Volume 64, Issue 2, 2020

Journal of Scientific Research

Institute of Science,

Banaras Hindu University, Varanasi, India.

A Comprehensive Study on Failure Detectors of Distributed Systems

Bhavana Chaurasia[*] and Anshul Verma

Department of Computer Science, Banaras Hindu University, Varanasi. India.
chaurasia.bhawana93@gmail.com[*], anshulverma87@gmail.com

Abstract: **In distributed systems, failure detectors are used to monitor the processes and to reduce the risk of failures by detecting them before system crashes. Accuracy and completeness are the key attributes to measure the quality of failure detectors. Failure detectors are said to be unreliable because sometimes they suspect a correct process as a faulty process or they treat a faulty process as a correct process. In this paper various failure detector algorithms are discussed. A comprehensive study is presented based on properties, methodologies used, the applicability of systems, and outcomes of the failure detectors. The paper helps readers for the enhancement of knowledge about the basics of failure detectors and the different algorithms which are developed to solve the failure detection problems of distributed systems.**

Index Terms: **Asynchronous System, Consensus, Distributed Systems, Failure Detectors, Failure Detector Classes.**

I. INTRODUCTION

Reliability is a key attribute to measure the quality of any system. The reliability of a system depends on the less probability of failures. A failure detector is used to detect the failures in a system. In a distributed system, a failure detector is a module or algorithm running at every process/node to collect the operational states of other processes. The information concerning the operational status of a process given by two failure detectors may vary at dissimilar processes (Cortinas, 2011). In this situation, the reliability of the failure detector is measured with the help of two abstract properties: completeness and accuracy. Completeness is classified into strong completeness and weak completeness, similarly, accuracy is classified into strong accuracy, weak accuracy which is hard to attain therefore it is further classified into eventual strong accuracy and eventual weak accuracy. The combination of these two abstract properties like Completeness and Accuracy form a class of Failure detector which is shown in Table I (Chandra & Toueg, 1996). Each combination builds a different class of failure detectors by using two completeness and four accuracy properties. The conceptual view of unreliable failure detectors was presented in Chandra and Toueg, 1996, for the reliable distributed system. Sometimes, failure detectors suspect a correct process as a faulty process or they treat a faulty process as a correct process. Therefore, the failure detectors are said to be unreliable due to their inaccuracy in detecting faulty processes.

The conceptual view of failure detectors proposed by Chandra and Toueg (1991) is used to solve consensus and other related problems like reliable broadcast and atomic broadcast in an asynchronous distributed system. A distributed system has a set of processes coordinate with each other through message passing. Asynchronous and failures are the fundamental issues of distributed computing (Raynal, 2016).

Distributed systems are classified into two categories based on their topological arrangement, and process or event completion time. The first category consists of Fully connected (Mesh), Partially connected, Hierarchical (Tree), Ring, and Star topology-based distributed systems. The second category has Synchronous, Asynchronous, and Partially-synchronous systems, which are based on two-time attributes, the first one is the message transmission time, and another is the task execution time (Cortinas, 2011).

The primary issue in a distributed system is coordination. As every component/node in the distributed system has only a partial view of the global state of the system. That is why all the processes have to agree on some value. The problem of consensus is found to be similar to the agreement in which all the participated processes propose some values then every (correct) process has to agree on one of the proposed values (Pease, Shostak, & Lamport, 1980). The problem of consensus is solvable in the synchronous system by considering lower and upper time bounds on processes' processing speed and communication delays (Lynch, 1996). But, in an asynchronous

[*] Corresponding Author

system, consensus cannot deterministically be solved due to the lack of proper time-bound on events. In an asynchronous system, the condition of reliable failure detection cannot be fulfilled due to its inefficiency to differentiate a crashed process and a slow process (Fischer, Lynch, & Paterson, 1985). Dolev et. al. (1987) examined 32 partial synchrony systems and proved that 4 systems out of those 32 systems can solve the problem of consensus. Dwork et. al. (1988) assumed two partially synchronous systems and proved that a consensus algorithm can solve as long as $f<n/2$, where f denotes the number of faulty processes and n denotes the number of total processes, and also proposed some eventually synchronous consensus algorithms.

The previously defined failure detectors are based on eight failure detector classes and are used to solve the different types of agreement problems like Consensus and other related problems like Atomic Broadcast, k-set agreement, non-blocking atomic commitment, and uniform reliable broadcast, etc. The consensus is a model that describes a family of agreement problems (Pease et. al., 1980; Turek and Shasha, 1992; Barborak & Malek, 1993). A failure detector can detect different types of failures such as crash failure, crash recovery failure, omission failure, link failure, timing failure, and byzantine failure, and can work for synchronous, asynchronous, or partially synchronous systems. The readers can refer (Chandra and Toueg, 1992; Chandra and Toung, 1996; Aguilera, Chen and Toueg, 1997; Larrea et. al., 1999; Larrea, & Ferna´ndez, 2004; Gallet et. al., 2007; Macêdo & Gorender, 2009; Soraluze et. al., 2011; Xiaohui and Yan, 2014; Verma, et. al., 2016; Sozinov & Hammar, 2017; Jiménez, López-Presa, & Martín-Rueda, 2020) articles to get more information about the various failure detectors.

The rest of the paper is organized systematically, Section II describes the system model and the basic terminologies. In Section III, a detailed introduction of failure detectors and their properties with failure detector classes are presented. Section IV contains a detailed analysis and comparison of various types of failure detector algorithms. The final section presents the conclusion and future directions.

II. System Models

Models are the building blocks to design and find the solution for a given problem. It is a representation of a problem as well as its solution. A good model has properties like accuracy and traceability, which provides easiness for validation and evaluation of the solution (Schneider, 1993). The components are modeled in the distributed systems based on their possible behavior.

