الجمهوريـة الجزائـريـة الديمقراطيـة الشعبيـة
People's Democratic Republic of Algeria
وزارة التعليـم العالـي والبحـث العلمـي
Ministry of Higher Education and Scientific Research

Nº Réf :……

Abdelhafid Boussouf
University Center– Mila

Institute of Mathematics and Computer Sciences

Department of Computer Sciences

Submitted for the degree of Master
In: Computer Sciences
Specialty:Artificial Intelligence

Failure Recovery and tolerant in large Distributed Graph Systems

Presented by: -RAWNAK Hammadi
- RYANE Amira

Defended on .../06/2024 In front of the jury

Abdelkader Kimouche	MAA	Abdelhafid Boussouf U. C. of Mila	Chairman
Mourad Geuttiche	MCB	Abdelhafid Boussouf U. C. of Mila	Supervisor
Adil Merabet	MAA	Abdelhafid Boussouf U. C. of Mila	Examiner

Academic year: 2023/2024

Remerciement

Nous remercions d'abord et avant tout Allah qui nous a donné le courage, la santé, la possibilité et la patience pour réaliser ce travail.

Un remerciement particulier à notre encadreur ***DR. MOURAD GUETTICHE*** *pour son soutient, son sérieux, sa disponibilité, ses précieux conseils et son aide tout au long de l'élaboration de ce travail.*

Nous remercions également, les membres du jury d'avoir accepté d'examiner et d'évaluer notre travail.

Sans oublier tous les enseignants du département d'informatique pour la qualité de l'enseignement qu'ils ont bien voulu nous prodiguer durant nos études afin de nous fournir une formation efficiente.

Nous n'aurions garde d'oublier tous ceux qui ont contribué de près ou de loin à la réalisation de ce mémoire et à tous ceux qui ont partagé avec nous les moments les plus difficiles dans la réalisation de ce travail et tous ceux qui nous souhaite le bon courage.

Finalement, nous remercions très sincèrement tous nos familles pour leur encouragement sans limite.

Rawnak et Ryane

إهداء

الحمد لله وما توفيقي الا به وبعد فإني أهدي عملي هذا :

الى من حملتني وهنا على وهن.... الى معنى الحب والحنان الى من ساندتني في رحلتي, **أمي** الغالية حفظك الله ورعاك الى من أحمل اسمه بإفتخار، **أبي** العزيز أدامك الله لنا، الى من لا أجد لها كلمات تعبير عن قيمتها **أمي الثانية** **جدتي** شفاك الله لنا ، إلى من تربيت على يده **جدي** الى من تمنت رؤية نجاحنا **جدتي** ، نجاحي ينقصه فخركم بي إلى **خالي الغالي** رحمكم الله واسكنكم فسيح جناته

ولا انسى اخوتي احبائي وجزء من روحي **كوثر** **عبد الرحمان أريج مهدي و جود** وبهجة عائلتنا **رتيل** صديقتي واختي **أية** ، إلى شريكة نجاحي **رونق** دمتم لي سندا لا يميل .إلى **رفيقات** **المشوار** اللواتي قاسمتني لحظاته رعاهم الله ووفقهم .إلى كل من كان لهم أثر في حياتي .
إلى كل من ساندني من قريب او بعيد لاتمام هذا العمل الذي

تم بحمد الله وفضله وكرمه.

ريان

اهداء

من قال أنا لها "نالها"

وأنا لها إن أبت رغما عنها أتيت بها.

الحمد لله الذي ما نجحنا وما علونا ولا تفوقنا إلا برضاه الحمد لله الذي ما اجتزنا دربا ولا تخطينا جهدا إلا بفضله وإليه ينسب الفضل والكمال والإكمال؛

إلى من جعل الله الجنة تحت أقدامها, من سهلت لي الشدائد بدعائها, سر قوتي ونجاحي. جنتي

والدتي "سليمة" متعها الله بالصحة و العافية؛

إلى من زين اسمي بأجمل الألقاب, من دعمني بلا حدود وأعطاني بلا مقابل سندي وقوتي وملاذي بعد الله. فخري واعتزازي؛

والدي"الوزناجي"حفظك الله لنا؛

ولا انسى احبائي و جزء من روحي الى من بهم يشد ساعدي وتعلى هامتي هم سندي وركائز نجاحي

إخواني "عبد الرؤوف , إمام, عماد"وكل **عائلتي،**

إلى **اسرتي الثانية من كانت سندي** اخص بالذكر **"ياسين"**

و أختي وصديقة الطفولة **"نجلاء"** الى شريكة نجاحي" **ريان"**

واخوتي اللاتي لم تنجبهم امي **"مريم، فدوى ،صفاء "**

دمتم لي سندا لا يميل الى رفيقات المشوار اللواتي قاسمتني لحظاته رعاهم الله ووفقهم،إلى من كان عونا وسندا في هذا الطريق..... للأصدقاء الأوفياء ورفقاء السنين لأصحاب الشدائد والأزمات، إلى من ضاقت السطور عن ذكرهم فوسعهم قلبي

اهديكم هذا الإنجاز وثمرة نجاحي الذي لطالما تمنيته ,ها أنا اليوم أتممت أول ثمراته بفضل من الله عز وجل؛

فالحمد لله على ما وهبني وأن يعينني ويجعلني مباركة أينما كنت.

رونق

Abstract

In this work, we explore fundamental concepts for studying a Failure Recovery and Tolerant in large Distributed Graph Systems.
We propose a novel approach based on Luby's Maximal Independent Set (MIS) algorithm, designed to enhance fault tolerance in distributed settings. Our approach involves distributed selection of coordinator nodes using the modified MIS algorithm, ensuring robustness and resilience to nodes failures.a guard node is assigned Each coordinator to take over in case of failure.
Furthermore, we assign each node in the system to a specific coordinator to ensure balanced network load and efficient resource utilization. The proposed system has been validated, demonstrating its effectiveness in managing large-scale graph computations while ensuring resilience to node failures.
Keywords: Distributed graph system,Luby's Maximal Independent Set (MIS) algorithm, distributed algorithm

Résumé

Dans ce travail, nous explorons des concepts fondamentaux pour atteindre la tolérance aux pannes et la récupération dans les grands systèmes de graphes distribués. Nous proposons une nouvelle approche basée sur un algorithme Luby de l'ensemble indépendant maximal (MIS) , conçu pour améliorer la tolérance aux pannes dans les environnements distribués. Notre approche implique la sélection distribuée des noeuds coordinateurs en utilisant l'algorithme MIS modifié, garantissant la robustesse et la résilience aux pannes des noeuds. Chaque coordinateur se voit attribuer un noeud de garde pour prendre le relais en cas de panne. De plus, nous attribuons chaque noeud du système à un coordinateur spécifique afin d'assurer une charge réseau équilibrée et une utilisation efficace des ressources.
Le système proposé a été validé, démontrant son efficacité dans la gestion des calculs de graphes à grande échelle tout en assurant la résilience aux pannes de noeuds.
Mots clés : Système de graphes distribués, algorithme d'ensemble indépendant maximal de Luby (MIS), algorithme distribué.

الملخص

في هذا العمل، نستكشف مفاهيم أساسية لتحقيق تحمل الأخطاء والاستعادة في الأنظمة الموزعة الكبيرة المعتمدة على الشبكات .نقترح نهجًا جديدًا يعتمد على خوارزمية Luby MIS ، المصممة لتحسين تحمل الأخطاء في البيئات الموزعة .يتضمن نهجنا اختيارًا موزعًا للعقد المنسقة باستخدام خوارزمية luby MIS المعدلة، مما يضمن المتانة والقدرة على تحمل أعطال العقد .يتم تعيين حارس لكل منسق لتولي المهمة في حالة حدوث عطل .بالإضافة إلى ذلك، نقوم بتعيين كل عقدة في النظام إلى منسق محدد لضمان توازن الحمل الشبكي واستخدام الموارد بشكل فعال، تم التحقق من فعالية النظام المقترح، مما يظهر كفاءته في إدارة حسابات الشبكات الكبيرة مع ضمان تحمل أعطال العقد.

الكلمات المفتاحية:

الشبكات، خوارزمية luby mis ،الخوارزمية الموزعة.

Contents

List of Figures

List of Algorithms

List of Tables

General introduction

In today's interconnected world, distributed systems have become an integral part of our daily lives, supporting various applications such as cloud computing, social media, and online banking. However, the complexity of these systems also introduces challenges, particularly in maintaining fault tolerance. Failure Recovery and Tolerance in Large Distributed Graph Systems is a thesis that aims to explore the strategies and techniques for enhancing fault tolerance in distributed systems, with a specific focus on large distributed graph systems.

The primary objective of this thesis is to investigate the importance of fault tolerance in distributed systems and the various methods used to achieve it. We will discuss the benefits of fault tolerance, such as increased availability, scalability, and performance, as well as the challenges it poses, including complexity, resource overhead, and balancing consistency and availability.

To address these challenges, our approach involves a modified version of Luby's algorithm. In this modified algorithm, a guard node is designated for each node in the Maximum Independent Set (MIS). If an MIS node fails, its guard immediately replaces it, ensuring continuous system operation without significant interruptions. This method aims to enhance fault tolerance by providing a reliable mechanism for node replacement in large distributed graph systems.

This general introduction provides an overview of the topic, context, and problem. The rest of the thesis is organized as follows:

Chapter 1: This chapter entitled **Graph Theory and Parameters**, in which we introduce the most appropriate model in the field of distributed systems which is graph. First, we explores the foundational concepts of graph theory including nodes, edges, node degree, sub graphs, connected components.Then we examine essential parameters such as dominating set, independent set, matching and critical nodes. Since the complexity of the algorithms that determine the values of these parameters is very high in general, it is very important to review the different distributed and parallel approach to determine these parameters. The self-stabilizing approach appears to be a promising approach for overcoming the problem of complexity, in addition, it is fault-tolerant, which allows dealing with the main problem of distributed systems.

Chapter 2: This chapter will be devoted to **Distributed Systems and Fault Tolerance**, The second chapter covers distributed systems from various perspectives, defining their core concepts and examining their usage in distributed com-

puting. It illustrates fault tolerance through concrete examples, such as blockchain technology, and reviews the state of the art of fault tolerance in sensor networks.

Chapter 3: the final chapter **Topological fault tolerance approach in distributed computing system** in which we explain our contribution for fault tolerance in distributed computing systems. We introduce a distributed approach that strengthens the coordinator nodes set represented by the maximum stable set as a set of replacemnt ones called guard set. This lead to reinforce the system allows Failures recovery. The approach also dispose a network decomposition which facilitate fault recovery. Finally, the approach is tested and its performance is heiglighted.

Chapter 1

Graph Theory and Parameters

1.1 Introduction

In the realm of general graphs, graph theory stands out for its ability to model and analyze a wide range of structures and phenomena. Continuously evolving, this theory provides a powerful framework for understanding complex relationships among various sets of objects. Applied across diverse fields such as transportation networks, information networks, and electronic circuits, it enables the exploration and interpretation of data from various contexts, thus offering fresh and valuable insights.

In this chapter, we will delve into the key parameters of graphs, starting with their basic definitions. We will then examine distributed algorithms designed to solve problems in environments where resources are distributed across a network of interconnected nodes. Additionally, we will address self-stabilizing algorithms aimed at ensuring the system returns to a correct state regardless of its initial condition. Finally, we will discuss fault tolerance in self-stabilizing algorithms, highlighting the importance of maintaining system stability even in the presence of failures. By combining these concepts, we will establish a comprehensive framework for understanding and analyzing graph parameters, providing valuable insights for effectively solving various problems in distributed and resilient contexts.

1.2 General concepts

Definitions

Undirected Graph: An undirected graph G is a pair G = (V, E) such that:

- V is a finite set of vertices, the graph is mentioned in case of confusion as V(G).
- E is a set of unordered pairs of vertices $\{v_i, v_j\} \in V^2$, the graph is denoted as E(G) to avoid confusion.
  A pair $\{v_i, v_j\}$ is called **an edge**. We say that vertices v_i and v_j are **adjacent**.The edge $\{v_i, v_j\}$ is called an edge **incident** to v_i and v_j **[01]**.

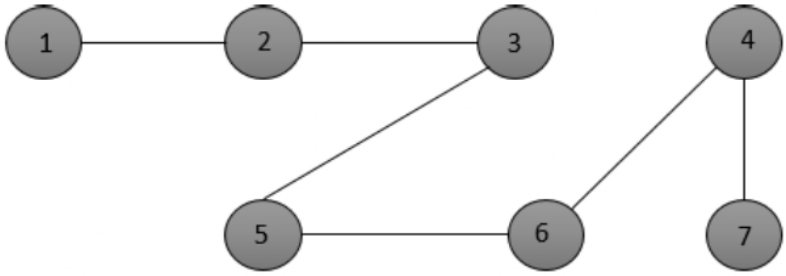

Figure 1.1: *Undirected Graph*

Exemple 01:
The graph in Figure 1.1 represents an undirected graph $G = (V, E)$ with $V = \{1, 2, 3, 4, 5, 6, 7\}$ and $E = \{\{1, 2\}, \{3, 5\}, \{2, 3\}, \{5, 6\}, \{4, 6\}, \{4, 7\}\}$.

Directed Graph: A directed graph G is a pair $G = (V, E)$ such that:

- V is a finite set of vertices.
- E is a set of ordered pairs of vertices $(v_i, v_j) \in V^2$.

An ordered pair (v_i, v_j) is called an arc, and is graphically represented by $v_i \to v_j$. v_i is the initial or source vertex, and v_j is the terminal or destination vertex. The arc $a = (v_i, v_j)$ is said to be outgoing from v_i and incident to v_j, and v_j is a successor of v_i, while v_i is a predecessor of v_j. The set of successors of a vertex $v_i \in V$ is denoted by $Succ(v_i) = \{v_j \in V \mid (v_i, v_j) \in E\}$. The set of predecessors of a vertex $v_i \in V$ is denoted by $Pred(v_i) = \{v_j \in V \mid (v_j, v_i) \in E\}$**[01]**.

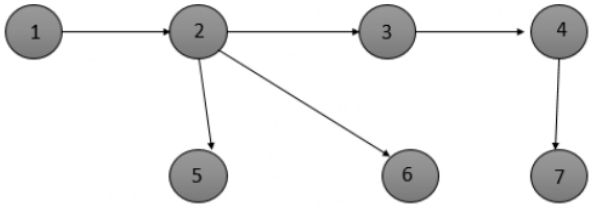

Figure 1.2: *irected Graph*

Example 02: The graph in Figure 1.2 represents a directed graph $G = (V, E)$ with $V = \{1, 2, 3, 4, 5, 6, 7\}$ and $E = \{\{1, 2\}, \{2, 5\}, \{2, 6\}, \{2, 3\}, \{3, 4\}, \{4, 7\}\}$.

