System-Level Fault Diagnosis

Tamás Bartha

30 June, 1996.

0.1 Introduction

0.1.1 System-level diagnosis and fault tolerance

It has long been the goal of system designers to connect independent computing resources together to create a multiprocessing system with greater computing power and availability than any of its parts. Unfortunately, the opposite can happen if faulty resources are allowed to corrupt the system. In the area of fault-tolerant computing, the diagnosis problem is to form a common view among the fault-free members about the fault state of system resources in order to maintain the performance and integrity of the system. Such an a common view or *agreement* may be also made on the configuration of the system, the synchronization of its clocks, the contents of its communications, or any other value requiring global consistency. The proposed approach is to diagnose and/or contain faults at the system level which will facilitate the consensus process. That is, this paper looks at general techniques for detecting and locating failed system components. Work in the area has increased with the appearence of distributed systems that range from small, local-area networks to large, real-time, fault-tolerant systems. Consensus solutions give a convenient and, sometimes, vital picture of the condition of the system.

0.1.2 The consensus problem in system-level diagnosis

The simple idea of consensus is to share information among a set of processing elements (units), preferably in a fault-tolerant manner. That is, the fault-free units should be able to consistently agree on and produce correct results despite the arbitrary actions of the faulty units in the system. This problem is at the core of protocols handling synchronization, reliable communication, resource allocation, task scheduling, reconfiguration, replicated file systems, sensor reading, and other functions.

Figure 1 shows a general layered approach to fault management in which higher layers are dependent on lower layers to produce a fault-tolerant system from a basic system consisting of a group of processors connected by some unreliable communication network. The synchronization layer uses time to allow processors to recognize untimely messages, to detect faulty processors, and to order timely messages in implementing reliable communications. Reliable communications let fault-free processors pass fault-free messages that are used to either diagnose or mask faulty processors in order to agree on a correct sequence of computations and a correct result. Finally, the ability to agree on a diagnosis allows the fault-free processors to consistently reconfigure after a fault.

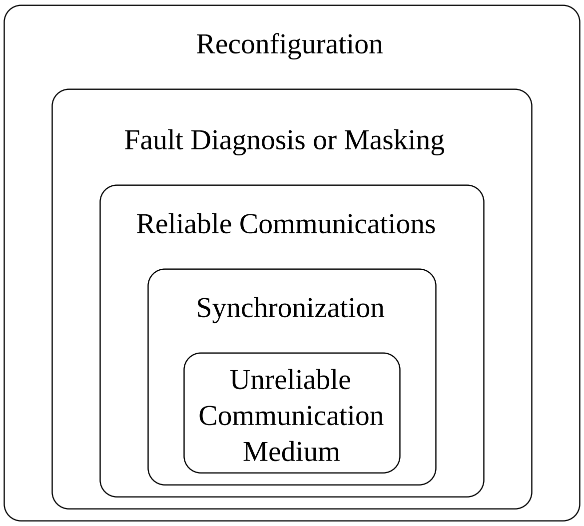

Figure 1: Consensus problems in fault management

Note, that each of these layers is a separate consensus problem. First, the synchronization level maintains a global timepiece—which is simply a consensus of all the fault-free units on a particular time value and a alteration rate of that value. Second, using the results of the synchronization layer, a reliable communication is one processor forming a consensus with another processor on some set of information and the order of transmission of that set. Third, using the reliable communications a consensus on the diagnosis of the system may be formed by the fault-free units. Finally, based on the diagnosis of the system, reconfiguration is a concurrence by all service users on the status of their servers. Therefore, at a high level, the fault-tolerant, distributed operating system of Figure 1 consists entirely of consensus procedures. In turn, this operating system may be used to implement fault tolerance with the state machine approach. Here, a service is a deterministic process or state machine whose execution is based solely on the reception of a message and which occasionally produces output. A fault-tolerant service is created by replicating the desired server and the service requests of its clients. At the heart of this technique is the coordination of the service population such that the failure of a member will be recognized and tolerated. In other words, a client, or any other destination for a service output, must be able to determine a correct result from the many results produced by the service population. Thus, the client uses a consensus procedure to uncover the correct result, i.e. the client provides a voting service to its servers and itself.

Traditionally, the formation of a consensus among several processors has been implemented with n-modular reduradancy (NMR) at a great cost of resources while only attaining the throughput (jobs per unit time) of a single unit. With NMR, n units perform the same task. Thus, t faulty units $n > 2t + 1$, may be masked by taking a majority vote of the n results. The throughput of the n-processor system could be inereased by the number of fault-free units if one could reliably determine what components of the system were faulty. Then, rather than mask the faulty processors, the system could identify and ignore them, thus allowing unique tasks to be seheduled on each fault-free processor, increasing the performance of the system over the NMR technique by the number of fault-free unit. Therefore, a natural goal is to detect and locate faulty processors and to disseminate this information to the fault-free processors. If the diagnosis is correct, then the result of each processor is as reliable as the majority result of NMR with a perfect voter. In other words, diagnosis allows consensus to be performed in what is assumed to be a fault-free environment. Researchers in the field of system diagnosis have explored solutions of this type for over 25 years [50], and the results are applicable to wafer scale integration [63, 56], large, loosely-coupled, distributed computer networks [34], tightly-coupled multiprocessors [16], and to other kinds of multicomputer systems.

A problem with diagnosis is that the fault status of the system is obsolete, although possibly correct, as soon as it is calculated. Most likely a fault will require a recovery action. Therefore, NMR techniques may still be needed when any recovery procedure would be too costly. But implementation of NMR requires a voting mechanism that maps the n results into a single output. Obviously, the reliability of any process that uses this output is directly related to the reliability of the voting mechanism. If a single process is using this result, then it is sufficient for it to act as its own voter as the voting process will fail exactly when the process fails. But when multiple processors rely on the output of the NMR system and their subsequent computations must be consistent, then every processor must be able to agree on the output of the NMR system. In this case, a single point of failure is unacceptable, and the voting mechanism must be distributed. Work on the Byzantine Generals Problem or Byzantine agreement explores the consensus problern given the need to distribute the voting process.

Despite their different characteristics, the Byzantine agreement and system diagnosis problems have very similar goals, namely to produce a correct agreement despite a number of faults. That is, given a distributed system subject to failures, methods are needed to allow the system to progress in a consistent manner. Specifically, such methods may provide synehronization or reliable communication, but generally they allow the fault-free processor population to reach conelusions on the state of the system. On the one hand, system diagnosis does this by identifying faulty processors in order that their impact may be avoided, and on the other hand, Byzantine agreement does this with protocols that will mask any possible impact of the faulty processors.

The conditions of reaching consensus

In this section we examine the conditions for the components a distributed system to reach consensus. Dolev et al. identified a set of system parameters for classifying distributed systems [17]. The following main items form a subset of those parameters:

Synchronous **or** *asynchronous* **processing elements.** The processors are synchronous if and only if there exists a finite constant $s \geq 1$ such that for every $s+1$ atomic steps taken by any processor, every other processor will take at least one atomic step.

Bounded **or** *unbounded* **communication delay.** A communication delay is unbounded if and only if every message sent by a processor arrives to its destination within t real-time steps, for a predetermined t.

Ordered **or** *unordered* **messages.** Messages are partially ordered (or simply ordered) if and only if processor P_i receives message m_j before message m_k, when processor P_j sends m_j at real time t_j, processor P_k sends m_k at real time t_k, and $t_j \leq t_k$. A complete ordering is achieved when messages are assigned different priorities, and in the case $t_j = t_k$ their order is decided on the basis of their priorities.

A consensus is always guaranteed if the system can implement ordered atomic broadcast protocols. Such protocols guarantee, that if two messages m_1 and m_2 are disseminated in the system, either all processors will receive m_1 first or all processors will receive m_2 first.

Table 1: Conditions of consensus in distributed systems

Processors	Communication	Message order: Unordered / Transmission: Point-to-point	Message order: Unordered / Transmission: Broadcast	Message order: Ordered / Transmission: Broadcast	Message order: Ordered / Transmission: Point-to-point
Asynchronous	Unbounded	No	No	Yes	No
Asynchronous	Bounded	No	No	Yes	No
Synchronous	Bounded	Yes	Yes	Yes	Yes
Synchronous	Unbounded	No	No	Yes	Yes

***Point-to-point* or *broadcast* transmission mechanism.** A transmission mechanism is point-to-point if a processor can send a message to only one other processor in an atomic step. It is broadcast if the processor can send a message to each other processor in the system in one atomic step.

Table 1 summarizes all possibilities for consensus [68]. As the table shows, consensus can be achieved in three minimal cases:

1. Processors are synchronous and communication is bounded.

2. Messages are ordered and the transmission mechanism is broadcast.

3. Processors are synchronous and messages are ordered.

However, the third case is included only for completeness. The best known consensus algorithms for this case require an exponential number of messages, so it is of little practical importance.

The first case describes a situation in which processors can detect errors in others using time-outs. This assumption is the basis of standard commit protocols, such as the three-phase commit. The second case deals with asynchronous processors, some of which can also fail. Still, they have an ordered atomic broadcast protocol (possibly because they are connected to a reliable bus).

A variation of the second case is k-casting. This protocol assumes that the transmission mechanism provides a broadcast to at most k other processors. It was shown, that if k-casting is possible and the messages are ordered, the system can attain deterministic consensus in the presence of up to $k-1$ failed processors.

Another variant requires that processors are "nearly" synchronous. This applies to processors being able to read a value from the network, process it, and write it back in one atomic step. The addition of bounded communication delay and broadcast transmission will be sufficient for reaching consensus. In other words, if processors can execute a critical section of the instruction flow in a predictable amount of time, then the problems with processor asynchrony can be overcome.

0.1.3 Definitions and notation

This section describes the main concepts of system-level diagnosis and their notation, as they are used throughout this chapter:

A system is assumed to be composed of n units (processing elements), denoted by the set $U = u_1, \ldots, u_n$. Each unit u_i is capable to execute tests on a particular subset of other units in U. Units are not allowed to test itselves. The set of testing assignments T is represented by a directed graph $G = (U, E)$, called the *testing graph*. In the testing graph, a vertex $u_i \in U$ corresponds to a unit, and a directed edge $(u_i, u_j) \in E$ represents the test $t_{ij} \in T$ executed by unit u_i on unit u_j.

With each unit $u_i \in U$ the following sets can be associated:

$$\begin{aligned}
\Gamma(u_i) &= \{u_j | t_{ij} \in T\} \\
\Gamma^{-1}(u_i) &= \{u_j | t_{ji} \in T\} \\
N(u_i) &= \Gamma(u_i) \cup \Gamma^{-1}(u_i).
\end{aligned}$$

Similarly, the following sets can be defined for a subset of units $U_k \subseteq U$:

$$\begin{aligned}
\Gamma(U_k) &= \bigcup_{u_i \in U_k} \Gamma(u_i) - U_k \\
\Gamma^{-1}(U_k) &= \bigcup_{u_i \in U_k} \Gamma^{-1}(u_i) - U_k
\end{aligned}$$

The set $\Gamma(u_i)$ contains the units tested by a given unit u_i whereas $\Gamma^{-1}(u_i)$ corresponds to the units testing unit u_i. $N(u_i)$ is the set of units which are neighbors of unit u_i. A unit is an *internal unit* of a set U_k if it belongs to that set, otherwise it is an *external unit* to the set U_k. Then, $\Gamma(U_k)$ and $\Gamma^{-1}(U_k)$ are the set of external unit to the set U_k, which are tested by and testing the units internal to the set U_k.

When discussing the faults, tests, and their relationship in particular, we use the following notation:

A potential fault in the system is denoted by f_i, and the set of all potential faults under consideration by $\mathcal{F} = f_1, \ldots, f_n$. Since multiple faults may occur, the subset of $F^j \subseteq \mathcal{F}$ faults present in the system is referred to as a *fault pattern*. The set of considered fault patterns is $F = F^0, \ldots, F^{m-1}$. Here, F^0 symbolizes the fault-free case: $F^0 = \emptyset$. The number of faults in a fault pattern is $\phi^j = |F^j|$, while the number of faults actually present in the system is simply ϕ.

Tests have a terminology and notation similar to faults. A test is denoted by t_i, and the set of all tests devised for the system by $\mathcal{T} = t_1, \ldots, t_p$. A *test pattern* T^j contains the tests that failed on the application of the test set T. The set of all test patterns that are expected to occur is $T = T^0, \ldots, T^{q-1}$. Correspondingly, the $T^0 = \emptyset$ test pattern represents the situation when all of the tests passed.

0.1.4 Preliminary discussion of diagnostic algorithms

The fundamental model for multiprocessor system-level fault diagnosis was developed by Preparata, Metze, and Chien [50]. This model formed a framework for much of the research in the area. We present the model as a basis for all main issues in further discussion.

Preparata et al. made the following assumptions considering diagnosable systems:

- A large system is partitioned into units.
- Units can execute test on each other to determine the condition of other units.
- A test outcome is evaluated as "pass" or "fail" on the basis of the test responses. The tester unit classifies the tested unit as fault-free or faulty.
- Units are *intelligent*. They are able to perform testing task individually, independent on other units.

- A fault-free unit is always able to identify accurately the fault state of the tested unit.
- A faulty unit produces unreliable test results, independent on the fault state of the tested unit.
- All faults are hard or permanent faults.

Test are executed periodically in the system, according to a predefined schedule. Test results are sent from every unit to a hypothetical central diagnostic device. It analyzes the syndrome, and produces the diagnosis.

Originally, Preparata et al. were interested in the so-called t-diagnosable systems. These systems allow correct and complete diagnosis under the presented conditions if at most t units can become simultaneously faulty. A diagnosis is said to be *complete* if all faulty units can be identified from a syndrome resulted in the presence of this fault set. A diagnosis is *correct* when no fault-free units are improperly classified to be faulty. Two different diagnostic strategies can be applied to t-diagnosable systems: one-step diagnosis and sequential diangosis. In *one-step diagnosis* or *diagnosis without repair* any possible fault set can be diagnosed correctly and completely based on a single syndrome produced by the system. This is the highest quality of diagnosis, but at the same time it has the strongest conditions. In *sequential diagnosis* or *diagnosis with repair* the location of faulty units is uncovered in more phases. At first only a subset of the faults present is diagnosed. These faulty units are replaced or repaired. Then the set of tests in the system is executed again, and the same process is repeated until no more errors are detected. To succeed, the diagnosis algorithm has to identify at least one faulty unit decoding every single syndrome.

Three important problems arise in connection with the Preparata, Metze, and Chien (PMC) model:

Characterization problem. To find necessary and sufficient conditions for the test connection assignment of a system to achieve a given level of diagnosability under a certain fault model and an allowable family of fault sets.

Diagnosability problem. To determine a uniquely diagnosable family of fault sets in a given test connection assignment and fault model.

Diagnosis problem. To determine a fault set present in a systems based on a given syndrome, test connection assignment, and fault model;

assuming that the set of faults present belongs to an allowable family of fault sets.

Note, that the characterization problem and the diagnosability problem are not the same. The former is a theoretical problem while the latter is an algorithmic problem. Thus, a good answer to the characterization problem may result an efficient algorithm for the diagnosability problem.

The necessary conditions of one-step t diagnosability for a system consisting of n units under the PMC model were given by the authors:

$$\begin{aligned} n &\geq 2t+1 \qquad (1)\\ |\Gamma^{-1}(u_i)| &\geq t, \forall u_i \text{ such that } u_i \in U \qquad (2) \end{aligned}$$

Hakimi and Amin showed that for the special case when mutual tests between units are not allowed the necessary conditions are also sufficient [24]. To achieve general characterization of one-step t-diagnosable systems they also added a third condition:

$$\begin{aligned} |\Gamma^{-1}(U_k)| &\geq p, \forall p \text{ such that } 0 < p \leq t, \text{ and} \\ & \forall U_k \text{ such that } U_k \subset U \text{ and } |U_k| = n - 2t + p \qquad (3) \end{aligned}$$

The most significant result for sequential t-fault diagnosability deals with single-loop systems (i.e. systems in which there is exactly one test on each unit and one test performed by each unit) [50]. A single loop system is sequentially t-diagnosable if and only if n, the size of the loop, satisfies

$$n \geq \left\lfloor \left(\frac{t+2}{2}\right)^2 \right\rfloor + 1 \qquad (4)$$

The diagnosability problem was solved by Sullivan [66]. He used *network flow* to constuct an $O(|E|n^{2.5})$ algorithm that computes the t-diagnosability of a given test connection assigment under the PMC model. Recently, Ranghavan and Tripathi improved the efficiency of the algorithm to $O(nt^{2.5})$ [51]. They also proved that calculating the sequential diagnosability of repairable systems is co-NP complete [52].

The problem of finding a minimal cardinality set of faulty units in whose presence a system can produce a given syndrome (i.e. the diagnosis problem) is NP complete according to Fujiwara and Kinoshita [21]. Still, much work has been done in restricted situations. For t-diagnosable systems under the PMC model Kameda, Toida, and Allan gave an $O(t|E|)$ algorithm of one-step diagnosis. Later, Sullivan improved the KTA solution and extended it by employing weighted digraphs in his system model [67]. The algorithm has $O(t^3 + |E|)$ time complexity, which is the best known solution in the practically relevant case when t is small compared to n. The best method in terms of worst case efficiency was given by Dahbura and Masson [12, 13]. Their algorithm first calculates the union of implied faulty sets for all units, represented by the graph G_{L} called the *L-graph*. Then, the set of faulty units is identified as the *unique minimum vertex cover set* of G_{L}. The authors proved, that for the class of graphs that include G_{L} the problem to find a unique minimum vertex cover set can be solved in polynomial time. The total time complexity of the diagnosis algorithm is $O(n^{2.5})$.

0.2 System model

0.2.1 Fault models

Faults are *impairments* to system dependability. They are undesired—but preferably not unexpected—circumstances causing that reliance cannot be placed justifiably on the service the system delivers.

The sources of faults may be classified according to three main viewpoints: their *nature*, their *origin*, and their *persistence*. The main fault classes according to the three viewpoints are presented in Figure 2. A detailed description of the various fault classes can be found in [36].

System-level diagnosis distinguishes the following main fault cathegories:

Permanent faults. Permanent faults remain to exist after their occurrence until the related faulty component is repaired, or replaced. Their presence is not affected by pointwise conditions, thus they always produce errors when they are fully exercised. Internal physical or design faults belong to this cathegory.

Intermittent faults. Intermittent faults are temporary internal faults. These faults are esentially permanent faults, but their activation depends on rare combinations of conditions. By their nature, intermittent faults do not occur consistently, which makes their detection a probabilistic event over time. Recurrent faults, whose activating conditions can be reproduced are also considered as intermittent faults in system-level diagnosis.

Transient faults. Transient faults are originating from the effects of the physical environment. They only affect the operation temporarily,

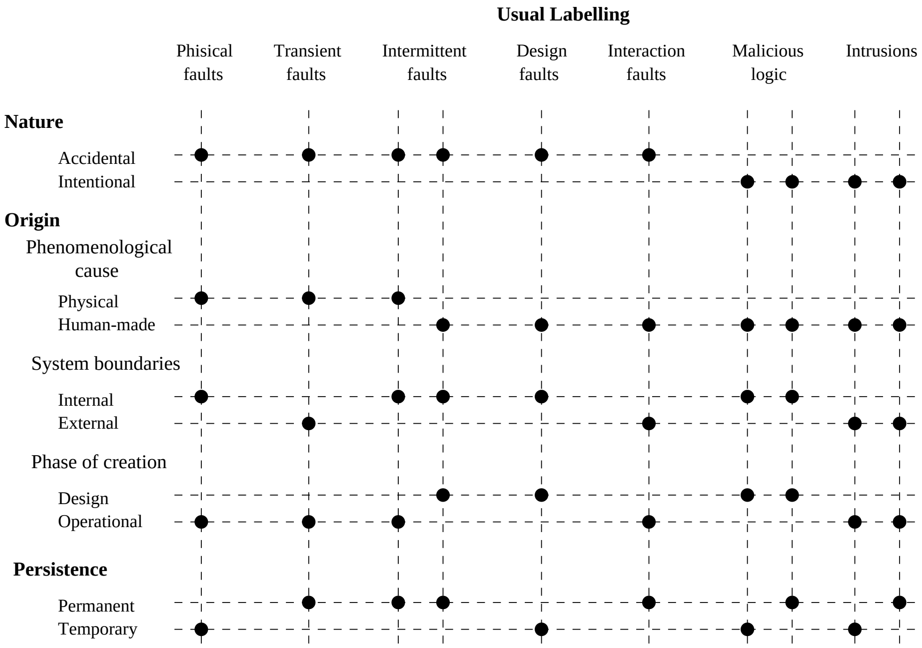

Figure 2: The classification of faults

usually without causing a permanent fault in the corresponding component. Thus, transient faults do not imply the system itself is faulty.

