



**ADAPTIVE FAULT DIAGNOSIS AND RESOLUTION  
SYSTEM FOR ENTERPRISE DATA REPLICATION  
SYSTEM USING DEEP REINFORCEMENT LEARNING**

Submitted by

Chee Keong Wee

BSc (Hons) in Computing & IS

Master of Business in IT

Oracle Certified Professional Database Administration

Graduate Cert in Australian Migration Law

For the award of

Doctor of Philosophy

2021

## ABSTRACT

Modern business IT systems in large organisations have high levels of collaboration and interoperability to support various business functions. In heterogeneous IT systems, data is one of the most important entities that are constantly exchanged. The method of data exchange or transfer among these collaborating IT systems can occur in near real-time or in batches, and they are arranged in either hierarchical or mesh structure relationships. There are several ways of conducting these data transfers and one of the methods is to use data replicating software. Maintaining both the business IT system and the data replication services is always a challenge to the IT administrators, and with mission-critical systems that demand 24x7 uptime, the data replicating services are expected to have a high level of operational standards and services to the organisation with minimum downtime.

The job of the IT administrator is to maintain and support all the IT systems and infrastructure to meet the expected service level agreement (SLA). This includes monitoring the IT systems and data replications for anomalies or defects and rectifying them as soon as possible to minimize downtime. However, humans need rest, are prone to fatigue, and are unable to scale their operational work effectively. Therefore, an alternative is needed to overcome these limits.

It is the goal of this thesis to meet this challenge by developing a novel autonomous and adaptive system in monitoring and proactively rectifying any technical problems encountered in the data replicating environment. This novel approach utilizes the research in the domain of deep learning and reinforcement learning that can take appropriate actions to rectify faults encountered in the data replication environment to maximize the concept of cumulative rewards. The proposed system will go through a series of learning cycles starting by learning through trial-and-error by interacting with the data replicating environment, then gradually move to learn to predict the course of faults' resolution actions and their associated scores of successes. It will refine and build up its knowledgebase incrementally and for any faults that it cannot resolve, it will need an IT administrator to help it out, which enrich its knowledgebase at the same time. The approach is novel as there has been no precedence in the use of Reinforcement learning in the domain of software's fault

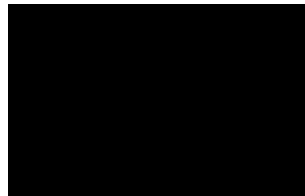
diagnosis and resolution for Near real-time Data replication before. The result will be an autonomous fault diagnostics and rectifying system that can function at near human's IT level troubleshooting skills to support the data replicating environment. It is evaluated based on the results of the cost functions from the fault diagnosis and resolution of intelligent agents, against the guiding software routines that perform similar activities.

The contribution that this thesis makes can be classified into two main groups: adaptively intelligent fault diagnosis and resolutions. The first group is to develop an adaptive self-learning approach that can learn to diagnose the service outage across the multitudes of software services which cannot be ascertained by manual IT system administration. This feature has significant benefits as it defies the traditional rule-based diagnostic procedures which are limited to the set of pre-assigned rules that they are strictly designed for. It has the flexibility to scale and augment its coverage adaptively. For the second group, the self-learning approach is used to resolve specific software faults adaptively discovered in the diagnosis phase. This gives an edge over the rule-based procedures of fault resolution which depend on predefined rules and conditions to act, and they have the limitation of scalability and adaptiveness. Given the complexity of a large enterprise data replication setup with tens of thousands of software's configuration and parameters, including a high volume of statistics and logs outputs, this thesis can contribute a significant value to the IT support and management community to automate their operations intelligently.

## CERTIFICATION OF THESIS

I, Chee Keong Wee, declare that the PhD Thesis entitled, Adaptive Fault Diagnosis and Resolution System for Enterprise Data Replication System Using Deep Reinforcement Learning, is not more than 100,000 words in length including quotes and exclusive of tables, figures, appendices, bibliography, references, and footnotes. The thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated, this thesis is my own work.