Simple Graph: A simple graph is a graph that does not contain any self-loops or multi-edges. In other words, each edge connects two distinct vertices, and there is at most one edge between any pair of vertices **[03]**. The graph in Figure 1.3 represents a Simple Graph.

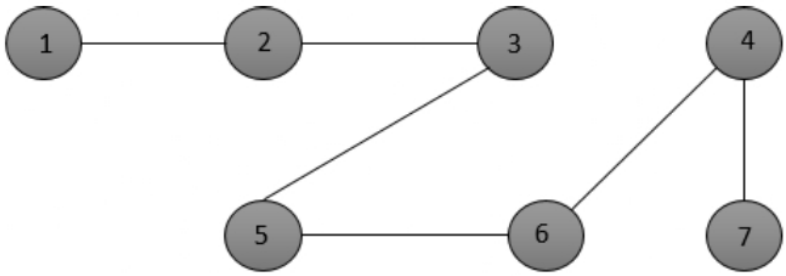

Figure 1.3: *Simple Graph*

Multi-graph: A multi-graph is a graph that may have self-loops and/or multi-edges. This means that it allows for multiple edges between the same pair of vertices and also permits edges that connect a vertex to itself**[04]**. The graph in Figure 1.4 shows a multi- Graph.

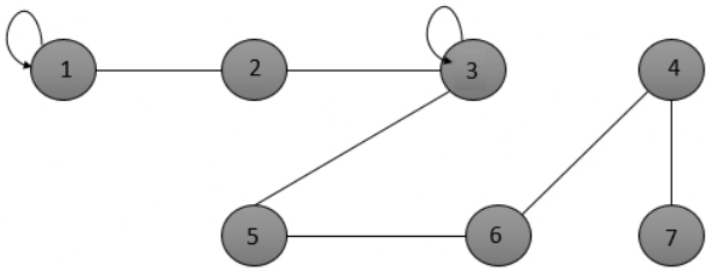

Figure 1.4: *multi- Graph*

Complete graph: A complete graph is an undirected simple graph in which every pair of vertices is adjacent. The complete graph with n vertices is denoted K_n. The complement of an undirected simple graph G is the graph H such that $G + H$ is a complete graph.

Subgraph: Graph G' (V', E') is an induced subgraph of G if $V' \subseteq V$ and E' is the collection of edges in G among that subset of vertices.

Consider a given graph $G(V, E)$.

Example 03: The graph in Figure 1.5 represents Undirected Graph and Figure 1.6 represents an induced subgraph of G by $V' = \{1, 2, 3, 5\}$: $G' = G[V']$.

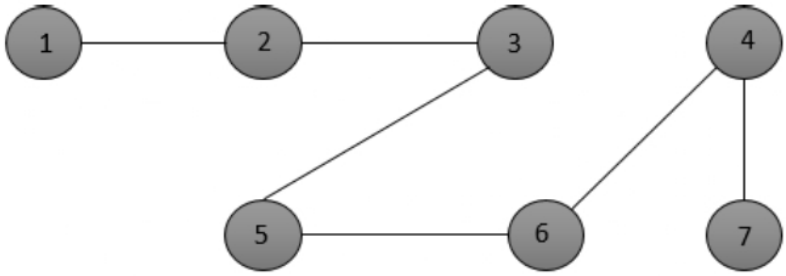

Figure 1.5: *Undirected Graph*

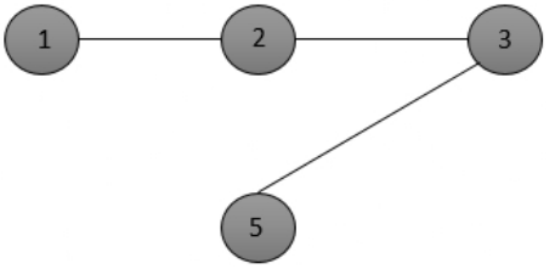

Figure 1.6: *subGraph*

Degree: The degree d_v of a vertex v in a graph G is the number of edges incident to v. In other words, it is the number of edges connected to the vertex v.

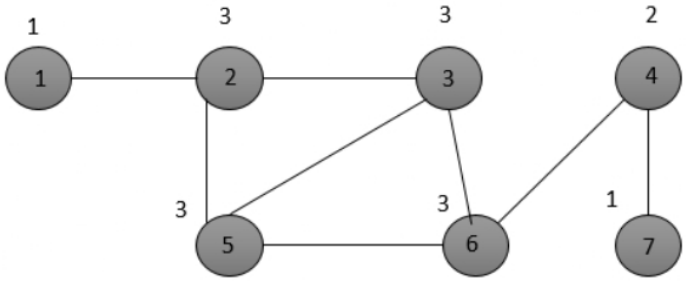

Figure 1.7: *degree Graph*

Example 04: The graph in Figure 1.7 represents the degree of a vertex $V' = \{1, 2, 3, , 4, 5, 6, 7\}$: $dv = \{1, 3, 3, 2, 3, 3, 1\}$.

1.3 Graph Parameters

Typically, we solve problems using graphs, by interpreting the problem in question using one of the graph parameters. For example, to disseminate information in a telecommunication network, we choose a certain number of nodes represented by a dominant set, which enables efficient dissemination protocols while minimizing cost. To this end, in what follows, we will try to review the most important graph parameters.

1.3.1 Graph Matching

An exact graph matching is a correspondence between the vertices of two graphs, G_1 and G_2, such that if there is an edge between two vertices in the first graph, then the corresponding vertices in the second graph are also connected by an edge. Various variants of graph matching exist, including isomorphism, subgraph isomorphism, monomorphism, homomorphism, and maximal isomorphic subgraph**[01]** .

1.3.2 Dominating Set

A dominating set in a graph comprises vertices where each vertex either belongs to the set or is adjacent to a set vertex. Essentially, these sets enable monitoring or controlling the entire graph through their vertices. For instance, in a social network, a dominating set could represent influential users capable of influencing others' behaviors **[12]**. A dominating set is said to be minimal dominating set (MDS) if it does not contain another dominating subset **[12]**. Dominating set has several variants, among which we cited:

- **Total Dominating Set (TDS)**:

  In a total dominating set, every graph vertex either belongs to the set or is adjacent to a set vertex. This concept is crucial in communication networks where every vertex requires surveillance or control. For example, in a sensor network, a total dominating set ensures complete coverage and monitoring . A minimal total dominating set (MTDS) is a total dominating set that does not contain another total dominating subset .

- **Distance-k Dominating Set (DKDS)**:

  A distance-k dominating set (DKDS) is a subset of nodes such that for each node (i) in the set of nodes (V) that is not in the dominating set (V-S) there exists a node (j) in the dominating set (S) such that the distance between them is less than k .

- **k-dominating Set (MKDS)**: A minimal k-dominating set (MKDS) is a subset of nodes such that each node (i) in the set of nodes (V) that is not in the dominating set (V-S) is adjacent to at least k nodes in the dominating set (S).

- **Connected Dominating Set (CDS)**:

  A connected dominating set is a dominating set where the induced subgraph (formed by set vertices and their edges) remains connected. This is vital in wireless sensor networks to maintain connectivity. By having a connected dominating set, communication pathways stay intact even if some sensors fail or get disconnected .

- **Independent Dominating Set (IDS)**:

  An independent dominating set is a dominating set where no two set vertices are adjacent. This parameter is significant in reducing interference in wireless networks. For instance, in frequency assignment scenarios, arranging transmitting devices in an independent dominating set configuration helps minimize signal interference .

- **Strongly Dominating Set (ISDS)**:

  A strongly dominating set is dominating set in the graph such that evry vertex not in the set is adjacent to a vertex in the set of greater degree. The strongly dominating set is said to be strongly independent dominating set if no two vertices are connected by an edge in it **[13]**.

The graph in Figure 1.8 represents some dominanting set vriant

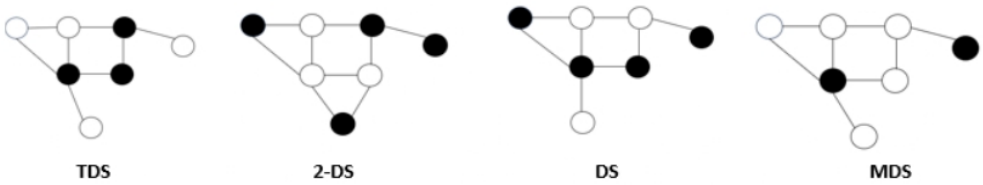

Figure 1.8: *example of dominant set*

1.3.3 Critical nodes

Critical nodes represent the most important nodes in the the graph i.e deletion of which destroy maximally the network connectivity. For that, find the critical nodes and protect them leads to keep the reliability and enhance network security. This enables them to be the key players in communication, social and transport networks.
There exists several vairant of critical nodes problem (CNP) including, K-CNP which seeks to determine a set of k-nodes deletion of which minimizes the pairwise connectivity (the set of connected pairs in the network), CCNP **[54]** for Component-cardinality-constrained critical node problem which aims the deletion of a minimal set of nodes to bound the network pairwise connectivity and 3CCNP **[55]** which seeks to deletion of $k-$nodes to minimises the largest connected component in the residuel graph.

Interested reader can be reffered to the recently published survey for more details on critical nodes and their applications .

Most decision problem problem related to aforementioned parameters determining are NP-complete ones.Thus, it is worth interesting to use distributed approach for dealing with the heigh complexity of sequential classical appraoch.

1.3.4 Independent set

An Independent set in a graph is a set of vertices where no two vertices are connected by an edge. A stable set is said to be maximum if there is no larger stable set in the graph, and maximal if the set cannot be enlarged to a bigger stable set. The number of vertices in a maximum independent set of a graph G is referred to as the independence number of G, denoted as $\alpha(G)$**[04]**.

Exmaple 05: Figure 1.9 illustrates a maximal independent set consisting of three vertices in the Petersen graph, while Figure 1.10 depicts a maximum stable

- **Strongly Dominating Set (ISDS)**:

  A strongly dominating set is dominating set in the graph such that evry vertex not in the set is adjacent to a vertex in the set of greater degree. The strongly dominating set is said to be strongly independent dominating set if no two vertices are connected by an edge in it **[13]**.

The graph in Figure 1.8 represents some dominanting set vriant

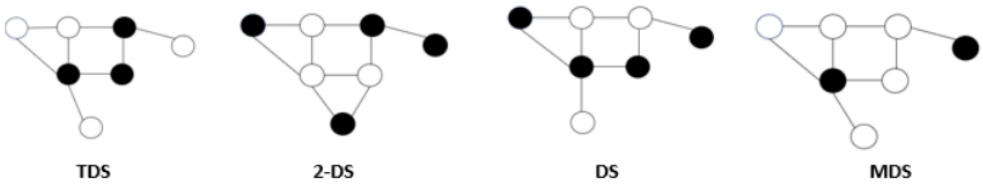

Figure 1.8: *example of dominant set*

1.3.3 Critical nodes

Critical nodes represent the most important nodes in the the graph i.e deletion of which destroy maximally the network connectivity. For that, find the critical nodes and protect them leads to keep the reliability and enhance network security. This enables them to be the key players in communication, social and transport networks.
There exists several vairant of critical nodes problem (CNP) including, K-CNP which seeks to determine a set of k-nodes deletion of which minimizes the pairwise connectivity (the set of connected pairs in the network), CCNP **[54]** for Component-cardinality-constrained critical node problem which aims the deletion of a minimal set of nodes to bound the network pairwise connectivity and 3CCNP **[55]** which seeks to deletion of $k-$nodes to minimises the largest connected component in the residuel graph.

Interested reader can be reffered to the recently published survey for more details on critical nodes and their applications .

Most decision problem problem related to aforementioned parameters determining are NP-complete ones.Thus, it is worth interesting to use distributed approach for dealing with the heigh complexity of sequential classical appraoch.

1.3.4 Independent set

An Independent set in a graph is a set of vertices where no two vertices are connected by an edge. A stable set is said to be maximum if there is no larger stable set in the graph, and maximal if the set cannot be enlarged to a bigger stable set. The number of vertices in a maximum independent set of a graph G is referred to as the independence number of G, denoted as $\alpha(G)$**[04]**.

Exmaple 05: Figure 1.9 illustrates a maximal independent set consisting of three vertices in the Petersen graph, while Figure 1.10 depicts a maximum stable

set consisting of four vertices in the Petersen graph (Petersen graph P, $\alpha(P) = 4$).

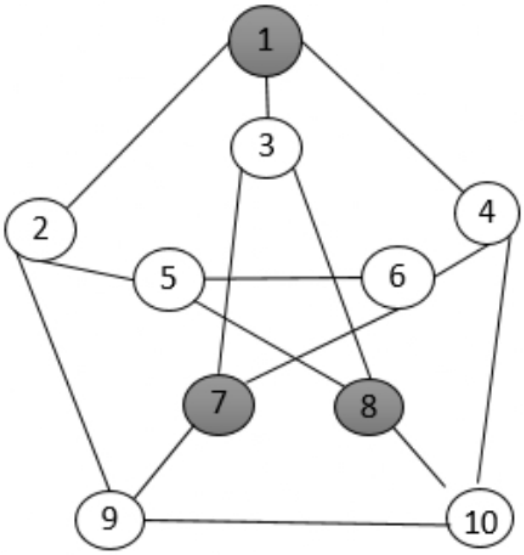

Figure 1.9: *a maximal stable set*

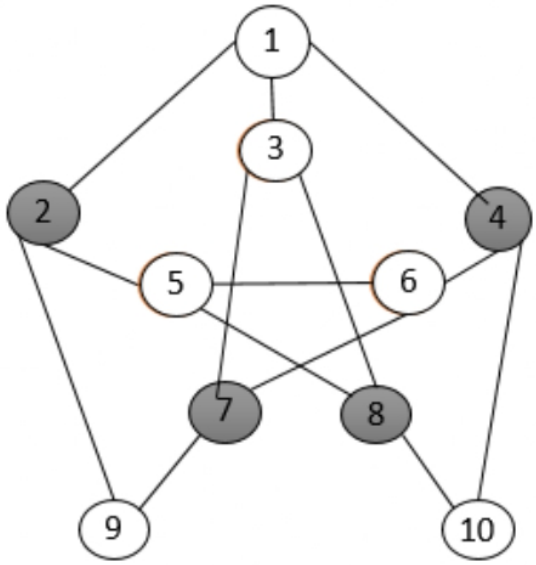

Figure 1.10: *a maximum stable set*

1.4 Complexity Classes

Since the majority of problems relating to graphs are NP-hard problems, so it is worth interesting to highlight the different complexity classes of problems .