Transient faults are hard to deal with in system-level diagnosis. They cannot be distinguished from intermittent faults just by simply observing the system. However, the difference is very important: the affected component cannot be declared faulty if a transient fault occured (although a restart, or a temporary shutdown might be necessary to restore the integrity of the system).

Hybrid-fault model. Transient and intermittent faults are the major source of errors in systems. Recent studies have shown, that generally more than 80 percent of the field failures were caused by these fault types [29]. Therefore, Mallela and Masson [45] introduced the concept of a *hybrid-fault* situation. The hybrid-fault model specifies bounded combinations of permanently faulty and intermittently faulty components in the system. The concept includes as special cases the case where each faulty unit may only be either permanently or intermittently faulty (see Section **??** for a more complete description of the model.)

Occurring faults are at first *dormant*. They become *active* when they produce an error. An internal fault may alternate between its dormant and active states.

An error is a change in the state of the system, so that it is no longer correct. The error is *latent* when it has been not recognized, but it can be *detected* by a detection algorithm or mechanism. An error may disappear before it is detected. If the error exist for a sufficient time it can—and usually does—propagate. By propagating it produces new errors.

If the system is treated from a hierarchical viewpoint, it can be divided into levels. Two levels are connected by a well-defined interface. Faults and errors exist on the same level in this hierarchy. When an error "passes through" such an interface, it results in a failure [32]. Failures manifest themselves as faults on the next level of the hierarchy. Thus, the activation mechanism of faults, errors, and failures completes the following "fundamental chain:"

$\ldots \rightarrow$ failure $\rightarrow$ fault $\rightarrow$ error $\Rightarrow$ (level interface) $\Rightarrow$ failure $\rightarrow$ fault $\rightarrow$ $\ldots$

A system generally does not alway fail in the same way. The ways a system can fail are specified as its *failure modes.* They also may be cathegorized according to different viewpoints. The two main viewpoints significant in system-level diagnosis are the *failure domain* viewpoint, and the *failure perception* viewpoint. The interested reader is again referred to [36] for a more complete discussion of the topic.

From the failure domain viewpont one can distinguish:

Value failures. The value of the delivered service does not comply with the specification.

Timing failures. The service is not delivered within the specified response time frame. This is sometimes called a *performance failure*. Early and late timing failures can be defined depending on wether the service was delivered before or after the specification.

A class of failures relating to both value and timing failures are the stopping failures: the system activity is not any more perceptible and a constant value service is delivered. The special case of stopping failures when no service is derivered is called *omission failures*. If the omission of service delivery is persistent, a *crash failure* has occurred.

When a failed service is delivered to multiple users, the failure perception leads one to distinguish:

Consistent failures. All users have the same perception of the failure.

Inconsistent failures. The system users may have different perception of an inconsistent failure. This is usually termed as *Byzantine failures* [35].

Some examples to illustrate the notions of faults, errors, and failures:

- an α-particle of sufficient energy to cause a change in the charge stored in a memory cell is a transient fault. The changed logical value in the memory results a latent error. It can disappear if a new value is written to the corresponding memory position. If the memory is protected using error checking and correction (ECC), the error can be masked and the correct value is read. Without such protection a value failure will occur on the functional level upon the subsequent read operation from the affected memory position.

- an open circuit in a MOS gate is a physical failure. It may manifest itself as a permanent stuck-at fault on the gate level. The fault will remain dormant until the respective gate is activated. Then, in certain conditions an error will be produced. If the error affects the operation of the component containing the faulty gate, a failure occurs. Note, that in a complex component this failure will likely appear as an intermittent fault on the functional level.

- an incorrect algorithm in a programmer's mind is an error, it becomes a failure during the coding of a subroutine, resulting a design fault in the software. This dormant fault is activated when the faulty subroutine is executed. An error is created, and the result might be a computational failure on the user level.

Byzantine faults

In a centralized environment, every unit shares a common media for information exchange. This is equivalent to a broadcast mechanism, therefore inconsistent behaviour of a unit to other units can be detected. However, in distributed systems no central recources exist, so inconsistent failures must be taken into consideration. The *Byzantine agreement* problem is a special case of reaching consensus in distributed systems.

The original problem, presented by Pease et al. in [48], was the following: n units must reach general agreement in a synchronous system. Of these, at most t can be faulty, behaving in arbitrary manners. Each unit has a private value that it wishes to share with the other units. Any two units can communicate via a point-to-point network, having the following characteristics:

- it is assumed to be fail-safe,
- it is not affected by the failure of the connected units,
- a recipient can identify the sender of an incoming message,
- the communication delay is negligible.

The goal was to achieve *iterative consistency*, satisfying the conditions:

Consistency. Each fault-free unit computes an identical vector of values whose ith elements corresponds to the ith unit in the system.

Meaningfulness. An element in the local vector corresponding to a fault-free unit is the actual private value of the unit.

The authors proved that consistency can be reached if and only if $n \geq 3t + 1$. They also gave an algorithm, which assures interactive consistency under this condition. The algorithm requires $t + 1$ rounds of message exchange among units. After the communication, the units use a recursive computation based on majority voting of private values to calculate the correct vector.

Lamport et al. transformed the above problem into a situation, where a Byzantine commanding general wishes to organize a concerted plan of action: e.g. an attack or a retreat. Each division of the army is lead by a lieutenant general. The generals can communicate with the commander and one another only by messengers. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement. Despite this, all of the loyal Byzantine lieutenant generals must decide on the same plan of action, and it must agree with the commanding general's order (provided that he is loyal). An algorithm which completes the problem successfully is said to reach *Byzantine agreement.*

In a distributed environment the Byzantine generals are substituted by processing elements. A single unit (representing the commanding general) wants to broadcast its private value to every fault-free unit. Faulty units may send inconsistent information to other units, trying to prevent consensus. It was shown, that similarly to iterative consistency $n \geq 3t + 1$ must hold even if only approximate agreement is required. Note, that the two problems are different: in Byzantine agreement only one unit needs to share its private value opposed to all of the units sharing their values. By assigning the role of the commanding general to every unit and solving the resulting consensus problems concurrently, the two problems become identical.

Byzantine agreement is made simpler, if the messages are *signed* or *authenticated* [48, 35]. A message signed by a fault-free unit:

- cannot be forged by faulty units,
- any alteration of its contents can be detected,
- any unit can verify the authenticity of the signature.

In this situation, algorithms were created which cope with any number of faulty units (yet, the problem is meaningless if $n < m + 2$). The connectivity of the connection graph must be $t + 1$. Dolev and Reischuk gave and algorithm that uses authenticated messages and requires $O(t)$ comminication rounds and $O(n + t^2)$ messages. They have also proven this to be a lower bound [18].

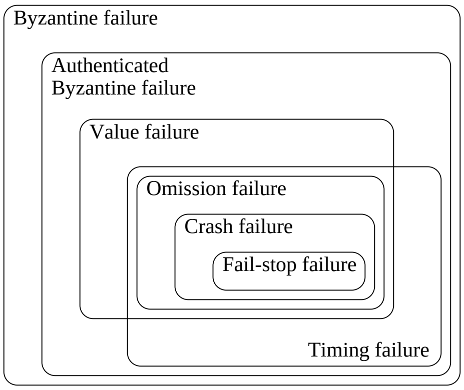

Figure 3: An ordered failure classification

The very pessimistic failure model of Byzantine agreement (i.e. in worst case faulty units are allowed to work in collusion with complete knowledge about the system) is the safest approach to handle inconsistent failures. Methods like message authentication or reliadle hardware broadcast mechanisms limit the capability of faulty units to behave maliciously. Therefore, another classification of failures was adopted, which introduces several new failure groups to the traditional ones (presented in Section 0.2.1) [1]. The classes are (from strongest to weakest): *fail-stop* failures, *crash* failures, *omission* failures, *timing* and *value* failures, and Byzatine failures. Figure 3 shows the graphical representation of their relationship.

The definition of the new failure groups:

Fail-stop failure. The failure that occurs when a unit ceases its operation, but first alerts other units of the termination.

Authenticated Byzantine failure. An arbitrary or malicious failure. Units can send inconsistent messages to different other units, but they cannot imperceptibly alter a message signed by a fault-free unit.

Byzantine failure. Every failure is possible in the system model. This is considered to be the universal set of failures.

0.2.2 Testing models

Testing is a *dynamic verification* of the system. A test, in a general sense, can be imagined as the application of some sequence of input patterns and the observation of one or more outputs for one or more time frames [32]. Tests are aimed at uncovering active faults by detecting the errors they caused. The generation of the test inputs can be either deterministic or probabilistic:

Deterministic testing. Test patterns are predetermined according to the adopted criteria and the given fault model. Test generation can be *functional* or *structural*, depending on whether the selection process is related to the function or the structure of the system.

Random, or statistical testing. Test patterns are selected on the basis of a defined probability distribution on the input domain. The distribution and the number of input patterns are determined according to the adopted criteria and the given fault model.

A test is said to be *complete* for a certain fault if it always fails when the fault is present, and passes when all faults are absent. On the system level, a test performed by unit u_i on unit u_j is complete if it detects all low-level faults that can cause a failure in unit u_j. Yet, for highly complex components such as processors exhaustive testing can take a lot of time and processing capacity, and still not produce accurate results. If the test detects only the subset $F^d \subset \mathcal{F}, |F^d| = \phi^d$ of faults from the set of all possible faults $\mathcal{F}, |\mathcal{F}| = n$, it is called *incomplete*. The performance of test can be characterized by the *test coverage*, defined as $\mathrm{t_{cov}} = \phi^d/n$. Naturally, complete tests have 100 percent test coverage.

A second problem that reduces the effectiveness of tests is *test invalidation*. Ideally, the outcome of a test on a fault is not influenced by the presence or absence of other faults. A complete test is called *invalid* if some faults can make it erroneously pass when the fault for which the test is complete exists, or can make it fail when the fault does not exist. For example, a memory test in a processing element may require the assistance of the memory management (MMU) unit and the internal bus. Consequently, a faulty MMU or bus can cause the memory test to fail, even if the memory is fault-free. In other words, the MMU and bus faults *invalidate* the memory test.

The impairments to testing (incompleteness, invalidation) can be characterized by the test result in the four possible configurations, respective to the fault state of the tester and tested units. Table 2 presents a complete tabulation of the nonequivalent testing models [20].

Table 2: Nonequivalent testing models

Notation	Name	Reference	Test outcome			
			$\odot \rightarrow \odot$	$\odot \rightarrow \otimes$	$\otimes \rightarrow \odot$	$\otimes \rightarrow \otimes$
T_{00}	0-fail-safe tester		0	1	0	0
T_{01}	perfect tester		0	1	0	1
$\mathrm{T}_{0\mathrm{X}}$			0	1	0	X
T_{11}	irreflexive invalidation (1-fail-safe tester)	HK2 [33]	0	1	1	1
$\mathrm{T}_{1\mathrm{X}}$	reflexive invalidation	HK1 [33]	0	1	1	X
$\mathrm{T}_{\mathrm{X}1}$	asymmetric invalidation	BGM [2]	0	1	X	1
T_{XX}	symmetric invalidation	PMC [50]	0	1	X	X
T_{PT}	partial tester		0	X	X	X
$\mathrm{T}_{\emptyset}$	0-information tester	Blount [10]	X	X	X	X

(Key: $\odot$ fault-free unit, $\otimes$ faulty unit, $\rightarrow$ test assignment)

Complete tests

Most of the deterministic diagnosis methods make the complete test assumption, i.e. they suppose that the test results generated by a fault-free tester unit always correctly reflect the actual fault state of the tested unit. In this situation only test invalidation can cause diagnostic uncertainty by corrupt the tests of faulty tester units. Invalidation models are strongly depending on the applied testing mechanisms:

Direct testing. In direct testing, the units of the system execute tests on each other. Typically, unit u_i tests unit u_j by giving it certain inputs and comparing the resulting outputs with some set of correct responses. Usually only the normal communication facilities can be used for testing purposes. Since units are able to execute the tests individually, these tests can be invalidated only by a fault in the tester unit. This is also known as Single Invalidation Per Test (SIPT).

The main invalidation schemes of direct testing are the following:

- *Symmetric test invalidation.* This model was introduced in the late sixties by Preparata et al. [50]. Since then, it became the most widely used test invalidation model in system-level diagnosis. A faulty tester unit in symmetric invalidation generates an arbitrary test result, i.e. the test may pass or fail independent on the state of the tested unit. If these conditions hold, a $t_{ij} = 1$ test outcome indicates a fault in either the tester or the tested unit, or in both.

The main advantage of the model lies in its generality: all other complete test SIPT invalidation models produce syndromes compatible with it. Thus, the algorithms designed for symmetric test invalidation are applicable to a wide range of systems.

- *Asymmetric test invalidation.* Barsi et al. proposed a less restrictive invalidation model in [2]. They assumed, that the testing units must apply a large number of test stimuli for a complete test. Therefore, it is reasonable that at least one mismatch occurs between the actual and expected results for a faulty tested unit, even if the tester itself is faulty. Yet, a faulty tester unit testing a fault-free tested unit still can produce an arbitrary output. The above model is called asymmetric invalidation. The limitation of the possible syndromes that can occur make the decoding of syndrome easier: all tests with outcome $t_{ij} = 0$ unambiguously identify a fault-free tested unit (the $t_{ij} = 1$ test results have the same interpretation as for symmetric invalidation.

  Later, Kreutzer and Hakimi [33] used the same assumptions to extend this model. They suggested, that a faulty tester unit should always find a fault-free tested unit as being faulty. Two mechanisms comply with this situation: when both the tester and tested units are faulty. in *reflexive invalidation* both pass or fail test outcomes are possible, while in *irreflexive invalidation* a faulty tester always generates a failed test result. They also convey more diagnostic information than the original Preparata model: for reflexive invalidation, the $t_{ij} = 0$ test outcome means that both units associated with the test have the same fault state; for irreflexive invalidation it even implies that they are unambiguously fault-free.

- *General test invalidation models.* Kavianpour examined the various testing models for system-level diagnosis (shown in Table **??**. He compared the models and discussed their most interesting properties.

  Selényi introduced a unified framework for complete test SIPT invalidation schemes. His work includes the diagnosability analysis of all the nine possible invalidation mechanisms, as well as the optimal testing arrangements for them. He also developed a diagnosis algorithm for the unified model. The algorithm can handle inhomogeneous systems consisting of various components having different invalidations. Recently, the idea was expanded to multi-valued test using Constraint Satisfaction (CSP) methods [47, **?**].

Somani et al. presented a generalized characterization theorem for system-level diagnosis [64]. They considered permanent and intermittent faults, and symmetric and asymmetric test invalidation in their model. For the model they derived the necessary and sufficient conditions for any fault pattern of any size to be uniquely diagnosable. Their results are described in Section **??**.

Self-testing Testing can also be performed by each processing element on itself in a series of *self-tests*. Kuhl and Reddy proposed a hierarchical system of built-in fault tolerance techniques and monitoring devices [34]. These include hardware or software checkers, watchdog timers and processors, error-checking codes, or redundancy with voting. Thus, the testing of a unit becomes a simple request for the actual fault status of the tested unit.

Multiple invalidations per test A test may consist of a large number of stimuli to obtain a good test coverage. The testing can be done by a single unit, but as another possibility, it can be a cooperation of multiple testing units. In the latter case, a fault in any of the participating units can invalidate the result. If more than one independent modules are required to perform a test, the system is described as Multiple Invalidations Per Test (MIPT).

The first MIPT scheme was introduced by Russel and Kime [31], [57, 58]. The model is formulated of faults, tests, and their relationships. It assumes, that there exists at least one complete test for each possible fault in the system (no undetected faults). The tests are complete for one and only one fault: Single Unit Per Test (SUPT). They can be invalidated by any number of faults, that is *hard-core* tests (not invalidated at all) are also included in the model. The authors defined two different types of systems (the notation $T(F^i)$ represents the set of tests invalidated by the presence of fault pattern F^i):

1. *semimorphic* systems: $T(F^i \cup F^j) \subseteq T(F^i) \cup T(F^j)$

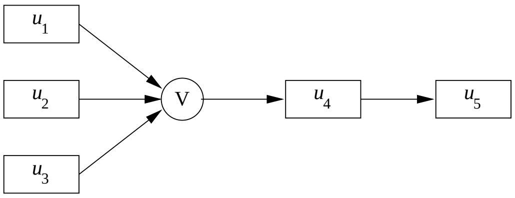

Figure 4: Illustration of a semimorphic system

2. *morphic* systems: $T(F^i \cup F^j) = T(F^i) \cup T(F^j)$

Figure 4 illustrates a semimorphic system. Test t_4 on unit u_4 is performed by using the voted output if units u_1, u_2, and u_3. In the example, test t_4 is valid only if at most one of the units u_1, u_2, and u_3 is faulty. It is invalid, if two ore more of these units are faulty. Restricting the set of potential fault patterns to single faults the example system becomes morphic.

Kime [31] and later Maheswari and Hakimi [43] extended the model by allowing tests that are complete for more than one fault. More precisely, a valid test t_i will fail if one or more faults in $F(t_i)$ are present, and will pass if all faults are absent ($F(t_i)$ denotes the set of faults t_i is complete for). This testing scheme is called Multiple Units Per Test (MUPT).

As Gupta and Ramakrishnan [23] pointed out, multiple invalidation can sometimes may result in loss of diagnostic information. They considered systems, where several units conduct multiple independent tests on a unit. Each of the tests probes a different part of the unit under test, and the results are combined into one complete test. Under these conditions, a faulty testing unit does not necessarily make the combined results unreliable.

Comparison testing A practical method of detecting faults in components is comparison. Two or more units are assigned the same task, and the results are compared. Comparing the results of two units one can detect an error, but there is no way to tell apart the faulty unit from the fault-free one. When comparing result from many units, one can diagnose up to $\lceil n/2 \rceil - 1$ faults using n-multiple redundancy (N-MR) techniques.

Table 3: Comparison testing model

Compared unit u_i	**Compared unit u_j**	**Comparator**	**Comparison result**
fault-free	fault-free	fault-free	0
fault-free	fault-free	faulty	1
fault-free	faulty	fault-free	1
fault-free	faulty	faulty	1
faulty	both states	both states	X

Whereas the tests of direct testing are mostly performed in rounds between normal system activities, comparison tests can take place in conjunction with productive tasks. An error is detected right when it would propagate. Comparison tests can be made more compact by creating and comparing *signatures*—such as checksums or cyclic redundancy codes—of the results.

Malek [45] introduced the comparison approach for system-level diagnosis and gave a method to assign comparison edges in the system graph. A syndrome of comparison results is created by labeling an edge with a "0" if it connects two units that agree, and with a "1" if it connect two disagreeing units. Maeng and Malek [42] suggested that a third unit should be employed to compare the results, thus decentralizing the arbitration of comparisons. Their testing model is shown on Table 3.

Time Domain testing A large portion of the potential failures is based in the time domain (see Figure 3. Therefore, it is necessary to also test system component with respect of time. When a unit fails to complete a task, or a message is not received within a certain time frame, a failure occurs. Errors causing timing failures can be detected by using time stamps and timers in the presence of a global set of synchronized clocks. A common time domain based testing technique is the use of ¡I'm alive¿ or "heartbeat" messages. Units send periodically ¡I'm alive¿ messages to their neighbors, and validate the incoming messages by time-outs. When an ¡I'm alive¿ message is late or is omitted, the origin of the message is declared as faulty.

Note, that the time domain testing model is orthogonal to the value domain methods described earlier. Timing errors may remain undetected in a value domain fault model as well as value errors may stay uncovered in a time domain fault model.