The components of distribute system such as state, trace and step are defined as follows: processes and channels are defined as state and work as an individual component, traces represent the behavior of a system which is also known as sequences of states, and the transition between the states is known as a step (Cortinas, 2011). It is very important to define the properties by which any implementation should efficiently satisfy its requirements. The properties are classified into two classes: safety and liveness (Lamport, 1977). The identification of properties of a system into these classes improves the better understanding of those properties, better specifications, subsequently possible and clear implementations, and proofs. The liveness property implies that the process or event will eventually produce a result. Similarly, the safety property implies that a wrong result will never be executed. According to the formal definitions of safety and liveness, the classification of the properties of a dependable system can be done into one or both classes (Alpern and Schneider, 1985). Many papers are there for more information on safety & liveness properties (Lamport, 1977 and 1979; Charron-Bost, Toueg, and Basu, 2000; Alpern and Schneider, 1985 and 1987; Benenson, Freiling, Holz, Kesdogan, and Penso, 2006b).

A new property is presented by (Neiger and Toueg, 1993) named as uniformity. It is quite interesting due to its nature. It is considered that a process, which holds the uniform property, is potentially correct because it does not fail (Hadzilacos and Toueg, 1994).

A. Processes

The process is stated as a computational entity. In the distributed system model, there is a set of processes $\prod= \{p1, p2, p3\ldots pn\}$, where $\prod$, n and p denote the set of processes, number of processes in a set, and a process respectively. The set of processes is finite and constant throughout the execution of the system. All processes in the system are connected with the help of pairwise bidirectional communicational channels in a fully connected topology (Cortinas, 2011).

Processes may be characterized as correct processes that never faced any failure and faulty processes that are affected by any type of failure or crash. Correct processes are represented by c and faulty processes are represented by f.

Time-driven and message-driven are the two classical approaches of event generation. In the time-driven algorithms, events generation implies the passage of time and measured by clocks or times. In message-driven algorithms, event generation only depends on message response.

B. Communication Channels/Links

An abstraction of the network is represented by communication channels or links. The processes are communicating with each other by sending messages to their corresponding links. A unidirectional communication link allows a one-way communication but a bidirectional communication link allows two-way communication by providing a pair of links. E.g., by a unidirectional link, a process p can only send messages to another process q, but by a bidirectional link, both the processes p and q can send messages to each other. The omission of messages can be happened after receiving and before delivering. Links can be differentiated based on their properties such as reliable links, unreliable links, and fair lossy links.

A reliable link is also known as a perfect link with the properties no creation, no duplicity, and no loss, that means no message is created, no message is received again, and every sent message has to be received (Basu, Charron-Bost, and Toueg, 1996).

A link with the properties no creation, finite duplication, and fair loss or fairness, is known as fair lossy link or fair-loss link. No creation means that if a process *q* receives a message *m*, this message *m* was sent by another process *p*. Finite duplication states that a message *m*, which is sent by finite times from a process *p* to process *q*, is also delivered to process *q* by the finite times. Fair loss or fairness means a message *m* was sent from a process *p* to correct process *q* by infinite times, whereas the message *m* was eventually received by *q* from *p* just once. In the crash failure model, a problem that is solved by using reliable links can also be solved by using fair lossy links (Basu et. al., 1996).

An eventually reliable link initially losses messages but after a time all the messages sent from a process *p* via this link to process *q* are eventually received. It has properties such as no creation, no duplication, and finite loss.

C. *Time and timing models*

Let us suppose, each process has a local clock which is used for passage measurement. In a distributed system, sometimes it is difficult to keep all the clocks in synchronous mode. The logical time concept was proposed (Lamport, 1976) that allows to abstract time. Every process augmented with a logical clock that permits the ordering of events at that process. A discrete global clock simply works as a functional device, events at processes are related to global time but processes cannot directly access to this clock.

A synchronous system is defined by (Hadzilacos & Toung, 1994) in which every step of a process is bounded with time, every send message is received within a given time-bound, and every clock has a known bounded deviation from real-time.

In an asynchronous system, there is no time-bound for message transmission or task execution (Chandra & Toueg, 1996). Processes transmit the messages without any time-bound therefore it is very difficult to decide whether the process is slow or crashed.

A partially synchronous system has features of both synchronous and asynchronous systems (Dolev, Dwork, and Stockmeyer, 1987). Initially, the partial synchronous system does not have any time-bound for processing messages and works as an asynchronous system. However, after a time it creates time-bounds based on the time taken by the previous events. This time is known as Global Stabilization Time (GST). After using GST, this system acts as a synchronous system therefore it is also known as an eventually synchronous system (Dwork, Lynch, & Stockmeyer, 1988). A partial synchronous timing model named as a theta model is proposed by (Widder and Schmid, 2009). This model shows that after an unidentified GST, the ratio between the shortest and largest time is bounded for message transmission.

D. *Process Failure Models*

Processes are classified into two types, correct or faulty as discussed above. The processes may be suffered from many types of failures. Crash, crash-recovery, general omission, and byzantine are some types of process failures found in a distributed system.

Crash failures were firstly discussed in the context of consensus (Lamport and Fischer, 1982; Hadzilacos and Toueg, 1994). When a process stops it's working like execution, sending or receiving messages, and when it becomes inactive then it is called that the process is suffered from the crash failure. The crash is permanent therefore there is no possibility of recovery. This model is also known as a crash-stop or crash-prone model. A fail-stop crash was discussed by (Schlichting & Schneider, 1983), it is a strong type of crash. If a process is suffered from this type of crash than all the correct processes are learned about this failure.

The crash- recovery model is an advanced version of the crash failure model in which a crashed process can be recovered. When a recovering process behaves like a new process, it forgets the pre-crash information. To overcome this problem, processes need stable storage to store the information which is accessed later.