Class NP (Non-deterministic Polynomial:)

The class NP contains all decision problems for which solutions can be verified in polynomial time **[08]**.

NP-complete Problems (NPC):

A problem is NP-complete if it is both in the class NP and every problem in NP can be reduced to it in polynomial time **[09]**.

NP-hard Problems (NPH):

A problem is NP-hard if it has the property that all problems in NP can be reduced to it in polynomial time **[10]**.
Figure 1.11 illustrates the Relationship between classes of complexity NP,NP-complete and NP-hard.

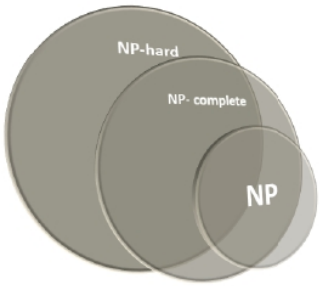

Figure 1.11: *Relationship between classes of complexity*

1.5 Distributed algorithms

Distributed algorithms are specialized algorithms designed for a group of independent computing units, each executing its own code to achieve a common goal. These algorithms operate under the premise of limited global state awareness, where each process only has knowledge of its local state and possesses partial information about the system. It is commonly assumed that processes can receive information from others, but this information may become outdated due to system changes or sender state alterations.
Classic problems addressed by distributed algorithms include **[07]**:

- **Leader election**: the process of selecting a known process in the system to make decisions and act as a reference point.
- **Information dissemination**: the act of sending information to a group of identified processes, necessitating specific communication primitives.
- **Mutual exclusion**: ensuring that shared resources are accessed in a serialized and fair manner to prevent simultaneous use.
- **Virtual topology construction**: some algorithms are tailored for specific topologies that aid in program execution control. However, real distributed systems may lack these ideal topologies, requiring higher-level distributed algorithms to virtually construct them.

1.5.1 Types of Distributed Algorithms

- **Uniform Algorithm**: In this type of algorithm, all processes execute the same code.
- **Semi-Uniform Algorithm**: In a semi-uniform algorithm, all processes execute the same code except for one process that runs a different code.
- **Non-Uniform Algorithm**: In a non-uniform algorithm, each process has its own unique code.

1.6 Self-stabilization

Self-stabilizing algorithms are distributed algorithm which ensure to fault tolerance constraint. This enables a prominent approach in graph parameters computation.

In what follows, we introduce such an approach and review its application in field of graph theory.

Self-stabilization is a fault-tolerance technique that addresses transient faults in a distributed system, it was introduced by E. W. Dijkstra in 1974. A distributed algorithm is considered self-stabilizing if, from any initial state, it is guaranteed to reach a correct state after a finite time, in other words, when a transient fault (temporary interruption) affects one or more components, the algorithm may pass transiently but an incorrect configuration known as an illegitimate configuration and recover the correct execution (legitime state) in an infinite time **[11]**.

Now, we present some self-stabilizing distributed algorithms that enable the construction of dominant and independent sets in an arbitrary graph.

1.7 self-stabilizing distributed algorithms of dominant and independent sets

Hedetniemi et al. **[14]** introduce few self-stabilizing distributed algorithms, for computing a domnating set and maximal independent set. these one utilize a centralized demon (coordinator). The proposed algorithms are straightforward and operate in an arbitrary topology.

Turau **[11]** introduced two algorithms for calculating a Maximal Independent Set (MIS) and a Minimal Dominating Set (MDS) using a distributed demon. Additionally, Turau proposed an enhancement to improve the performance of these algorithms.

Chiu et al. **[20]** recently proposed a self-stabilizing algorithm for the MDS problem that reaches a stable state with only $4n - 2$ movements, utilizing a distributed demon. This algorithm is simple and works in an arbitrary topology.

Goddard et al. **[15]** presented an algorithm for computing a Maximal Independent Set (MIS) that is deterministic and utilizes a distributed demon. The algorithm assumes that each node has a unique identifier. They also proposed an algorithm to calculate a minimal dominating set (MDS) using a distributed demon. The algorithm assumes that each node i has a variable (i) indicating whether the node is in the dominating set S or not.

Neggazi et al. **[16]** presented an auto-stabilizing algorithm for calculating an Independent Strongly Dominating Set (ISDS) in a graph, using a distributed demon and operating in an arbitrary topology. Each node is assumed to have a unique identifier ID. The idea of this algorithm is that a node i becomes dominant if it does not have a stronger neighbor. In other words, it becomes dominant if it has the highest degree and the highest identifier. The neighbors of i then become strongly dominated nodes.

Srimani et al. **[21]** introduced a deterministic and uniform algorithm that computes a minimal total dominating set (MTDS) in an arbitrary graph. The algorithm employs a centralized demon and assumes that nodes have unique identifiers.

Huang et al. **[19]** presented an algorithm for finding an M2DS in an arbitrary graph using a centralized demon. This algorithm is an extension of the result found

Algorithme	Resultat	Anonymity	Demons	Complexity
Hedetniemi[8]	DS	Yes	centralized	O(n)Mouvements (n-1 Mouv.)
Hedetniemi[8]	MIS	Yes	centralized	O(n)Mouvements (2nMouv.)
Hedetniemi[8]	MDS	Yes	centralized	$O(n^2)$Mouvements
Turau[8]	MIS	No	Distributed	O(n)Mouvements
Turau[8]	MDS	No	Distributed	O(n)Mouvements
Chiu[8]	MDS	No	Distributed	O(n)Mouvements (4n-2 Mouv.)
Goddard[8]	MIS	Yes	Distributed	O(n)Rounds
Goddard[8]	MDS	Yes	Distributed	O(n)Mouvements
Neggazi[8]	ISDS	No	Distributed	O(n)Rounds (n+1 Round.)
Srimani[8]	MTDS	No	Centralized	not mentioned
Huang[8]	MKDS(K=2)	No	Distributed	not mentioned
Lin[8]	DKDS(K=2)	No	Centralized	$O(n^i)$Mouvements

Table 1.1: Summary of the self-stabilizing algorithms present.

by Shukla et al. **[18]** that allows for the computation of an MIS, exploitaing the concept of a Maximal Independent Set (MIS) can be viewed as a minimal I-dominating set, therefore Minimal 2-Dominating Set (M2DS) is an extension of the MIS construction problem.

Lin et al. **[17]** proposed a self-stabilizing distributed algorithm to find a Minimal Distance-2 Dominating Set (MD2DS) in an arbitrary graph.

1.8 Conclusion

In this chapter, we have presented the basic concepts relative a graph tool and its parmeteres. We have seen that almost decision problem relatives to these parameters determining are NP-Complete ones. Thus we use the distributed approach to deal with heigh complexity and also de self-stablizing approach to deal with the fault occur whenever we distributed the computation to seeral machines. To better understand our problematic, the second chapter is devoted to the topic of fault tolerance and recovery in distributed system.

Chapter 2

Distributed Systems and Fault Tolerance

2.1 Introduction

Fault tolerance in distributed systems constitutes a critical area of modern computer science, where the reliability and availability of systems are paramount to ensure continuous and effective operations. In an interconnected digital world, distributed systems are ubiquitous, powering our online interactions, cloud services, social networks, and much more. However, these systems face a constant challenge: the possibility of hardware or software failures that could compromise their normal operation. Fault tolerance aims to address this challenge by ensuring that distributed systems can maintain proper functioning even in the presence of faults or failures occurring at the level of individual components, such as servers, network connections, or software.

This general introduction will explore the fundamental principles of fault tolerance in distributed systems, shedding light on the challenges encountered, the strategies employed, and the implications of this discipline for the reliability and resilience of modern computing infrastructures.

2.2 Distributed systems

Distributed systems are defined as a collection of multiple independent systems connected together as a single system. Each independent system has its own memory and resources, while some common resources and peripheral devices are shared among the connected devices. The design of distributed systems involves connecting all nodes or devices, even if they are located at long distances. The challenges faced by distributed systems include fault tolerance, transparency, and communication primitives **[45]**.

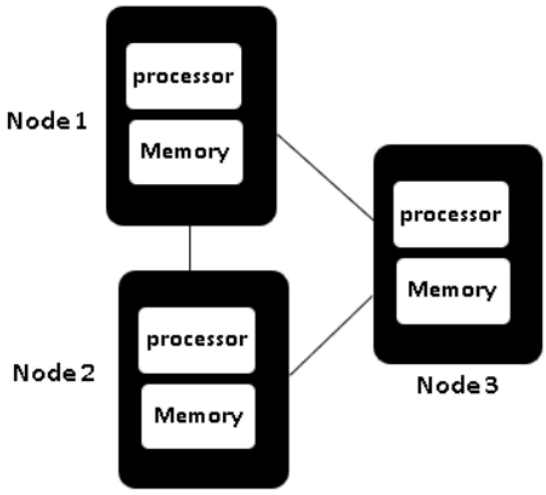

Figure 2.1: *Distributed system.*

2.2.1 Types of fault distributed systems

- **Transient Faults:** Transient faults are characterized by their occurrence once and subsequent disappearance. While they generally don't cause substantial harm to the system, they pose a significant challenge in terms of detection and localization. The typical example of transient faults is the processor fault **[22]**.
- **Intermittent Faults:** Intermittent faults manifest as recurring occurrences, where the fault arises, resolves on its own, and then recurs periodically. A classic example of an intermittent fault is when a functioning computer freezes intermittently **[22]**.
- **Permanent Faults:** Permanent faults persist within the system until the faulty component is replaced. While they have the potential to cause severe damage to the system, they are relatively easy to identify. A prime example of a permanent fault is a burnt-out chip **[22]**.

2.3 Fault tolerance system

A fault tolerance system is designed to maintain proper program execution and continue functioning correctly even in the face of partial failures. While system performance may be impacted by these failures, fault tolerance mechanisms aim to ensure operational continuity [**Bib45**]. Failures in fault tolerance systems can be attributed to hardware or software issues (Node Failure) or unauthorized access (Machine Error). These events are categorized into performance, omission, timing, crash, and fail-stop errors [**45**].

- **Performance:** Occurs when hardware or software components fail to meet user demands .
- **Omission:** Involves components unable to execute specific commands .
- **Timing:** Refers to components failing to execute commands at the correct time .
- **Crash:** Indicates components that fail without recovery .
- **Fail-stop:** When software detects errors, it halts the process, which is straightforward but may not handle complex situations effectively. Respond to the different types of errors mentioned above, three distinct error scenarios can be identified:

  Permanent Error: Causes lasting damage to software components, requiring program restarts after crashes .

  Temporary Error: Results in brief software component damage, resolving over time to restore normal function .

  Periodic Errors: Occur intermittently, like software conflicts when multiple programs run simultaneously, necessitating program exits to resolve conflicts.

Most computers incorporate fault tolerance techniques like microdiagnosis, parity checking, and watchdog timers. Incompletely fault-tolerant systems may reduce computing capabilities by removing programs or slowing down processes due to hardware configuration issues or design flaws .

2.3.1 Replication mechanism

When considering fault tolerance, replication is commonly employed as a general method to safeguard against system failures. Sebepou et al.[**45**] pointed out three principal replication mechanisms, namely :

- **State Machine:** In this method, the process state of a computer system is replicated across autonomous systems simultaneously. All replica nodes handle data in a analogous or matching manner, ensuring coordination among them. Additionally, all inputs are distributed to every replica at the same time. An active replica serves as an illustration of a state machine .

- **Process Pairs:** The process involves a master (primary)/slave (secondary) relationship in replication coordination. The primary workstation assumes the role of a master, transmitting its respective input to the secondary node. Both nodes maintain a reliable communication link .

- **Roll Back Recovery (Check-Point-Based):** This mechanism momentarily collects checkpoints and then transfers these checkpoint states to either a stable storage device or backup nodes. This enables a roll back recovery to be done successfully during or after the recovery process. The checkpoint is reconstructed to its state prior to the most recent state .

2.3.2 Fault tolerance objectives :

- **Availability:** Availability refers to how ready and accessible the system is for immediate use. It means the probability that the system is operating correctly at any given time, so that it can perform its intended tasks for users. In simple terms, a highly available system is one that is expected to be up and running whenever it is needed **[22]**.

- **Reliability:** Reliability is measured over a set timeframe, while availability is an instantaneous metric. A highly reliable system is expected to function without interruption for a long duration. The distinction is subtle but important. For example, a system that briefly goes down every hour would have high availability at any moment, but low reliability over time due to the repeated outages. On the other hand, a system that shuts down for two weeks every August may have high reliability when operational, but lower overall availability that year. So reliability and availability measure different things - one tracks failure-free operation over time, while the other looks at the readiness to execute tasks right now. They are separate metrics that can't be used interchangeably **[22]**.

- **Safety:** Safety refers to preventing catastrophic events from happening, even if the system experiences a temporary failure or malfunction. Safety is extremely important in certain critical industries like nuclear power plants, spacecraft systems, etc. These systems absolutely must maintain a high level of safety because even a brief failure could lead to disastrous consequences. There have been many historical examples that show how difficult it is to build truly safe systems that never allow failures to cause catastrophes. Ensuring safety is an ongoing complex challenge, as even momentary system failures cannot be allowed to result in dangerous or catastrophic outcomes in these safety-critical environments.

  The key point is that safety goes beyond just system uptime or reliability - it's about ensuring no failures can cause catastrophic events, no matter how brief the failure is. This makes achieving true safety an immense challenge, especially for critical systemsr **[22]**.

- **Maintainability:** Maintainability refers to how easily a failed system can be repaired and restored to proper working condition. A system with high maintainability can be fixed quickly and efficiently after a failure occurs. Maintain-

able systems often also have high availability, especially if they can automatically detect and recover from failures on their own. An ideal scenario is for the system to repair itself without manual intervention. However, achieving this level of automatic recovery from failures is very difficult in practice. While self-healing systems are the goal, implementing this level of maintainability is a major challenge. So in summary, maintainability focuses on how rapidly a broken system can be repaired and made operational again after a failure, either through manual repairs or ideally through automatic recovery capabilities built into the system itself[**22**].

2.3.3 Fault handling steps

The fault handling approahc involves the following steps(see Figure 2.2):

- **Fault Detection:** Fault detection represents the initial phase of continuous system monitoring. It involves comparing actual outcomes with expected ones, and any discrepancies are promptly identified and notified. These faults may arise from hardware failures, network issues, or software malfunctions. The primary objective of this phase is to swiftly detect faults as they arise to prevent delays in assigned tasks. [**22**].