Incomplete tests

The complete test assumption is not applicable in many practical cases: either the tested unit is too complex to gain 100 percent test coverage, or the timing and performance requirement necessitate a quick but incomplete test. These cases can be handled by using the *partial tester* model, examined by Simoncini and Friedman [61]. They consider the problem that a fault-free unit may be able to detect only a fraction $0 \leq p < 1$ of all the faults in the tested units.

Many authors use the same model to take into account intermittent faults, or even mixed systems with both incomplete tests and intermittent faults. The syndromes for all these situations are compatible. Yet, if the partial tester scheme is applied to intermittently faulty systems, the value of parameter p has no clear physical meaning and thus is hard to estimate correctly.

The 0-information tester

The most general testing model, referred to as *0-information tester* model, does not restrict the possible test outcomes. As illustrated in Figure 4, all combinations of fault states and test results are associated with a probability parameter ($\mathrm{P}(a_{ij})$ is the probability of the test t_{ij} being a_{ij} given the fault states of the testing and tested units). The parameters enable the model to address several error detection requirements:

- Assuming that a fault-free unit can fail in judging another fault-free unit, the errors in the interconnections between two units can be modeled (parameter q_{ij}).
- Considering that a fault-free unit can be unable to detect an error of a faulty unit, the incomplete tests can be taken into account (i.e. under a permanent fault model, parameter p_{ij} is the test coverage).
- Parameters r_{ij} and s_{ij} are the probabilities of a faulty unit correctly identifying another fault-free and faulty unit, respectively.

It is easy to see that the 0-information tester model incorporates all of the SIPT/SUPT testing schemes mentioned above. The more restricting models can be derived by adjusting certain parameters. The partial or incomplete

Table 4: Probability parameters for the 0-information testing model

Tester unit u_i	**Tested unit** u_j	**Test result** a_{ij}	**Parameter** $\mathrm{P}(a_{ij})$
fault-free	fault-free	0	q_{ij}
fault-free	fault-free	1	$1 - q_{ij}$
fault-free	faulty	1	p_{ij}
fault-free	faulty	0	$1 - p_{ij}$
faulty	fault-free	0	r_{ij}
faulty	fault-free	1	$1 - r_{ij}$
faulty	faulty	1	s_{ij}
faulty	faulty	0	$1 - s_{ij}$

testing model results using the value $q_{ij} = 1$. Further restricting the value of $p_{ij} = 1$ gives the complete testing model. Different test invalidation mechanisms also can be attained by manipulating the parameters r_{ij} and s_{ij}.

0.2.3 Representations of the fault-test relationship

- tabular forms
- Boolean equations
- digraph representations

0.3 Diagnostic algorithms

0.3.1 Classification of diagnostic algorithms

The original system model and diagnostic goals formulated by Preparata et al. make a number of not quite practical assumptions about the underlying hardware [50]. As a result, system-level diagnosis did not become the determining design principle in fault-tolerant mulitprocessor systems. Much research have been therefore conducted toward more widely applicable diagnostic models. The shortcomings of the PMC approach addressed are:

Complete tests. Algorithms based on the PMC model assume, that the result of a test performed by a fault-free tester unit always reflects the real fault state of the tested unit. However, in most practical cases it is impossible to set up tests that probe all of the complex components of a unit exhaustively. Thus, tests usually do not have 100 percent coverage. The *comparison testing* technique can improve on this situation, but it still does not assure the test to be complete. *Probabilistic methods* attempt to provide the most probable diagnosis to cope with imperfect tests.

Permanent faults. The only fault class the PMC model considers is the class of permanent faults. It was also assumed, that fault-free and faulty units do not change their fault states (i.e. do not fail or get repaired) during the whole testing procedure. In reality, the majority of faults occurring in the systems are transient or intermittent faults. Many different fault models were proposed to include *intermittent* and *hybrid fault* situations in system-level diagnosis. Most of the diagnosis methods based on these models can handle not only intermittent faults but also incomplete test. This is not surprising, since the syndromes test to syndrome projection schemes are identical in both cases.

Lack of knowledge about faulty units. Early fault models did not make assumptions about the behavior of faulty units. This general approach may be applicable to many systems, but more efficient algorithms can be formulated if there are additional information available on faulty units. *Asymmetric invalidation* by Barsi et al. placed some realistic restrictions on faulty units [2], later Hakimi and Amin suggested different invalidation mechanisms [24]. Still, these models assume that the system consist of identical components, i.e. it is homogeneous. *Generalized test invalidation* [60] integrates all of the possible SIPT/SUPT models and can also deal with inhomogeneous systems. Recently, a constraint-based representation was introduced in the framework of generalized invalidation, further extending its modelling power [47, **?**].

Intelligent tester units. The majority of testing mechanisms require the tester unit to carry out the testing task individually. A test may be so complex that it requires the cooperation of several units. Unfortunately, this also means that the test is invalidated by multiple faults. Only *MIPT diagnostic models* are able to represent these situations. Another disadvantage of assuming intelligent tester units is that it excludes passive components (not participating in the diagnosis; like communication devices, coprocessors, etc.) from the scope of the diagnosis. Liaw et al. suggested a solution to the problem by introducing *heterogeneous systems* [41]. Their model is composed of testing and nontesting units. The authors also gave a classification and a distributed diagnosis algorithm for their model.

The number of faults is bounded. Preparata et al. proved that $n \geq 2t+1$ must hold for the number of units in one-step t-diagnosable systems. This bound was the basis of characterization and diagnosis of these systems, resulting in inefficient diagnosis strategies. In practice it cannot be guaranteed that the number of faults will not exceed a statis limit if the system was not built specifically to fulfill this purpose. There are two ways to handle larger amounts of faulty units in the system: *sequential diagnosis* and *probabilistic methods.* Sequential diagnosis concentrates on locating only one fault at a time, and repairing the found faulty units. If the repair-rate is larger than the fault-rate, this strategy will eventually lead to a completely fault-free system. Probabilistic methods try to identify every faulty unit, but correctness of the resulting diagnostic image has a less than 100 percent probability when the conditions of one-step t-diagnosability are not met.

Centralized diagnosis. Often it has been assumed that a hard-core central supervising module exists in the system to analyze syndromes. Every unit transfers its local test results to this module via an ultra-reliable communication link. This assumption violates two important design principles. The first principle states that a fault-tolerant system must be free of single-point failures; or in other words it cannot have hard-core components. According to the second principle, reliable massively parallel systems should be scalable and gracefully degradable. The diagnostic algorithm must be *distributed* itself to avoid these pitfalls.

Test organization. Tests are organized in space (testing graph) and in time (test schedule). The testing graph defines the tester—tested unit relationships, and so it mainly depends on the interconnection topology of the system. Most of the diagnostic algorithms is restricted to a specific topology. *General topology* algorithms have no such restrictions, thus they can be applied to all kinds of 'topologically static' systems. Sometimes the interconnection topology is also subject to change (e.g. due to failures in communication devices). In this case *adaptive* algorithms—which automatically accomodate to the dinamically changing circumstances—can be used.

Many diagnostic algorithms ignore the timing requirements of tests—the test schedule. They treat the syndrome as a static 'snapshot' of the system. Thus, all of the units are supposed to be tested simultaneously, and there cannot be a new fault occurrence during testing and diagnosis. In reality, the execution of tests and the distribution of test results have a significant execution time. Units must perform the testing in a coordinated way to avoid deadlocks and performance bottlenecks. They also must be able to reconstruct the temporal order of the tests, no matter how the communication delays and the lack of sincronity among units has distorted it. The time spent with communication should be kept minimal. These problems are addressed in *event-driven* algorithms.

Reliable communication. Communication is a vital issue in system-level diagnosis. Since testing is performed locally at processing elements, the test results must be transferred to the central supervising module (centralized diagnosis) or broadcasted to a set of other units (distributed diagnosis). Usually the algorithms leave the failures of the interconnection links and devices out of consideration. In this case the underlying hardware must guarantee the reliable message transport using broken line detection, error-checking and correction (ECC) codes, message authentication. Adaptive algorithms can operate in weaker conditions: it is enough if the communication has fail-stop or crash failue semantics. Finally, the scope of diagnosis can be broadened to include the location of faults in the interconnection links and devices as well. It is most advantageous in fine-grain large distributed systems, where proper diagnostic support can reduce the time and cost of maintenance.

Diagnostic goals. A natural expectation of a diagnostic algorithm to find every faulty unit at once. However, this *one-step* diagnosis is usually quite a time and resource consuming task. Other methods achieve less precise results, but have much lesser requirements. Sequential diagnosis finds only one fault at a time. *Set diagnosis* [19] aims at finding a set of units, containing all of the faulty ones and potentially some fault-free ones as well. This can be acceptable when either single-unit diagnosability is not practical, or the system granularity is so coarse, that replacing a set of units can be done by replacing a single field-replaceable unit (FRU).

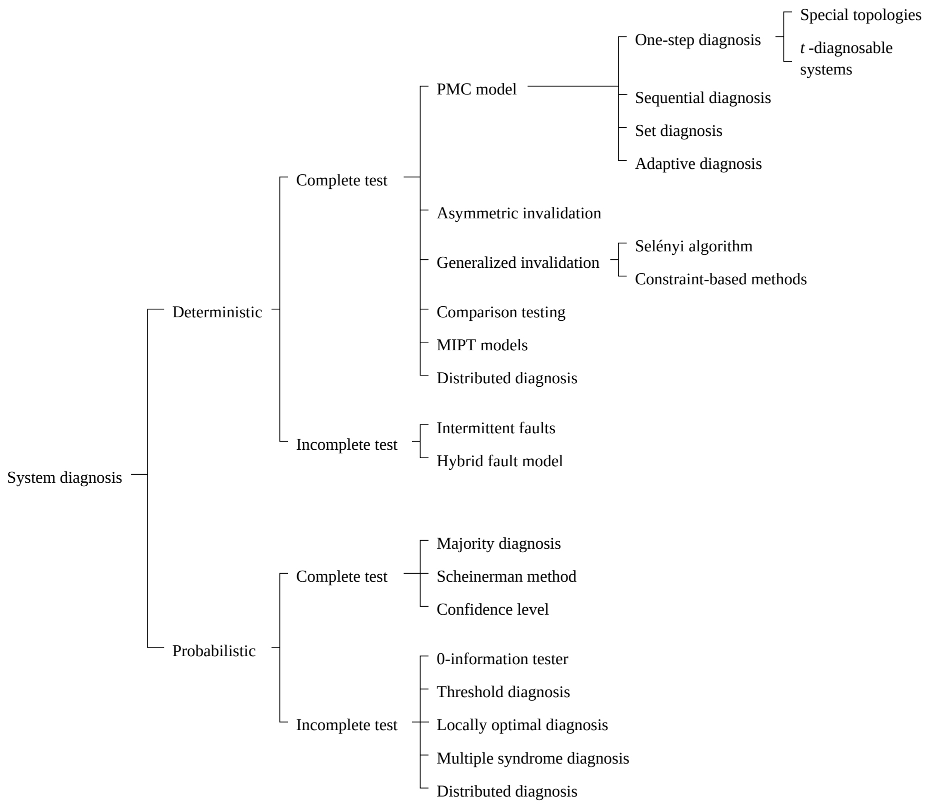

Figure 5: Classification of diagnosis algorithms

0.3.2 Deterministic algorithms

Centralized diagnosis

To get acquainted with the problem of system-level fault diagnosis and the possible solution methods, let us consider a small example taken from [60] of a fault-tolerant multiprocessor system consisting of 7 units. Each unit is capable of testing two other units, as illustrated in Figure **??**. The directed arcs represent tester-tested unit relationships, the arrow points from the tester unit to the unit under test. In other words, Figure **??** is a 7-node, directed $\mathrm{D}_{1,2}$ testing graph. Every unit conforms to the symmetric test invalidation model.

In centralized diagnosis first each test is carried out and the results are collected by the central supervising module. The complete set of test results is then analyzed. In our example, the outcome of the tests is written on the arrows (passed tests are denoted by a 0, failed test by a 1). From the syndrome we can draw the following conclusions on a purely logical bases without any particular knowledge of system-level diagnosis:

- The system contains at least one faulty unit. Otherwise all of the test would pass. This conclusion provides a *fault-detecting* information.

- Unit u_1 is surely faulty. Let us suppose that u_i is fault-free. In this case, units u_2 and u_3 also must be fault-free, since they were tested by fault-free units and the tests passed. However, at the same time u_3 is faulty, according to u_1. Since there is a contradiction among the conclusions, we must reject our original supposition. This conclusion provides a *fault-localizing* information.

- The fault states of units u_2, u_3, u_5, u_6, and u_7 are the same. The reason is that these units are involved in a loop of 0-links (passed tests). Suppose one of these units is fault-free. Then the unit tested by the fault-free one must also be fault-free. On the other hand, if one of the units is faulty, then its tester must be faulty as well. The same rule applies for each unit in a loop (or a strongly connected subgraph) composed of 0-links.

- A possible diagnosis is that all of the units in the system are faulty. Since faulty units can generate any test outcome independent on the state of the tested unit, every syndrome is is compatible with this situation, including the one shown in Figure **??**.

The last conclusion is not very useful in practice, however it cannot be left out of consideration without some *a priori* information. Assume we can guarantee (with an acceptable probability), that at most two units can be faulty at the same time—based on the thorough knowledge of the system. In this case units u_2, u_3, u_5, u_6, and u_7 are surely fault-free. This can be inferred, because these units have the same fault state, and the existence of 6 faulty units would exceed our assumed limit. Consequently, u_4 is faulty, according to units u_2, u_3. We managed to classify all of the units in the system! Our classification is contradiction-free and compatible with the syndrome, thus it is another possible diagnosis.

Further experimenting with our example, we can state the following:

- No diagnosis exists if no more than one unit can be faulty.
- When at most 2 to 5 units can be faulty the above described diagnosis is the only solution.
- When at most 6 units can be faulty, there are two possible diagnoses: the one described above, and the "only unit u_4 is fault-free" solution.

System-level diagnosis tries to solve such and similar problems in the form of a general algorithm. Many approaches to generalize the diagnostic task were developed over the years. One of the first methods was presented for one-step t-fault diagnosable systems by Kameda, et al. [30]. They treated the syndrome decoding as a state-space search. Their state-space consists of the different unique fault state to unit assignments. The search terminates when a complete unit classification is achieved. The control algorithm assures that the generated diagnosis is unique, free of contradictions, and consistent with the syndrome.

At the beginning an arbitrary unit is selected and assigned to be fault-free: $F(u_i) \leftarrow f^0$ (explicit assignment). Then two sets, the 'implied fault-free' L_i^0 and 'implied faulty' L_i^1 sets are calculated. The sets are initialized to $L_i^0 \leftarrow u_i$ and $L_i^1 \leftarrow \emptyset$. New elements are appended to the sets using the following implication rules:

The algorithm encounters a contradition when the $L_i^0 \cap L_i^1 \neq \emptyset$ condition is met during the generation of the sets. The explicit assignment $F(u_i) = f^0$ is not consistent with the syndrome, so it is changed to $F(u_i) \leftarrow f^1$ (forced assignment), $L_i^0 \leftarrow \emptyset$ and the L_i^1 set is recomputed. Another condition to evaluate is the number of the faulty units found so far: $|U_f| = |L_i^1 \cup L_j^1 \cup \cdots \cup L_k^1|$. If this number is below the diagnostic t limit, the units in L_i^0 and L_i^1

Unit	Test outcome	Implication
$u_j \in L_i^0$	$a_{jk} = 0$	$L_i^0 \leftarrow L_i^0 \cup u_k$
$u_j \in L_i^0$	$a_{jk} = 1$	$L_i^1 \leftarrow L_i^1 \cup u_k$
$u_j \in L_i^0$	$a_{kj} = 1$	$L_i^1 \leftarrow L_i^1 \cup u_k$
$u_j \in L_i^1$	$a_{kj} = 0$	$L_i^1 \leftarrow L_i^1 \cup u_k$

are tentatively labeled as fault-free and faulty, respectively. If it exceeds the limit, again the last explicit assignment is changed to $F(u_i) \leftarrow f^1$, $L_i^0 \leftarrow \emptyset$ and L_i^1 is recomputed.

Forced assignments are sure classifications and cannot be changed. Normal explicit assignments, however, are just supposed fault states and eventually they need to be changed. When such an assigment is changed its consequences must be cancelled (backtracking step). Therefore the algorithm maintains a stack structure to store the explicit assignments and the corresponding L_i^0 and L_i^1 implied sets. Due to the above 'branch and bound' technique, the algorithm traverses the state space in a depth-first manner. The search terminates when either all of the units were assigned a fault state (diagnosis is complete), or a backtracking would be necessary and the stack is empty (diagnosis is impossible).

Meyer and Masson published a much simpler algorithm for the same problem in [46]. They store the explicit assignments and their consequences in a matrix representation to avoid backtracking. Forced assignments are not used, the conditions of one-step t-diagnosability guarantee the solution to be unique anyway. Their algorithm also requires a specific topology to work: the testing graph is a $D_{1,t}$ interconnection design. However, the algorithm can be easily adopted to other testing structures.

The consequence matrix $\mathbf{B}$ is a binary matrix: $\forall i, j : b_{ij} \in 0, 1$. It is initially empty. The B_i row vector contains the implications drawn from the explicit assignment $F(u_i) \leftarrow f^0$. First the b_{ii} elements in the main diagonal of $\mathbf{B}$ are set to 0. Then the row vectors are filled one-by-one. Let us imagine that the $B_0, B_1, \ldots, B_{i-1}$ vectors are already complete. The b_{ii} element of B_i is 0, reflecting that unit u_i is supposed to be fault-free. Take the test results local to unit u_i: $a_{ij}, j \in i+1, i+2, \ldots, i+t$. Let $b_{ij} = a_{ij}$ until $a_{ij} = 1$, including the first passed test, say a_{ik}. Now take the test results local to u_k: $a_{kl}, l \in k+1, k+2, \ldots, k+t$ and repeat the previous procedure. It is terminated if either the row vector is completely filled, or the number of 1s in the row reaches t (in this case the remaining row vector elements are set to 0).

The completed **B** consequence matrix has the following two properties:

1. The row vectors corresponding to fault-free units are identical. The b_{ij} element of these vectors reflect the correct fault state of unit u_j.

2. The elements of row vectors corresponding to faulty units contain arbitrary values.

Dahbura and Masson approached the diagnostic problem from a purely graph-theoretical viewpoint [12, 13]. Their solution is composed of three main steps. In the first step they generate the so called 'implied faulty' graph. Then a maximum matching of this graph is computed. Finally, a labeling method is applied to the graph, where the labels represent the classification of the corresponding units.

The 'implied faulty' G^{01} graph is an undirected graph, which contains an edge from each node u_i to the nodes corresponding to units in the 'implied faulty' L_i^1 set of unit u_i. The 'implied faulty' set can be created from the syndrome using the following equation:

$$L_i^1 = \Delta_1(D_0(u_i)) \cup A_0(\Delta_1(D_0(u_i))).$$

The authors propose a more effective method to obtain the information required to construct the G^{01} graph. Let $\mathbf{X}$ be an $n \times n$ matrix such that $x_{ij} = 1$ if and only if $a_{ij} = 0$ or $i = j$, and 0 otherwise. It follows that $\hat{\mathbf{X}}$, the transitive closure of $\mathbf{X}$ is an $n \times n$ matrix in which $x_{ij} = 1$ if and only if $u_j \in D_0(u_i)$. Let $\mathbf{Y}$ be an $n \times n$ matrix such that $y_{ij} = 1$ if and only if $a_{ij} = 1$ or $a_{ji} = 1$, and 0 otherwise. Then the matrix $\mathbf{Z} = \hat{\mathbf{X}} Y \hat{\mathbf{X}}^T$ is an $n \times n$ matrix where $z_{ij} > 0$ if and only if $u_j \in L_i^1$. The so obtained $\mathbf{Z}$ matrix is the adjacency matrix of the graph G^{01}.

