where for any* $j \in [d]$*, the* j*th dimension of* μ *is* $\mu_j = median(\{(v_1)_j, \ldots, (v_n)_j\})$*,* $(v_i)_j$ *is the* j*th dimension of the vector* v_i*, median is the one dimensional median*

The mean-around-median aggregation, or $MeaMed$, also takes the same arguments. It is defined in Defition 4

Definition 4 $MeaMed(\{v_i : i \in [n]\})$*, where for any* $j \in [d]$*, the* j*th dimension of* ρ *is* $\rho_j = \frac{1}{n-q} \sum_{\mu_j \to i} (v_i)_j$*,* $mu_j \to i$ *is the indices of the top* $(n-q)$ *values in* $\{(v_1)_j, \ldots, (v_n)_j\}$ *nearest to median* μ_j*,* $(v_i)_j$ *is the* j*th dimension of the vector* v_i*.*

Regarding practical tests, it was found that the three proposed aggregation functions react similarly to Krum when there are no attacks, and when a Gaussian attack is in place. When an omniscient attack takes place, $MeaMed$ reacts as though there is no attack, and Multi-Krum, although not as good, reacts similarly. Krum converges slowly, but $GeoMed$ and $MarMed$ converge to bad solutions. However, in a gambler attack, $MeaMed$ and $MarMed$ can converge, when neither Krum nor Multi-Krum can.

Later in the same year, the same authors proposed the usage of the trimmed mean and $Phocas$, a newly proposed algorithm based on the trimmed mean defense [23]. $Phocas$, mathematically, is defined in Equation 4.

$$Phocas_b(\{u_i : i \in [m]\}) = \frac{\sum_{i=1}^{m-b} u_i / TrimmedMean(b)}{m-b} \tag{4}$$

In plain English, the result of $Phocas$ is the average of the first $m-b$ elements closest to the b-trimmed mean.

When testing $TrimmedMean$ and $Phocas$ against Krum and Multi-Krum, the results are interesting. In a Gaussian attack, all approaches converge; however, $Phocas$ reacts as if there were no Byzantine failures at all. In an omniscient attack, $Phocas$ remains the best approach and $TrimmedMean$ converges to bad solutions. Regarding both the bit-flip attack and the gambler attack, $Phocas$ and $TrimmedMean$ are able to react, while Krum is not.

Another work, this time focusing on the problem of distributed stochastic convex optimization, is [2]. The authors propose ByzantineSGD, an algorithm that theoretically is the first tight (up to logarithmic factors) sample and time complexity bounds for distributed SGD in a Byzantine setting. The authors accomplish this by using concentration bounds to obtain a new set of detection criteria for malicious machines. However, there are some problems with this work. First, it requires there to be strong concentration of the gradients, meaning that in more scattered settings this approach may not work. Another problem is the fact that, although supposedly robust, it is not applicable in a fully decentralized setting. Finally, there is the problem that the work is merely theoretical, meaning that there is no practical evaluation of their approach.

However, there is no guarantee that, in a Byzantine system, the Byzantine nodes will remain the same over time. At any point, a Byzantine node could become benign, and a benign node could become Byzantine.

DynaBRO, proposed by [11], deals with these identity changes while achieving the asymptotic convergence rate of a static setting. The authors use a Multi-Level Monte Carlo (MLMC) gradient estimation technique and integrate it with a fail-safe filter, enhancing its robustness against dynamic Byzantine strategies.

However, another problem with Byzantine defense is its computational overhead. What some authors have proposed to counter this issue is to implement Byzantine defense at the network level. Some, such as [18], propose that the layer two switch has this responsibility, meaning it should both filter and aggregate packets according to a predefined defense scheme. As a proof of concept, they implement two defense methods – median-based defense and trimmed-mean-based defense. Preliminary results show that the solution, Netshield, can offer significant performance advantages in terms of computational overhead and communication efficiency when compared to current state-of-the-art CPU-based solutions.

4 Byzantine Faults in Other Areas

However, it is vital to note that Byzantine faults are not necessarily limited to ML.

When it comes to the blockchain, there exists, of course, the risk of a Byzantine node. A common approach to dealing with Byzantine faults in blockchain is state machine replication [13].

Hyperledger Fabric[1] is an open-source blockchain framework that has been considered useful by both industry and academia. It has a BFT system in place, called SmartBFT. This protocol is based on PBFT and, according to [1], it has challenges in terms of scaling and performance. Thus, the authors propose the usage of a protocol based on the DLS approach, which has also been adapted to the blockchain. This protocol is known as BDLS, and it was proposed by [21]. The way it works is by first establishing a reliable broadcast channel, stopping communication as soon as an honest participant decides on a message. Another adaptation is that for each blockchain height h, the BDLS protocol runs from round to round until it reaches an agreement on height h. Then and only then does it move to height $h + 1$.