A general omission is a type of failure, which happens at a process end due to loss of messages during transmission. Omission failures help to model communication channels that lose messages (Fich and Ruppert, 2003). This failure has two types: send-omission and receive-omission. In send-omission failure, a process executes the send-message instruction but the message is not dispatched into the channel because it is deleted in the out-going buffer of the process. Whereas, in receive-omission failure, a message is received at its destination process but never delivered to it because it is deleted in the in-coming buffer of the process. A send-omission failure model is proposed in (Hadzilacos and Toueg, 1994) in which processes suffer from send-omission and do not recover. In the general omission failure model, processes suffer from send omission as well as received omission (Perry and Toueg, 1986).

Timing failure is a type of failure which is related to contravention of time bounds in synchronous as well as partially synchronous systems (Attiya and Welch, 2004; Coulouris, Dollimore, and Kindberg, 2005).

In byzantine failure, a process makes arbitrary state transitions and sends arbitrary messages instead of assigned messages, it starts deviating from the given algorithm. This type of failure is also called an arbitrary failure or malicious and is modeled as a byzantine failure model (Lamport, Shostak, and Pease, 1982).

III. Failure Detectors

A failure detector is like an abstract module or device established at each process so that it can provide information related to the operational status of the other processes in the system. The concept of failure detector was introduced by (Chandra and Toueg, 1996), which is used to encapsulate timing assumptions when solving the problem of consensus in asynchronous systems and the list of suspected processes was given as a result. Failure detectors are said to be unreliable due to its wrong suspicion of processes, as they may suspect correct processes in place of faulty processes. Completeness and accuracy properties are used to measure the reliability of the failure detectors (Chandra and Toueg, 1996). A detailed study of failure detectors with crash failures is presented in (Nie et. al., 2011). A layered structure of the process is proposed by (Cortinas, 2011) has four layers as shown in (Figure 1). Processes are provided by the transport layer with the capability to send and receive messages through a communication channel. Failure detector layer observes the processes and provides probably unreliable information about the faulty processes to the upper layer named as a consensus layer, which uses this information to achieve agreement among correct processes. After that the set of potential applications signified by the application layer are benefitted from the consensus service.

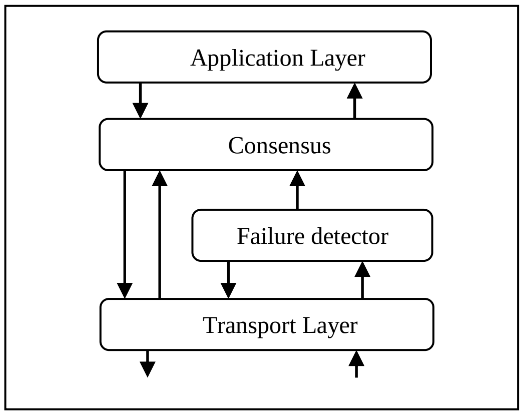

Figure 1: Layered structure of process with the failure detector (Cortinas, 2011)

Failure detectors which provide a list of suspected processes are known as suspicion based failure detector, e.g. eventually perfect failure detector symbolized as $\Diamond P$. Whereas, the failure detectors which provide a list of non-suspected processes (trusted processes) are known as trust-based failure detectors, e.g. eventual leader failure detector symbolized as Ω (Hutle, 2005).

A. *Formal Definitions*

Failure detectors were introduced by (Chandra and Toueg, 1996) for the crash failure model. The different types of failure models have been discussed above. Some basic terminologies related to failure detectors are defined as follows:

1) *Failure Patterns:* Process failure happens due to crashing. The crash is permanent therefore crashed processes do not recover. A failure pattern defined as a function $F: T \rightarrow 2^{\prod}$, where $F(t)$ represents the set of processes that have been crashed till time t. When a process is crashed, it does not recover that is $F(t) \subseteq F(t+1)$. Similarly, the set of processes is defined as $\prod = c + f$ where $\prod$, c, and f denote the set of processes, set of correct processes, and set of faulty processes respectively.

2) *Failure Detector History:* According to (Chandra and Toueg, 1996), failure detector history is defined as a function that provides the list of suspected processes, which have been suspected by a failure detector till a given time period. Mathematically, failure detector history is defined as a function $H: \prod \times T \rightarrow R$, where $R=2^{\prod}$ is a range, T is a time, $\prod$ is a set of processes and H is a failure detector history.

3) *Failure Detector:* A failure detector D is defined as a function $D: F \rightarrow D(F)$, where F and $D(F)$ denote the failure pattern and set of failure detector histories respectively. It maps the failure pattern to a set of failure detector history. The behavior of a failure detector for failure pattern was shown in each respective history. Every process in a system has its local failure detector module.

B. *Failure Detector Properties*

Failure detectors have two main properties: completeness and accuracy (Chandra and Toueg, 1996). Completeness describes the capability of a failure detector for detecting faulty processes, whereas accuracy ensures that a failure detector should not suspect a correct process as faulty. Both are the opposite of each other. It is not beneficial to use both the properties individually.

Completeness:

There are two types of completeness properties (Chandra and Toueg, 1996), which are satisfied by a failure detector D: $\forall F$, $\forall H \in D(F)$. These properties are described in (Verma, Singh, & Pattanaik, 2019) as follows:

- *Strong Completeness:* Eventually every process that crashes is permanently suspected by *every* correct process. This means every correct process will eventually and permanently suspect every faulty process.
$$\exists t \in T, \forall p \in F, \forall q \in C, \forall t' \geq t: p \in H(q, t')$$
- *Weak Completeness:* Eventually every process that crashes is permanently suspected by *some* correct process. This means every faulty process will be eventually and permanently suspected by at least one correct process.
$$\exists t \in T, \forall p \in F, \exists q \in C, \forall t' \geq t: p \in H(q, t')$$

Accuracy:

There are four variations of accuracy property (Chandra and Toueg, 1996) that a failure detector D can satisfy: $\forall F$, $\forall H \in D(F)$.