The 'implied faulty' graph is a bipartite graph due to the rule of symmetry: for $u_i, u_j \in U, u_j \in L_i^1$ if and only if $u_i \in L_j^1$. The authors proved that this graph contains a maximum matching which saturates each node corresponding to faulty units in a one-step t-diagnosable system. The minimum vertex set of this maximum matching is the set of faulty units. The algorithm employs a simple labeling method to determine the minimum vertex set. Those units not saturated by the maximum matching can be immediately labeled as fault-free, and placed into a queue. One of these units, say unit u_i is then removed from the queue. If u_i is labeled as fault-free, then the units adjacent to it are labeled as faulty and placed into the queue. If u_i is faulty, then the unit adjacent to u_i in the maximum matching is labeled as fault-free and placed into the queue. This procedure is repeated until the queue is empty, that is the labeling is complete. The generated node labeling constitutes the diagnosis.

A similar, but much more efficient algorithm was given by Dahbura et al. for a special subclass of t-diagnosable systems, the so called 'self-implicating' structures. The faulty unis in a self-implicating system can always be identified using at least one of two straightforward criteria associated with the test outcomes. In each case, the given faulty units in effect implicates itself as faulty. A unit $u_i \in U$ is diagnosable as faulty if

$$|\Delta_1(D_0(u_i)) \cup A_0(\Delta_1(D_0(u_i)))| > t,$$

and/or

$$u_i \in A_0(\Delta_1(D_0(u_i))).$$

Since the problem of characterizing systems in which faulty units satisfy these two conditions becomes too unwieldy, the authors defined a unit u_i to be 'self-implicating' if and only if

$$|\Delta_1(u_i)| > t,$$

and/or

$$u_i \in A_0(\Delta_1(D_0(u_i))).$$

Note, that the first condition of self-implication is a stronger version of the first condition of faultiness. Several necessary and sufficient condition for a system to be 'self-implicating' were also given. These are also stronger and harder to verify than those required for general diagnosability, the cost of satisfying such conditions results the benefit of a more efficient diagnosis procedure.

For a given syndrome in a t-self-implicating system represented by $G = (U, E)$ consider the directed graph $G_0 = (U_0, E_0)$ defined as $U_0 \equiv U$ and $E_0 = (u_i, u_j) : a_{ij} = 0$. Let $C_1, C_2, \ldots, C_r$ be the vertex sets of the strongly connected components of G_0, and $U(C_i)$ denote the set of units belonging to C_i. It is clear from the discussion in the beginning of this section that all of the units in $U(C_i)$ are either uniformly fault-free or uniformly faulty. Define $G_{C0} = (U_{C0}, E_{C0})$ to be a directed graph such that $U_{C0} = C_1, C_2, \ldots, C_r$, and $E_{C0} = (C_i, C_j) : \exists u_k \in C_i, u_l \in C_j, C_i \neq C_j \text{such that} a_{kl} = 0$. Since G_{C0} is a directed acyclic graph, there exists a node in U_{C0} with out-degree 0, called a *sink*. Define $G_{C1} = (U_{C1}, E_{C1})$ to be an undirected graph in which $U_{C1} = C_1, C_2, \ldots, C_r$, and $E_{C1} = (C_i, C_j) : \exists u_k \in C_i, u_l \in C_j \text{such that} a_{kl} = 1$.

Note, that if there exists a node $C_i \in U_{C1}$ for which $(C_i, C_i) \in E_{C1}$, then the units contained in $U(C_i)$ are faulty. Otherwise, for each sink $C_i \in U_{C1}, (C_i, C_i) \notin E_{C1}$ the units contained in $U(C_i)$ are faulty if and only if

$$\sum_{C_j \in N(C_i)} |U(C_j)| > t. \tag{5}$$

The algorithm based on the above construction combines the results of Dahbura and Masson [12, 13], and the KTA algorithm [30]. Given a t-self-implicating system and a syndrome, it first creates G_{C0} and G_{C1}, and a queue containing the sinks in G_{C0}. For each component $C_i \in U_{C1}, (C_i, C_i) \in E_{C1}$ the units in $U(C_i)$ are declared to be faulty, and C_i is deleted from G_{C0}, G_{C1}, and the queue. Newly formed sinks in G_{C0} are added to the queue. For each sink contained in the queue the inequality 5 is tested. If it holds, then the units in $U(C_i)$ are declared to be faulty; otherwise they are fault-free and the units located in $N(U(C_i))$ are declared to be faulty. In both cases C_i is deleted from G_{C0}, G_{C1}, and the queue, and newly formed sinks in G_{C0} are added to the queue. If the units in $U(C_i)$ were declared to be faulty, then the t upper bound on the number of faulty units is decreased by $|U(C_i)|$. The previous procedure is repeated until the queue is empty, when the classification of units is completed.

One common underlying assumption in the traditional methodology of system-level fault diagnosis is that all of the system components have an equal probability of failure. Maheswari and Hakimi [43] first considered the probabilistic nature of the occurrence of faults in multiprocessor systems. They introduced a probabilistic measure ρ and presented necessary and sufficient conditions for a system to be *probabilistically t-diagnosable* or *rho-diagnosable*. If the failures are assumed to be independent and the *a priori* probability of failure of unit u_i is denoted as $p(u_i)$, then the *a priori* probability that $F \subseteq U$ is the set of faulty units in the system is

$$P(F) = \prod_{u_i \in F} p(u_i) \prod_{u_i \in U-F} (1 - p(u_i)).$$

A system is then ρ-diagnosable if for any syndrome there exists a unique allowable fault set $F \subseteq U$ such that $P(F) \geq \rho$ assuming that the *a priori* probability of occurrence of the set of faulty units is at least ρ. Therefore, in a ρ-diagnosable system

$$\log(P(F)) = \sum_{u_i \in F} \log(p(u_i)) + \sum_{u_i \in U-F} \log(1 - p(u_i)) \geq \log(\rho).$$

It follows that

$$\sum_{u_i \in F} \log \frac{1-p(u_i))}{p(u_i)} \leq -\log(\rho) + \sum_{u_i \in U} \log(1-p(u_i)) = K(\rho).$$

If with each unit u_i a weight $W(u_i) = \log[(1-p(u_i))/p(u_i)]$ is associated, then the problem of locating the set of faulty units is equivalent to that of determining the allowable fault set $F' \subseteq U$ for which

$$W(F') = \sum_{u_i \in F'} W(u_i) \leq K(\rho).$$

It is assumed that $p(u_i) < 1/2$ for all $u_i \in U$, so $W(u_i) > 0$. Thus, the diagnosis problem is to determine the unique allowable fault set of minimum weight in the system. However, the problem of actually identifying of this fault set on the basis of a valid collection of test results in a ρ-diagnosable system remained open.

Dahbura was first to solve the above problem. He presented an algorithm for locating the most likely faulty units in a ρ-diagnosable system. The approach is a generalization of the diagnostic algorithm given in [12]. The first step is the construction of of a directed bipartite graph $B = U_B with bipartition such that$X

One-step implications are stored in a $2n \times 2n$ $\mathbf{M}$ hypermatrix. The $\mathbf{M}$ matrix consists of four $n \times n$ binary minormatrices: $\mathbf{M}^{00}$, $\mathbf{M}^{01}$, $\mathbf{M}^{10}$, and $\mathbf{M}^{11}$. The $\mathrm{m}^{xy}[i,j]$ element of the $\mathbf{M}^{xy}$ minormatrix $(x, y \in \{0,1\})$ is 1, if there exists an $f_i^x \to f_j^y$ one-step implication between units u_i and u_j.

Transitive closure can be computed by the logical closure of the $\mathbf{M}$ matrix. Units corresponding to "1"s in the main diagonal of $\mathbf{M}^{01}$ and $\mathbf{M}^{10}$ are involved in a contradiction, and so they can be surely classified. For other u_i units, the sum of $f_j^0 \to f_i^0$ and $f_j^0 \to f_i^1$ implications is counted. These implications are stored in the jth column of the $\mathbf{M}^{00}$ and $\mathbf{M}^{01}$ matrices, so if $\sum_j \mathrm{m}^{01}[j,i] > \sum_j \mathrm{m}^{00}[j,i]$ the unit is diagnosed as faulty, otherwise it is fault-free.

Later, Pataricza et al. presented a novel modelling technique for system-level fault diagnosis in massive parallel multiprocessors, further enhancing the theoretical basis of the Selényi algorithm. The authors re-formulated the problem of syndrome decoding to a constraint satisfaction problem (CSP). The CSP-based approach is able to handle detailed and inhomogeneous functional fault models on a level similar to the Russel-Kime model [57, 58].

Multiple-valued logic is used to describe system components having multiple fault modes. The granularity of the model can be adjusted to the diagnostic resolution of the target without altering the methodology.

Constraint satisfaction problems (CSP) deal with the estimation of a single or all consistent solutions in large-scale relation systems. More formally, a CSP is a (X, D, C) tuple, where $X = X_1, \ldots, X_n$ is a set of variables, defined over the set $D = D_1, \ldots, D_n$ domains; and $C = C_1, \ldots, C_k$ is a set of constraints. Constraints are relations between the variables. A solution of a CSP is a vector $[x_1, \ldots, x_n]$ of values that satisfies all the constraints.

The ultimate goal of syndrome decoding and CSP is very similar: the algorithm must classify the fault state of system components in a consistent way that conforms to the given diagnostic model, test invalidation rules and the actual outcomes of the elementary tests (syndrome elements). These restrictions can be represented as binary relations between the fault state of the tester and the tested units. The use of relations instead of logical functions supports the handling of diagnostic uncertainty appearing in some test invalidation models. A diagnosis problem can be very easily reformulated to a constraint satisfaction problem. The variables of the CSP represent the fault states of the system components. The constraints correspond to the restrictions derived from the test invalidation model and the current syndrome elements, thus the test invalidation rules determine a set of candidate relations, from which the actually used one has to be selected, when knowing the actual test outcome.

Testing of system components is done by time-out protected mutual "I'm alive" messages, sent periodically between neighboring processors. An extended PMC-like test invalidation is used to describe the fault state-test result relationships.

Syndrome decoding is driven by implication rules, represented by constraints. They originate from the system structure, the test invalidation model and the actual test outcomes. All of the constraints are binary over the fault state of the tester and the tested component, to achieve a greater simplicity: the test results (syndrome bits) are eliminated from them as variables after receiving the test outcome. The generated constraint network is then preprocessed. The applied simple consistency algorithms greatly reduce the amount of comuptation required to solve the CSP problem. After the preprocessing phase the CSP solver is executed. When all of the constraints are satisfied, the domains of the variables corresponding to the system units contain the possible classifications for the units, in other words, the transitive closure. If not all the components have a sure classification in the transitive closure, the remaining ones are classified using further restrictions on the fault model (limitation on maximal number of faults, exclusion of certain faults).

The diagnostic algorithm has the ability to create diagnosis "on-the-fly", using only partial syndrome information. In this case the diagnostic part is invoked several times at the arrival of partial test results. Implications from new syndrome elements are added to this dynamic constraint network during each call and a path-consistency based preprocessing of the network is performed, successively restricting the solution domain. As a result, the algorithm will extract all of the useful diagnostic information available in the partial syndrome, and the diagnosis is improved at the arrival of each new test result.

If one-pass diagnosis is required, simply a static binary CSP is produced after performing all tests. However, in the case of on-the-fly diagnosis, newly generated test results must be processed immediately. Only a few syndrome elements are present at the beginning, so the complete set of relations cannot be created at once. Every incoming test result adds new relations to this set, while the previously constructed ones remain still valid. Thus the solution space is gradually reduced, in other words a monotone dynamic CSP is produced.

As mentioned, decreasing the complexity of the CSP solution be achieved by preprocessing the constrint network. The so-called consistency algorithms exclude locally inconsistent value combinations from the domains of variables, since these values surely cannot appear in a globally consistent solution. These methods work generally only on binary CSPs, because every variable can be evaluated independently in this subclass of CSPs. Moreover, such an evaluation of the variables guarantees the global consistency as well. Consistency algorithms can be grouped according to the number of the nodes they consider during an elementary step, while searching for local inconsistencies:

- Node-consistency or 1-consistency considers only a single node at a time. It simply checks unary constraints, and deletes all values not complying to them. As unary constraints can be previously eliminated by restricting the domains, this algorithm is used only as a supplementary step in more complex algorithms.

- Arc-consistency or 2-consistency considers at a time two variables X_i and X_j and a binary relation R_{ij} between them. Every value is excluded from the domain of X_i without a value of X_j satisfying R_{ij}.

Full consistency can be achieved by checking appropriate vertex pairs and relations.

- Path consistency or 3-consistency algorithms check the transitivity of the consistency of a candidate value assignment, thus path consistency between two variables X_i and X_j connected by a binary relation R_{ij} means that all value pairs in a solution allowed by R_{ij} must be also allowed by all paths between X_i and X_j. The entire constraint network is path consistent if every vertex pair is path consistent. Full path consistency in a complete constraint graph is equal to the consistency for length 2 paths. Checking path consistency requires only the checking of all length 2 paths, since any constraint network can be virtually extended to a complete constraint graph by inserting dummy ("always true") constraints.

- k-consistency examines a set S_k of k variables at a time. If a completely consistent subset of value $k-1$ tuples exist on $S_{k-1} \subset S_k$, then any value from the domain of the kth variable can be eliminated which cannot form a consistent value set with any one of the consistent $k-1$ tuples. Global consistency can be achieved by successive elimination for increasing values of k until all variables are involved or a domain becomes empty indicating the insatisfiability of the CSP.

Distributed diagnosis

In massively parallel computing systems several problems arise employing diagnosis with a centralized control:

- The space and time complexity of the diagnosis algorithm increases. The processors cannot transmit their local test results to the central observer simultaneously, thus the communication phase becomes longer. The diagnosis can take place only after the complete syndrome has been collected by the central observer. The computing power of the parallel processors is not utilized during diagnosis (the central observer executes the diagnostic algorithm alone). Yet, the system cannot proceed with its normal tasks, since diagnostic results are necessary to validate the correctness of the computation completed up to that point.

- The information storage and processing capacity requirements of the central unit grow. The time of generating the larger diagnostic image becomes longer as well. Let the system employ table-look-up diagnosis, then $O(n)$ new tests are required and $O(nt)$ new fault patterns may occur, assuming a one-step t-fault diagnosable system was extended by n additional units. The increase in the size of the look-up table is $O(nt+1)$. Using a binary search algorithm to retrieve the look-up table entries, $O(\log(nt+1))$ extra steps are necessary to locate a certain fault pattern in the average case. The amount of memory in the central observer is finite, and the time available for the diagnosis process is also bounded (since the diagnosis is almost always a real-time task). The existing methods to overcome these two limitations have contradicting effects. To reduce the memory space used, data compression can be applied. Unfortunately, data representations taking up smaller space in memory can be handled only with complex, computation-intensive algorithms. On the other hand, a large amount of fast access memory is required to speed-up diagnosis. A good compromise for both parameters is hard to find [32, 44].

- Due to the finite amount of resources in the central observer, it tends to become overloaded and hence diagnosis might be inefficient or even impossible.

- The central observer is exclusively responsible for locking out faulty components, in other words it is a diagnostic hard-core. A failure in this hard-core causes a single-point failure which is inadmissible in a fault-tolerant system.

Since massively parallel systems are distributed, there is an opportunity to use a *distributed observer* approach, where no global monitoring and control is present. There exists no central facility in the coordination or mediation among the processors. The distributed nature of such a diagnostic process requires that the units are able to make local decisions regarding certain tasks like scheduling, resource allocation, etc. These decisions can be based only on the information obtained via normal, unit-to-unit communication links. Moreover, any action taken in response to the detection and location of a fault must be local, and may again involve only normal communication.

Any synchronized or voted effort among units to physically or logically remove a failed node is impossible, due to the autonomy of units, the lack of tight synchronization, and the limited nature of communication facilities. Any physical equipment employed for removing a faulty component would constitute a potential single-point failure (even if it were to act only in response of a voted effort among several processors). Consequently, all responsibility for fault detection and subsequent removal of a unit should not be assigned to any single device. The reason of this observation is that if such liability is localized, then a failure of the single equipment implementing the detection and removal mechanism may make the system incapable of protecting itself from subsequent failure in the unit reviewed by the equipment. In this sense the entire system is dependent upon the unit supervising the othr one, despite of the localized nature of the responsibility.

To avoid the above problems a concept of *distributed fault-tolerance* was proposed by Kuhl and Reddy [34]. As its name implies, distributed fault-tolerance allows diagnosis and subsequent isolation of faulty processing elements in a way that does not centralize the failure handling devices in the system. The requirements for fully distributed fault-tolerance are as follows: each fault-free unit must be able to independently achieve correct diagnosis of all other units in the system. Once a unit has recognized the failure of another one, it can "discriminate" against the faulty processor simply by refraining from further cooperation. Since all fault-free units locate the fault(s) and refuse to interact with the faulty component(s), the erroneous part is in effect isolated from the rest of the system.

According to the authors, most of the problems with testing can be solved by incorporating means of fault-tolerance into the architecture of the processing elements. First of all, a processor may be constructed in a way, that it tolerates and recovers from certain faults within itself. These failures can be left out of consideration on the system level. Also it can include self-testing circuitry, together with checkers monitoring various data and control paths to detect errors during normal operation. Note however, that it is not desirable to localize the entire responsibility for error detection and subsequent fault isolation within the processor itself, since the failure of this component would then render the whole system unreliable. Therefore, the application of additional fault detection techniques like error correcting codes and digital signature algorithms can be useful. The protection of inter-processor messages with coding methods assures both information integrity and validation of correct operation of the self-testing component. And so the execution of a test on a unit u_j by unit u_i is reduced to a simple interrogation of the self-test circuitry in u_j and obtaining the value a_{ij}. It is assumed that self-tests are valid and complete, and fault-free units collect and distribute test results correctly. Faulty processors may behave unpredictably, therefore the diagnostic information reported by such units are unreliable and useless.

In other words, the distributed fault-tolerance concept uses symmetric test invalidation model with distributed control.

A processing element is capable of testing only neighboring units. The local test results must be broadcasted throughout the whole system to enable the diagnosis of remote units. All of the messages are forwarded through the interconnection network. Units may rely only on indirect information passed via a neighbor in order to obtain the condition of non-neighbors. Since the model allows a faulty processing element to transmit incorrect test results, or alter and even destroy the information routed through it, fault-free units may only trust data received from neighbors they "personally" tested and found to be fault-free. Cooperation and communication with faulty units is forbidden. For this reason, fault-free processors perform a test before transmitting and after receiving a message to a certain neighbor. If the neighbor was found to be faulty, the message is not sent or the received message is ignored. Due to this mechanism, all of the messages are distributed over a path of fault-free processors to assure their reliability.

Kuhl and Reddy introduced a new diagnosability measure for distributed multiprocessor systems: *t-fault self-diagnosability.* A system is considered to be t-fault self-diagnosable if and only if each fault-free processing element can correctly identify all faulty (and fault-free) units in the system, provided that no more than t faults are present. To formalize conditions for t-fault self-diagnosability, for a given syndrome consider the directed graph $G_0 = (U_0, E_0)$ defined as $U_0 \equiv U$ and $E_0 = (u_i, u_j) : a_{ij} = 0$. The different strongly connected components of G_0 (containing units of the same fault state) do not have any common vetrices. This is a significant theorem in the field of distributed diagnosis. It implies that if there are more than one strongly connected components in G_0, the system is cut up into isolated areas of fault-free units that cannot exchange information. In these conditions it is impossible to generate a system-wide diagnostic image. Therefore a multiprocessor system with a testing graph $G = (U, E)$ is t-fault self-diagnosable on the basis of the distributed fault-tolerance concept if and only if $\kappa(G) > t$.

It is interesting to note that the level of diagnosability is tied solely to the connectivity of the testing graph. This implies that in designing a multiprocessor system, which is to have a given level of self-diagnosability, the designer must only assure that the interconnection structure is rich enough to support a testing graph of the required connectivity. On the other hand, massive parallel systems generally have a regular interconnection structure with the communication speed being primary objective in the design process.

The condition of t-fault self-diagnosability gives a simple way to determine the maximum number of tolerable failures in such systems. The connectivity of the system graph may also allow a higher level of self-diagnosability than really necessary (e.g. an a priori knowledge about the system suggests that the probability of more than t faults present simultaneously in the system is negligible). In this case, a testing assignment with lower connectivity will reduce both the testing load on individual units, and the number of diagnostic messages spreading over the multiprocessor network.