Returning to [1], the authors found that their approach offers a substantial performance advantage over SmartBFT. This is in part due to the change in

[1] https://www.lfdecentralizedtrust.org/projects/fabric

the consensus mechanism, which effectively handles transaction validation and block generation while ensuring BFT. In fact, performance is similar to that of Raft, which is surprising, as Raft is not a BFT solution. With smart contracts, the problem of nondeterminism becomes apparent, which in term is a problem for most state machine replication protocols.

A modular asynchronous system called Block-ND [13] has approached this complex problem. In short, Block-ND separates agreement on transaction ordering from agreement on replica state. The authors also propose a new distributed computing primitive, DO-MBA, and provide an efficient construction. Preliminary tests show that the solution incurs marginal overhead when compared to conventional BFT systems dealing with deterministic operations.

In the area of post-quantum cryptography, BFT may even bridge the gap, making aggregate signature schemes viable. At least, that is what [14] proposes. They present two BFT aggregate signatures. Their first proposition is a synchronized few-time aggregate signature scheme based on lattice assumptions which, based on the authors' knowledge, is the first to secure against quantum adversaries. The second construction is a practically and asymptotically efficient BFT aggregate signature scheme. To the best of the authors' ability, theirs is the most efficient signature scheme under the assumptions they make that is also secure against quantum adversaries. Moreover, the authors were able to reduce their mathematical problems, which attackers would have to "guess", to problems that are assumed to be hard, even for quantum computers.

5 Future Directions

Distributed workflows are very diverse, as their architectures, trust models, and operational priorities differ across application scenarios. Scientific workflows like climate simulations rely on deterministic task scheduling and high-fidelity data dependencies. At the same time, decentralized business processes (e.g., multi-party supply chains) prioritize adversarial-proof coordination among mutually untrusted entities. These structural and environmental disparities inherently shape the threat landscape. Byzantine faults in a tightly coupled scientific workflow might propagate as cascading data corruption, damaging the experiments/simulations, causing the results to be misleading/counterproductive/damning to the reputation of those asserting them as true. Whereas, for instance, in a decentralized business process, these faults could manifest as maliciously altered contractual terms or

fraudulent consensus, with consequences more directly linked to monetary losses.

Even within similar application scenarios, workflows diverge. Some employ centralized orchestrators vulnerable to single-point compromise, while others adopt decentralized choreography susceptible to omniscient attacks. Trust assumptions further fragment the landscape; workflows in controlled cloud environments may assume semi-honest participants, while those in public edge networks must guard against fully malicious actors. Such variability ensures that Byzantine faults cannot be reduced to a universal adversary model or mitigated through one-size-fits-all protocols.

The explored BFT solutions can be divided into two distinct categories – calculated BFT, and protocol-based BFT.

Calculated BFT refers to cases in which the system has a centralized server, and the calculations made with the provided values already deal with the Byzantine fault. This is the case of, for example, $Phocas$, $GeoMed$, and many other defenses typical to distributed ML systems. Transposing this concept to distributed workflows, these methods will be most practical for MLOps, and other settings in which model training occurs on the workflow. Moreover, other workflows in which distributed calculations exist will also benefit from these methods.

Protocol-based BFT, in this case, refers to when the protocols used are, in and of themselves, BFT. These methods are extremely useful when the system cannot be changed – for example, legacy workflows, or workflows in mixed-ownership situations. In these cases, the system may not be BFT, however, an added protocol makes it so. This is useful in cases such as smart contract transactions.

However, a known problem with BFT is the underlying resource consumption. Some works already take this into account and try to create methods to offload these tasks. While task offloading is feasible, distributed workflows are ultimately constrained by the finite number of available processing units. As the system reaches capacity, the ability to further offload tasks becomes limited, leading to resource saturation. Instead of offloading, then, there should be a focus on optimizing already existing BFT mechanisms.

Moreover, most BFT solutions do not take node heterogeneity into account. Considering how they may have vastly different computing power or operating systems, the resource consumption mentioned above is a bigger problem. Moreover, the solution used must be able to deal with these differences, which is more of an implementation problem.

Adaptive workflows, such as those that need to react to external events by scaling, changing the functions used, or reconfiguring existing executions, are another problem. In this case, BFT will necessarily need to be able to adapt to an arbitrary number of nodes on the fly, changing functions, and impacts of the reconfiguration.