- *Strong Accuracy:* No process is suspected before it crashes. This means, no correct process can ever be suspected.
  $\forall t \in T, \forall p, q \in (\prod - F(t)): p \notin H(q, t)$
- *Weak Accuracy:* Some correct process is never suspected. This means some correct processes can be suspected.
  $\exists p \in C, \forall t \in T, \forall q \in (\prod - F(t)): p \notin H(q, t)$

Since it was very difficult to achieve both the accuracy properties so that (Chandra and Toueg, 1996) proposed an eventual accuracy property which is further classified into two more properties as follows:

- *Eventual Strong Accuracy:* There is a time after which correct processes are not suspected by any correct process. Means, eventually no false suspicion will be made.
  $\exists t \in T, \forall t' \geq t, \forall p, q \in (\prod - F(t)): p \notin H(q, t')$
- *Eventual Weak Accuracy:* There is a time after which some correct process is never suspected by any correct process. Means, eventually some false suspicion will not be made.

$\exists t \in T, \exists p \in (\prod - F(t)), \forall t' \geq t, \forall q \in (\prod - F(t)): p \notin H(q, t')$

C. Failure Detector Classes

Eight classes of failure detectors are defined with the combination of two completeness and four accuracy properties as shown in Table I (Chandra and Toueg, 1996).

Another failure detector class named "omega Ω" was proposed by (Chandra, Hadzilacos, and Toueg, 1996), it provides eventual agreement on a common and correct leader among all correct processes in a system. The Ω is a trust-based failure detector. The perfect failure detector $\mathcal{P}$ is said to be reliable because it detects only faulty processes. Whereas, the eventually perfect failure detector $\diamond\mathcal{P}$ is unreliable because it may suspect a correct process instead of a faulty process but eventually it will not make any mistake.

D. Reducibility

The "weaker than" relation is used for the comparison of failure detectors. With the help of an asynchronous algorithm, failure detector D_1 can be emulated by using another failure detector D_2, this implies D_1 is weaker than D_2 i.e. $D_1 \leq D_2$. Reducibility is achieved by using a reduction algorithm. The failure detector D_2 is reducible to D_1 using a reduction algorithm. Such that:

Equivalent relation: If $D_1 \leq D_2$ & $D_2 \leq D_1$ then $D_1 \equiv D_2$.

Strictly-weaker relation: If $D_1 \leq D_2$ & $D_2 \not\leq D_1$ then $D_1 < D_2$.

A reduction algorithm does not implement any failure detector function because it is assumed to be asynchronous. The weaker than relation is only valid for the assumed system model as an algorithm is system dependent. The output of the emulated failure detector with a distributed variable often implemented by this algorithm.

The weakest failure detector problem: The capability to comparing the failure detector, provides the opportunity to look for the weakest failure detector *D* which can solve a given problem *P*. If there is an algorithm that solves problem *P* using failure detector *D* and $\forall D'$ such that there is an algorithm that solves *P* using D' so that $D \leq D'$. In this concern, the weakest failure detector Ω was proposed by (Chandra and Toueg, 1992) for solving the problem of consensus with a majority of correct processes. Many researchers (Aguilera, Toueg, and Deianov, 1999; Delporte-Gallet et. al., 2004; Jayanti and Toueg, 2008) have used this weakest failure detector for solving various problems.

E. Implementation Approaches

Failure detectors are like abstract modules or devices therefore there is no specific implementation approach. The implementation of a failure detector does not compete until it meets the properties of the expected class. Failure detectors monitor the other processes and provide their operational status. *Polling* and *Heartbeat* are the two main approaches to implement the monitoring in the failure detectors.

The *polling* approach is a query/reply based approach used to monitor the processes (Larrea, Arévalo, and Fernández, 1999; Larrea, Fernández, and Arévalo, 2004). In this technique, when a process *p* tracks another process *q*, process *p* sends a query message to *q* and wait until it gets a reply to this message from *q*. *p* will suspect *q* if *p* does not get a reply after a given time.

In the *heartbeat* approach, every monitoring process *p* receives periodic heartbeat messages from every monitored process *q* as a notification that it is still alive. Monitoring process *p* will suspect *q* if *p* does not receive a heartbeat message from *q* after a given time.

By observing both the approaches, the waiting time depends on the timing of the system model. In synchronous systems time is bounded, and in partially synchronous systems time is adjusted as per need.

Along with the monitoring approaches, different communication patterns are also followed for the implementation of failure detectors. Efficiency in communication is an important issue, for example, an all-to-all communication pattern has the quality of responsiveness but inefficient in terms of communication. Similarly, a higher degree of communication efficiency would be achieved by a linear communication pattern.

F. Solving Consensus with Failure Detectors

(Chandra and Toueg, 1996) used the weakest class of failure detectors $\Diamond S$ to solve consensus with the majority of correct

Table I. Eight Classes of Failure Detectors

Completeness	**Accuracy**			
	Strong	Weak	Eventual Strong	Eventual Weak
Strong	Perfect $\mathcal{P}$	Strong $\mathcal{S}$	Eventually Perfect $\diamond\mathcal{P}$	Eventually Strong $\diamond\mathcal{S}$
Weak	$\mathcal{Q}$	Weak $\mathcal{W}$	$\diamond\mathcal{Q}$	Eventually Weak $\diamond\mathcal{W}$

processes. They developed a centralized consensus algorithm by using $\Diamond\mathcal{S}$ failure detector and round-based (rotating coordinator) approach. (Hurfin and Raynal, 1999) and (Schiper, 1997) also used this approach to develop algorithms. A comparison of these three algorithms is presented in (Guerraoui et al., 2000).

IV. Existing Failure Detector Algorithms

Failure detector algorithms that belong to different classes for asynchronous, partially synchronous and synchronous systems are discussed in the literature. These algorithms are mostly designed for the all-to-all process communication model and can handle the majority of failures. Polling or heartbeat approaches are adapted to monitor the process. Failure detectors provide a list of suspected processes or trusted processes by using these approaches. Thus, the analysis of failure detector algorithms depends on their class, monitoring approach, failure types, and outcomes.