- **Fault Diagnosis:** Fault diagnosis entails thoroughly examining the fault identified in the initial phase to determine its root cause and likely characteristics. This process can be carried out manually by an administrator or through automated techniques to rectify the fault and fulfill the designated task. [**22**].

- **Evidence Generation:** Evidence generation refers to the process of compiling a fault report based on the diagnosis conducted in a preceding phase. This report outlines the causes and characteristics of the fault, potential solutions for resolution, as well as alternative measures and preventative actions to be taken into account. [**22**].

- **Assessment:** Assessment involves analyzing the damages resulting from faults. This evaluation often relies on messages relayed from the affected component. Subsequent decisions are then based on this assessment. [**22**].

- **Recovery:** Recovery is the endeavor to restore the system to a fault-free state. This involves implementing various techniques such as reconfiguration and resynchronization, aiming to return the system to both its forward recovery and backup recovery states [**22**].

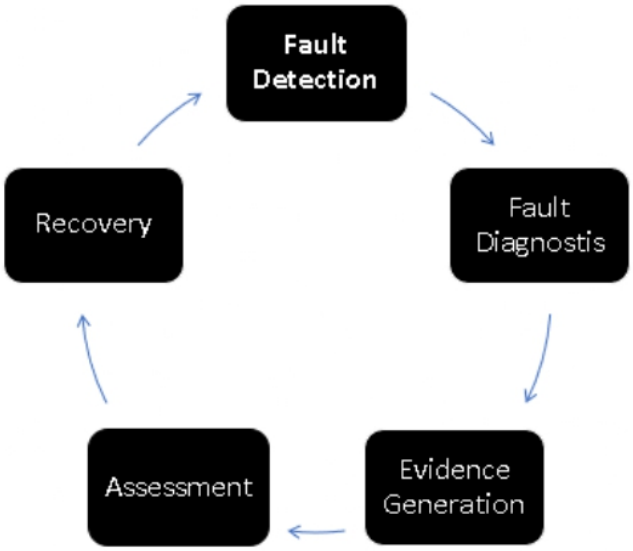

Figure 2.2: *Distributed system fault dealing.*

2.3.4 Fault-tolerant mecanism

- **Hardware Fault Tolerance:** Hardware fault tolerance involves the provision of additional backup hardware components such as CPUs, memory, hard disks, and power supply units. While hardware fault tolerance can't prevent or detect errors like accidental program interference or program errors, it provides a backup system for hardware failures. Systems employing hardware fault tolerance automatically address faults arising from hardware components. Typically, this technique divides the system into units, each equipped with redundant backup, so if one module fails, others can assume its function. Two common approaches to hardware fault recovery are Fault Masking and Dynamic Recovery **[45]**.

- **Software Fault Tolerance:** Software Fault Tolerance refers to a strategy employed to mitigate errors in software systems. It involves utilizing specialized software to identify and rectify invalid output, runtime glitches, and programming mistakes. This approach employs both static and dynamic techniques for error detection and resolution. Additionally, Software Fault Tolerance incorporates mechanisms like recovery rollback and checkpoints to enhance resilience **[45]**.

- **System Fault Tolerance:** System fault tolerance encompasses a comprehensive system that not only stores checkpoints but also detects errors within applications. It automatically stores memory blocks and program checkpoints. In the event of a fault or error, the system offers a corrective mechanism to rectify the error **[45]**. the table2.1 illustrates Comparison of three fault tolerance mechanism.

Mechanism	Hardware Fault Tolerance	Software Fault Tolerance	System Fault Tolerance
Major technique	Hardware backup	Checkpoint storage Rallback	Architecture with error detecting and correcting
Design complexity	Low	Medium	High
Time/cost expenditure	Low	Medium	High
Fault-tolerance Level	Low	Medium	High

Table 2.1: Comparison of fault tolerance mechanism.

2.4 Fault tolerance in distributed computing system

A distributed computing system with fault tolerance refers to a network of interconnected and autonomous computing entities that collaborate to achieve a common goal despite potential failures. Fault tolerance in distributed computing involves the ability of the system to continue operating reliably even in the presence of faults or failures, ensuring uninterrupted service and preventing catastrophic system breakdowns.

2.4.1 Exemple of Blockchain:

The blockchain technology can also be considered as an example of a distributed computing system with a certain degree of fault tolerance, although fault tolerance in this context is somewhat different from that of other systems.

Blockchain is a decentralized and distributed database that contains an immutable ledger of transactions, spread across multiple network nodes. Each block of transactions is cryptographically linked to the previous one, thus forming a chain of blocks.

Fault tolerance in a blockchain relies on the principles of decentralization and distributed consensus. Here's how it works:

Decentralization: In a blockchain, data is replicated across many network nodes, meaning there is no single point of failure. Even if some nodes fail, others continue to operate normally, ensuring continuous availability of data and services.

Distributed consensus: Blockchains use distributed consensus algorithms, such as proof of work or proof of stake, to reach agreement on the state of the ledger and the validity of transactions. These mechanisms help maintain the integrity of the blockchain even in the presence of malicious or failing nodes.

Although blockchains are designed to be resilient to faults and attacks, they are not entirely immune. Issues such as chain forks and 51 attacks can compromise the security and availability of the network under certain circumstances. However, security mechanisms and consensus protocols continue to evolve to strengthen blockchain resilience against these potential threats.

We can classify distributed computing solutions into two gategories, Checkpoint-Based and checkpoint free ones.

2.4.2 Checkpoint-Based Solutions:

Checkpoint-based solutions involve the system creating checkpoints or maintaining replicas during normal execution. These checkpoints or replicas are utilized for

recovery in case of failure[**26**].

Systems like Pregel[**30**], GraphLab, GraphX, and PowerGraph employ checkpoint-based approaches to tolerate failures. They periodically create checkpoints during normal execution and reload the latest checkpoint upon failure.

There exists several techniques to improve Checkpoint-Based techniques such as lightweight check-pointing reduce the volume of checkpoint data by saving vertices and incrementally storing edges. Other strategies, like unblocking checkpointing, aim to decrease the overhead associated with blocking checkpoints. For example, CoRAL applies unblocking checkpointing to asynchronous graph processing systems.

While checkpoint-based solutions are effective in tolerating failures, they can incur significant overhead. These methods may not be suitable for all scenarios, particularly those involving topological mutations.

2.4.3 Checkpoint-Free Solutions:

Checkpoint-free solutions achieve failure recovery without relying on checkpoints or replicas. Instead of storing checkpoints, these solutions employ alternative methods for recovery [**26**]. Optimistic recovery, proposed by Schelter et al.[**42**], reloads lost partitions from input data and applies compensations algorithmically upon failure. Zorro [**43**]utilizes implicit replicas of vertices to recover lost vertex values and accelerate recovery. Phoenix [**44**]classifies existing graph algorithms into categories and provides tailored APIs for each category to achieve failure recovery.

Checkpoint-free solutions offer advantages such as reduced overhead and improved efficiency in recovery, especially in scenarios involving topological mutations. However, they may require careful consideration of the recovery process and its impact on system performance.

2.5 Literature review

Xu, Chen, et al.[**26**] proposed ACF2 to address the limitations of checkpoint-free solutions. ACF2 introduces two key components: a partition-aware backup strategy and an incremental protocol. Unlike traditional methods that require reloading input data, the partition-aware backup strategy retrieves lost sub-graphs directly from backups stored on the DFS (Distributed File System). This approach minimizes the recovery overhead typically associated with checkpoint-free solutions.

Shenet al.[**23**] proposed an innovative approach for rapid failure recovery within distributed graph processing systems. Their task involved emphasizing the necessity for an effective recovery strategy and advocating for parallelizing the recovery process to align with the evolving demands of modern graph-based Big Data applications. The article outlines a novel method involving partitioning the lost graph components among a subset of operational nodes to enhance system performance compared to conventional checkpoint-based methods. Their notable contributions include formalizing the issue of failure recovery, introducing a partition-centric recovery approach, and proposing a parallel recovery mechanism.

Han et al.[**57**] propose a comprehensive evaluation framework for Pregel-type graph processing systems, focusing on Giraph, GPS, Mizan, and GraphLab. Their study systematically compares these systems by analyzing their use of graphical and algorithmic optimizations to efficiently handle large-scale graph processing. By

assessing performance metrics such as execution time, scalability, and resource utilization, the evaluation provides a detailed and balanced comparison. This approach aims to inform users and researchers about the most suitable systems for various graph processing tasks within distributed computing models.

Yinghua Tong et al.[**24**] focuses on a fault tolerance mechanism tailored for cluster heads in wireless sensor networks. This mechanism integrates static backup and dynamic timing monitoring to enhance reliability and fault detection capabilities. It introduces a CH reliability model based on the Markov model, determines the minimum required number of CHs for reliability, and quantitatively analyzes energy consumption and latency. The study's contributions lie in its innovative approach to improving fault tolerance for cluster heads, offering practical and theoretical insights in the field.

Weiyu Zhong et al.[**25**] offers a comprehensive review of Byzantine Fault-Tolerant (BFT) consensus algorithms, which are crucial for ensuring safety and liveness guarantees in distributed systems. The study of BFT algorithms has gained significant attention due to the rapid growth of blockchain applications, as they enable tolerance of arbitrary faults in server actions. The paper delves into fundamental BFT algorithms that achieve consensus and fault tolerance, including classic consensus algorithms such as PBFT and Hotstuff. These algorithms have been widely implemented in blockchain applications, providing high throughput and low latency. However, they also face challenges such as bad behavior of master nodes, high network communication overhead, and low system flexibility. The paper also discusses improvements to the PBFT consensus algorithm, including those based on blockchain technology, which can enhance the performance and reliability of BFT algorithms.

Since sensor and Internet of things(IOT) are two sensitives area of distributed system, it worth interesting to review the articles that addressed the Fault-tolerance in such domains.

2.5.1 Fault tolerance in sensor networks

Wireless Sensor Networks (WSNs) consist of decentralized arrays of small, energy-efficient devices furnished with sensors, processors, and wireless communication capabilities. These sensors are strategically dispersed throughout a geographic region referred to as the coverage zone or areas of interest [**31**]. Collaboratively, they gather, analyze, and relay data pertaining to the surrounding environment or system to a central device known as a sink node or base station. This central node further processes the data, often computing key metrics such as maximum, average, or median values [**32**].

Fault tolerance in sensor networks refers to the network's ability to continue operating reliably despite hardware or software failures that may occur in individual sensors or within the network itself. As sensors are often deployed in harsh and dispersed environments, they are susceptible to various types of failures, such as sensor failures, radio interference, communication losses, etc[**33**]. To ensure fault tolerance in sensor networks, several techniques can be utilized:

- **Sensor Redundancy:** Data is collected by multiple sensors, and in the event of one sensor failure, others can take over to ensure continuity of data collection.

- **Resilient Routing:** Routing protocols in sensor networks can be designed to account for failed nodes or degraded communication links by selecting alternate paths to transmit data.
- **Auto-Configuration and Self-Repair:** Sensor networks can be equipped with auto-configuration and self-repair mechanisms, where defective sensors can be automatically detected and replaced without human intervention.
- **Energy Conservation:** Energy management protocols can be used to extend the lifespan of sensors in case of failures by adjusting energy consumption and disabling non-essential sensors.
- **Congestion Control Protocols:** To avoid network congestion that could be caused by failures or faults, congestion control protocols can be implemented to regulate data flow and prevent bottlenecks.

By incorporating these and other techniques into the design and deployment of sensor networks, it is possible to ensure effective fault tolerance and maintain the reliability and availability of collected data even in challenging environments.

2.5.2 Fault tolerance approaches in wireless sensor networks and IOT

2.5.3 Checkpoint Approach

Belkadi, et al.[**36**] introduced a novel method for fault tolerance in WSNs, comprising two distinct phases. The initial phase identifies critical nodes within the network whose failure substantially impacts network performance. Subsequently, the second phase implements an augmentation technique to sustain network connectivity in the event of failure of one of these critical nodes.

In[**37**]A. Ghaffari and S. Nobahary . proposed a new method focusing on a fault detection method in clustered wireless sensor networks utilizing a genetic algorithm. The proposed method is based on a majority vote mechanism that can accurately detect permanently faulty sensor nodes. It emphasizes high accuracy and low false alarm rates in identifying faulty nodes within the network clusters.

In [**38**],Fan et al. introduced an optimized machine learning technology scheme for fault detection in wireless sensor networks. Additionally,Fan et al. propose an innovative approach to enhance fault detection in these networks by integrating Particle Swarm Optimization (PSO) with machine learning. The authors present a novel technical solution that combines evolutionary computing and machine learning to address the challenge of fault detection in data collection within wireless sensor networks

Rajasegarar, Sutharshan, et al. [**39**]proposed a solution to the challenge of identifying misbehavior in wireless sensor networks, focusing on monitoring, fault diagnosis, and intrusion detection. Their approach involves a distributed, cluster-based anomaly detection algorithm aimed at minimizing communication overhead and energy consumption. By clustering sensor measurements and merging clusters before transmitting cluster descriptions to other nodes, they reduce communication requirements.

Cheng, Yong, and colleagues.[40] proposed a novel fault detection mechanism for sensor networks that addresses the limitations of existing approaches, which often result in low precision and high complexity. To improve fault detection, they introduced a mechanism based on support vector regression and neighbor coordination. The mechanism builds a fault prediction model using support vector regression and meteorological data from a multi-sensor. This model generates residual sequences that are used to identify node status through mutual testing among reliable neighbor nodes. The proposed mechanism also reduces communication to sensor nodes and is suitable for fault detection in meteorological sensor networks with low node densities and high failure ratios.

Munir, Arslan, Joseph Antoon, and Ann Gordon-Ross [41] proposed a fault-tolerant approach for Wireless Sensor Networks (WSNs) to meet the application requirements of lifetime and reliability. They utilized NS-2 to simulate fault detection algorithms and develop Markov models for reliability characterization. The study emphasizes the significance of integrating fault detection and fault tolerance in modeling Fault-Tolerant (FT) WSNs and provides insights into future research directions for enhancing the reliability and trustworthiness of WSNs. The results demonstrate that an FT WSN comprising duplex sensor nodes can notably enhance Mean Time to Failure (MTTF) and reliability compared to a Non-Fault-Tolerant (NFT) WSN.

2.5.4 Conclusion

In this chapter, we have address fault tolerance in distributed systems from various perspectives.First, We hae defined distributed systems and examined their basic concepts. Additionally, we have looked into distributed computing systems and their fault tolerance, highlighting the specific challenges encountered in this domain. Finally, we have explored fault tolerance in sensor networks.