The algorithm SELF presented by Kuhl and Reddy in [34] conducts the testing in the system in rounds. A round of testing consists of the execution of each test represented by the testing graph. It is assumed, that no failures occur in the system from starting a round of testing till every fault-free processor completes its diagnosis. This assumption means that the diagnostic algorithm analyzes a static "snapshot" of the status of different components at a fixed point of time.

In the algorithm SELF each unit u_i computes a fault vector containing its conclusions about the condition of each other unit. Initially, the local test results are obtained. The units then label themselves as 'good' and all other units as 'unknown' in their fault vector. They examine and record the local test results in the fault vector. Every unit u_i prepares a message containing the local test results for broadcasting to each $u_j \in \Gamma^{-1}(u_i)$. A unit u_j records any new test results received from a neighbor that it tested and found to be fault-free in the fault vector, and forwards these test results to each $u_k \in \Gamma^{-1}(u_j)$. The messages sent by units tested and found to be faulty are ignored. Note, that diagnostic information travels backward over the directed arcs in the testing graph via only fault-free units. This process iterates at each unit until either their fault vector is complete or contains indications of t faults. In the latter case, the remaining unspecified entries in the fault vector are filled with fault-free classification since at most t faults can coexist in the system.

The SELF algorithm developed for the distributed fault-tolerance concept had some imperfections. The algorithm did not take into consideration faults in the communication links and reintegration of repaired, replaced, or simply recovered faulty components was not supported. An improved algorithm called NEW_SELF dealing with these problems was introduced in [26, 27]. The new algorithm is based on the same theoretical concept, thus the same assumptions and notion presented above are used.

The validation of diagnostic messages possibly altered, destroyed or erroneously issued by faulty units has been changed. A unit u_i accepts a diagnostic message from a neighbor unit u_j only if $u_j \in \Gamma(u_j)$ and it has ascertained that unit u_j is fault-free. This is accomplished by the following mechanism: when unit u_j sends a diagnostic message, unit u_i temporarily stores this message in a buffer, provided u_i has found u_j to be fault-free last time it tested. Upon the next test on u_j, if the result indicates that it is still fault-free, unit u_i confirms all pending messages waiting in the buffer. Of course, this strategy requires the assumption that a unit cannot fail (undetectedly) and recover from the fault in the interval between two consecutive tests on it. If the self-testing processing elements are equipped with latch circuitry for storing error indications, then even transient faults can be detected by acquiring the condition of these lathes.

The possibility of reentry of previously faulty units and/or communication links, or entry of new units adds considerable complexity to the diagnostic process. This complexity is due to the necessity of keeping track of temporal ordering among different messages travelling over the interconnection network. To deal with this problem each unit contains a local clock, and messages incorporate a time stamp. This mechanism is similar to that commonly used in distributed systems for message sequencing and maintaining consistency in distributed databases, but it is complicated by the fact that a failure may cause the unit to lose its old clock value.

The algorithm maintains at each unit u_i two arrays: $\mathrm{ACCUSERS}_i$ and ENTRY_i. The elements of both arrays are sets of messages stored by u_i for diagnostic purposes. The jth element of these arrays is associated with the unit u_j. At any time, a fault-free unit u_i will have in $\mathrm{ACCUSERS}_i[j]$ a number of messages in the form (nil, $u_k(t_k)$, u_j) representing that unit u_k started a message at its local time t_k indicating that u_j was faulty. In the array element $\mathrm{ENTRY}_i[j]$ will be messages corresponding to tests of u_j that have passed. There are two types of these tests: ($u_l(t_l)$, $u_l(t_l)$, u_j) or ($u_l(t_l)$, $u_k(t_k)$, u_j), both indicating that u_l tested u_j at its local time t_l and found it to be fault-free.

A unit u_i tests a neighbor $u_j \in \Gamma(u_i)$ according to some locally determined schedule. In addition to periodically scheduled tests, u_i may test in response to some suspicion concerning unit u_j (e.g. not having communication with u_j for an abnormally long time) or in response to a request for entry from unit u_j (which is possibly added to the network or was repaired, replaced, or recovered from a transient failure). If u_i finds u_j to be faulty, it composes a message (nil, $u_i(t_i)$, u_j) and broadcasts it to each $u_k \in \Gamma^{-1}(u_i)$, where t_i is the local clock value of the time of test. This is the initiation of the reliable dissemination of the message throughout the network backward along paths in the testing graph. After each test performance u_i increments t_i. When u_i finds u_j to be fault-free it composes and broadcasts a message $(u_i(t_i),\ u_i(t_i),\ u_j)$. In addition, if u_i currently has in ACCUSERS$_i$ any message in the form (nil, $u_k(t_k)$, u_j) indicating that u_k found u_j to be faulty at its local time t_k, then it composes and broadcasts a message $(u_i(t_i),\ u_k(t_k),\ u_j)$ to its testers. This message is essentially an announcement that u_j has recovered from the failure detected by u_k.

When a unit u_m has received and validated an entry message $(u_i(t_i),\ u_i(t_i),\ u_j)$ or $(u_i(t_i),\ u_k(t_k),\ u_j)$, it searches the ACCUSERS$_m$ array for any message of the form (nil, $u_k(t'_k)$, u_j) or (nil, $u_k(t'_i)$, u_j) with $t'_k \leq t_k, t'_i < t_i$. Such messages can be ignored and removed, as they are indicative of failures which recovered. Then the entry message is forwarded to each $u_i \in \Gamma^{-1}(u_j)$ to continue its dissemination throughout the network. Entry messages associated with unit u_j are stored in ENTRY$_m[j]$. When a failure message (nil, $u_i(t_i)$, u_j) reaches u_m, it searches ENTRY$_m[j]$ for a message $(u_i(t'_i),\ u_i(t'_i),\ u_j)$ or $(u_l(t_l),\ u_i(t'_i),\ u_j)$ with $t'_i \geq t_i$. If such a message could be found, the failure message is no longer valid and is simply ignored. If not, u_m adds the failure message to ACCUSERS$_m[j]$ and also forwards it to each $u_i \in \Gamma^{-1}(u_j)$.

In order to prevent the ACCUSERS and ENTRY arrays from growing unboundedly, messages are added to these arrays only if they represent new and useful information. Any messages superseded by newer information are removed. For example, when u_m receives a message $(u_l(t_l),\ u_i(t_i),\ u_j)$ this message will be added to ENTRY$_m[j]$, if

1. there is not a message (nil, $u_i(t'_i)$, u_j) with $t'_i > t_i$ already in ACCUSERS$_m[j]$ (i.e. u_m has not received report from u_i of a failure in u_j occurring subsequent to local time t_i), and

2. there is not a message $(u_l(t''_l),\ u_i(t''_i),\ u_j)$ with $t''_i \geq t_i$ in ENTRY$_m[j]$ (i.e. u_m has not received an entry message from u_l indicating recovery of unit u_j from a failure detected by u_i at some time subsequent to local time t_i).

A unit uses similar criteria to determine whether it should or should not forward a message to its testers, thus ensuring finite message lifetimes (a message never passes through a unit more than once).

The actual diagnosis is accomplished by examining the ENTRY$_m$ array. The presence of a message (nil, $u_k(t_k)$, u_j) in ACCUSERS$_m[j]$ does not necessarily imply that u_j is faulty. It may also represent an old failure from which u_j has recovered, but for which there no entry message ($u_l(t_l)$, $u_i(t_i)$, u_j) will ever reach u_m. Similarly, the presence of a message ($u_k(t_k)$, $u_k(t_k)$, u_j) or ($u_k(t_k)$, $u_i(t_i)$, u_j) in ENTRY$_m[j]$ does not imply that u_j is fault-free, because it is possible that u_k has failed after sending the message, and subsequent to this u_j has also failed. However, u_m can discover which units are fault-free by performing a simple computation on the ENTRY$_m$ array. First, u_m can directly certify the condition of the units it has tested. Then, for any unit $u_i \in \Gamma(u_m)$ found to be fault-free, if there is a message ($p_i(t_i)$, $p_i(t_i)$, u_j) or ($p_i(t_i)$, $u_k(t_k)$, u_j), then u_j must be fault-free. Also, for any message ($u_j(t_j)$, $u_j(t_j)$, u_l) or ($u_j(t_j)$, $u_k(t_k)$, u_l), u_l must be fault-free, and so on. The diagnosis is achieved by a simple breadth-first like search of the ENTRY$_m$ array. Upon completition all of the fault-free units generate a correct and complete system-wide diagnostic image, provided the testing graph remains strongly connected when the vertices and edges corresponding to faulty units are removed.

The NEW_SELF algorithm also allows the classification of interconnection links. In order to take the inter-processor communication devices and their potential faults into consideration, the notion of a *link failure domain* was incorporated in the diagnostic model of the algorithm. A link failure domain of a distributed computing system is any single source of failure in the inter-processor communication devices. The *link failure domain set* is the set of all link failure domains for the network. The detection and location of interconnection link faults are accomplished in one of two ways:

1. If a test on a unit u_j performed by unit u_i fails, then it may be due to an error in u_j or in the communication path between u_i and u_j, or both. Therefore, the failure message (nil, $u_i(t_i)$, u_j) must be interpreted by the units receiving it as being possibly indicative of any of these situations.

2. A unit u_i may have difficulties accomplishing normal, non-diagnostic communication with a neighbor u_j due to a faulty inter-processor link. In such a case, if $u_i \in \Gamma^{-1}(u_j)$ it can broadcast a message identical to any other failure message (nil, $u_i(t_i)$, u_j). If $u_i \notin \Gamma^{-1}(u_j)$, then the generated message has the form of (link, $u_i(t_i)$, u_j).

A processor u_m, receiving a failure report (nil, $u_i(t_i)$, u_j) initially treats the message as indicative of a fault in processor u_j, and adds it to ACCUSERS$_m[j]$. If, in fact, u_j is fault-free and only the communication path between u_i and u_j is faulty, then u_m should eventually receive an entry message ($u_k(t_k)$,

$u_i(t_i)$, u_j). Note, that this message is generated when the message (nil, $u_i(t_i)$, u_j) reaches $u_k \in \Gamma^{-1}(u_j)$. Upon receiving this message u_m removes the entry indicating the failure of u_j from ACCUSERS$_m[j]$, and adds a new entry to the array FAULTY_LINKS$_m$.

In the event of repair of the faulty link, the message (link, $u_i(t_i)$, u_j) will be removed from FAULTY_LINKS$_m$ when u_m receives a valid message of the form ($u_i(t_i')$, $u_k(t_k)$, u_j) with $t_i' > t_i$. In case of $u_i \notin \Gamma^{-1}(u_j)$ and u_i certifies that it is no longer having problems communicating with u_j, it sends a message (link-up, $u_i(t_i)$, u_j). This message is distinct from a normal entry message for u_j, since u_i is no longer capable of confirming the correct operation of u_j. It is only stating that u_i does not have communicational difficulties caused by a faulty interconnection path any more. Of course, messages indicating a fault in a communication link may arrive to a certain unit later than messages reporting the subsequent recovery of that link. Hence, the most recent link-up messages are also stored in the ENTRY$_m$ array and before adding a message (link, $u_i(t_i)$, u_j) to FAULTY_LINKS$_m$, unit u_m first checks whether a message (link-up, $u_i(t_i')$, u_j) or ($u_i(t_i')$, $u_i(t_i')$, u_j) with $t_i' > t_i$ already exists in ENTRY$_m[j]$. If so, the arriving message is ignored.

An adaptive diagnosis approach based on the Meyer and Masson algorithm [46] was introduced by Bianchini and Buskens int [4, 5]. The algorithm, called Adaptive DSD, is similar to the centralized adaptive methods in that the testing assignment is determined by the actual fault pattern present in the system. The algorithm is only applicable for systems with a fully connected interconnection topology (these include bus-oriented systems or networked workstations). Unit failures and repairs are considered; link failures are not. The number of faulty units in the fault set is not bounded, the fault-free processing elements correctly diagnose the fault state of all units in the system.

In the algorithm each unit u_x maintains a local array called TESTED_UP$_x$. TESTED_UP$_x$ contains n elements, indexed by a unit identifier $i \in 0, 1, \ldots, n-1$. Each element of TESTED_UP$_x$ contains a unit identifier. The entry TESTED_UP$_x[i] = j$ indicates that u_x has received diagnostic information from a fault-free unit specifying that unit u_i has tested u_j and found it to be fault-free. Figure **??**b describes the TESTED_UP$_2$ array generated at unit u_2 in the eight unit system with fault vector (f_1, f_4, f_5) present, shown on **??**a.

Each unit first tries to identify another fault-free unit. List the units in sequential order: $u_0, \ldots, u_{n-1}$. Unit u_x identifies the first fault-free unit sequentially after itself in this ordered list. This is accomplished by testing consecutively units $u_{x+1}, \ldots, u_{x+n-1}$ (modulo n) until a fault-free unit, say unit u_y, is found. Then u_x sends a request to u_y to transmit back the contents of its local TESTED_UP_y array. A test on the unit u_y is assumed to pass, if u_y has remained fault-free since the last test performed by u_x including the period required for unit u_y to send the TESTED_UP_y array. This guarantees that the diagnostic information represented by TESTED_UP_y is accurate. This information is utilized to update the local data structures of unit u_x in the following manner:

1. the entry $\text{TESTED\_UP}_x[x]$ is set to y, specifying that unit u_x has tested unit u_y and determined it to be fault-free,

2. all other elements of TESTED_UP_x are updated to the values of corresponding entries in TESTED_UP_y.

Note, that the diagnostic information in the TESTED_UP arrays are communicated among units in the reverse direction of tests.

Diagnosis is achieved at unit u_x by following the fault-free paths from u_x to other fault-free units. Results of the analysis of TESTED_UP_x are recorded in array STATE_x, where the entry $\text{STATE}_x[i]$ represents the diagnosed fault state of unit u_i. Initially, all units are classified as faulty in STATE_x. a variable called *node_pointer* (denoted by NP) is set to x, the identifier of the unit performing the diagnosis. The algorithm traverses the forward fault-free paths in the test set, labeling each of the units visited as fault-free. This is accomplished by marking the unit represented by $\text{STATE}_x[NP]$ as fault-free. Then setting NP is set to $\text{TESTED\_UP}_x[NP]$, which is the sequentially next fault-free unit in the ordered list. This procedure is repeated until NP is set to every fault-free unit and returns to x.

Later the same authors applied the Adaptive DSD algorithm for systems using a comparison-based unit validation scheme. They omitted explicit tests as impractical to detect arbitrary failures. Instead, a unit monitors the communication of two other units to decide whether they are reliable or not. Two new models for the behavior of faulty units were defined. Online distributed diagnostic algorithms that operate correctly under the new failure semantics were also given.

The standard "black box" method for testing consists of applying the input test sequences at the control points and observing the outputs of the tested device. The test passes when the outputs conform to the specification of the system given its inputs. In the model of Buskens and Bianchini, the input of each unit corresponds to diagnostic messages the unit receives, and its outputs are the diagnostic messages the unit generates. Unit u_i attempts to validate the operation of unit u_j by observing its input and output messages. However, u_i may not able to directly observe the input messages of u_j. In this case it is required to request this information from any unit that sends messages to u_j (e.g. unit u_k). Thus, u_k transmits messages to two different units. There is no guarantee that the two messages will be identical, unit u_k may send message m_1 to u_j and message m_2 to u_i. There are two classes of faulty units depending on the relation of m_1 to m_2: *consistent liars* always send incorrect diagnostic information on response to a request; *inconsistent liars* arbitrary disseminate correct and incorrect diagnostic information. Faulty units that distribute correct diagnostic information are indistinguishable from fault-free units. Explicit tests are not used to detect faults, redundant message passing provides evidence of faulty behaviour instead.

There are two variations of the so called *Robust* algorithms for systems with consistent and inconsistent liars. Every message transmitted by fault-free units during the operation of the algorithms is assumed to be delivered correctly. The receiver of a message can identify the sender of the message. The absence of a message can be detected. Every unit can communicate with every other unit (fully connected topology).

The version of Robust for systems with only consistent liars is very similar to the Adaptive DSD algorithm. If there is only one faulty unit, then the algorithm tries to identify a unique fault-free unit and update the local Syndromes array with the information received from that unit. (The Syndromes array is equivalent to the TESTED_UP array of the Adaptive DSD algorithm.) Units are listed in sequential order: $u_0, \ldots, u_{n-1}$. Unit u_x requests Syndromes information from the next two consequtive units in the ordered list: u_y, u_z. If u_y does not validate with u_z, then u_x requests the Syndromes information from a subsequent unit u_w and identifies the first unit that validates with u_w as fault-free. Diagnostic information from the validated unit is then used to update the local Syndromes array. By this mechanism, the contents of the Syndromes arrays at different nodes will be identical to the TESTED_UP arrays of the Adaptive DSD algorithm for symmetric test invalidation. Consequently, processing of the Syndromes array to achieve diagnosis of each unit is done with the same method of the TESTED_UP array described before.

Three conditions must be satisfied by u_a to be validated by u_b. The first is that u_a must validate u_b, that is the Syndromes$_a$ array local to unit u_a must contain u_b. Second, u_a and u_b must agree on the information generated by the remaining system units. Third, the Syndromes$_a$ information must not agree with the Syndromes data from any node $u_a < u_i < u_b$. Otherwise u_a is faulty, because it should identify u_i as fault-free.

The above procedure works if there are no more than one faulty unit in the system. It can be generalized to handle at most t consistent liars present by finding a validation path of length t. This ensures that the first unit in the path is fault-free.

Diagnosis of inconsistent liars is complicated by the fact that an inconsistent liar may forward valid information in addition to corrupt information. Therefore, in systems with inconsistent liars it is not always possible to uniquely identify a fault-free unit. To ensure that diagnosis is always correct in such situations, multiple units may be validated by a single unit. Therefore, each fault-free unit validates with at least one other fault-free unit. Suppose, there is no more than one faulty unit in the system. The version of Robust for systems with only inconsistent liars first attempts to identify a unique fault-free unit. If it fails, it evaluates two units as fault-free. In either case, the Syndromes array is updated from the found fault-free unit(s).

Each unit u_x executes the algorithm at predefined testing intervals. First, u_x selects the next three consequtive units following it in the ordered unit list: u_y, u_z, u_w. Unit u_y is identified as fault-free if it is validated by unit u_z. Otherwise, exactly one of u_y, u_z is faulty and u_w is fault-free. In this case there are three possible fault situations:

1. u_y is faulty and lies to u_x,

2. u_z is faulty and lies to u_y, but not to u_x, and

3. u_z if faulty and lies to u_x.

If only u_y or u_z validate with u_w, the other unit is the faulty one and the Syndromes array is updated from u_y or u_z, respectively. If both of the units validate with u_w, then u_y and u_z are indistinguishable as the faulty unit and u_x must validate both of them. For this reason, the definition of the Sydromes array is modified. A single entry in u_x.Syndromes$[i] = j$ specifies that u_x received information that u_i classifies u_j as fault-free. An entry containing multiple unit identifiers in u_x.Syndromes$[k] = l \ldots m$ indicates that u_k detected inconsistent behavior among $u_l, \ldots, u_m$, but it is unable to identify the faulty unit(s). One of the units in $u_l, \ldots, u_m$ must be fault-free. Diagnosis is incomplete (but not incorrect), because the faulty units was not located. Note, that the lies u_y or u_z propagates to u_x in our example must be very specific: u_y (u_z) can only lie about the Syndromes entry of u_z (u_y) without being detected as faulty.

The algorithm can be generalized to handle at most t inconsistent liars similarly to the case of consistent liars. A validation path of length t must be found to guarantee that at least one fault-free unit is identified. Faulty units may appear on the path if they lie inconclusively. As uch, fault-free units may exist that are not on the validation path. To ensure that fault-free units are not incorrectly diagnosed, every unit validating with the farthest unit on the validation path must be classified as fault-free.

Bagchi and Hakimi presented an off-line adaptive distributed diagnostic agorithm for general topology systems. Testing decisions are made locally at each unit depending on the fault state of system components. These decisions are based on graph-theoretical methods. The algorithm tries to create a spanning tree of the vertices representing fault-free units in the testing graph. This is achieved by communicating *tree-forming packets* among the units. The traversal of the packets creates the trees. Different packets are merged together to avoid having multiple spanning trees. Packet 'capture' and 'merge' operations are controlled by a different set of packets, called *information packets.* At the termination of the algorithm every fault-free unit knows the fault state of all units in the system, provided that the $G - G_f$ is strongly connected (where G_f is the graph formed by vertices of faulty units and edges incident to these vertices). If, however, $G - G_f$ is not strongly connected, then the fault-free units in each strongly connected component would know each other unit in that component and the faulty units which are neighbors to some units in the component.