To solve the consensus problem in asynchronous systems, unreliable failure detectors were proposed by (Chandra & Toueg, 1996) which identifies only crash failures. The classification of unreliable failure detectors was based on eight classes in terms of two abstract properties: completeness and accuracy. A suspicion based unreliable failure detector for asynchronous systems with the majority of crash failures is discussed in this paper.

Another type of failure detector was introduced by (Aguilera, Chen, & Toueg, 1997) named Heartbeat failure detector which depends on a non-suspicion based algorithm that accomplishes a reliable communication with quiescent algorithms. It handles crash failures as well as link failures and offers a list of suspected processes in the matrix form representing both correct and faulty processes as an output. The failure detector was implemented for asynchronous systems without using the timeout concept. Problems like a consensus, k-set agreement, atomic broadcast, and atomic commitment can be solved with the help of the heartbeat failure detector.

(Chandra & Toueg, 1996) proposed a failure detector algorithm that follows an all-to-all process communication pattern therefore several messages are exchanged periodically. (Larrea, Fernández, & Arévalo, 2004) introduced an advancement of this failure detector algorithm which detects crash failures in a partially synchronous system by using the polling concept. In this algorithm, the processes are arranged as a logical ring, and they exchange a linear number of messages in a periodical manner.

(Chandra and Toueg, 1992) introduced the weakest failure detector Ω which solved consensus in an asynchronous system with a majority of correct processes. Weak and eventually weak failure detectors that are used to solve consensus can also solve the atomic broadcast problem in asynchronous systems. A modified version of this failure detector was introduced by (Gallet et. al., 2007) which is the weakest failure detector to solve the consensus problem in an asynchronous system model. Failure detectors and problem specifications are two different transformations for algorithms. This algorithm is used to solve crash-stop and permanent omission failures by assuming that the majority of processes are correct.

Partitioned synchronous systems are weaker than synchronous systems. A suspicion based partially perfect failure detector for Partitioned synchronous systems was proposed by (Macêdo & Gorender, 2009). It uses a polling concept for monitoring the processes. A failure detector of the Perfect P class was developed with properties like strong partition synchrony and timeliness oracle for this type of system.

A communication efficient failure detector of eventual Perfect $\Diamond P$ class is proposed by (Soraluze et. al., 2011) that deals with crash failures and omission failures. The heartbeat approach is used to monitor the process failure in a partially synchronous system which follows the all-to-all process connected network. This algorithm follows only a linear number of links for message transmission to achieve communication efficiency. The results are shown in the form of a matrix with both correct and faulty processes.

(Arevalo, Anta, Imbs, Jimenez, & Raynal, 2015) presented a new failure detector classes $H\Omega$ and $H\Sigma$ which are the counterpart of weakest failure detector classes Ω and Σ for homonymous distributed systems. Two trust-based failure detector algorithms were proposed to resolve the problem of consensus in homonymous asynchronous systems with the assumption of the majority of correct processes. The polling approach is used for monitoring the processes and to detect crash failures.

Based on Chandra and Toueg (1996) unreliable failure detector, (Larrea and Ferna´ndez, 1999) proposed a family of distributed algorithms and implemented four unreliable failure detectors in partially synchronous systems. It was proven that these algorithms are efficient in comparison with the weakest failure detector algorithm proposed by Chandra and Toueg.

In (Larrea and Ferna´ndez, 2004), the implementation of different classes of failure detectors for partially synchronous systems was studied. They implemented the four classes of unreliable failure detectors with eventual accuracy (named $\Diamond P$, $\Diamond Q$, $\Diamond S$, *and* $\Diamond W$). These distributed algorithms arrange the processes in a logical ring.

(Most´efaoui and Raynal, 1999) proposed a quorum-based, generic dimensional, and simple consensus protocol. The basic algorithmic structures and principles of the protocol permit early decision and use messages shorter than previous solutions. This protocol works with any failure detector belongs to the class $\mathcal{S}$ (provided that $f \leq n - 1$) or to the class $\Diamond\mathcal{S}$ (provided that $f < n/2$). They proved that this protocol solves the consensus problem with the help of unreliable failure detectors when at most *f* process may crash out of *n* processes.

The previous failure detection algorithms which are used in asynchronous and partially synchronous systems are not working efficiently in synchronous systems because they generate more computation and communication overhead. (Verma, & Pattanaik, 2016) proposed a suspicion based failure detector of Perfect *P* class which has strong completeness and strong accuracy properties. This failure detector is developed for synchronous environments and real-time hierarchical distributed systems (Verma, Pattanaik, & Goel, 2014; Verma, & Pattanaik, 2014; Verma, & Pattanaik, 2015a; Verma, & Pattanaik, 2015b). This failure detector algorithm detects Crash failures, crash recovery failures, omission failures, link failures, and timing failures through polling-based health monitoring methodology that reduces the number of messages flooded in the network. Each process maintains the operational information of all child processes so that they have a list of faulty processes. For hierarchical distributed systems, a suspicion-based failure detector of Strong *S* class proposed in (Verma, Singh, & Pattanaik, 2019) working in time-synchronous environments. The algorithm of Strong *S* class has properties like strong completeness and weak accuracy and detects permanent crash failures, omission failures, link failures, and timing failures.

The Modal failure detector star, denoted by M*, was proposed by Park et. al. (2013) to solve the problem of a non-blocking atomic commitment in a message-passing asynchronous environment. This failure detector belongs to a new type of failure detector class with the properties' strong completeness, strong accuracy, and modal accuracy. This failure detector class is weaker than Perfect $\mathcal{P}$ class and stronger than Eventually Perfect $\diamond\mathcal{P}$ class. Initially, this failure detector suspects every process, but when it found the process is correct then it does not suspect it again before the crash. It is effective and efficient than the failure detector of class $\mathcal{P}$.