Chapter 3

Topological fault tolerance approach in distributed computing system

3.1 Introduction

In an increasingly interconnected world where computer systems play a crucial role in our daily lives, fault tolerance becomes a major concern. Failures, whether due to hardware malfunctions, software errors, or malicious attacks, can seriously compromise system operation, causing disruptions and significant damage. Yet, even in a fault-prone environment, it is imperative that computer systems continue to function reliably and efficiently.

In this chapter, we will explore an effective approach, based graph topological, for fault tolerance in distributed computing system. The approach consists of distributing checkpoint in some selected nodes while assign to each one a guard that replace it whenever a fault occurs. In this case we consider a maximal independent set as the coordinator nodes, in which the checkpoints are saved, while implementing distributed algorithms that allows computing a maximal dependent set, assigning guards and ensure a network decomposition that enable an equiload-balancing between selected nodes. The efficiency of the proposed approach is validated in large graphs and its validity is highlighted in a case study of a real distributed system.

In what follows, we explain our approach based distributed checkpoint to ensure failure recovery and tolerant in distributed system .

3.2 A distributed checkpoint based approach for fault tolerance in distributed system

As aforementioned, the first step of the proposed approach consists on selecting the set of nodes in which the checkpoints are saved. For that, we opt for a maximal independent set which computing using Luby's algorithm.

3.2.1 Maximal independent set computing

The Luby's algorithm, pioneered by Michael Luby, is a probabilistic method for approximating a Maximum Independent Set (MIS) in a graph. It assigns random probabilities to each vertex and then selects vertices to form a maximum independent set using a probabilistic greedy strategy. While it provides a polynomial-time approximation, the resulting set's size may vary. This algorithm is utilized when precise accuracy is not crucial, but a rapid solution is required **[47]**.

3.2.1.1 Distributed Luby's Algorithm for MIS Determining

Algorithm 1 Distributed MIS

1: $I \leftarrow \emptyset$ **Input:** Graph(V, E)
Output: A maximal independent set $I \in V$.
2: status(v) = undecided
3: **while** status(v) = undecided **do**
4: status(v) = yes // belongs to MIS
5: **if** $d(v) = 0$ **then**
6: status(v) = yes // belongs to MIS
7: **else**
8: v marks itself with probability 1/(2d(v))
9: **end if**
10: **end while**
11: Round 1:{
12: notifies neighbors that it is marked and also sends its current degre
13: **if** v receives a message from a marked neighbor of higher degree (or equal degree but higher ID) **then**
14: v unmarks itself
15: **end if**
16: **if** v is still marked **then**
17: status(v) = yes
18: **end if**}
19: Round 2:{
20: v notifies all neighbors its status
21: **if** v receives a message from a neighbor that is in MIS **then**
22: status(v) = no
23: **end if**}

In what follows, we provide the distributed running of Luby's algorithm.

Distributed Luby's Algorithm for MIS computing

- The algorithm operates in synchronous rounds, grouped into phases. A single phase is as follows:
  - Node v marks itself with probability $2/d(v)$, where $d(v)$ is the current degree of v
  - If no higher degree neighbor of v is also marked, node v joins the MIS. If a higher degree neighbor of v is marked, node v unmarks itself again. (If the neighbors have the same degree, ties are broken arbitrarily, e.g., by identifier).
  - Delete all nodes that joined the MIS and their neighbors, as they cannot join the MIS anymore.

For example, in the graph of figure 3.1 we have $d(A) < d(B)$ then we kept the node B in MIS and reject A .

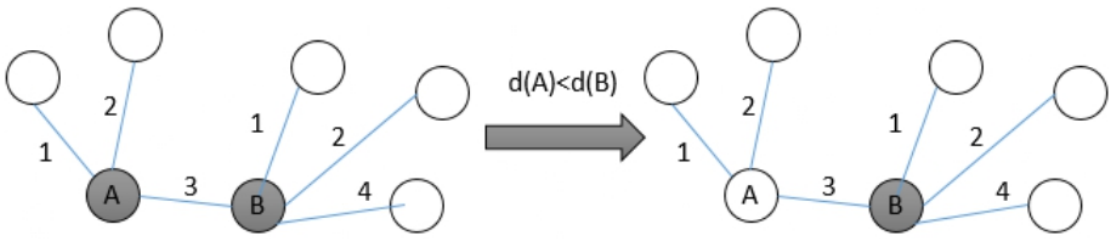

Figure 3.1: *Coordination between neighboring nodes*

- The Correctness in the sense that the algorithm produces an independent set is relatively simple: Steps 1 and 2 make sure that if a node v joins the MIS, then v's neighbors do not join the MIS at the same time. Step 3 makes sure that v's neighbors will never join the MIS.
- The algorithm halts when the state of each node is different to undecided.

3.2.2 The Complexity of Luby's Distributed MIS Algorithm

Luby's Distributed MIS Algorithm offers an efficient solution to the maximal independent set problem in distributed systems. With an expected time complexity of $O(\log n)$ and a high probability time complexity of $O(\log d \cdot \log n)$, where n is the number of vertices and d is the maximum vertex degree, the algorithm ensures scalability and practicality. It operates efficiently for large graphs, with logarithmic time relative to the number of vertices, and also achieves logarithmic time complexity with high probability concerning both the maximum degree and vertex count. This highlights the algorithm's effectiveness and adaptability in various network configurations and graph sizes within distributed computing environments [**48**].

3.2.3 Simulated Annealing for Graph Optimization

Simulated Annealing (SA) is an optimization technique inspired by the annealing process in metallurgy, where a material is heated to a high temperature and then slowly cooled to remove defects, thereby reaching a stable, low-energy state. The technique mimics the physical process of annealing by allowing the algorithm to probabilistically accept worse solutions early on, facilitating the escape from local optima, and focusing more on the exploitation of the best solutions as the process progresses.

Simulated Annealing is particularly effective for optimization problems that have a large search space and may contain many local optimum.

In this context, we apply Simulated Annealing to enhance the Maximum Independent Set (MIS) solution obtained from Luby's algorithm.

3.2.3.1 Simulated annealing for Luby MIS

Algorithm 2 Simulated annealing for Luby MIS

$I \leftarrow \emptyset$ **Input:** Graph G=(V, E), $X_0 < -Luby(G(V, E))$
Output: An optimized maximal independent set $X_b \in V$.
Initialization:
$T = 1000$ //
$X_B = zero[n]$ // Vector of n zeros
while $T > 1$ **do**
  Reset G(V,E) // reset the graph to its initial state
  $X_n < -neighbo\_solution(X_0)$
  $F(X_n)$ // Evaluate the fitness of the new solution by summing the elements of the solution vector.
  Validity Check(X) // Ensure the new solution is valid by checking that no two nodes in the MIS are adjacent.
  $r < -Randomnumberbetween0and1$
  $c = F(S_N) - F(S_0)$
  if ($\Delta > 0$ or $r < e^{-(\Delta/T)}$) **then**
    $S_0 < -S_n$
    if ($S_n > S_b$) **then**
      $S_b < -S_n$
    end if
  end if*update*(T) // Use the Boltzmann factor ($np.e^{(-\Delta/T)}$)
end while

3.2.3.2 Algorithm Process:

0. Initialization:

   - The process begins by obtaining an initial solution using Luby's algorithm.
   - An initial high temperature (temp = 1000) is set to permit wide exploration of the solution space.

- A decay rate (alpha = 0.9) is defined to control the rate of temperature reduction.

0. Iteration

- Graph Reset: For each iteration, reset the graph to its initial state.
- Solution Generation: Run Luby's algorithm on the reset graph to generate a new potential solution
  .
- Fitness Calculation: Evaluate the fitness of the new solution by summing the elements of the solution vector.
- Validity Check: Ensure the new solution is valid by checking that no two nodes in the MIS are adjacent.
- Acceptance Probability: Calculate the change in fitness (delta) between the new solution and the current solution. Use the Boltzmann factor (np.exp(-delta / temp)) to probabilistically decide whether to accept the new solution, allowing acceptance of worse solutions early in the process.
- Update Solution: If the new solution is accepted, update the current solution. If it improves upon the best known solution, update the best solution.
- Temperature Decay:Reduce the temperature by multiplying it by the decay rate (alpha).

0. Termination:

- Continue the process until the temperature drops below a certain threshold (e.g., $T > 1$).
- The best solution found during the process is retained as the optimized result.

3.3 Performance Comparison: Luby's Algorithm and Simulated Annealing Method

In the context of computing the maximum independent set (MIS), we have assessed the performance of two approaches: Luby's algorithm and a simulated annealing method. We conducted a series of tests on graphs of varying sizes, ranging from 16 to 500 nodes. Luby's algorithm, known for its efficiency in graph theory, employs a probabilistic approach to identify a maximum independent set. The table below

summarizes the results obtained using Luby's algorithm, highlighting the number of MIS nodes determined and the total execution time for each graph size.

number of node	**16**	**30**	**50**	**100**
number of MIS node	6	10	12	13
total execution time	2.35	3.38	23.78	21.13

Table 3.1: Performance of Luby's Algorithm on Different Graph Sizes.

Compared to the simple execution of Luby's algorithm, the simulated annealing method generally yielded larger maximum independent sets, particularly for larger graph sizes. However, this improvement in solution quality came at the cost of increased execution time, as the annealing process required additional computational effort to explore and refine the solutions. The table below illustrates the performance of the simulated annealing method, showcasing the number of MIS nodes obtained and the corresponding execution times for each graph size.

number of node	**16**	**30**	**50**	**100**
number of MIS node	8	12	15	18
total execution time	60.35	175.13	646.12	758.31

Table 3.2: Performance of Simulated Annealing Method on Different Graph Sizes.

3.3.1 Augmented Luby's Algorithm: Introducing Fault Tolerance with Guard Nodes

To ensure resilience against failures, we have developed a new algorithm that aims to designate guards to replace failing nodes in the maximum independent set (MIS). This approach guarantees the persistence of critical functions in a distributed environment, even in the event of hardware or software malfunctions. The proposed algorithm to ensure this resilience in the designation of guards to replace failing nodes in the maximum independent set (MIS) can be described as follows:
Replacement Selection: When a MIS node is identified as failing, the other members of the maximum independent set take action to replace it. They select a replacement from among the available neighbors based on well-established criteria. First, the neighbor with the highest degree is chosen to ensure continuity in monitoring. In case of a degree tie, priority is given to the node with the lowest ID. Furthermore, if multiple failing MIS nodes share a neighbor with the maximum degree, that neighbor is selected as the replacement to ensure consistency and efficiency of the monitoring.
The procedure that allows guard attributing is called automatically after adding a node to the MIS and it work as follows.
Notice that the procedure is also called after each iteration of Simulated annealing whenever a MIS node is replaced.

Exemple. The graphs in following Figures explain the mechanism of guard selection. In the graph of Figure 3.2, the algorithm selects the maximal node degree which is the node 4. Whenever the maximal degree is confused between several nodes, the algorithm chooses the minimal identifier one (see Figure 3.3). confused

Algorithm 3 Procedure for Guard Designation in Maximum Independent Set (MIS) with Fault Tolerance

Input: Graph(V, E)
Output: Guards set of the maximum independent set (MIS)
if v has not already a guard in its neighborhood **then**
    v Send message its max degree neighbor: "You are guard"
    Wait for message of guard designation confirmation
    if message received **then**
        $max_d efgre(V).guard < -True$
    end if
end if

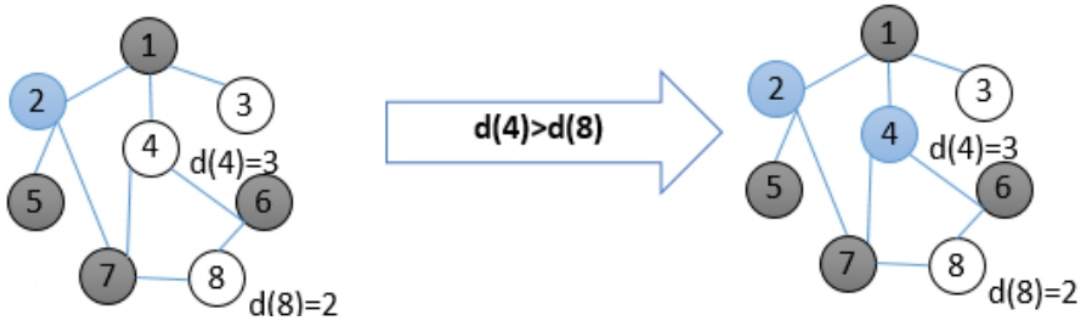

Figure 3.2: *case of maximum degree*

Note also Once the replacement is designated, all members of the maximum independent set send a message to the new guard to complete the tasks.

If the MIS node fails, the newly appointed sentinel responds immediately by not only sending the MIS node messages to all neighbors of the failed node, but also these neighbors send a message to the sentinel to take over. This approach aims to maintain maximum continuity Independent cluster monitoring operations despite node failure, thus ensuring the robustness of the distributed system even under degraded conditions.

Exemple 3.4. The graph in Figure 3.4 represents node Mis 1 has broken down the guard node 2 has to be replaced by making the node MIS tasks

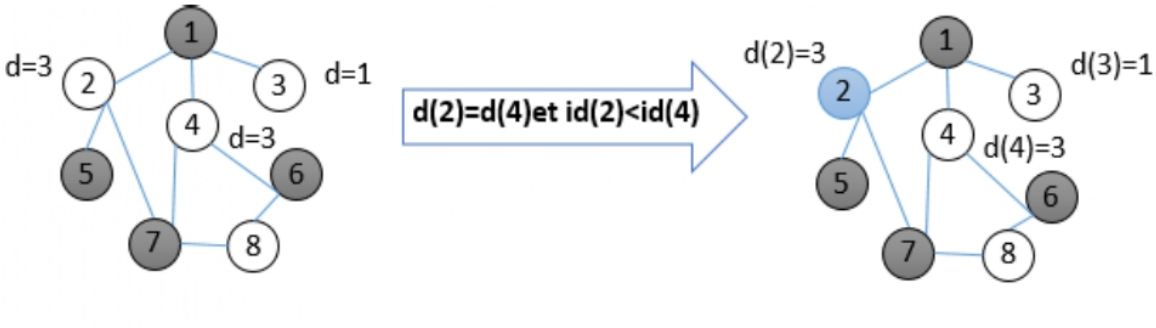

Figure 3.3: *case of equal degrees*

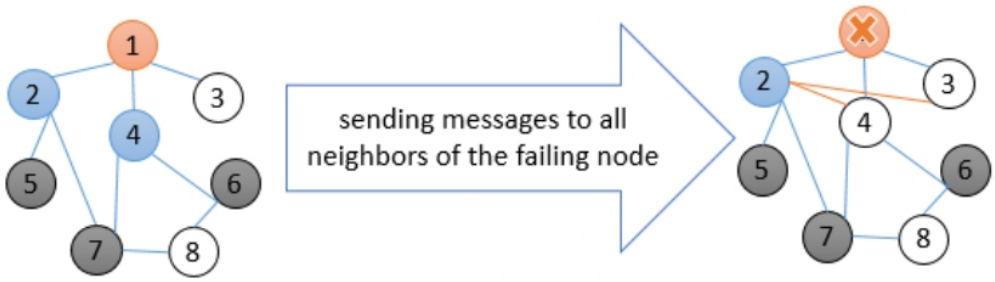

Figure 3.4: *replace MIS node failure*

3.4 Network Equilibrium Achieving

In simple terms, an equilibrium state is reached in a network whenever:

3.4.1 Balanced Allocation of Nodes

The balancing problem among nodes entails determining the optimal allocation of common neighboring nodes to different designated nodes. It involves deciding which designated node should be assigned the common neighboring nodes to ensure a balanced and efficient distribution of loads.