Signed:



Date: 25-11-2021

Endorsed by:

Principal Supervisor: Associate Professor Xujuan Zhou

Associate Supervisor: Professor Raj Gururajan

Associate Supervisor: Associate Professor Xiaohui Tao

Student and supervisors' signatures of endorsement are held at the University

## LIST OF PUBLICATIONS

During the research, several research papers were published. I was the primary author of the following co-authored papers:

1. Wee, C.K., Zhou, X., Gururajan, R., Tao, X. and Wee, N., 2022, February. Adaptive Fault Resolution for Database Replication Systems. In International Conference on Advanced Data Mining and Applications (pp. 368-381). Springer, Cham.
2. Wee, C.K. and Wee, N., 2021, June. Outlier Detection for GP Referrals in Otorhinolaryngology. In International Conference on Artificial Intelligence in Medicine (pp. 444-451). Springer, Cham.
3. Wee C.K., Wee N. (2021) Adaptive Fault Diagnosis for Data Replication Systems. In: Qiao M., Vossen G., Wang S., Li L. (eds) Databases Theory and Applications. ADC 2021. Lecture Notes in Computer Science, vol 12610. Springer, Cham.
4. Wee, C.K. and Nayak, R., 2020, December. Adaptive Data Replication Optimization Based on Reinforcement Learning. In 2020 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1210-1217). IEEE.
5. Wee, C.K. and Nayak, R., 2019, December. Data replication optimization using simulated annealing. In Australasian Conference on Data Mining (pp. 222-234). Springer, Singapore.
6. Wee, C.K. and Nayak, R., 2019, August. Adaptive database's performance tuning based on reinforcement learning. In Pacific Rim Knowledge Acquisition Workshop (pp. 97-114). Springer, Cham.
7. Wee, C.K. and Nayak, R., 2019. Adaptive load forecasting using reinforcement learning with database technology. *Journal of Information and Telecommunication*, 3(3), pp.381-399.
8. Wee, C.K. and Nayak, R., 2019. A novel machine learning approach for database exploitation detection and privilege control. *Journal of Information and Telecommunication*, 3(3), pp.308-325.

9. Wee, C.K. and Nayak, R., 2018, November. An approach to compress and represents time series data and its application in electric power utilities. In Australasian Conference on Data Mining (pp. 107-120). Springer, Singapore.
10. Wee, C.K. and Nayak, R., 2018, February. An alternate approach to Time Series reduction. In 2018 International Conference on Soft-computing and Network Security (ICSNS) (pp. 1-4). IEEE.
11. Wee, C.K. and Nayak, R., 2018. A novel load forecasting system leveraging database technology. In Modern Approaches for Intelligent Information and Database Systems (pp. 491-503). Springer, Cham.
12. Wee, C.K. and Nayak, R., 2018. A novel database exploitation detection and privilege control system using data mining. In Modern Approaches for Intelligent Information and Database Systems (pp. 505-516). Springer, Cham.

**Under review:**

13. Wee, C.K, Zhou, X., Gururajan, G., Tao, X. and Wee, N. (2021) Automated Triaging Medical Referral for Otorhinolaryngology using Data Mining and Machine Learning Techniques. *IEEE Access (Q1)*. *Under second round review*.

## ACKNOWLEDGEMENTS

This dissertation is the result of my six years of part-time study with the University of Southern Queensland and the Queensland University of Technology. I would like to express my appreciation to my supervisory team, Dr XuJuan Zhou, Dr Xiaohui Tao, Dr Raj Gururajan at USQ and Dr Richi Nayak at QUT, for their invaluable contribution and advice throughout my higher degree study journey of six years duration that spanned across QUT and USQ.

I want to thank Vince Currie, my database team leader, who encouraged me to take on this academic journey. His leadership, enthusiasm and vision have shaped my professional outlook and motivated me to upgrade my professional and academic pursuits.

I want to thank especially my family, Clara and Nathan, who has been a great pillar of support in my life. They gave me the purpose and the perseverance to endure 15 years of part-time study while working full-time across multiple jobs with factories and small companies. A special thanks to Comsertrac and Informatic Computer school, for giving us poor working adults in Singapore a chance to upgrade ourselves and earn tertiary qualifications. The Singapore Government had set a cap on the number of local students eligible to study at the local 23% ([https://wikileaks.org/plusd/cables/07SINGAPORE394\\_a.html](https://wikileaks.org/plusd/cables/07SINGAPORE394_a.html)). Without these private education institutes, I wouldn't be able to have the necessary qualifications to upgrade my profession, let alone qualified for my emigration to Australia, and PhD study.