(Xiaohui & Yan, 2014) proposed an adaptive failure detector algorithm (*A-FD*) to increase the adaptive capacity of failure detectors through which their detecting quality adjusts dynamically according to the variations of the network. This model implements a heartbeat detection strategy based on PULL mode. It is proven by the experiments that this algorithm can reduce average detection time and the effect of the network delay to some extent. The main purpose of this algorithm is to increase the fault tolerance capacity of a distributed system with the help of adaptive failure detectors.

A Two-window failure detector (2W-FD) was introduced by (Tomsic, Sens, Garcia, Arantes, & Sopena, 2015). It is an adaptive algorithm that provides Quality of Service in terms of speed and accuracy. Owing to its adaptive nature, it can respond to sudden changes in network scenarios as LAN or WAN scenarios. It uses two sliding windows of past receiving messages, which are different in sizes and store recent heartbeat history. The large window allows the failure detector to estimate on a stable period or unstable periods. The small window allows the failure detector to quickly respond to those abrupt condition changes in the network. The large window increases the accuracy and a small window increases the speed of failure detector. The failure detector shows the best performance in both stable and unstable network conditions.

An adaptive binary machine learning-based failure detector (MLFD) was proposed by (Sozinov, & Hammar, 2017). It is based on an online linear regression model and follows Monte Carlo simulations of a distributed system. The network and clock synchronization increase the difficulty of failure detection. This model considers failure detection as a learning problem where round trip times (RTTs) are predicted using incremental learning on data from recorded heartbeats. It monitors every heartbeat timeout δ and also sends a liveness request to every process. To detect failures, it works with a sliding window of RTTs and a machine learning model that is updated incrementally. It is suitable for partially synchronous systems and adaptable in various network conditions.

An Omega failure detector was proposed by (Jiménez et. al., 2020) to solve consensus in the asynchronous system. It deals with degenerative byzantine failure with a reliable new broadcast primitive called RFLOB. It gives the optimal result for failure detection because it is applied when $f < n/3$. It follows unknown membership and minimum connectivity. At the initial stage, every process does not know the identity of all the processes in the system. The algorithm referred to as failure detector Ω (Chandra & Toueg, 1996), with degenerative byzantine failures and RFLOB (Reliable FIFO and Local Order Broadcast) broadcast primitive to communicate among the processes through messages. This RFLOB confirms guaranteed reliability and reduces the complexity in design and understanding of the algorithm.

We provide an analysis of various failure detectors based on many properties. Table II shows the comprehensive analysis of some failure detector algorithms based on their specific properties, the capabilities to solve the type of problems, applicabilities on system modes, and methodologies used.

Conclusion

In this paper, the fundamental of failure detectors is discussed and a detailed analysis of various failure detector algorithms is presented. Various suspicion-based and trust-based failure detector algorithms are discussed. Some of the failure detectors can detect only a single type failure and others can detect more than one failure but only a few can able to solve all types of failures. This study helps to naïve researchers to review all the algorithms systematically.

Developing failure detectors for the systems with various topologies (other than all-to-all connected networks) and dealing with all types of failures with high accuracy may be a good research direction for the researchers.

Table II. A comparative view of failure detector algorithms

Author	Name of Failure Detector (FD)	Applicable System Mode	Problems	Results	Properties	Methodology used
Chandra et. al., 1992	Weakest Failure Detector $\Diamond\mathcal{W}$ (suspicion based)	Asynchronous systems	Detect failure and solve Consensus	Provides a list of suspected processes with failure pattern	Weak Completeness & Eventually Weak Accuracy	To solve consensus when (*n>2f*), connected with a reliable communication channel, Reduction algorithm.
Chandra et. al., 1996	Unreliable failure detectors (suspicion based)	Asynchronous systems	Crash failure, Solve consensus as well as atomic broadcast	Provides suspected list of processes	Abstract properties completeness and accuracy,	Reliable Broadcast communication channel, Heartbeat monitoring approach, Reduction algorithm.
Aguilera et. al., 1997	Heartbeat FD (Non- suspicion based)	Asynchronous systems	Crash failures & link failures, consensus, k-set agreement, atomic broadcast & atomic commitment problem	Matrix of correct & faulty processes	HB-Completeness & HB-Accuracy	Reliable communication with the quiescent algorithm. Periodically send heartbeat without timeout
Larrea et. al., 2004	Enhancement algorithm for unreliable failure detector (suspicion based)	Partially synchronous systems	Crash failures, Consensus	Detects the crashed processes	Abstract properties completeness and accuracy,	Uses polling concept, & also uses a logical ring arrangement of processes and periodically exchanges at a most linear number of messages
Gallet et. al., 2007	Omega (Ω) failure detector (suspicion based)	Asynchronous systems	Consensus with Crash-stop failures & permanent omission failures	Provides suspected list of processes	Weak Completeness & Eventually Weak Accuracy	Use transformation for algorithms, enhancement of weakest failure detector Ω
Soraluze et. al., 2011	Communication efficient eventually perfect $\diamond\mathcal{P}$ Failure Detector	Partially synchronous systems	Omission failure, Crash failure while receiving messages	Matrix of correct & faulty processes	Strong completeness & eventually strong accuracy	Use the heartbeat approach for failure monitoring, the all-to-all process connected network
Arevalo et. al., 2015	Failure detector (called HΩ, $\Diamond H\overline{P}$ and HΣ) (Trust-based)	Homonymous distributed systems	Consensus, Crash failures	Majority of correct processes	Classes HΩ and HΣ	Polling approach for failure monitoring, and initially unknown membership
Park et. al., 2013	Modal failure detector star represented by M^* (suspicion based)	Crash-prone asynchronous system	Non-blocking atomic commitment problem	Provides the information of correct processes	Strong completeness, strong accuracy & modal accuracy	Heartbeat approach is used for monitoring, assume a majority of correct processes
Xiaohui et. al., 2014	Adaptive failure detector A-FD	Distributed system	Failure detection & fault tolerance	Decrease average detection time & impact of network delay	Strong completeness & eventually strong accuracy	Adaptive-algorithm heartbeat detection strategy based on PULL mode, measure QoS of failure detectors
Tomsic et. al., 2015	Two-Window failure detector	Asynchronous Systems	Unstable network conditions, time-constraints problem	Fast and accurate information about the faulty processes	Provides Quality of Service in the form of speed and accuracy	Uses two sliding windows of past receiving messages, that allows failure detector to enhance the estimation power as well as quick responsiveness