Units "wake up" spontaneously at arbitrary times or they are "awakened" by some neighbor. Actions of a faulty unit when awakened are unpredictable. A fault-free unit u_i waking up creates a packet and tries to create a tree consisting of other fault-free units. Assume that the packet created is $\mathrm{PKT}(r_i)$ (initially r_i is u_i) and its traversal is spanning a tree T_i ($u_i \in T_i$). A unit possessing a packet initates testing one of its neighbors, in our example $u_j \in N(u_i)$, that has not yet been visited by $\mathrm{PKT}(r_i)$. This also wakes up unit u_j. The two units mutually test each other, thus u_i knows a_{ij} and u_j knows a_{ji}. If $a_{ij} = a_{ji} = 0$, then u_i includes u_j among the set of nodes the packet visited (this information is stored in the $\mathrm{PKT}(r_i)$.visited field of the packet body), sends the packet to u_i, and sets its "latest child" $l(u_i) \leftarrow u_j$.

Unit u_j receives the packet, sets its "father" $f(u_j) \leftarrow u_i$, and repeats the above procedure.

However, in case of $a_{ij} = 1$ unit u_i does not send the packet to u_j. Instead, it marks u_j as faulty in the set of visited nodes, and tries to find some other unit to send the packet to. Note, that a faulty unit u_k might also create a packet and initiate a spanning tree, but by the same mechanism $a_{ik} = 1$ and unit u_i will just ignore the received packet.

As T_i traverses the system, at some time packet $\mathrm{PKT}(r_i)$ will arrive to a unit u_m for which $\mathrm{PKT}(r_i)$.visited $\subseteq N(u_m)$. This implies that the states of all neighbors are already in the packet. At this point, unit u_m sets a flag $\mathrm{PKT}(r_i)$.alldone indicating that the packet is in the returning phase, and sends it to $u_k = f(u_m)$. Receiving the packet, unit u_k includes u_m in its set of "other children" $c(u_k)$, sets $l(u_k) \leftarrow 0$, and clears the $\mathrm{PKT}(r_i)$.alldone flag. It now posesses the packet and tries to find some other units to include in T_i. Due to the depth-first search nature of the algorithm, eventually $\mathrm{PKT}(r_i)$ will return to unit r_i, the root of the tree T_i. Also, when $N(r_i) \subseteq$ $\mathrm{PKT}(r_i)$.visited the packet contains the states of every unit in the system.

If there are multiple tree forming packets in the different parts of the system, they have to be merged together to assure that only a single packet completes. There is a priority between two packets based on the number of rooted trees merged to get the present tree. The notion $\mathrm{PKT}(r_i) >$ $\mathrm{PKT}(r_j)$ means that $\mathrm{PKT}(r_i)$ has higher priority than $\mathrm{PKT}(r_j)$. Every unit $u_k \in T_i$ stores a local copy of the packet $\mathrm{PKT}(r_i)$ in a variable called *current packet* $p(u_k)$. Suppose the packet $\mathrm{PKT}(r_j)$ arrives to u_k. If $p(u_k) =$ $\mathrm{PKT}(r_i) \geq \mathrm{PKT}(r_j)$, then $\mathrm{PKT}(r_j)$ is made to wait at unit u_k. Eventually, when $\mathrm{PKT}(r_i)$ comes back to u_k from $l(u_k)$ it will "capture" $\mathrm{PKT}(r_j)$. The two packets are then merged together: T_j will become a subtree of T_i and the priority of $\mathrm{PKT}(r_i)$ increases. On the other hand, if $\mathrm{PKT}(r_i) < \mathrm{PKT}(r_j)$, then $\mathrm{PKT}(r_j)$ is sent along the same path as $\mathrm{PKT}(r_i)$, thus creaing the intuition of "chasing". Due to the FIFO discipline on the links and because packets cannot cross each other, the two packets will meet at some unit u_m and then be combined into one packet. In this case, T_i becomes a subtree of T_j.

When $\mathrm{PKT}(r_j)$ captures $\mathrm{PKT}(r_i)$ an informing packet is broadcasted among the units in T_i and every such unit updates its current packet to the combined packet $\mathrm{PKT}(r_j + r_i)$. During the merging of the two packets the *merging point* (suppose it is u_k) has a special function. Only this point changes its father to reflect that T_i became a subtree of T_j. The edges along the path $u_k \ldots u_m$ are not affected, any unit in this path still has the same father and latest child. However, the direction of the edges along the path $u_k \ldots r_i$ have to be reversed, the units have to interchange their father and latest child.

The algorithm creates a single spanning tree saturating only fault-free units, although the process could be started by multiple initiators. The states of every unit are collected at the root of this tree. After the collection stage, the states are broadcasted in a single packet to the units in the tree.

Stahl, Buskens, and Bianchini developed an on-line diagnosis algorithm called Adapt on the same theoretical background as the algorithm of Bagchi and Hakimi [65]. The testing assignment is also adaptive. The algorithm is decomposed into two main steps. In the first step, tests are added to the testing graph locally by each unit to restore its strong connectivity. Execution of this step occurs in parallel to minimize diagnosis latency. Redundant tests are removed in the second step, maintaining correct diagnosis. The algorithm locates any number of faulty units provided that the testing graph of the whole system remains strongly connected (i.e. there are sufficient number of reliable unit available in the first step of the algorithm).

The Adapt algorithm does not classify the interconnection links, but it handles blocked communication paths due to its adaptive nature. The tests in the system are periodic. A unit may not fail and subsequently recover undetectedly during the interval between two consecutive test by the same tester unit. A message validation scheme, similar to that of the NEW_SELF algorithm is used to ensure that messages received from faulty units are ignored. Both unit failures and repairs are considered.

The algorithm consists of three phases. Immediately after a fault event occurs, the algorithm enters into the *Search phase* to reconstruct a strongly connected testing graph. Once the Search phase completes, the algorithm begins the *Destroy phase* to remove redundant tests (i.e. to minimize the number of arcs in testing graph). Finally, the *Inform phase* is invoked to notify all units about the most recent testing assignment. When the testing assignment is updated, units execute the same diagnosis procedure as described at the Adaptive DSD algorithm.

The algorithm uses an adaptive testing assignmentto ensure that a directed path exists in the testing graph from the vertices representing fault-free units to every other vertex. When a unit detects the failure or repair of any unit it tests, it begins the Search phase. First it generates a *Search packet* containing information about the fault event, then it sends the packet to its neighbors. All other units enter in the Search phase upon receiving a Search packet, and forward the Search packet to units according to the testing graph. If a certain unit cannot find a directed path to any of its neighbors, tests on these neighboring units are added to the testing graph, and the Search packet is updated to reflect the changes. After the search procedure has terminated at all units, the resulting testing graph is guaranteed to be strongly connected. The Search phase is finished when a single Search packet completely traverses the whole network, indicating that all units have accomplished the search procedure.

Completion of a Search packet signals the start of the Destroy phase. During this phase a single *Destroy packet* is circulated to all units of the network, causing each unit to execute a destroy procedure on the testing graph. A unit excludes a test if it can remove the test and still finds a directed path to all other units in the testing graph. A strongly connected testing graph containing no redundant edges exists at the completion of the Destroy phase. Finally, the algorithm begins the Inform phase. In this last phase an Inform packet is circulated to distribute the new testing graph to each unit.

All packets are forwarded over the network using a routing procedure. This procedure performs a distributed depth-first search of the system graph, traversing a spanning tree of all fault-free units. Packets traverse a distribution tree by recording their information at each unit visited. Every unit that receives a packet belongs to the tree created by that packet, and the packet is current at the unit. Message validation occurs whenever packets are exchanged between units.

The search procedure can be executed on several units in parallel and it will be still functioning correctly. The Destroy procedure, however, must run sequentially to assure that a unit does not remove tests required by another unit for connectivity. Parallelism occurs in the Search phase when multiple Search packets are created for the same fault event by two or more units. Since several Search packets can simultaneously exist in the network, coordination is necessary for correct operation. Packet dominance defines a competitive relationship between two packets. Packets are compared on the basis of syndrome time stamps generated by the units upon entering the Search phase, included in every Search pocket. Given two packets A and B, packet A is said to be dominant (denoted by A ¿ B), if either all time stamps in packet A are greater than the corresponding time stamps in packet B, or the time stamps are equal but the sender identification code (also included in the packets) of packet A is greater than of packet B. Two packets A and B are bidominant (denoted by A $\leftrightarrow$ B) if some of the syndromes in packet A have greater time stamps than the corresponding syndromes in packet B and some of the time stamps in packet B are greater than in packet A.

If the arrival of a dominant Search packet causes a unit to add tests, then the diagnostic information stored on the unit is considered to be bidominant with the packet. In this case, the combination of the Search packet information and the unit information results in a new dominant Search packet. If a unit receives a packet that is of equal dominance to the one it has recorded as current, then this packet is returning from a child unit and is immediately forwarded. Packet A collides with packet B if pocket A arrives at a unit which has pocket B recorded as current. The algorithm uses the following rules to determine the outcome of pocket A colliding with pocket B on unit u_i:

- if A ¿ B, then A overwrites B, u_i becomes part of the tree of A, and u_i forwards A.
- if A ¡ B, then u_i deletes packet A, and remains part of tree of B.
- if A $\leftrightarrow$ B, then u_i forms a new packet C containing the newest information from both A and B, and forwards C with setting the sender identification code to u_i. Packet A is deleted. Note, that $\mathrm{C} > \max(\mathrm{A}, \mathrm{B})$ by definition of dominance.

The dominance and collision rules assure that if multiple Search packets were generated in the network, then only a single Search packet completes. Therefore only one Destroy packet is created, ensuring the destroy procedure is executed sequentially. The collision rules are modified for a Destroy or Inform packet to provide that the Destroy or Inform phase is terminated when a new Search phase begins. If a Destroy or Inform packet is bidominant with the information stored at a certain unit, then a (new) search packet must be created.

In the distributed fault-tolerance concept processing elemens were considered to be identical from the diagnostic viewpoint, i.e. the system was homogenous. A more general, non-homogeneous diagnosis model was presented by Liaw, Su and Malaiya [41]. A processing element in the model can be either a *testing unit* capable of performing tests on neighbors, and a *non-testing unit* only tested by others. Accordingly, interconnection links also have different types. A *test-and-transfer edge* implies a message transfer capability in the direction indicated by the corresponding directed arc in the testing graph, and a testing capability in the opposite direction. A *transfer-only edge* implies a message transfer capability in the given direction, but no test is made via the link represented by this arc. A *test-only edge* indicates that the corresponding link is not capable of transferring data, yet a test can be performed via the link in the direction opposite to the respective directed arc in the testing graph. Note, that only test-and-transfer and test-only edges imply testing, the arrows are drawn from the tested unit to the testing unit. If there exists both a transfer-only and a test-only edge between two units in the same direction, then they can be logically replaced by a single test-and-transfer edge. In the testing graph, there is at most one edge from a unit to another.

The tests in the model are assumed to be complete, that is the test covers all faults in the tested unit and the interconnection link. If the testing unit is fault-free, but a fault exists in either in the tested unit or the interconnection link between the two units, then the testing unit obtains a failed test result. Otherwise the test passes. The inter-processor communication network is considered to be a passive component having a simple and easy-to-test structure. As long as both of its terminal units are fault-free, the receiving unit can verify the correct operation of an interconnection link with a simple test. A failure of a unit or a link involved in the communication can corrupt any message passed through. In order to protect essential information from unnoticed alterations during transmission, error-correcting codes are proposed. Each message will include a *private identification code* of the sending unit. Each non-testing unit can decode and detect errors in codes, so any information with valid identification code received by a non-testing unit will very likely be correct.

The authors also developed a new diagnosability measure for their extended model. They defined a system to be *unweighted if* all of its elements have equal reliability. When different components are not equally reliable, a positive integer weight may be assigned to each component to reflect their relative reliability. The weight assignment should satisfy the following condition: a set of faulty units with smaller total weight is (assumed to be) more probable to fail than other fault sets with larger total weight. Such a system is called a *weighted* system. A weighted system is *w-weight self-diagnosable* if each fault-free unit can correctly identify all faulty (and fault-free) units in the system, provided that the weight of the fault set present is not greater than w. A *testing path* is a sequence of testing edges in the same direction. The first edge may either be a test-and-transfer or a test-only edge, but all the subsequent edges must be test-and-transfer edges. A *message path* is a sequence of test-and-transfer and transfer-only edges in the same direction, that cover only testing units except a non-testing unit at the receiving end.

A set of paths sharing sending and receiving units are blocked by a fault set (not including the two terminal units) if none of the paths remain completely connected after the removal of faulty components. The following conditions must be fulfilled by a weighted system to be w-weight self-diagnosable:

- if w_l is the total weight of testing units, then $w \leq \lfloor \frac{w_l+1}{2} \rfloor$,
- from every unit to a testing unit there exists either a test-and-transfer or a test-only edge, or a set of testing paths that cannot be blocked by any fault set with a weight less than w,
- from every testing unit to a non-testing unit there exists either a test-and-transfer or a transfer-only edge, or a set of message paths that cannot be blocked by any fault set with a weight not greater than w.

Self-diagnosis of a weighted, non-homogeneous system incorporating both testing and non-testing units is achieved in three steps. First, diagnosis of individual units via test links is performed. Testing units record test outcomes in the local syndrome. Then the testing units exchange diagnostic information among each other to complete their diagnosis about the whole system. Finally, testing units broadcast information to non-testing units and let them achieve their diagnosis.

In the first step of the algorithm all of the tests included in the testing graph are executed by testing units. Non-testing units can also verify the transfer-only edges connecting them to testing units. The second step is very similar to the SELF algorithm with some differences: diagnostic messages are only broadcasted over test-and-transfer edges, and the process is iterated until either the fault vector is fully specified, or the weight of the identified fault set is reaches or exceeds w (unclassified units are labeled as fault-free). In the third step each testing unit generates a message for each fault-free non-testing unit, containing the completed fault vector and the personal identification code. Testing units only start those messages that can be forwarded over a chain of testing units they diagnosed to be fault-free. Upon receiving a message, non-testing units first decode the identification code to verify the integrity of the information. Valid messages are stored in a stack structure, invalid messages are ignored. Since faulty testing units may transmit a diagnostic message more than once, such messages are counted and are taken into consideration only when received for the first time. The weight of subsequently received fault vectors is added to the weight of the identical fault vector already recorded in the message stack. If the weight of a certain fault vector on the stack becomes greater than or equal to $w \leq \lfloor \frac{w_l+1}{2} \rfloor$, then the procedure is terminated and the fault vector is accepted. The fault vector selecting process is similar to the algorithm of Meyer and Masson [46].

The MIPT/MUPT model of Russelt and Kime was adapted to distributed multiprocessor systems by Holt and Smith [25]. The diagnostic goals of their distributed model are slightly different from the traditional ones. These goals can be expressed most easily in terms of the required *diagnostic information* and the so called *diagnostic sinks*. Diagnostic information in the notion of Holt and Smith is the information that is to be deduced through the diagnosis process. Diagnostic sinks are the set of units in the system that must obtain this information. By specifying different combinations of diagnostic information and diagnostic sinks, a wide variety of self-diagnosis goals can be defined. The following two possible purposes for diagnosis were considered in the model:

- *Repair*: the system is periodically diagnosed so that faulty units can be repaired, thus keeping the system operating at full capacity. The repair might be manual, or might be in the form of standby spares that are activated by the remaining good units.

- *Graceful degradation*: the system initially has excess computing capacity, and as units fail the system continues operation by distributing the tasks of failed units among the remaining good units. Diagnosis is needed in order to identify the portion of the system that is still useful.

In the case of diagnosis for repair it is assumed that the repair need not be done all at once; the diagnosis/repair cycle can be repeated until the entire system is working. Thus, it is not necessary to locate all the faults within one diagnostic round, identification of one unit is sufficient. A system that is diagnosed for repair must contain some diagnostic controller facilities that carry out the repair or replacement. A controller might be a reconfiguration control unit capable of activating spares automatically, or it might be a human repairman. Furthermore, there might be one centralized controller or several distinct controllers, each of which can perform the required repairs (this assumption violates the requirement of not involving any form of diagnostic hard-core stated by distributed fault-tolerance). Then a suitable diagnosis goal for a reparable system is to correctly locate at least one faulty unit (diagnostic information), and to deliver this information to a fault-free controller (diagnostic sink).

In diagnosis for graceful degradation, the goal is no longer to precisely identify faulty units, but rather to locate some good set of units that can remain in operation. Diagnostic sinks would be those units that are responsible for disconnecting faulty (or suspected faulty) units. Unlike the diagnosis for repair case, the presence of special controllers capable of system-wide reconfiguration is not assumed. The responsibility for disconnecting local faulty units is rather distributed among the "intelligent" units of the system called *analyzers*. The analyzers perform diagnosis, then disconnect themselves from any portion of the system that cannot be verified to be fault-free. Disconnection can be carried out by shutting off or powering down units, by electronically breaking communication paths or by ignoring the faulty (or suspected faulty) units. Having good units ignore the suspected faulty ones is likely the preferable method, since switching would probably require hard-core or fail-safe circuitry and special cooperation among analyzers. Then a suitable goal for a gracefully degrading system is to identify as many remaining useful units as possible for a set of fault-free analyzers.

The system is modeled by Holt and Smith with two graphs, a *diagnostic graph* D and a *communication graph* C. The diagnostic graph is a modified version of the Russel and Kime test invalidation graph model. The vertex set of the graph contains both the units of the system and a collection of pass/fail tests on these units. A test may be any sort of check by one or more units on the operation of one or more others, including applying test vectors on the input of tested units and the observing the resulting outputs, running an "exerciser" program on the tested unit and checking the program outcome, monitoring the output of self-testing circuits, using special built-in testing modes, or making a comparison check on two identical units. The model employs asymmetric test invalidation. The diagnostic graph is a directed bipartite graph, the edge set consists of test edges from a test unit t_i to a unit f_i indicating that test t_i is a complete test for unit f_i, and invalidation edges from a unit f_j to a test unit t_j indicating that a fault in unit f_j might cause test t_j to become invalid. Also shown in the graph are the units at which test results originate. For each test unit t_k at least one of the test edges from f_k is distinguished from the others to indicate that the pass/fail result of t_k is originally determined at f_k (in the graph representation this is denoted by a solid block drawn on the f_k end of the edge). Such an edge is called an origin edge, and f_k is referred to as an origin for t_k. Because many systems rely to some degree on hard-core, modeling of hard-core units is also included in the Holt and Smith model. Hard-core units are not allowed to appear in fault patterns (in the graph representation hard-core units are denoted by hexagons).

The effects of communicating test results through potentially faulty units are considered by means of a communication graph. This is a directed graph in which the vertices represent the system units and the edges indicate the possible transfers of test result messages (an undirected edge may be used to represent two directed arcs in opposite directions). A path from f_i to f_k is a sequence of unit vertices $f_i \ldots f_k$, such that each successive pair of elements in the sequence is connected with an edge in the communication graph. It is assumed that test results are transmitted only when they are requested by a certain analyzer. Communication of any test result requires paths in both directions (yet the to and from paths are need not be the same).

In case of *self-diagnosis for repair* a set of intelligent units called *analyzers* are responsible for collecting and analyzing test results in order to locate faulty units. At periodic intervals or when an error is in some way detected, analyzers send messages to units throughout the system requesting that tests be performed and the outcome of tests be transmitted back to them. Given the test result reports received (which may not be all those requested), analyzers attempt to identify a fault. Finally, the analyzers send reports of found faulty units to *controllers*. Controllers are the set of units capable of performing a repair. Upon receiving an acceptable report of failure(s) a particular controller repairs the faulty units, and the testing process begins again. Diagnosis terminates when no more faults are detected. Note, that in this model the analyzers are the intelligent instruments that actually decide what units to be repaired, while the controllers simply carry out the order (however a component may be simultaneously analyzer and controller).