A special thanks to the Australian government, for believing in me and supporting my PhD study through the provision of research training grants. To me, education is a privilege and that is something I learnt to cherish dearly from an early age. My sincere gratitude to my parents, Wee Chin Tiong and Tay Keng Moi, both had gone through arduous hardship and difficulty to raise my siblings and me up.

# TABLE OF CONTENTS

ABSTRACT.....	2
CERTIFICATION OF THESIS.....	4
LIST OF PUBLICATIONS .....	5
ACKNOWLEDGEMENTS.....	7
TABLE OF FIGURES.....	11
TABLES OF TABLES .....	13
CHAPTER 1: INTRODUCTION.....	1
1.1. Motivation.....	1
1.2. Problem Statement .....	3
1.3. Research Question .....	4
1.4. Research Objectives.....	6
1.5. Research Contribution.....	7
1.6. Research Significance.....	8
1.7. Outline of The Thesis.....	9
CHAPTER 2: LITERATURE REVIEW.....	11
2.1. Fault Detection and Diagnosis .....	11
2.2. Anomaly Detection (AD).....	14
2.2.1. Anomaly detection techniques .....	15
2.2.2. Clustering-based techniques .....	18
2.2.3. Classification based techniques .....	19
2.2.4. Other techniques .....	20
2.3. Organization IT Infrastructure .....	20
2.3.1. Data Replication System - Shareplex.....	20
2.3.2. Relational database management system (RDBMS) - Oracle .....	23
2.3.3. Energy Queensland’s Data Replication System.....	25
2.3.4. System Anomalies detection in DRE.....	30
2.4. Reinforcement learning.....	31
2.4.1. Markov Decision Process (MDP) – model-based.....	33
2.4.2. Q-learning – Model-free .....	36
2.4.3. Actor-Critic Reinforcement Learning.....	36
2.4.4. Artificial Neural Network .....	37
2.4.5. Deep Reinforcement Learning .....	40
2.5. Research Gap and Summary .....	41
CHAPTER 3: RESEARCH METHODOLOGY & DESIGN.....	44
3.1. Overview.....	44



3.2.	Data Input and Analysis Phase.....	45
3.2.1.	Environment Dataset.....	46
3.2.2.	Evaluation Procedure .....	47
3.3.	Classification And Learning Phase .....	50
3.4.	Fault Diagnosis Phase .....	51
3.4.1.	Defining Data Source for System Anomalies .....	51
3.5.	Faults Resolution Phase .....	53
3.5.1.	Development of Fault Diagnosis and Resolution Process .....	54
3.6.	Integration/Process Learning Phase .....	57
3.7.	Develop an Intelligent Fault Resolution System.....	58
3.8.	FDR's RL Agent Learning Process .....	61
3.9.	Approximation Between State and Actions .....	64
3.10.	Scoring the Environment's State.....	65
3.11.	Action for the Environment .....	67
	CHAPTER 4: DESIGNING THE FAULT DIAGNOSTIC (FD) MODULE.....	69
4.1.	The Current Approach Toward DRE's Fault Diagnosis .....	69
4.2.	Problem Formulation .....	72
4.3.	Adaptive Fault Diagnosis (FD) Module Design .....	73
4.3.1.	Information Acquisition (IA) module .....	75
4.3.2.	Diagnostic Reinforcement Learning (DRL) for FD Module .....	75
4.4.	System Diagnostic (SD) Module .....	77
4.5.	Data Replication Environment (DRE)'s State Representation .....	78
4.6.	DRE's Action of Diagnostic Prediction.....	81
4.7.	Approximation Between DRE's Symptoms-States and Diagnosis-Actions .....	81
4.8.	FD's Algorithm.....	82
4.9.	Empirical Analysis.....	85
4.9.1.	Software used for FDR tests .....	86
4.9.2.	The experimental set-up.....	88
4.9.3.	True Negative test results.....	91
4.10.	Evaluation Criteria and Benchmarking.....	92
4.10.1.	Test results .....	93
4.10.2.	Service outage Classification results.....	95
4.10.3.	Service Outage prediction accuracy .....	95
4.10.4.	FD's DRL-NN performance results .....	96
4.11.	FD's Conclusion .....	99
	CHAPTER 5: DESIGNING THE FAULT RESOLUTION (FR) MODULE .....	101