3.4.2 Adherence to Thresholds

Each MIS node adheres to the following defined thresholds:

0. **Minimum Threshold:** This is equal to the minimum degree of the graph.
0. **Maximum Threshold:** This is calculated as: (Totalsize - (Number of MIS nodes + Number of already assigned guards)) / Number of MIS nodes.

3.4.3 Equitable Distribution

By following these thresholds, each MIS node receives an equitable share of adjacent nodes. This ensures a uniform distribution of assignments and prevents any overload or under-utilization of nodes within the network. In summary, the equilibrium state is achieved when the allocation of adjacent nodes to the MIS nodes is balanced, and

each MIS node stays within the defined minimum and maximum thresholds, leading to a fair and efficient distribution of resources across the network.

3.4.4 Drawbacks of Balanced Allocation

While the balanced allocation of nodes to MIS nodes is an effective approach to achieving equilibrium in a network, there are some drawbacks to consider. One of the main issues is that certain nodes MIS may reach their maximum threshold while others do not, leading to some nodes remaining unaffected in the graph. This can occur when unaffected nodes are solely adjacent to nodes MIS that have already reached their maximum threshold.

3.4.5 Solution to Imbalance

In such scenarios, the only solution to balance the allocation is to assigned these unaffected nodes (which are adjacent to MIS node) to the corresponding node MIS, even if it means deviating from the strictly defined thresholds. This approach ensures that all nodes in the network are utilized, preventing any overload or underutilization of nodes within the network.

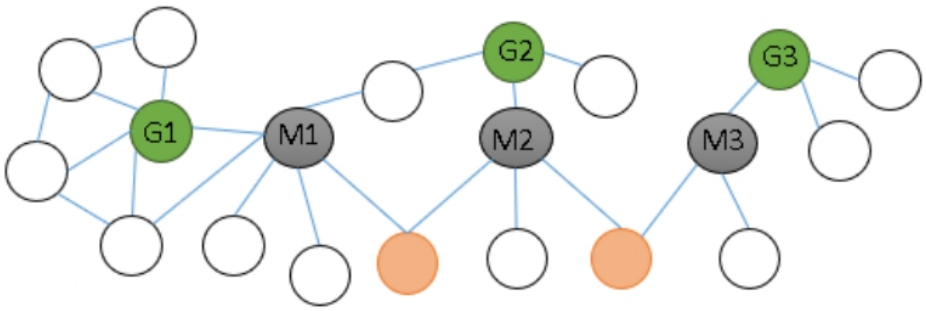

Figure 3.5: *Load Balancing Problem*

The algorithm that allows equiload balancing achieving is as follows.

Algorithm 4 Equiload balancing

Input: Graph(V,E),MIS node M„MaxT
Output: $S = \{S_i\}$
a network decomposition such as each node v_j is assigned to only one components S_i
status(v) = unassigned
while status(v) = unassigned **do**
  v send message to all MIS neighborhood to have theirs components order $|S_i|$.
  all MIS send their components $|S_i|$ to v.
  v send message to the MIS of Minimal order.
  if $|S_i| <= MaxT$ **then**
    while $|S_i| <= MaxT$ **do**
      MIS send acceptance message to v_j.
      $S_i = S_i \cup \{v_J\}$
    end while
    MIS send rejected message to all remaining node .
  else
    v send imergency message to MIS
  end if
  if v receive acceptance message **then**
    status(v) = unassigned
  end if
  if MIS receive imergency message **then**
    MIS send acceptance message to v.
    $S_i = S_i \cup \{v\}$
  end if
end while

Figure 3.5 illustrates the process of assigning nodes to Maximum Independent Set (MIS) nodes to achieve a balanced allocation in a network. The node red represent MIS node ,and green nodes represent the guard node, The algorithm ensures that nodes adjacent to MIS nodes are assigned to them, even if it deviates from predefined thresholds, to prevent overload or underutilization of nodes.

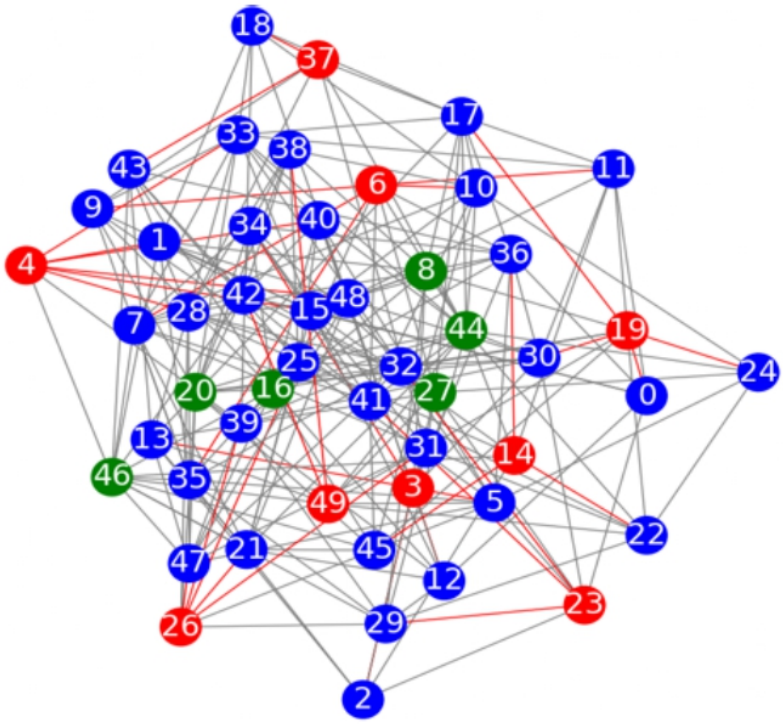

Figure 3.6: *Balanced Node Allocation to Maximum Independent Set (MIS) Nodes*

3.5 Approach simulation

Now, we simulate the proposed approach and evaluate its performance. Beginning by presenting our development environment.

3.5.1 development tools:

In this part, we will specify the tools used to develop our application:

0. **Hardware platform:**The implementation of the application is carried out on a laptop computer having the following characteristics:

- Machine: HP
- Processor: i5-8250U
- Frequency: 1.60GHz 1.80 GHz
- RAM: 8.00 GB
- Graphics card: Intel(R)
- Operating system: Windows 11 Professional

3.5.2 Programming Languages and Development Tools

To develop our simulation, we used Python as the primary programming language due to its simplicity and power for data processing and modeling. The integrated

development environments (IDEs) Spyder and Google Colab were chosen for their complementary features.

- **Spyder :** is a powerful IDE for Python, particularly suited for scientific development. It offers an advanced code editor, debugging tools, and interactive execution capabilities, making it an ideal choice for the initial development phase of our simulation.
- **Google Colab :** is a cloud-based platform that allows running Python code in a Jupyter Notebook environment. Colab is particularly useful for real-time collaboration and access to powerful computing resources without complex hardware setup. It is also convenient for saving and sharing notebooks, facilitating documentation and reproducibility of results.

To keep the results of our simulation, we used Python functions that let us write to text files. By using Python, Spyder, and Google Colab together, we were able to take advantage of each tool's strengths. This gave us a solid and flexible solution for creating our complex simulation, running it, and saving the results.

3.5.3 Testing and Evaluation

We present a detailed evaluation of the system's performance using MIS nodes for distributed computations, along with node removal experiments to test the system's resilience. This evaluation is essential to understand the efficiency of our approach in parallel and distributed computing environments. Figure 3.7 illustrates the initial graph visualizing the MIS nodes, where the nodes belonging to the Maximal Independent Set are colored in red.

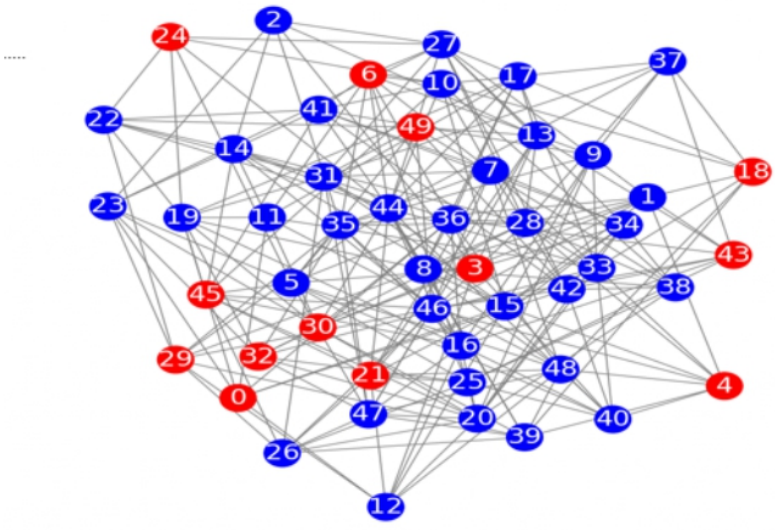

Figure 3.7: *the initial graph*

3.5.4 Matrix Multiplication by MIS Nodes

We begin by describing the process of matrix multiplication by the MIS nodes. Each MIS node computes the product of two square matrices, each with a size equal to the number of nodes allocated to that MIS node. Subsequently, each allocated node (neighbors after rebalancing) computes the product of each row of the first matrix with the second. The results are then transmitted to the corresponding MIS node, which forwards them to the guardian node for storage.Figure 3.8 illustrates this task distribution process The node red represent MIS node ,yellow node represent the neighbours by MIS node selection, and green nodes represent the guard node.

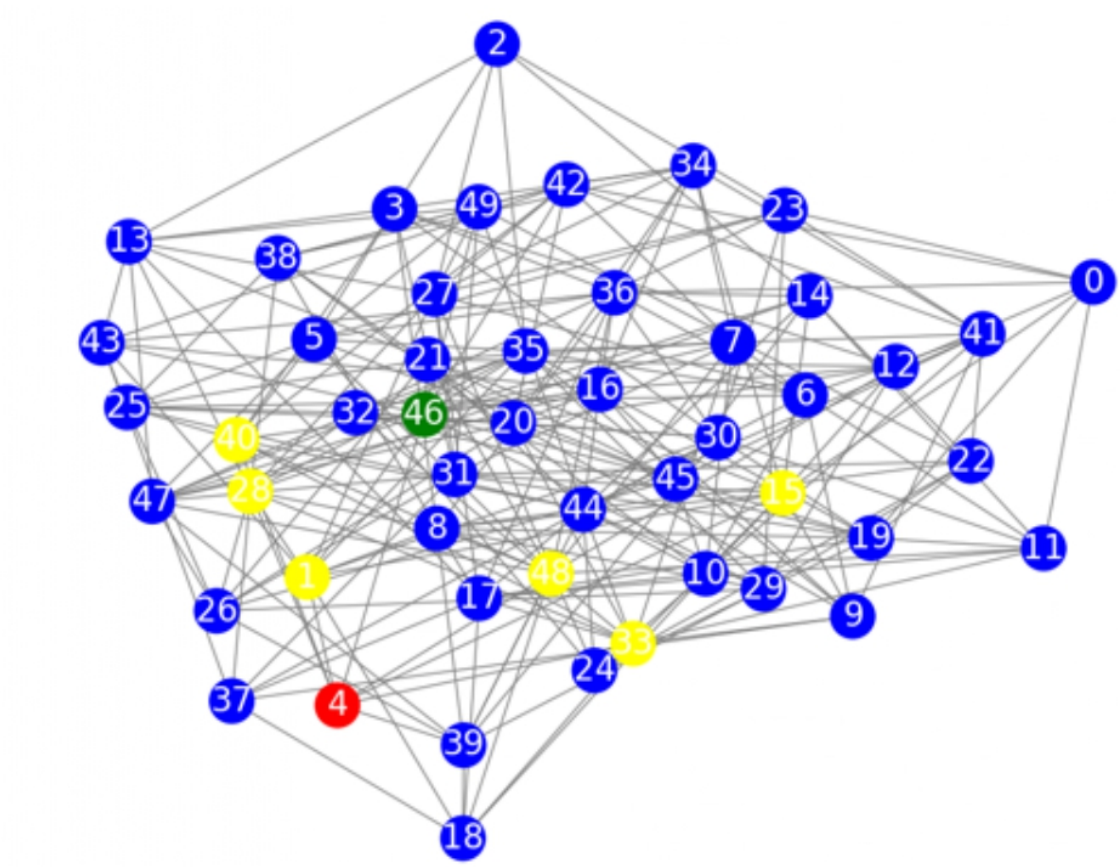

Figure 3.8: *Sending tasks to allocated nodes by an MIS node 4*

3.5.5 Node Removal Experiment

in our simulation, we have removed node 19 and replaced its guard (node 16). Node 19 was a part of the list of MIS nodes, and its removal required a replacement to maintain the system's functionality. Node 24 was selected as the replacement guard for node 44. Figure 3.9 illustrates this process.