The collection and analysis of test results by analyzers is similar to the diagnosis with repair process presented in [57], except that faulty communication of test result report to analyzers must also be considered. The effect of such communication is that each analyzer is limited to performing fault location based on its own, possibly incomplete view of the system. A report including test result t_i will reach an analyzer f_a if there exist fault-free paths in the communication graph from f_a to f_i for the test request, and from f_i to f_a for the test result. If the origin f_i is a faulty unit, then the received test result report from this unit may be false or mislabeled and therefore useless. Hence analyzer f_a can only accept a report from origin f_i if it can ascertain that f_i is a fault-free unit. The following strategy can be employed to find reliable origins: at first f_a begins by assuming itself to be a hard-core unit. (If f_a is really a fault-free unit, then its decisions will be correct, while if it is faulty, its results will be discarded by the other units.) Then it initiates the tests that have f_a as origin. Among these there are tests on units, which are origins of other tests. Any test performed by f_a that pass identify fault-free units. Test result reports can now be accepted from any newly found fault-free origins. These origins can test other origins, thus further expanding the set of reports that can be accepted by f_a. Note, that this procedure is very similar to the mechanism involved in the distributed fault-tolerance concept of Kuhl and Reddy.

In accordance with the above a test t_j is defined to be *accessible* by f_a in the presence of a fault pattern F^x, either if $f_a \in R(t_j)$ (f_a is an origin of t_j), or

- there exist paths in the communication graph from $f_j \in R(t_j)$ to f_a and from f_a to f_j such that none of these paths contain members of F^x,
- there exists a test t_k on f_j such that none of the units that affect t_k are members of F^x, and t_k is accessible by f_a.

It is assumed that fault location algorithms, which take a diagnostic graph and a syndrome consistent with a certain fault pattern F^x as their input and produce a computed fault set F^y as their output, can be constructed. Any such algorithm can identify a member of F^x only if the computed F^y is always a non-empty subset of F^x. A fault pattern F^x is defined to be *locatable* by f_a in system (D, C) if the fault location algorithm can identify a member of F^x using D as the input graph, and the results of all tests in D are accessible by f_a as the input syndrome (assuming f_a to be a hard-core unit).

In order to successfully complete the diagnosis process, the analyzers must not only locate a fault, but also send this information to a controller in some fault-tolerant manner. Transmitting fault information from analyzer f_a to controller f_s requires a directed path in the communication graph from f_a to f_s. It is assumed that no response from f_s is involved, thus only a one-way path is required. To prevent the controller to accept information from a faulty analyzer, any fault information sent to a controller must be accompanied with by a passed test on the analyzer. (Note, that with asymmetric invalidation a single passing test is sufficient to assure that a certain unit is fault-free.) The passed test result is called a *testimonial*. The testimonial might be "signed" by the test origin to reduce the possibility of a faulty analyzer producing a corrupted testimonial. The controller is responsible for checking that the fault information contains a valid testimonial. Accordingly, a fault pattern F^x is defined to be *reportable* by analyzer f_a, if f_a passes a test t_i accessible by itself, and there exists a path from analyzer f_a to some controllers f_s.

Based upon the previous definition, a diagnosability measure can be derived for a system of the Holt and Smith model employing self-diagnosis for repair: a system is *t-self-diagnosable for repair* if for each fault pattern F^x there exists some analyzer f_a such that F^x is locatable and reportable by f_a. The self-diagnosability for repair measure is the largest integer t such that the system is t-self-diagnosable for repair.

In case of *self-diagnosis for graceful degradation* it is assumed that the system contains intelligent analyzer units, which periodically request execution of enough tests to detect a fault in any of the currently active processing units. If a fault was detected, analyzers perform tests to locate as many good units as possible. Any remaining units are potentially faulty and hence they are disconnected. In disconnecting the suspected faulty units some tests on the selected set of remaining active units may be eliminated as well. To ensure that the future occurrences of faults in these units will not be unnoticed, the remaining active units must be checked in order to assure that there exists at least one complete passing test on each unit. Any units that do not satisfy this requirement must also be disconnected. It may be necessary to repeat this procedure several times, disconnecting some units at a time, until a testable set of units is found. The remaining active units may then reallocate tasks and resume normal operation.

As more faults accumulate, the size of the remaining useful system gradually decreases. At some point the system becomes incapable of accomplishing the assigned tasks, yet this limit depends on the application. Hence, the number of faults cannot be used as an estimation of diagnosability. Instead, a reasonable measure of diagnosability in a gracefully degrading system indicates how much "reduction" occurs in the system as the number of faults increases. Such a measure called the *survival curve* is to be introduced.

The active set of units seen by analyzer f_a in the presence of fault pattern F^x is the largest set of units F^0 such that: $f_a \in F^0$, all members of F^0 are fault-free, and for each $f_i \in F^0$ there exists a test t_i accessible by f_a and performed on f_i only by the members of F^0 (i.e. f_a has access to valid tests on all units of F^0).

Note, that a fault pattern F^x may divide the communication graph into separate strongly connected components, resulting in several active sets which may be capable of independent operation. If the system breaks up in this manner, it is assumed that only the largest active set is of interest. The largest active set seen by a certain analyzer f_a in the presence of fault pattern F^x is called the *surviving set.* The *survival function*, denoted by $V(f)$, for a given system expresses the minimum surviving set size as a function of fault pattern size. The *survival curve* is a plot of the survival function versus the number of faults present.

In defining the survival curve, the size of surviving set is used as a rough measure of the quality of the gracefully degrading system. This measure does not take into consideration neither the (possibly different) types of remaining active units, nor the way the remaining units are connected. A measure incorporating these factors could be defined or an active set containing a minimum set of facilities could be determined. Formulation of such a measure, however, would require application-dependent knowledge about the various unit types and the necessary amount of communication between them. The survival curve is a worst-case degradation measure. Each point of the curve gives the size of the surviving set for the worst possible arrangement of f faults. For most system configurations, various points on the survival curve correspond to completely different arrangements of faults. There is usually no particular sequence of faults that will result in a system degradation as shown on the survival curve. If particular sequences of faults were considered, a whole family of corresponding degradation curves could be drawn, all of which would lie above or on the survival curve. To give a better abstraction of typical system behavior other survival-type measures may be defined, e.g. average surviving set size or probabilistic measures with confidence levels.

0.3.3 Probabilistic algorithms

The deterministic diagnosis methods presented in previous sections identify the entire fault set (or a predefined subset of all faults) from the syndrome, provided that certain restrictions on the testing structure and the behavior of the units are satisfied. By contrast, probabilistic diagnosis methods try to locate the faulty units *with high probability* and require no restrictive assumption on the set of faulty units or the structure of the testing assignments (i.e. the concept of diagnosability is not applicable to probabilistic diagnosis). On the other side, probabilistic methods cannot guarantee that a correct and complete diagnosis is made in every situation.

In probabilistic diagnosis a probability model is used to model the operation of fault-free and faulty units. On the basis of this model, a procedure—which may be heuristic—is used to diagnose a set of units as faulty. The important issues in probabilistic methods are the complexity of the diagnosis procedure and the quality of the diagnoses generated. Many of these methods use probabilistic parameters to describe the behavior of units and to evaluate diagnosis performance. However, the way of obtaining these parameter values in real computing systems is still an important unsolved problem [40]. Another limitation of probabilistic diagnosis algorithms is that they assume the outcome of test performed by different units to be statistically independent. Statistically dependent test results do not make these diagnosis algorithms useless, but they may lower the level of the diagnostic accuracy.

Since diagnosability is not a valid measure for the quality of probabilistic methods, they must be evaluated by other means. The three arguments used to support probabilistic diagnosis algorithm are:

1. analyzing the diagnosis accuracy produced in certain situations,
2. guaranteeing the set of most likely faulty units given the syndrome is found,
3. proving that the diagnosis accuracy will approach 100 percent as the number of units in the system grows to infinity.

The second argument may seem particularly appealing from a practical viewpoint. Yet, it has been shown that finding the most probable diagnosis based on the global syndrome information is an NP-hard problem [8, 37]. From the same point of view, the third argument is insufficient since an acceptable diagnosis accuracy is required for finite systems. In spite of this, asymptotically correct and complete diagnosis is a desirable property of any probabilistic diagnosis algorithm due to the significance of automated diagnosis for large and massively parallel systems.

Probability models.

Probabilistic methods differ in the probabilistic parameters used and the probability model employed to define the likeliness of the different syndromes. A probability model is characterized by defining the probability space, which is a triple (Ω, Θ, P). In this triple Ω is the *sample space*, Θ is the *event space*, and P is a probability measure.

The most often used probability model is called the *common* probability model. In this model, the sample space Ω is defined to be the set of all possible syndrome and fault set pairs in a given testing graph $G = (U, E)$. Formally,

$$\Omega_G = \{\,(S, F) : F \subseteq U \text{ and } S \text{ is a function from } E \text{ to } \{0,1\}\,\}$$

The event space Θ_G consists of the set of all possible subsets of Ω_G. The probability measure P is determined using the probability parameters describing the behavior of fault-free and faulty units, assuming that all the units in F are faulty and all the units in $U - F$ are fault-free. In the case of the 0-information tester model, every possible syndrome and fault set pairs can have a nonzero probability value. However, for the partial tester or complete testing models certain syndrome and fault set pairs will have zero probability. For example, in these testing models a situation of $u_i, u_j \in U - F$ and $S(t_{ij}) \equiv a_{ij} = 1$ can never occur, so the probability of this syndrome and fault set pair is 0.

Blough introduced a more general probability model. He modeled the actions of faulty units by assuming that they behave in any arbitrary manner, even the most detrimental to the diagnostic algorithm. The sample space Ω_G is the same in the Blough model as for the common model, but the set of basic events is different. Since test outcomes by faulty units are random values, in general no probability value can be assigned to a syndrome, fault set pair. Therefore the basic events of the model consist of syndrome, fault set pairs that have the same fault set and whose syndromes are identical except the test results of faulty units. Thus, a syndrome, fault set pair (S', F') is contained in a basic event B is defined as

$$B = \{\,(S, F) : F = F' \text{ and } \forall t_{ij} \in T \text{ with } u_i \in U - F, S(t_{ij}) = S'(t_{ij})\,\}$$

Let B_{set} denote the set of basic events, then the event space Θ_G is composed of all subsets of B_{set}. The probability of an algorithm producing a correct and complete diagnosis using the model can be calculated as the minimum of the probabilities assigned to the syndrome, fault set pairs in the basic event B, for which the set of faulty units F is the set of units classified as faulty and B contains the actual syndrome observed.

Lee and Shin introduced another generalization of the common probability model. Since most probabilistic algorithms use less than the global syndrome information for diagnosis, the authors assumed a distributed self-diagnosis method in which each unit declares itself faulty-free or faulty.

For this, units use only a limited form of the global syndrome information, referred to as *partial* syndrome. Since the information analyzed during diagnosis varies for every unit, each of them uses a different probability space. For a given unit u_i, let S_i represent the partial syndrome used in the diagnosis of u_i, and S_i^{all} be the set of all such partial syndromes. Then, the sample space for unit u_i is defined as

$$\Omega_i = \{\, (S_i, f(u_i)\,) : S_i \in S_i^{\text{all}}, f(u_i) \in \{f_i^0, f_i^1\}\,\}$$

The event space Θ_i is the set of all possible subsets of Γ_i. The definition of of the probability measure P_i depends on the partial syndrome information used.

All of the probability models presented have advantages and disadvantages. The common probability model provides exact calculation of the a posteriori fault probability of each fault set given a syndrome. In this model an algorithm that produces the most probable diagnosis can be formulated. Such an algorithm is also proven to be optimal in diagnostic accuracy [8]. The Lee and Shin model makes it possible to generate the most probable diagnosis for a certain partial syndrome, assuring locally optimal algorithms (i.e. these algorithms are optimal in diagnostic accuracy among all methods analyzing the same partial syndrome information). While the problem of optimal diagnosis in the Lee and Shin model can be solved in polynomial time, the same problem has exponential computational complexity in the common model. Also a serious drawback of both models is that the probability parameters describing the operation of units most be known or estimated before any probability analysis can be done.

Here lies the main advantage of the Blough probability model: the behavior of faulty units do not need to be parameterized before its application. On the other side, this causes many diagnosis algorithms that generate different diagnostic output to be evaluated as being equal under the model. Thus, the Blough model can not distinguish between two methods, one of which may perform significantly better in terms of diagnostic accuracy. Consequently, exact analysis and optimal diagnosis is not possible using this model, but the asymptotic or upper-bound characteristics of the algorithms can be estimated.

Complete test methods

It is clear from the prevoius sections that if 100 percent test coverage is assumed, then there are numerous deterministic methods available to solve the diagnostic problem. Note, that in particular cases—like one-step t-diagnosable systems—even a unique set of falty units can be identified. The reason for the introduction of probabilistic fault diagnostic algorithms in the complete-test model is to extract diagnostic information from situations with more than t faulty units and for general testing graph structures.

Scheinerman presented a probabilistic approach with desirable asimptotic properties [59]. In his algorithm, first a core group of nonfaulty units is identified. This is accomplished by finding a strongly connected subgraph of G in which all edges are labeled by a 0 test outcome and in which the more than the half of the total units is included. Since in this component each unit has the same fault state (all of them are involved in a circle of 0 edges), they are surely fault-free provided that only the minority of the system units may fail. Other units accessible from the core group via a path of 0 edges are then added to the set of fault-free units. All other units are finally classified as faulty.

The author showed that his algorithm produces asymptotically correct and complete diagnosis in random graphs in which a unit is connected to another unit with a probability $(c \log n)/n$, where $c > 1/(1-p_f)$. A significant weakness of the algorithm that it does not work if a fault-free core group cannot be found. Such situations can easily occur even for a moderate number of faults (e.g. when the testing graph is "cut up" into more than two strongly connected subgraphs by "chains" of faulty units). Scheinerman's work is important, however, since it provided the first proof of asymptotically correct and complete diagnosis.

Using Blough's probability model, Blough et al. presented a complete-test probabilistic diagnostic method in which each unit u_i simply evaluates itself to be fault-free or faulty based on the majority opinion of its testers [7]. Thus, u_i is diagnosed as faulty if and only if

$$|\{u_j : u_j \in \Gamma^{-1}(u_i), a_{ji} = 1\}| > \left|\frac{\Gamma^{-1}(u_i)}{2}\right|$$

The authors were able to demonstrate that asymptotically, as the size of the system approaches infinity, 100 percent accurate diagnoses can be obtained for hypercube testing topologies if $p_f < 0.067$. Additionally, it was shown that asymptotically correct and complete diagnosis can be generated for a special class of testing graphs with $n\omega(n)$, where $\omega(n)$ is any function that grows to infinity, albeit arbitrarily slowly.

Somani and Agarwal introduced three more complex algorithms for the same problem [62, 63]. These algorithms first assign *confidence levels* to indicate how likely a certain processor can be considered as faulty. There are two confidence levels: units are classified as either 'likely good' or 'likely faulty'. The decision between the two classes is based on a majority voting of the testers, just like in Blough's algorithm. Then, the found potentially fault-free units are used to locate at least one definitely fault-free unit (again using a majority voting, this time only taking into account the votes of likely good units). Finally, test results of the definitely fault-free units are propagated and the other units are diagnosed iteratively. The iteration continues until no futher diagnosis is required or possible. At the end the algorithms check the consistency of the generated diagnostic image with the symmetric test invalidation model.

The three algorithms differ in the voting technique they use. The first algorithm LDA1 employs majority voting in both phases. The second algorithm LDA2 differs from ht previous one only in that unanimous voting among the likely good units is required to identify a faulty processor. The third algorithm LDA3 uses unanimous voting to classify likely good and likely faulty units, and also to find definitely fault-free units. The LDA2 and LDA3 algorithms are meant to be gradually simpler methods than the LDA1 algorithm. Somani and Agarwal described the conditions under which their diagnostic algorithms produce correct and/or complete diagnosis. They also present several examples to show that a good diagnostic accuracy can be obtained with the LDA1 algorithm in $\sqrt{n} \times sqrtn$ mesh topologies.

Incomplete test methods

Probabilistic diagnosis is most useful in situations where the units can be intermittently faulty or the tests have significantly less than 100 percent coverage. Although, probabilistic methods by definition cannot guarantee that the produced diagnosis is accurate, neither can deterministic methods make such a claim in the case of intermittent faults or incomplete tests. The best possible probabilistic method in terms of diagnostic accuracy is the one that outputs the most probable set of faulty units given a syndrome. Blount gave an early approach for the solution of this problem [10]. He described the 0-information tester model and used the common probability model to define a mapping from syndromes to fault patterns. The diagnostic process uses a lookup table, which contains the fault set F that has the largest probability $P(S|F)$ for a given syndrome S. Whenever an observed syndrome needs to be decoded, the lookup table is consulted to find the most probable fault set. Though strightforward, this method is very resource consuming. Given $|E|$

edges and n units, there are $2^{|E|}$ possible syndromes and 2^n possible fault sets. Thus, to create the lookup table, $O(2^{n+|E|})$ calculations and $O(2^{|E|})$ memory locations are required.

Lee used the partial tester and common probability models to produce a more efficient method for finding the most probable diagnosis given a syndrome [37]. In his method, he introduced several heuristics to speed up the search for the most likely fault set by bounding the search tree as early as possible. Although the average behaviour of this algorithm is fairly good, its worst-case computational complexity is $O(2^{|F|})$, where F is the set of faulty units identified.

Realizing the limitations of finding the most probable diagnosis, Blough et al. used Blough's probability model to develop an $O(|E|)$ algorithm which is able to achieve asymptotically correct and complete diagnosis in systems of $O(n \log n)$ testing connections [6]. In the algorithm, the number of failed tests on a unit $|\Delta_{\text{in}}|$ is compared to an *a priori* threshold value k. The unit u_i for which $|\Delta_{\text{in}}(u_i)| - k_i$ is maximal (i.e. the unit most exceeding its threshold) is selected and placed in the set of faulty units. Next, all of the test results generated by u_i are set to 1 (to imitate that each of them has failed). The difference between the number of failed test and the threshold is the re-calculated at the units $u_j \in \Gamma(u_i)$. Again, the process looks for the maximal difference to place another unit in the set of located faults. This iteration terminates when there are no more u_k units for which $|\Delta_{\text{in}}(u_i)| > k_i$. The rest of the system is classified as fault-free.

The authors proved that asymptotically correct and complete diagnosis can be obtained if the k_i threshold values local to unit u_i are chosen as follows:

$$k_i = \left\lfloor \frac{p_f + p_{ij}(1-p_f)}{2} |\Gamma^{-1}(u_i)| \right\rfloor .$$

They also describe a heuristic method of chosing better threshold values, based on the following lower-bound estimate of the correct diagnosis:

$$\begin{aligned} P_{\text{correct}} \;\; &\geq \;\; 1 - n(1-p_f) \sum_{j=k+1}^{N} \binom{N}{j} p_f^j (1-p_f)^{N-j} - \\ &- \;\; np_f \sum_{j=0}^{N} \left[\binom{N}{j} (1-p_f)^j p_f^{N-j} \sum_{l=0}^{\min(j,k)} \binom{j}{l} (1-p_d)^l p_d^{j-l} \right], \quad (6) \end{aligned}$$

when a common threshold $k_i = k$, failure probability p_f, and detection probability $p_{ij} = p_d$ is used, and $N = \max_{u_i \in U} |\Gamma^{-1}(u_i)|$. To find the best common threshold, the above expression is evaluated for all possible values of k and the threshold resulting the highest probability of correct diagnosis is chosen. Note, that the threshold selected by this method is not necessarily an optimal one.

Dahbura et al. gave a comparison-testing based probabilistic diagnostic algorithm of $O(n^2)$ time complexity [14, 15]. It repeatedly selects the unit that is incident to the largest number of 1-links (failed test), i.e. $\max_{u_i \in U} |\Delta_{\text{in}}(u_i)|$. This unit is placed in the set of diagnosed faults and excluded from the testing graph, as well as its local test results are removed from the syndrome. This procedure is iterated until there are no 1-links in the testing graph. Using an assumed upper bound on the number of faulty units, it was shown that the probability of incorrect diagnosis is extremely small given a completely connected testing assignment. Later, Lee [37] was able to show that this algorithm has the same asymptotic properties as the method from Blough et al. [6] while performing significantly better for several different topologies and probability parameter values. This demonstrates that asymptotic accuracy is not sufficient in itself to indicate the usefulness of a probabilistic diagnostic algorithm.