5.1.	Adaptive Fault Resolution (FR) Module Design .....	101
5.1.1.	Diagnostic Reinforcement Learning (DRL) for FR Module.....	103
5.1.2.	System Correction (SC) module .....	105
5.1.3.	Representation and correlation of diagnosed faults to corrective actions ....	107
5.1.4.	Prioritization of the software groups' action.....	109
5.1.5.	Cost function and Q-Values for FR module .....	111
5.2.	FR's Algorithm .....	113
5.3.	Empirical Analysis .....	119
5.3.1.	Software setup.....	120
5.3.2.	Experiment setup and goal.....	120
5.4.	Evaluation Criteria .....	126
5.4.1.	Test results .....	127
5.4.2.	FR module - SC's results .....	127
5.4.3.	FR module – NN performance and result .....	130
5.4.4.	FR's efficacy test results .....	132
5.5.	FR's Conclusion.....	136
CHAPTER 6: FDR – FD AND FR INTEGRATION.....		137
6.1.	Background of integration testing.....	137
6.2.	FDR Test Analysis.....	138
6.2.1.	Usage of Software .....	138
6.2.2.	Experiment setup and goal.....	139
6.2.3.	Test analysis procedures .....	139
6.3.	Test results and analysis.....	142
6.3.1.	FDR modules performance results.....	142
6.3.2.	FDR integration test outputs and findings .....	142
CHAPTER 7: CONCLUSIONS .....		146
7.1.	Research Contribution.....	146
7.2.	Comparison of FDR to other diagnostic and resolution methods .....	148
7.3.	Future works and Enhancement.....	151
7.4.	Conclusion .....	153
References .....		154

## TABLE OF FIGURES

Figure 1 - Classifications of anomaly detection techniques.....	15
Figure 2 - Shareplex data replication flow [28] .....	22
Figure 3 – Shareplex’s SP_CTRL status and EVENT_LOG output with error messages.....	23
Figure 4 - Oracle RDBMS architecture [33].....	24
Figure 5 - Energy Queensland’s Shareplex and oracle integration overview .....	27
Figure 6 - Current architecture of EQ’s enterprise data replication.....	29
Figure 7 - Reinforcement learning's Agent processes.....	32
Figure 8- Actor-Critic Reinforcement Learning .....	37
Figure 9- Diagram of a neural network with 5 inputs, 1 hidden layer and one output .....	38
Figure 10 – Neuron’s transfer and activating function diagram. ....	39
Figure 11- Deep Reinforcement Learning .....	41
Figure 12 - Research plan .....	44
Figure 13 – Faults Diagnosis and Resolution (FDR) workflow .....	54
Figure 14 – Direct and complex fault goals’ resolution workflow .....	56
Figure 15 – Hierarchical relationship of DRE’s software systems and their components.....	57
Figure 16 – DRE’s Fault Diagnosis and Resolution (FDR) System model.....	59
Figure 17 - The detailed process of the FDR model.....	60
Figure 18 - Different learning stages of RL agent .....	61
Figure 19 - NN function approximation between states versus predicted reward and actions .....	64
Figure 20 – RL agent’s action state-action flow .....	65
Figure 21 – Relationship between Services, metrics and DRE’s sub-systems .....	73
Figure 22 – Adaptive Faults Diagnosis overview.....	74
Figure 23 – Faults diagnosis agent’s architecture and workflow.....	75
Figure 24 - Different phases of reinforcement learning in the RL agent. ....	76
Figure 25 – MASE score of True and positive predicted results .....	96
Figure 26 - Accuracy results of the SD's DRL-NN with different epoch/batch size .....	98

Figure 27 - Time spent between FD-SD and FD-NN for fault injection, reversal and diagnosis on DRE .....	99
Figure 28– Adaptive Faults Resolution overview.....	102
Figure 29 – Faults Resolution agent’s architecture and workflow .....	103
Figure 30 – DRL different phases of learning for the FR module .....	104
Figure 31 – FR-SC progress results against various faults .....	128
Figure 32 – NN’s performance with varying epoch and batch size against different activation functions and optimizers .....	132
Figure 33 – Corrective action cycles results between SC and NN.....	135
Figure 34 – Comparison of time taken to find corrective actions between SC and DRL-NN .....	135
Figure 35 – Time taken by FD to diagnose, FR to resolve faults using SC and NN unit ....	142