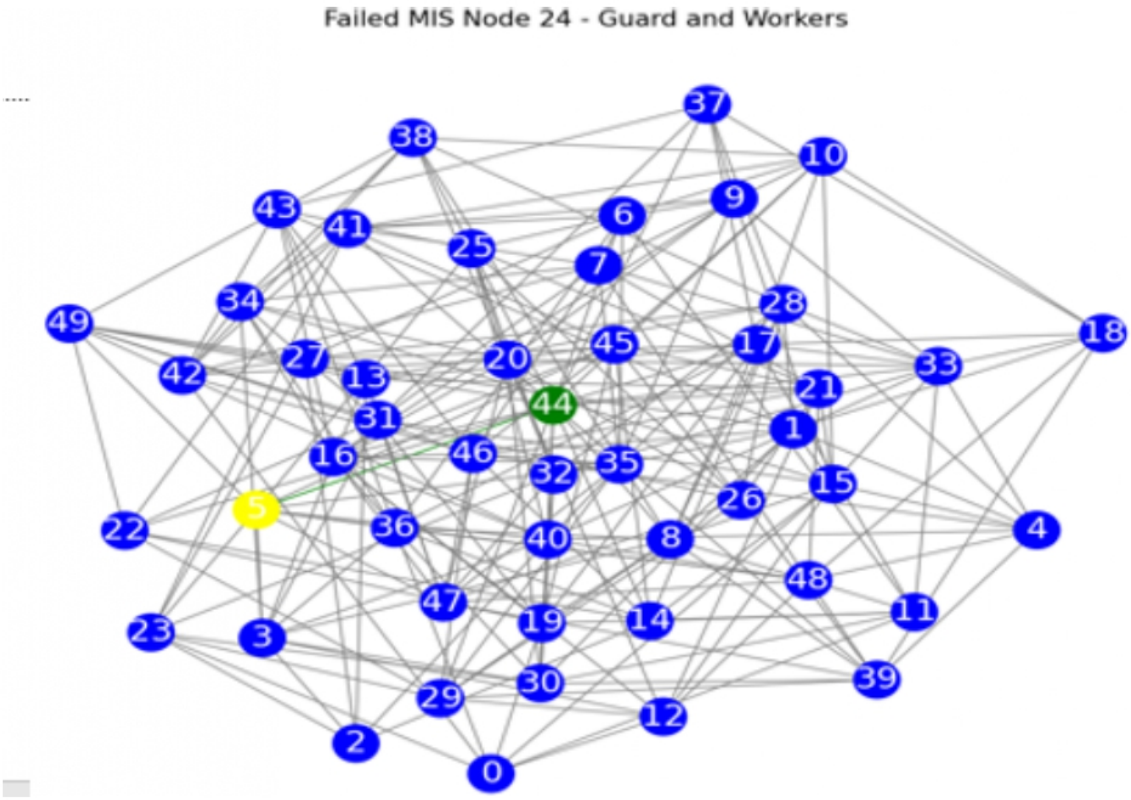

Figure 3.9: Node removal and replacement: Node 24 removed, Node 44 replaced as guard

3.5.6 Update of the New Node List

Following the removal and replacement operations, the updated list of MIS nodes is as follows: MIS Nodes updated: [0, 3, 4, 6, 18, 21, 44, 29, 30, 32, 43, 45, 49]. By updating the MIS node list in this manner, we ensure that the system remains resilient and can continue to operate effectively even in the face of node failures. Figure 3.10 depicts the updated node list. The node red represent MIS nodes new ,and green nodes represent the guard node.

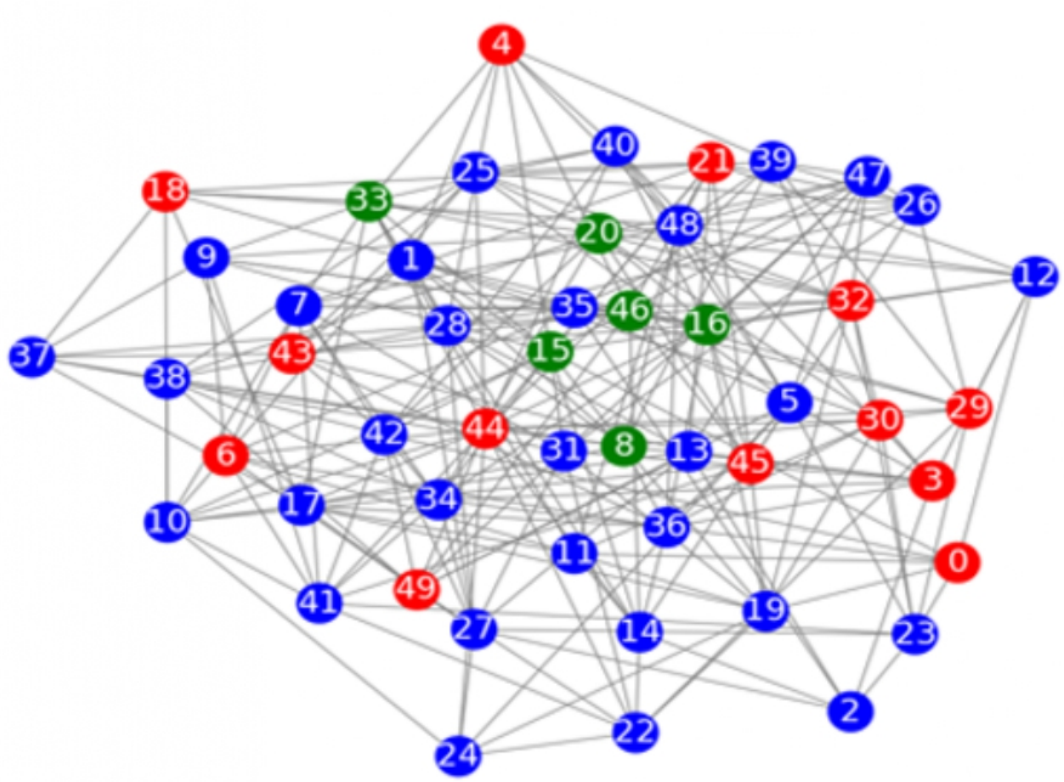

Figure 3.10: Updated MIS node list after removal and replacement

3.5.6.1 Selection of Nodes

Random Selection: Randomly select a fixed percentage of MIS nodes for removal to simulate random failures or drops in a real-world scenario. Replacement Strategy: Node Guard Replacement: Introduce "node guards" as replacements for the removed MIS nodes. Node guards are designed to take over the responsibilities of MIS nodes seamlessly.

Test Cases:

Baseline (0% Removals): No nodes are removed. This serves as a control scenario to measure the system's performance under normal conditions.

Mild Impact (20% Removals): Remove a small percentage of nodes to observe how the system copes with minimal disruptions.

Moderate Impact (30% Removals): Remove a moderate percentage of nodes to challenge the system's resilience and the effectiveness of guardian nodes.

Significant Impact (50-60% Removals): Stress the system by removing a higher

percentage of nodes.
Severe Impact (70% Removals): Test the system's limits by removing a significant percentage of nodes, evaluating its ability to maintain functionality under heavy stress.
Severe Impact (More than 80% Removal): Test the system's limits by removing a significant percentage of nodes, simulating a critical failure scenario.

3.5.7 Testing Results

We tested our approach on a graph of 200 nodes with 22 identified MIS nodes as follows: MIS Nodes: [14, 15, 31, 52, 66, 74, 78, 83, 85, 88, 90, 102, 109, 111, 123, 150, 169, 175, 180, 184, 189, 196]
Assigned Guards: {15: 35, 31: 20, 52: 110, 14: 48, 66: 35, 74: 163, 78: 35, 83: 20, 85: 20, 88: 176, 90: 48, 102: 163, 109: 48, 111: 35, 123: 20, 150: 35, 169: 35, 175: 176, 180: 35, 184: 48, 189: 56, 196: 48}
The recorded times for the matrix multiplication task under different test cases are as follows: The results from both tables show that the execution times (Table 3.3)

Percentage of MIS Removed	**20%**	**30%**	**40%**	**60%**	**70%**	**80%**	**90%**	**100%**
Removed MIS nodes	4	6	8	13	15	17	20	22
Total Execution Time (s)	25.53	25.83	25.38	23.76	23.31	23.02	22.96	21.96

Table 3.3: Execution times for different percentages of guard nodes

Percentage of MIS Removed	**20%**	**30%**	**40%**	**60%**	**70%**	**80%**	**90%**	**100%**
Removed MIS nodes	4	6	8	13	15	17	20	22
Repair Time (s)	34.26	43.853	53.54	80.37	91.05	98.57	100.61	120.63

Table 3.4: Repair times for different percentages of MIS nodes removed

remain relatively stable and even slightly decrease as the percentage of removed MIS nodes increases. In contrast, the repair times (Table 3.4) increase significantly with the percentage of removed MIS nodes. This indicates that while the system maintains performance in terms of execution time due to the replacement of MIS nodes with guard nodes, the repair times still rise with higher removal percentages. This suggests that the introduction of guard nodes effectively stabilizes execution time .

3.5.7.1 Visual Representation

To further illustrate the impact of removing MIS nodes on computation time, we present a plot. Figure 3.11 visually represents the relationship between the percentage of removed MIS nodes and the total execution time, providing a clear understanding of the system's performance under different scenarios.

Figure 3.11: *Excution time vs percentage of removed mis node*

3.6 Implementation

After evaluating the performance of our proposed algorithms through simulations, we explored the practical implementation of our solutions in a real-world scenario. One of the key components in building distributed systems and enabling communication between different nodes or processes is the sockets.

3.6.1 High Availability and Fault Tolerance in Distributed Systems:

3.6.2 Overview:

High availability and fault tolerance are critical aspects of distributed systems that ensure continuous service operation despite server failures. This section presents a practical scenario that demonstrates the implementation of these concepts through a simple server-client architecture involving a main server, a backup server, and clients. The example highlights the system's ability to handle server failures seamlessly and maintain service continuity for connected clients.

3.6.2.1 System Architecture

In the context of network computing, a socket is a software endpoint that establishes bidirectional communication between a server and one or more clients. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Essentially, a socket in an Internet application is a point where messages can be sent or received using the Internet Protocol (IP) suite.
The system consists of the following components:

- **Main Server:** Handles all client requests under normal operation and sends periodic "heartbeat" signals to the backup server to indicate it is active.the

figure 3,12 shows that signal

```
Server started, waiting for connections...
Connection established with ('192.168.56.1', 60749)
Received: bonjour moi client01
Connection established with ('192.168.56.1', 60760)
Received: bonjour moi cline02
```

Figure 3.12: *Main Server*

- **Backup Server:** Monitors the heartbeat from the main server. It remains in standby mode until it detects a failure in the main server, at which point it takes over as the main server.the figure 3,13 shows that If the backup server

```
Backup server started, waiting for heartbeat...
Heartbeat received
Heartbeat received
Heartbeat received
Heartbeat received
Heartbeat received
```

Figure 3.13: *Backup Server Monitoring Heartbeat*

detects a failure in the main server, it takes over as the main server, as shown in the following figure 3.14.

```
Heartbeat missed. Taking over as main server...
Backup server ready to take over...
Successfully started backup server as main server.
```

Figure 3.14: *Backup Server Replacing Main Server*

Once the backup server takes over, it connects with the clients to ensure continuous service, as illustrated in the next figure 3.15.

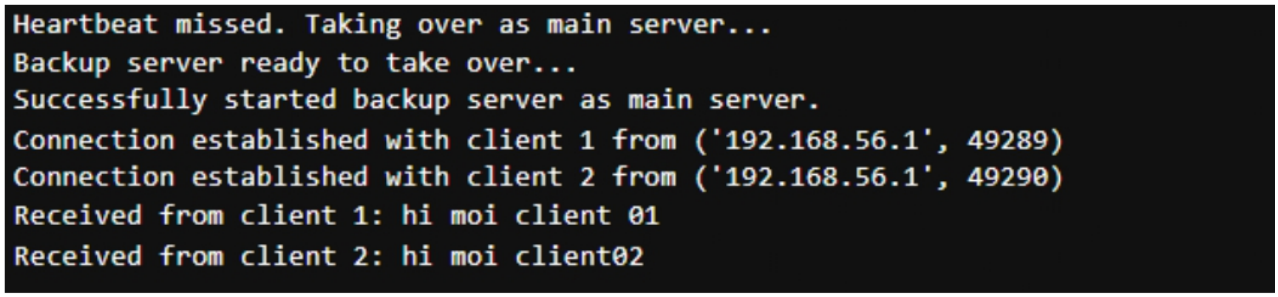

Figure 3.15: *Backup Server Connected with Clients*

- **Clients:** Two clients are configured to connect to the main server to send and receive messages. They are designed to reconnect automatically to the backup server if the main server fails. The following figures 3.16 and 3.17 show the clients.

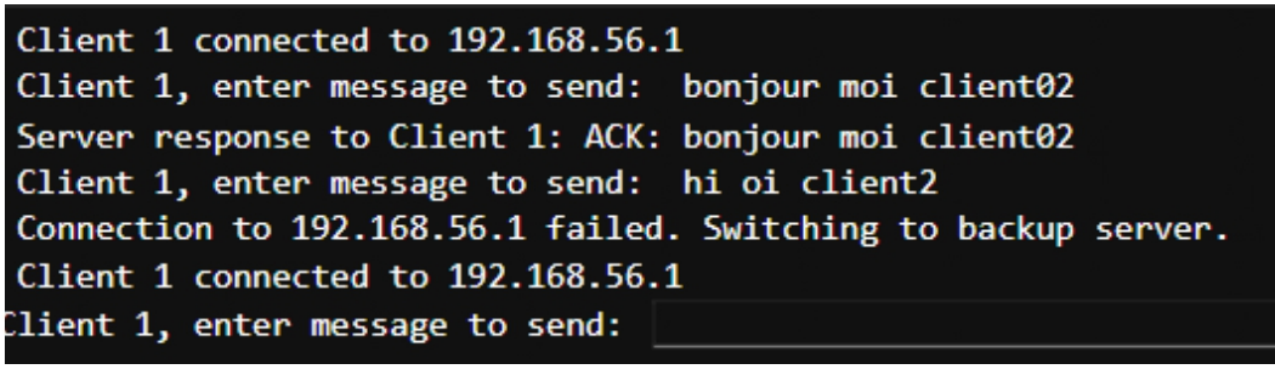

Figure 3.16: *Client 1*

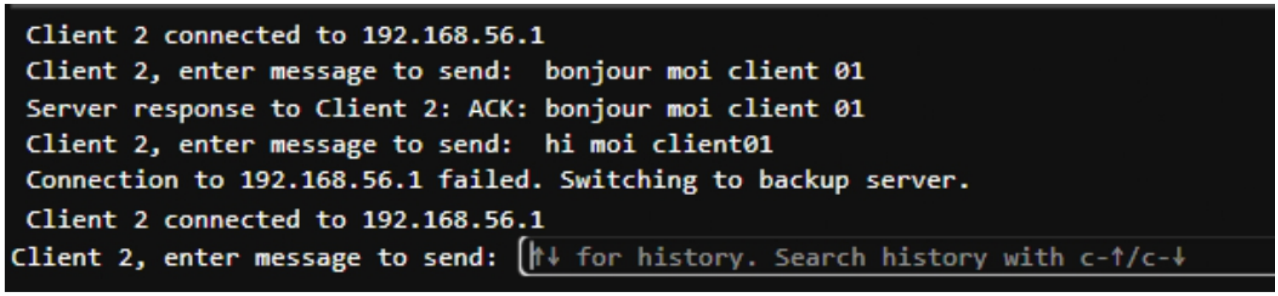

Figure 3.17: *Client 2*

3.6.3 Operational Flow:

Initial Setup and Operation:

- **Main Server Start-Up:** Begins its operation and starts listening for client connections. It also initiates a separate thread to send heartbeat signals to the backup server every five seconds.
- **Backup Server Monitoring:** The backup server listens for the heartbeat on a dedicated port. If it fails to receive any heartbeat within a predefined interval (e.g., 10 seconds), it prepares to take over.