Verma et. al., 2016	Failure Detector of Perfect P Class (suspicion based)	Real-time hierarchical synchronous distributed systems	Crash failures, crash recovery failures, omission failures, link failures, and timing failures	Gives the information of all faulty processes to the root process	Strong Completeness & Strong Accuracy	Failure detector algorithm, polling-based health monitoring methodology is used for the reduction of messages in the network
Sozinov et. al., 2017	MFLD (Machine Learning failure Detector)	Partially synchronous systems	Failure detection and also consider heartbeat timeouts	Record RTTs to adapt changing network conditions and record timeout to detect failures accurately	Strong Completeness & probabilistic eventually strong accuracy	Based on an online linear regression model, Round trip time of prediction, based on Monte Carlo Simulations of a distributed system, heartbeat monitoring approach
Jiménez et. al., 2020	Omega Failure Detector	Asynchronous systems	Consensus, degenerative byzantine failures,	Consensus can be solved totally and messages are sent in the same order	Failure detector class Ω with correctness and accuracy	Uses minimum connectivity, unknown membership & RFLOB (Reliable FIFO and Local Order Broadcast) broadcast communication primitive

REFERENCES

Aguilera, M. K., Chen, W., & Toueg, S. (1997, September). Heartbeat: A timeout-free failure detector for quiescent reliable communication. In International Workshop on Distributed Algorithms (pp. 126-140). Springer, Berlin, Heidelberg.

Aguilera, M. K., Chen, W., & Toueg, S. (1999). Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. Theoretical Computer Science, 220(1), 3-30.

Alpern, B., & Schneider, F. B. (1985). Defining liveness. Information processing letters, 21(4), 181-185.

Alpern, B., & Schneider, F. B. (1987). Recognizing safety and liveness. Distributed computing, 2(3), 117-126.

Arévalo, S., Anta, A. F., Imbs, D., Jiménez, E., & Raynal, M. (2015). Failure detectors in homonymous distributed systems (with an application to consensus). Journal of Parallel and Distributed Computing, 83, 83-95.

Attiya, H., & Welch, J. (2004). Distributed computing: fundamentals, simulations, and advanced topics (Vol. 19). John Wiley & Sons.

Barborak, M., Dahbura, A., & Malek, M. (1993). The consensus problem in fault-tolerant computing. ACM Computing Surveys (CSur), 25(2), 171-220.

Basu, A., Charron-Bost, B., & Toueg, S. (1996, October). Simulating reliable links with unreliable links in the presence of process crashes. In International Workshop on Distributed Algorithms (pp. 105-122). Springer, Berlin, Heidelberg.

Beneson, Z., Freiling, F. C., Holz, T., Kesdogan, D., & Draque Penso, L. (2006). Safety, liveness, and information flow: Dependability revisited. In ARCS'06, 19th International Conference on Architecture of Computing Systems. Gesellschaft für Informatik eV.

Chandra, T. D., & Toueg, S. (1991, July). Unreliable failure detectors for asynchronous systems (preliminary version). In Proceedings of the tenth annual ACM symposium on Principles of distributed computing (pp. 325-340).

Chandra, T. D., & Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. Journal of the ACM (JACM), 43(2), 225-267.

Chandra, T. D., Hadzilacos, V., & Toueg, S. (1992). The weakest failure detector for solving consensus. Proceedings of the11th Annual ACM Symposium on Principles of Distributed Computing Vancouver, BC, Canada (pp. 147-158). doi:10.1145/135419.135451

Charron-Bost, B., Toueg, S., & Basu, A. (2000, August). Revisiting safety and liveness in the context of failures. In International Conference on Concurrency Theory (pp. 552-565). Springer, Berlin, Heidelberg.

Cortiñas, R., (2011, January). Failure detectors and communication efficiency in the crash and general omission failure models. Ph.D. thesis, University of Basque Country, UPV/EHU.ISBN: 978-84-694-5838-9

Coulouris, G. F., Dollimore, J., & Kindberg, T. (2005). Distributed systems: concepts and design. pearson education.

de Araújo Macêdo, R. J., & Gorender, S. (2009, March). Perfect failure detection in the partitioned synchronous distributed system model. In 2009 International Conference on Availability, Reliability and Security (pp. 273-280). IEEE.

Delporte-Gallet, C., Fauconnier, H., Freiling, F. C., Penso, L. D., & Tielmann, A. (2007, September). From crash-stop to permanent omission: Automatic transformation and weakest failure detectors. In International Symposium on Distributed Computing (pp. 165-178). Springer, Berlin, Heidelberg.

Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Hadzilacos, V., Kouznetsov, P., & Toueg, S. (2004, July). The weakest failure detectors to solve certain fundamental problems in distributed computing. In Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing (pp. 338-346).

Dolev, D., Dwork, C., & Stockmeyer, L. (1987). On the minimal synchronism needed for distributed consensus. Journal of the ACM (JACM), 34(1), 77-97.

Dwork, C., Lynch, N., & Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. Journal of the ACM (JACM), 35(2), 288-323.

Fich, F., & Ruppert, E. (2003). Hundreds of impossibility results for distributed computing. Distributed computing, 16(2-3), 121-163.

Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM), 32(2), 374-382.

Guerraoui, R., Hurfinn, M., Mostéfaoui, A., Oliveira, R., Raynal, M., & Schiper, A. (2000). Consensus in asynchronous distributed systems: A concise guided tour. In Advances in Distributed Systems (pp. 33-47). Springer, Berlin, Heidelberg.