Looking at future directions for the compute continuum and emerging technologies (e.g., 6G, Internet of Everything, Industry 4.0+), it is more productive to select a few promising workflow types. First and foremost, we must verify if BFT is even a requirement for that workflow. If the distribution happens within a self-controlled environment, with our functions or curated functions, with our data or curated party data, then, BFT should not be a concern because Byzantine faults are unlikely to be in that threat model.

For distributed scientific workflows, it is necessary to ensure that the BFT mechanisms chosen are reliable, lightweight, and energy-efficient. Because scientific workflows usually have their custom-built functions, faults can arise from data, infrastructure, or the functions supply chain (e.g., a compromised library). Scientific workflows often span hundreds, if not thousands, of machines, so ensuring a scalable BFT solution is used is also critical. Blockchain technologies for the supply chain are a possible approach to achieving BFT, ensuring the provenance of data and functions. Moreover, there are many settings and computations being done across diverse environments, and resource availability tends to vary widely. Some machines may be extremely powerful, while others may not. Thus, any BFT mechanism for the computations must also be context-aware and is an open field of research.

For the serverless Internet of Things (IoT) that may power the smart everything trend and the next industrial revolution, attacks driven by monetary considerations are now a strong consideration. In these cases, the business logic may be centralized on the controlling entity (i.e., the business owner), but scaling to different platform providers and other opportunistic machine-to-machine type communications may still be a factor. BFT is required when crossing trust boundaries (i.e., within the untrusted domains) but may be relaxed within the controlled environments (i.e., trusted domains). Establishing trust and proper validation when crossing boundaries is crucial and the most significant open research question. Outside of these particulars caused by the monetary considerations, similar solutions from scientific workflows may also be applicable here whenever distributed data processing takes place.

When it comes to APIs that power the future web and enriched mobile applications (e.g., augmented reality), such as those with crowdsourced elements and active use of the edge in the compute continuum, additional considerations must take place regarding the context. Therefore, BFT from crowdsourced elements must follow the data and processing through the entire workflow, allowing for a late resolution (maybe even post-mortem) not to disrupt lower-latency communications, but still prevent continued abuse of the system. Such balance is a tightrope that needs further research.

6 Conclusion

Byzantine fault-tolerant systems are fundamental in maintaining the integrity, consistency, and reliability of distributed workflows, especially as these workflows become more complex and integral to domains such as machine learning, blockchain, and cloud computing. Byzantine faults, where system components can exhibit arbitrary or even malicious behavior, pose significant challenges to achieving consensus in such workflows. These faults can result from hardware failures, software bugs, or deliberate attacks, and their potential to disrupt distributed processes highlights the importance of BFT mechanisms.

Seminal works such as PBFT and the DLS algorithm have established the foundation for managing Byzantine faults in distributed systems. These protocols ensure that, even with faulty or malicious nodes, non-faulty nodes can agree on a shared state, keeping the workflow progressing correctly.

As distributed workflows expand into areas such as machine learning, cloud-native applications, and decentralized networks, new challenges have emerged, requiring adaptations of BFT techniques. Byzantine fault tolerance is now used to safeguard the integrity of distributed workflows, ensuring that tasks are coordinated correctly, even in the presence of faults or adversarial behavior. These developments highlight the ongoing importance of Byzantine fault tolerance for ensuring the stability and security of complex distributed workflows.

In conclusion, Byzantine fault-tolerant systems are key to secure and reliable distributed workflows. As workflows grow in scale and complexity, the ability to resist Byzantine faults remains crucial for maintaining reliability and robustness in the face of unpredictable behavior. Continued innovation in BFT protocols will be essential to addressing evolving threats and ensuring the resilience of distributed workflows in various domains.

Acknowledgments

This work has been partly developed in the scope of the project EXIGENCE. EXIGENCE has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101139120. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Smart Networks and Services Joint Undertaking. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Ahmed Al Salih and Yongge Wang. BDLS as a Blockchain Finality Gadget: Improving Byzantine Fault Tolerance in Hyperledger Fabric. *IEEE Access*, 2024.

[2] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine Stochastic Gradient Descent. *Advances in neural information processing systems*, 31, 2018.

[3] Adam Barker and Jano van Hemert. Scientific Workflow: A Survey and Research Directions. pages 746–753. 2008.

[4] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In I Guyon, U Von Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[5] Djamila Bouhata, Hamouma Moumen, Jocelyn Ahmed Mazari, and Ahcène Bounceur. Byzantine fault tolerance in distributed machine learning: a survey. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–59, 2024.