Client Interaction:

- **Client Connections:** Clients 1 and 2 attempt to connect to the main server. Upon successful connection, they send messages which the main server acknowledges.
- **Continuous Service:** The main server processes and responds to client requests, ensuring smooth communication.

Fault Tolerance Mechanism:

- **Detection of Main Server Failure:** If the main server stops sending heartbeats due to a failure, the backup server detects the absence of these signals.
- **Role Switching:** Upon detecting the failure, the backup server transitions to become the main server. It starts accepting client connections on the same port as the original main server.
- **Client Reconnection:** Clients automatically reconnect to the new main server (originally the backup) due to their built-in reconnection mechanism.

3.6.4 Failure Handling and System Resilience:

The transition from the main server to the backup server is seamless, with minimal disruption to client services. This demonstrates the system's resilience and its ability to maintain high availability despite server-side failures.

3.7 Conclusion

This chapter makes the kernel of this dissertation, in which we exposed our proposed approach for failure recovery and tolerance in distributed systems. The approach consists of determining a maximal independent set MIS in a distributed manner, this later is considered as a set of coordinator nodes in which the controllers will be installed, then it determines the set of guards that replace these controllers in case of failures in some of them and finally, it ensure a network decomposition which permits an equiload balancing between controllers. The performances of the approach have been validated in large generated graph system. Furthermore, the approach has been applied in real case of distributed system using socket. Further research works in this topic may lead to apply the approach in field of fault tolerance in Internet Of Things (IOT) systems.

General conclusion

This dissertation is devoted to study the failure tolerance in distributed system, especially whose used for computation. Given that such systems contains a huge number of elements connected between them, then it seems that graph is the most appropriate model to this kind of systems.

after having modeled by a graph, with nodes represent the system entities and arcs represent the communication links, we are looking for underlying problem solving. Notice that graph model allows related problem solving based certain parameters determining, therefore we have to review basic concepts of graph and its parameters including, dominating set, independent set and critical node. unfortunately, almost problems related to these parameters determining are NP-complete ones. Thus, we have to use effective technical such as parallel and distributed computing. Thereby, we introduced a literature review of such parameters determining using distributed algorithms.

It is clear that one of the most well known problem in distributed system is fault tolerance. Thus, the second chapter is devoted for such a problem beginning by basic concept introducing and reviewing the related work of literature for distributed computing systems, sensor networks and IOT systems.

As aforementioned, the third chapter is devoted to the proposed approach about failure recovery and tolerance in large graph system, which consists of a set of distributed algorithms that allows, i) to determine a maximal independent set of node that play the role of controllers. ii)to determine guards that replace these controllers whenever some failures occurs and iii) allows network decomposition that ensure to an equiload balacing between controllers. Finally, the proposed approach is validated based generated large graph system and its correctness is highlighted also throw a real case implementation.

The last chapter was about displaying the results of our application tests, which combined three models U-Net, Attention U-Net, and Attention Residual U-Net, in one place. We have chosen a medical problem related to the segmentation of electron microscopy images. We have augmented our dataset to get more images to work with, and we have built all three models and trained them all with the same hyperparameters to compare them. The results proved that our models were very efficient and adaptable in image segmentation problems, the results were close in terms of performance metrics, but the Attention U-Net model showed a slight superiority in some areas.

The major advantages of the proposed approach are the following:

- **Scalability:** In fact, its complexity of order O(log n), this can simply deal with graph of high order.
- **Extensibility:** Indeed, adding a new node is fluently made as follow, if the node is adjacent to some MIS then it add to the component of law order. Otherwise, it create its own component adding iteratively its adjacent node from highest component.

Further research in the topic may leading to apply the proposed approach in sensor networks and IOT systems which enable fault tolerance approach in these later.

Bibliography

[1] J.-C. Fournier, Graphes et applications, Hermes science publications-Lavoisier, 2007.

[2] Harary, Frank. Graph theory (on Demand Printing of 02787). CRC Press, 2018.

[3] Bondy,J.A., & Murty, U. S. R. (2008). Graph theory wih applications. Elsevier.

[4] Michalski, Adrian, et al. "On proper (1, 2)?dominating sets in graphs." Mathematical Methods in the Applied Sciences 45.11 (2022): 7050-7057.

[5] Yu, En-Yu, et al. "Identifying critical nodes in complex networks via graph convolutional networks." Knowledge-Based Systems 198 (2020): 105893.

[6] Tel, Gerard. Introduction to distributed algorithms. Cambridge university press, 2000.

[7] Cook, Stephen A. "The complexity of theorem-proving procedures." Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook. 2023. 143-152.

[8] Karp, Richard M. Reducibility among combinatorial problems. Springer Berlin Heidelberg, 2010.

[9] Garey, Michael R., and David S. Johnson. "A Guide to the Theory of NP-Completeness." Computers and intractability (1990): 37-79.

[10] Dijkstra, Edsger W. "Self-stabilizing systems in spite of distributed control." Communications of the ACM 17.11 (1974): 643-644.

[11] Turau, Volker. "Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler." Information Processing Letters 103.3 (2007): 88-93.

[12] Hedetniemi, Sandra M., et al. "Self-stabilizing algorithms for minimal dominating sets and maximal independent sets." Computers & Mathematics with Applications 46.5-6 (2003): 805-811.

[13] S. M. Hedetniemi, D. P. Jacobs, and P. K. Srimani. Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. Computers and Mathematics with Applications, 46(5-6) :805-811, 2003.

[14] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, P. K. Srimani, and Z. Xu. Self-stabilizing graph protocols. Parallel Processing Letters, 18(01) :189-199, 2008.

[15] B. Neggazi, N. Guellati, M. Haddad, and H. Kheddouci. Efficient self-stabilizing algorithm for independent strong dominating sets in arbitrary graphs. International Journal of Foundations of Computer Science, 26(6) :515-518, 2015.

[16] J. C. Lin, T. C. Huang, C. P. Wang, and C. Y. Chen. A self-stabilizing algorithm for finding a minimal distance-2 dominating set in distributed systems. Journal of Information Science and Engineering, 24(6) :1709-1718, 2008.

[17] S. K. Shukla, D. J. Rosenkrantz, and S. S. Ravi. Observations on self-stabilizing graph algorithms for anonymous networks. In Proceedings of the second workshop on self-stabilizing systems, volume 7, page 15, January 1995.

[18] T. C. Huang, J. C. Lin, C. Y. Chen, and C. P. Wang. A self-stabilizing algorithm for finding a minimal 2-dominating set assuming the distributed demon model. Computers and Mathematics Application, 54(3) :350-356, August 2007.

[19] W. Y. Chiu, C. Chen, and S. Y. Tsai. A 4n-move self-stabilizing algorithm for the minimal dominating set problem using an unfair distributed daemon. Information Processing Letters, 114(10) :515-518, 2014.

[20] W. Goddard, S. T. Hedetniemi, D. P. Jacobs, and P. K. Srimani. A self-stabilizing distributed algorithm for minimal total domination in an arbitrary system graph. In Parallel and Distributed Processing Symposium, pages 240?243, April 2003.

[21] M. van Steen and A.S. Tanenbaum, Distributed Systems, 4th ed., distributed-systems.net, 2023.

[22] Shen, Yanyan, et al. "Fast failure recovery in distributed graph processing systems." Proceedings of the VLDB Endowment 8.4 (2014): 437-448.

[23] Tong, Yinghua, et al. "Fault tolerance mechanism combining static backup and dynamic timing monitoring for cluster heads." IEEE Access 8 (2020): 43277-43288.

[24] Zhong, Weiyu, et al. "Byzantine Fault-tolerant consensus algorithms: a survey." Electronics 12.18 (2023): 3801.

[25] Xu, Chen, et al. "ACF2: Accelerating Checkpoint-Free Failure Recovery for Distributed Graph Processing." Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data. Cham: Springer Nature Switzerland, 2022

[26] Gonzalez, Joseph E., et al. "GraphX: Graph processing in a distributed dataflow framework." 11th USENIX symposium on operating systems design and implementation (OSDI 14). 2014.

[27] Low, Yucheng, et al. "Distributed graphlab: A framework for machine learning in the cloud." arXiv preprint arXiv:1204.6078 (2012)

[28] Vora, Keval, et al. "Coral: Confined recovery in distributed asynchronous graph processing." ACM SIGARCH Computer Architecture News 45.1 (2017): 223-236

[29] Malewicz, Grzegorz, et al. "Pregel: a system for large-scale graph processing." Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. 2010.

[30] Akyildiz, Ian F., et al. "A survey on sensor networks." IEEE Communications magazine 40.8 (2002): 102-114.

[31] Castelluccia, C., & Francillon, A. (2008)."Protéger les réseaux de capteurs sans fil (SSTIC08) ".19 11

[32] Chamberod, E., & Poupot, C. (1999). "Intelligence locale et processeur de signal dans Capteurs intelligents et micro actionneurs intégrés [Local intelligence and signal processor in Intelligent sensor and integrated microactuators] ". LAAS-CNRS, Cépaduès Editions, 199-200

[33] Gonzalez, Joseph E., et al. "PowerGraph: distributed Graph-Parallel computation on natural graphs." 10th USENIX symposium on operating systems design and implementation (OSDI 12). 2012.

[34] Yan, Da, et al. "Lightweight fault tolerance in pregel-like systems." Proceedings of the 48th International Conference on Parallel Processing. 2019.

[35] Belkadi, Khadidja, Mohamed Lehsaini, and Mohammed Amin Tahraoui. "Fault?tolerance based on augmenting approach in wireless sensor networks." Concurrency and Computation: Practice and Experience 34.28 (2022): e7359.

[36] Ghaffari, A., and S. Nobahary. "FDMG: Fault detection method by using genetic algorithm in clustered wireless sensor networks." Journal of AI and Data Mining 3.1 (2015): 47-57.

[37] Fan, Fang, et al. "An optimized machine learning technology scheme and its application in fault detection in wireless sensor networks." Journal of Applied Statistics 50.3 (2023): 592-609.

[38] Rajasegarar, Sutharshan, et al. "Distributed anomaly detection in wireless sensor networks." 2006 10th IEEE Singapore international conference on communication systems. IEEE, 2006.

[39] Cheng, Yong, et al. "Distributed fault detection for wireless sensor networks based on support vector regression." Wireless Communications and Mobile Computing 2018 (2018).

[40] Munir, Arslan, Joseph Antoon, and Ann Gordon-Ross. "Modeling and analysis of fault detection and fault tolerance in wireless sensor networks." ACM Transactions on Embedded Computing Systems (TECS) 14.1 (2015): 1-43.

[41] Schelter, Sebastian, et al. "" All roads lead to Rome" optimistic recovery for distributed iterative data processing." Proceedings of the 22nd ACM international conference on Information & Knowledge Management. 2013.

[42] Pundir, Mayank, et al. "Zorro: Zero-cost reactive failure recovery in distributed graph processing." Proceedings of the Sixth ACM Symposium on Cloud Computing. 2015.

[43] Dathathri, Roshan, et al. "Phoenix: A substrate for resilient distributed graph analytics." Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019.

[44] Sari, Arif, and Murat Akkaya. "Fault tolerance mechanisms in distributed systems." International Journal of Communications, Network and System Sciences 8.12 (2015): 471-482.

[45] Ajay, N.Tarasia, S. Dash, S.Ray, ARSwain, ?Une erreur dynamique tolérant protocole de routage pour prolonger la durée de vie des réseaux de capteurs sans fil?, Journal (IJCSIT), Vol. 2 (2), pp727-734, 2011

[46] Luby, Michael. "A simple parallel algorithm for the maximal independent set problem." Proceedings of the seventeenth annual ACM symposium on Theory of computing. 1985.

[47] Li, Liwu. "Ranking and unranking of AVL-trees." SIAM Journal on Computing 15.4 (1986): 1025-1035.

[48] Renard, Hélene. Équilibrage de charge et redistribution de données sur plates-formes hétérogenes. Diss. Ecole normale supérieure de lyon-ENS LYON, 2005

[49] Yagoubi, Bellabas. "Modèle d'équilibrage de charge pour les grilles de calcul." Revue Africaine de Recherche en Informatique et Mathématiques Appliquées 7 (2007).

[50] Cybenko, George. "Dynamic load balancing for distributed memory multiprocessors." Journal of parallel and distributed computing 7.2 (1989): 279-301.

[51] Renard, Hélene. Équilibrage de charge et redistribution de données sur plates-formes hétérogenes. Diss. Ecole normale supérieure de lyon-ENS LYON, 2005.

[52] A. Arulselvan, C.W. Commander, L. Elefteriadou, P.M. Pardalos, Detecting critical nodes in sparse graphs, Comput. Oper. Res. 36 (7) (2009) 2193-2200.

[53] A. Arulselvan, C.W. Commander, O. Shylo, P.M. Pardalos, Cardinality-constrained critical node detection problem, in: Performance Models and Risk Management in Communications Systems, Springer, 2011, pp. 79-91.

[54] Lalou, M., Tahraoui, M. A., & Kheddouci, H. (2016). Component-cardinality-constrained critical node problem in graphs. Discrete applied mathematics, 210, 150-163.

[55] Lalou, M., Tahraoui, M. A., & Kheddouci, H. (2018). The critical node detection problem in networks: A survey. Computer Science Review, 28, 92-117.

[56] Han, M., Daudjee, K., Ammar, K., Özsu, M. T., Wang, X., & Jin, T. (2014). An experimental comparison of pregel-like graph processing systems. Proceedings of the VLDB Endowment, 7(12), 1047-1058.

[57] ANDREW S, T. A. N. E. N. B. A. U. M., and W. E. T. H. E. R. A. L. L. DAVID J. "COMPUTER NETWORKS FIFTH EDITION." (2011).