Hadzilacos, V., & Toueg, S. (1994). A modular approach to fault-tolerant broadcasts and related problems. Cornell University.

Hurfin, M., & Raynal, M. (1999). A simple and fast asynchronous consensus protocol based on a weak failure detector. Distributed Computing, 12(4), 209-223.

Hutle, M. (2005). Failure detection in sparse networks. na.

Jayanti, P., & Toueg, S. (2008, August). Every problem has a weakest failure detector. In Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing (pp. 75-84).

Jiménez, E., López-Presa, J. L., & Martín-Rueda, J. (2020). Consensus using omega in asynchronous systems with unknown membership and degenerative Byzantine failures. Journal of Computer and System Sciences, 107, 54-71.

Lamport, L. (1977). Proving the correctness of multiprocess programs. IEEE transactions on software engineering, (2), 125-143.

Lamport, L. (1979). A new approach to proving the correctness of multiprocess programs. ACM Transactions on Programming Languages and Systems (TOPLAS), 1(1), 84-97.

Lamport, L., & Fischer, M. (1982). Byzantine generals and transaction commit protocols (Vol. 66). Technical Report 62, SRI International.

Lamport, L., & Time, C. (1976). The Ordering of Events in a Distributed System. Communications of the ACM, 21(7), 558.

Larrea, M., Arévalo, S., & Fernndez, A. (1999, September). Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In International Symposium on Distributed Computing (pp. 34-49). Springer, Berlin, Heidelberg.

Larrea, M., Fernández, A., & Arévalo, S. (2004). On the implementation of unreliable failure detectors in partially synchronous systems. IEEE Transactions on Computers, 53(7), 815-828.

Lynch, N. A. (1996). Distributed Algorithms, Morgan Kaufmann Publishers. Inc, 1, 997.

Mostéfaoui, A., & Raynal, M. (1999, September). Solving consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach. In International Symposium on Distributed Computing (pp. 49-63). Springer, Berlin, Heidelberg.

Neiger, G., & Toueg, S. (1993). Simulating synchronized clocks and common knowledge in distributed systems. Journal of the ACM (JACM), 40(2), 334-367.

Nie, C., & Leung, H. (2011). A survey of combinatorial testing. ACM Computing Surveys (CSUR), 43(2), 1-29.

Park, S. H., Lee, J. Y., & Yu, S. C. (2013, April). Non-blocking atomic commitment algorithm in asynchronous distributed systems with unreliable failure detectors. In 2013 10th International Conference on Information Technology: New Generations (pp. 33-38). IEEE.

Pease, M., Shostak, R., & Lamport, L. (1980). Reaching agreement in the presence of faults. Journal of the ACM (JACM), 27(2), 228-234.

Perry, K. J., & Toueg, S. (1986). Distributed agreement in the presence of processor and communication faults. IEEE Transactions on Software Engineering, (3), 477-482.

Raynal, M. (2016, June). A look at basics of distributed computing. In 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS) (pp. 1-11). IEEE.

Schiper, A. (1997). Early consensus in an asynchronous system with a weak failure detector. Distributed Computing, 10(3), 149-157.

Schlichting, R. D., & Schneider, F. B. (1983). Fail-stop processors: an approach to designing fault-tolerant computing systems. ACM Transactions on Computer Systems (TOCS), 1(3), 222-238.

Schneider, F. B. (1993). What good are models and what models are good. Distributed systems, 2, 17-26.

Soraluze, I., Cortiñas, R., Lafuente, A., Larrea, M., & Freiling, F. (2011). Communication-efficient failure detection and consensus in omission environments. Information Processing Letters, 111(6), 262-268.

Sozinov, K., & Hammar, K. (2017). Machine Learning for Failure Detection in Distributed Systems.(Project Report)

Tomsic, A., Sens, P., Garcia, J., Arantes, L., & Sopena, J. (2015, May). 2w-fd: A failure detector algorithm with qos. In 2015 IEEE International Parallel and Distributed Processing Symposium (pp. 885-893). IEEE.

Turek, J., & Shasha, D. (1992). The many faces of consensus in distributed systems. Computer, 25(6), 8-17.

Verma, A., Pattanaik, K. K., & Goel, P. P. (2014, April). Mobile agent based CBTC system with moving block signalling for Indian Railways. In Proceedings of the 2nd international conference on railway technology: Research, development and maintenance, Ajaccio, Corsica, paper (Vol. 278, pp. 8-11).

Verma, A., & Pattanaik, K. K. (2014). Mobile agent based train control system for mitigating meet conflict at turnout. Procedia Computer Science, 32, 317-324.

Verma, A., & Pattanaik, K. K. (2015a). Multi-Agent Communication Based Train Control System for Indian Railways: The Structural Design. JSW, 10(3), 250-259.

Verma, A., & Pattanaik, K. K. (2015b). Multi-agent communication-based train control system for Indian railways: the behavioural analysis. Journal of Modern Transportation, 23(4), 272-286.

Verma, A., & Pattanaik, K. K. (2016). Failure Detector of Perfect P Class for Synchronous Hierarchical Distributed Systems. International Journal of Distributed Systems and Technologies (IJDST), 7(2), 57-74.

Verma, A., Singh, M., & Pattanaik, K. K. (2019). Failure Detectors of Strong S and Perfect P Classes for Time Synchronous Hierarchical Distributed Systems. In Applying Integration Techniques and Methods in Distributed Systems and Technologies (pp. 246-280). IGI Global.

Widder, J., & Schmid, U. (2009). The theta-model: achieving synchrony without clocks. Distributed Computing, 22(1), 29-47.

Xiaohui, W., & Yan, Z. (2014, December). Adaptive failure detector A-FD. In 2014 IEEE 7th Joint International Information Technology and Artificial Intelligence Conference (pp. 455-458). IEEE.

***