[6] Xinyang Cao and Lifeng Lai. Distributed Gradient Descent Algorithm Robust to an Arbitrary Number of Byzantine Attackers. *IEEE Transactions on Signal Processing*, 67(22):5850–5864, nov 2019.

[7] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[8] Tainã Coleman, Henri Casanova, Loïc Pottier, Manav Kaushik, Ewa Deelman, and Rafael Ferreira da Silva. WfCommons: A framework

for enabling scientific workflow research and development. *Future Generation Computer Systems*, 128:16–27, 2022.

[9] Rafael Ferreira da Silva, Loïc Pottier, Tainã Coleman, Ewa Deelman, and Henri Casanova. WorkflowHub: Community Framework for Enabling Scientific Workflow Research and Development. In *2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pages 49–56, 2020.

[10] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, may 2009.

[11] Ron Dorfman, Naseem Yehya, and Kfir Y Levy. Dynamic Byzantine-Robust Learning: Adapting to Switching Byzantine Workers. *arXiv preprint arXiv:2402.02951*, 2024.

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

[13] Yue Huang, Huizhong Li, Yi Sun, and Sisi Duan. Byzantine Fault Tolerance with Non-Determinism, Revisited. *IEEE Transactions on Information Forensics and Security*, 2024.

[14] Quentin Kniep and Roger Wattenhofer. Byzantine Fault-Tolerant Aggregate Signatures. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 1831–1843, 2024.

[15] Leslie Lamport, Robert Shostak, and Marshall Pease. *The Byzantine Generals Problem*, page 203–226. Association for Computing Machinery, New York, NY, USA, 2019.

[16] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 739–753, 2019.

[17] Jing Qiao, Zuyuan Zhang, Sheng Yue, Yuan Yuan, Zhipeng Cai, Xiao Zhang, Ju Ren, and Dongxiao Yu. BR-DeFedRL: Byzantine-Robust Decentralized Federated Reinforcement Learning with Fast Convergence and Communication Efficiency. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, pages 141–150. IEEE, 2024.

[18] Qingqing Ren, Shuyong Zhu, Lu Lu, Zhiqiang Li, Guangyu Zhao, and Yujun Zhang. Netshield: An in-network architecture against byzantine

failures in distributed deep learning. *Computer Networks*, 237:110081, 2023.

[19] Nuria Rodríguez-Barroso, Javier Del Ser, M Victoria Luzón, and Francisco Herrera. Defense Strategy against Byzantine Attacks in Federated Machine Learning: Developments towards Explainability. In *2024 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8. IEEE, 2024.

[20] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In Jan Camenisch and Doğan Kesdoğan, editors, *Open Problems in Network Security*, pages 112–125, Cham, 2016. Springer International Publishing.

[21] Yongge Wang. Byzantine fault tolerance for distributed ledgers revisited. *Distributed Ledger Technologies: Research and Practice*, 1(1):1–26, 2022.

[22] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized byzantine-tolerant sgd. *arXiv preprint arXiv:1802.10116*, 2018.

[23] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Phocas: dimensional byzantine-resilient stochastic gradient descent, 2018.

Biographies

Sofia Vaz gained her B.Sc. in computer engineering in 2021 and her M.Sc. in cybersecurity in 2024 from the University of Aveiro. She is currently a Ph.D. student in a joint program with the Universities of Aveiro, Porto, and Minho. Additionally, she is a research fellow for EXIGENCE, with her research interests centered on security-based observation in distributed systems.

Vítor A. Cunha (Ph.D.'2022) is an Assistant Professor at the Univ. of Aveiro and a researcher at Instituto de Telecomunicações. He is currently working on sustainable networks and dynamic security mechanisms for softwarized and virtualized networks. Interests include network security, SDN, NFV, and the computing continuum.

Rui L. Aguiar received his degree in telecommunication engineering in 1990 and his Ph.D. degree in electrical engineering in 2001 from the University of Aveiro. He is currently a full professor at the University of Aveiro, responsible for the networking area, and has been previously an adjunct professor at the INI, Carnegie Mellon University. He was a visiting research scholar at Universidade Federal de Uberlândia Brazil and served as advisor to the portuguese governement on 5G policies. He is coordinating a research line nationwide in Instituto de Telecomunicações, on the area of networks and services. Over six years, he led the Technological Platform on Connected Communities, a regional cross-disciplinary industry-oriented activity on smart environments. His current research interests are centred on the implementation of advanced wireless networks and systems, with special emphasis on 5G networks and the Future Internet.