Lee and Shin presented another set of diagnostic algorithms [39]. They observed that many of the approaches to that time used only partial syndrome information in classifying the fault state of each unit. The authors defines several cathegories of methods respective to the type of partial syndrome information they used. Then, Lee's probability model was used to calculate *a posteriori* fault probability values for the individual units in each cathegories of diagnosis. Faulty unit were identified on the basis of these *a posteriori* fault probability calculations. All of the algorithms developed was shown to have the same asymptotic proprties as Blough's algorithm. The main advantage of Lee and Shin's work is that their methods find the most probable fault set given a particular type of syndrome information, resulting in a *locally optimal* diagnosis. Their most significant disadvantage is that the syndrome decoding process is relying on the use of probability parameter values.

Fussel and Rangarajan introduced an entirely different type of diagnostic methodology [22]. Given that the testing graph is the subgraph of the processor interconnection graph, the algorithms described so far require each unit to be tested by at least $\log n$ other units for asymptoticallty correct and complete diagnosis. Fussel and Rangarajan suggested to conduct multiple mutual tests in each pair of units. If the number of these tests on all of the units grows faster than $\log n$, then the same asymptotic properties can be obtained in systems with lower connectivity (e.g. meshes or rings), thus improving on the connectivity bound. Their algorithm uses several stages of comparison tesing. In testing stage i, all units are assumed to execute the same test task, then the local test results of each unit are compared to the results of all adjacent units. The testing link between two units u_i and u_j is labeled by $c_{ij} = c_{ji} = 1$ if the tests t_{ij} and t_{ji} gave identical results. In this manner, r independent syndromes are collected at the end of the testing stages. Two thresholds s_v (tester threshold) and k_v (test threshold) are used to classify the units. The algorithm assigns labels to the vertices of the testing graph in two steps. First, a *super label* of value 1 is associated with units for those adjacent to 1-links exceeding the tester threshold s_v. Given r syndromes, r super labels are created. In the second step, a *decision label* of value 1 is associated with units for which the sum of super labels exceeds the test threshold k_v. These units are diagnosed as fault-free, all others are faulty.

The authors chose $k_v = 1$ and a range of values was indicated as being acceptable for s_v. In later works [54, 55], they derive better k_v and s_v threshold values and discuss a hierarchical version of the above algorithm. The hierarchical method conducts testing and diagnosis in *clusters*. A third threshold h_v is used to compute a *hierarchy label* which has similar role in the diagnostic process as the previous two ones. The new method thus preserves the topological flexibility of the previous algorithm, while allowing the number of tests each tester must perform to be tailored to the requirements of the interconnection topology.

Referring to the use of multiple testing stages and multiple syndromes as a *multiple-syndrome* diagnosis method, Lee and Shin derived a *locally optimal* multiple-syndrome diagnosis algorithm using their own probability model [37]. The only change introduced by the authors is the way in which the thresholds k_v and s_v are chosen, but in the other aspects the algorithm is the same as Fussell and Rangarajan's algorithm. Calculations of the *a posteriori* fault probabilities help to derive optimal values for k_v and s_v. Due to the similarities and the optimal thresholds, the algorithm shares the same desirable asymptotic properties as Fussell and Rangarajan's method.

Similar to multiple-syndrome diagnosis is the *sequential diagnosis* strategy for probabilistic diagnosis. In the sequential approach, diagnosis is also conducted in stages. Units identified as faulty in the ith stage are replaced with spares before beginning the next stage of diagnosis. Blough and Pelc presented four algorithms for sequential diagnosis respective to the four possible combinations of complete and incomplete system-level tests, and perfect and imperfect spares [9]. Their algorithms require the total number of $o(n)$ tests produce asymptotically correct and complete diagnosis in the complete-test, perfect-spare model. The other models need somewhat more tests. Both of the intermediate models requires $O(n \log n)$ tests; while the incomplete-test, imperfect-spare model uses $O(n \log^2 n)$ tests. The basic diagnosis process takes place in two phases. A core set of fault-free units is identified in the first phase. This is followed by a second phase in which the fault-free units are used to determine the fault status of other units, similarly to the algorithms of Somani and Agarwal [62, 63]. The units found to be fault-free are added to the core set. The units found to be faulty are replaced by spares and tested. Multiple tests and multiple replacements of spares are used in the case of incomplete tests and imperfect spares, respectively. These sequential diagnosis algorithms permit asymptotically correct and complete diagnosis using testing graphs in the form of rings and meshes.

Distributed diagnosis

Distributed self-diagnosis can be executed with minimal message overhead if complete testing is assumed. In deterministic distributed methods such as in Hosseini et al. [28] and Kuhl and Reddy [34], diagnostic information is only transmitted once from a unit u_i to another unit u_j that it tests to be fault-free. By passing information along paths of units known to be fault-free (assuming complete test coverage), no redundant messages are transmitted. If a single test may be incomplete, but a collection of tests conducted by different testers of the same unit is assumed to be complete (as in Buskens and Bianchini [11]), then a similar method can be used to distribute diagnostic information.

Probabilistic complete-test methods can also use the same technique during diagnosis. However, if the complete-test assumptions does not hold for the system, then faulty units can never be identified with 100 percent certainty, and so the above methods cannot be used. There exist three general approaches to perform probabilistic distributed self-diagnosis using the incomplete-test testing model. In the first class of methods the communication of test results is mixed in with the actual testing and diagnostic activities. Thus, redundant copies of the test results and/or partial diagnostic results must be broadcasted over multiple paths. The diagnosis takes into account the redundant copies received. In the second class of methods a reliable broadcast procedure is used to distribute the local test results of the units to every other unit. After the communication, each fault-free unit can execute the appropriate syndrome-decoding algorithm to form its diagnostic view about the overall system. Although the reliable broadcast operation generates extremely high message traffic, methods such as in Lee and Shin [38] have shown that the implementation can be quite efficient on regular mesh and hypercube topologies. In the third class of methods each unit only diagnoses the fault status of its immediate neighbors. This requires much less communication overhead than the previous alternatives.

The diagnosis method developed by Berman and Pelc is an example of the first approach to distributed self-diagnosis [3]. The algorithm is designed for a special class of testing graphs with $O(n \log n)$ edges. The graph can be partitioned into completely connected subgraphs with $\lceil c \log n \rceil$ units each. Within a completely connected subgraph a maximum clique of units connected by 0-links is found and labeled as fault-free; all other units in the subgraph are labeled as faulty. In the second stage, all units in each completely connected subgraphs receive the diagnostic information of all other units. The classification a given unit u_k belonging to a different subgraph as u_i is the majority value of the messages received by u_i on u_k. The computational complexity of this algorithm is $O(n^c)$, where c is a constant that depends on the probability parameters p_f and p_{ij}.

Pelc later improved on this result by introducing a modified diagnosis algorithm with an $O(n)$ computational complexity [49]. In his improved algorithm, the computationally expensive step of finding a maximum clique of units connected by 0-links does not exist. Instead, the set of units is partitioned into subsets of units with $\lceil c \log n \rceil$ units each. The subsets are then connected in a loop. All units within a subset test one another and all units within an adjacent subset. All test results are then communicated to the same set of units. This 'test-and-send' procedure is repeated r times, for a fixed r. Then majority voting is used to determine the faulty units, and this diagnosis result is communicated to other units using a procedure similar to that in Berman and Pelc [3].

The second approach to distributed self-diagnosis is very general but requires a high communication overhead. Each unit must initiate a reliable broadcast procedure to disseminate its test results among the other units in the system. This reliable broadcast procedure (also called multicast operation) is described in general terms by Yang and Masson in [69]. Algorithms designed for specific architectures such as hypercubes are also available [53]. After testing, suppose that the test evaluation results of each unit are combined into a single message. Up to t faulty units can be tolerated by having each unit send $2t + 1$ copies of its message to every other unit along unitdisjoint paths. Since there are $n(n-1)$ possible sender-receiver pairs, there are $(2t+1)n(n-1)$ such message transmissions. Lee and Shin [38] describe a method for implementing such an "all-to-all reliable broadcast" operation that provides a few milliseconds transmission time even on large systems with over a thousand units by using wormhole routing. After this reliable broadcast procedure, each unit can use majority voting to determine the test evaluation results of all other units.

The third approach to distributed probabilistic self-diagnosis applies to those diagnostic methods that classify each unit as faulty or fault-free based only on the outcomes of tests performed by its direct neighbors. In these methods, each unit u_i must reliably receive the tests results of units $u_j \in \Gamma^{-1}(u_i)$. If comparison testing is being used, then no extra communication is required for diagnosis since each unit knows the results of its comparison tests with its neighbors after the testing phase. In case of other testing techniques, it may be necessary to execute a multicast procedure so that each unit sends a message reliably to all of its adjacent units. Although this diagnostic approach has extremely low communication overhead when compared to the previous two ones, the diagnostic result is dispersed throughout the system. Therefore, the results of the individual diagnoses must be combined if it is necessary to obtain a global diagnostic image.

Bibliography

[1] M. Barborak, M. Malek, and A. Dahbura. The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.

[2] F. Barsi, F. Grandoni, and P. Maestrini. A theory of diagnosability of digital systems. *IEEE Transactions on Computers*, C-25(6):585–593, June 1976.

[3] P. Berman and A. Pelc. Distributed probabilistic fault diagnosis for multiprocessor systems. In *20th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 340–346, June 1990.

[4] R. Bianchini and R. Buskens. An adaptive distributed system-level diagnosis algorithm and its implementation. In *21st IEEE Int. Symp. on Fault-Tolerant Computing*, pages 222–229. IEEE, New York 1991.

[5] R. Bianchini and R. Buskens. Implementation of on-line distributed system-level diagnosis theory. *IEEE Transactions on Computers*, 41(5):616–626, May 1992.

[6] D. Blough, G. Sullivan, and G. Masson. Efficient fault diagnosis of multiprocessor systems under probabilistic models. *IEEE Transactions on Computers*, 41(9):1126–1136, September 1992.

[7] D. Blough, G. Sullivan, and G. Masson. Intermittent fault diagnosis in multiprocessor systems. *IEEE Transactions on Computers*, 41(11):1430–1441, November 1992.

[8] D. M. Blough. *Fault Detection and Diagnosis in Multiprocessor Systems.* PhD thesis, The John Hopkins University, Baltimore, 1988.

[9] D. M. Blough and A. Pelc. Diagnosis and repair in multiprocessor systems. *IEEE Transactions on Computers*, 42(2):205–217, February 1993.

[10] M. L. Blount. Probabilistic treatment of diagnosis in digital systems. In *7th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 72–77, June 1977.

[11] R. Buskens and R. Bianchini. Distributed on-line diagnosis in the presence of arbitrary faults. In *23rd IEEE Int. Symp. on Fault-Tolerant Computing*, pages 470–479, New York, 1993. IEEE.

[12] A. Dahbura and G. Masson. An $o(n^{2.5})$ fault identification algorithm for diagnosable systems. *IEEE Transactions on Computers*, C-33(6):486–492, June 1984.

[13] A. Dahbura and G. Masson. A practical variation of the $o(n^{2.5})$ fault diagnosis algorithm. In *14th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 428–433, New York, 1984. IEEE.

[14] A. Dahbura, G. Masson, and C. Yang. Self-implicating structures for diagnosable systems. *IEEE Transactions on Computers*, C-34(8):718–723, August 1985.

[15] A. Dahbura, K. Sabnani, and L. King. The comparison approach to multiprocessor fault diagnosis. *IEEE Transactions on Computers*, C-36(3):373–378, March 1987.

[16] M. Dal Cin. A diagnostic device for large multiprocessor systems. In *12th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 357–360, New York, 1982. IEEE.

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

[18] D. Dolev and R. Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM*, 32(1):191–204, January 1985.

[19] A. Friedman. A new measure of digital system diagnosis. In *5th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 167–169, New York, 1975. IEEE.

[20] A. D. Friedman and L. Simoncini. System-level fault diagnosis. *IEEE Computer*, 13(3):47–53, March 1980.

[21] H. Fujiwara and K. Kinoshita. On the computational complexity of system diagnosis. *IEEE Transactions on Computers*, C-27(10):881–885, October 1978.

[22] D. Fussell and S. Rangarajan. Probabilistic diagnosis of multiprocessor systems with arbitrary connectivity. In *19th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 560–565, New York, 1989. IEEE.

[23] R. Gupta and I. Ramakrishnan. System-level diagnosis in malicious environments. In *17th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 184–189, 1987.

[24] S. Hakimi and A. Amin. Characterization of connection assignment of diagnosable systems. *IEEE Transactions on Computers*, C-23(1):86–88, January 1974.

[25] C. Holt and J. Smith. Self-diagnosis in distributed systems. *IEEE Transactions on Computers*, 34(1):19–31, January 1985.

[26] S. Hosseini, J. Kuhl, and S. Reddy. A diagnosis algorithm for distributed computing systems with dynamic failure and repair. *IEEE Transactions on Computers*, C-33(3):223–233, March 1984.

[27] S. Hosseini, J. Kuhl, and S. Reddy. On self fault-diagnosis of the distributed systems. In *15th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 30–35, New York, 1985. IEEE.

[28] S. Hosseini, J. Kuhl, and S. Reddy. On self fault-diagnosis of the distributed systems. *IEEE Transactions on Computers*, 37(2):248–251, February 1988.

[29] R. K. Iyer. Experimental evaluation. In *Special Issue, 25th Int. IEEE Symp. on Fault-Tolerant Computing*, June 1995.

[30] T. Kameda, S. Toida, and F. Allan. A diagnosing algorithm for networks. *Information and Control*, 29:141–148, 1975.

[31] C. Kime. An analysis model for digital system diagnosis. *IEEE Transactions on Computers*, C-19(11):1063–1073, November 1970.

[32] C. Kime. System diagnosis. In D. Pradhan, editor, *Fault-Tolerant Computing: Theory and Techniques*, volume 2, chapter 8, pages 577–626. Prentice-Hall, Englewood Cliffs, N. J., 1986.

[33] S. Kreutzer and S. Hakimi. Adaptive fault identification in two new diagnostic models. In *21st Allerton Conf. on Communication, Control and Computing*, pages 353–362, Urbana, Ill., 1983. University of Illinois.

[34] J. Kuhl and S. Reddy. Distributed fault-tolerance for large multiprocessor systems. In *7th Ann. Symp. on Computer Architecture*, pages 23–30, 1980.

[35] L. Lamport, R. Shostak, and M. Pease. Tha Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, June 1982.

[36] J. C. Laprie, editor. *Dependability: Basic Concepts and Terminology*, volume 5 of *Dependable Computing and Fault-Tolerant Systems*. Springer, Wien, 1992.

[37] S. Lee. *Probabilistic Multiprocessor and Multicomputer Diagnosis*. PhD thesis, University of Michigan, Ann Arbor, Mich., 1990.

[38] S. Lee and K. Shin. Optimal multiple syndrome probabilistic diagnosis. In *20th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 324–331, New York, 1990. IEEE.

[39] S. Lee and K. Shin. Optimal and efficient probabilistic distributed diagnosis schemes. *IEEE Transactions on Computers*, 42(7):882–886, July 1993.

[40] S. Lee and K. Shin. Probabilistic diagnosis of multiprocessor systems. *ACM Computing Surveys*, 26(1):121–139, March 1994.

[41] C. Liaw, S. Su, and Y. Malaiya. Self-diagnosis of non-homogeneous distributed systems. In *12th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 349–352, New York, 1982. IEEE.

[42] J. Maeng and M. Malek. A comparison connection assignment for self-diagnosis of multiprocessor systems. In *11th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 173–175, 1981.

[43] S. H. Maheswari and S. L. Hakimi. On models for diagnosable systems and probabilistic fault diagnosis. *IEEE Transactions on Computers*, C-25(3):228–236, March 1976.

[44] M. Malek and Y. Maeng. Partitioning of large multicomputer systems for effective fault diagnosis. In *12th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 341–348, New York, June 1982. IEEE.

[45] S. Mallela and G. Masson. Diagnosis without repair for hybrid fault situations. *IEEE Transactions on Computers*, C-29(6):461–470, June 1980.

[46] G. Meyer and G. Masson. An efficient fault diagnosis algorithm for symmetric multiple processor architectures. *IEEE Transactions on Computers*, C-27(11):1059–1063, November 1978.

[47] A. Pataricza, K. Tilly, E. Selényi, and M. Dal Cin. A constraint based approach to system level diagnosis. Internal Report 4/1994., University of Erlangen-Nürnberg, Erlangen, June 1994.

[48] M. Pease, R Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[49] A. Pelc. Efficient distributed diagnosis in the presence of random faults. In *23rd IEEE Int. Symp. on Fault-Tolerant Computing*, pages 462–469, New York, June 1993. IEEE.

[50] F. P. Preparata, G. Metze, and R. T. Chien. On the connection assigment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, EC-16(6):848–854, December 1967.

[51] V. Raghavan and A. Tripathi. Improved diagnosability algorithms. *IEEE Transactions on Computers*, 40(2):143–153, February 1991.

[52] V. Raghavan and A. Tripathi. Sequential diagnosability is co-NP complete. *IEEE Transactions on Computers*, 40(5):584–595, May 1991.

[53] P. Ramanathan and K. Shin. Reliable broadcat in hypercube multicomputers. *IEEE Transactions on Computers*, 37(12):1654–1657, December 1988.

[54] S. Rangarajan and D. Fussell. Probabilistic diagnosis algorithms tailored to system topology. In *21st IEEE Int. Symp. on Fault-Tolerant Computing*, pages 230–237, New York, 1991. IEEE.

[55] S. Rangarajan and D. Fussell. Diagnosing arbitrarily connected parallel computers with high probability. *IEEE Transactions on Computers*, 41(5):606–615, May 1992.

[56] S. Rangarajan, D. Fussell, and M. Malek. Built-in testing of integrated circuit wafers. *IEEE Transactions on Computers*, 39(2):195–204, February 1990.

[57] J. Russell and C. Kime. System fault diagnosis: Closure and diagnosability with repair. *IEEE Transactions on Computers*, C-24(11):1078–1089, November 1975.

[58] J. Russell and C. Kime. System fault diagnosis: Masking, exposure, and diagnosability without repair. *IEEE Transactions on Computers*, C-24(12):1155–1161, December 1975.

[59] E. Scheinerman. Almost sure fault tolerance in random graphs. *SIAM Journal of Computing*, 16(6):1124–1134, December 1987.

[60] E. Selényi. *Generalized Theory of System-Level Diagnosis.* D.Sc. thesis, Hungarian Academy of Sciences, Budapest, 1985.

[61] L. Simoncini and A. D. Friedman. Incomplete fault coverage in modular multiprocessor systems. In *ACM Ann. Conf.*, pages 210–216, December 1978.

[62] A. Somani and V. Agarwal. Distributed syndrome decoding for regular interconnected structures. In *19th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 70–77, New York, 1989. IEEE.

[63] A. Somani and V. Agarwal. Distributed diagnosis algorithms for regular interconnected structures. *IEEE Transactions on Computers*, 41(7):899–906, July 1992.

[64] A. K. Somani, V. K. Agarval, and D. Avis. A generalized theory for system level diagnosis. *IEEE Transactions on Computers*, C-36(5):538–546, May 1987.

[65] M. Stahl, R. Buskens, and R. Bianchini. On-line diagnosis of general topology networks. In *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pages 114–121, New York, July 1992. IEEE.

[66] G. Sullivan. A polynomial time algorithm for fault diagnosability. In *25th Symp. on the Foundations of Computer Science*, pages 148–156, 1984.

[67] G. Sullivan. An $o(t^3 + |e|)$ fault identification algorithm for diagnosable systems. *IEEE Transactions on Computers*, 37(4):388–397, April 1988.

[68] J. Turek and D. Shasha. The many faces of consensus in distributed systems. *IEEE Computer*, 18(6):8–17, June 1992.

[69] C. Yang and G. Masson. A distributed algorithm for fault diagnosis in systems with soft failures. *IEEE Transactions on Computers*, 37(11):1476–1480, November 1988.