## TABLES OF TABLES

Table 1 - Shareplex configuration table on EPM data warehouse .....	27
Table 2 - Typical Shareplex configuration file sample .....	27
Table 3 – DRE’s software stats and logs queries.....	66
Table 4 – list of system commands for DRE’s software and functions .....	67
Table 5 – Fault Diagnosis Method (low 1 to high 10) .....	72
Table 6 - Memory and logs checks .....	80
Table 7- Faults induction and restoration on DRE software’s component services (service status flag: 0 – good, 1 – faults).....	87
Table 8 - Detailed Test for Fault Diagnostic module, with break-fix routines .....	88
Table 9 - Results of service outage prediction & scores against DRE’s state .....	93
Table 10 – Outputs from SD’s simulated tests .....	94
Table 11- Confusion matrix of the classification of the service outage’s prediction.....	95
Table 12 – Example of diagnosed faults correlation to corrective actions .....	108
Table 13 – Association of software corrective actions to diagnosed faults for specific software elements.....	108
Table 14 – Sequence of actions’ consideration for series of faults.....	112
Table 15 – List of DRE’s software groups service outage fault and their corrective actions .....	120
Table 16 – SC’s results in response to injected faults.....	128
Table 17 – SC’s results in response to injected faults.....	133
Table 18 – FR module’s efficacy test results .....	134
Table 19 - Test scenario and increase the number of simultaneous occurring faults for each case.....	140
Table 20 - Results from FR integrated testing .....	143
Table 21 - Benchmarking FDR against other methods of fault diagnosis/resolution .....	149

## TABLE OF ABBREVIATIONS

<b>Abbreviation</b>	<b>Description</b>
3NF	Third Normal Form
ACID	Atomicity, Consistency, Isolation, and Durability
ACL	Access Control List
ADAM	Adaptive Moment Estimation
ADDM	Automatic Database Diagnostic Monitor
ADR	Automatic Diagnostic Repository
ASM	Automatic Storage Management
AWR	Automatic Workload Repository
DBA	Database Administrator
DBW	Database Writer Process
DDL	Data Description Language
DES	Data Encryption Standard
DLL	Dynamic-Link Library
DML	Data Manipulative Language
DNS	Domain Name System
DNN	Deep Neural Network
DR	Disaster Recovery
DRL	Deep Reinforcement Learning
DRE	Data Replication Environment
DSS	Decision Support System
EDW	Enterprise Data Warehouse
ELU	Exponential Linear Unit
ETL	Extract, Transform, Load
FD	Fault Diagnostics
FTP	File Transfer Protocol
FR	Fault Resolution
FDR	Fault Diagnostics and Resolution
GIS	Geographic Information System
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment

IT	Information Technology
IP	Internet Protocol
KB	Knowledge Base
LDAP	Lightweight Directory Access Protocol
LGWR	Log Writer Process
LOB	Large Binary Object
MASE	Mean Average Squared Error
ML	Machine Learning
NFS	Network File System
NI	Network Interface
NIC	Network Interface Card
NN	Neural Network
NRT	Near Real-Time
NW	Neural Network
ODBC	Open Database Connectivity
OLTP	Online Transaction Processing
OracleDB	Oracle Database
OS	Operating System
PGA	Program Global Area
RAID	Redundant Array of Inexpensive Disks
RAM	Random Access Memory
RDBMS	Relational Database Management System
RMAN	Recovery Manager
RELU	Rectified Linear Unit
SAN	Storage Area Network
SCN	System Change Number
SD	System Diagnostics
SC	System Correction
SO	Service Outage
SGA	System Global Area
SGD	Stochastic Gradient Descent
SID	Oracle System Identifier
SLA	Service Level Agreement

SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SPLX	Shareplex
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
TCPIP	Transmission Control Protocol/Internet Protocol
TNS	Transparent Network Substrate
XML	Extensible Mark-up Language
VM	Virtual Machine



# CHAPTER 1: INTRODUCTION

---

This chapter gives an overview of the research and its various aspects such as background, questions, and goals, including its significance and limitation. The structure of the thesis is described at the end of this chapter.

## 1.1. Motivation

In the modern world of Information Technology (IT), there are numerous heterogeneous types of computer systems that work cohesively to deliver a multitude of services to organizations and individuals[1]. One of the main driving forces behind these IT services is the acquisition, manipulation, dissemination, and consumption of data. Contemporary systems and machines, together with the proliferation of modern electronic devices, generate several types of data at exceptionally high volumes and rates. As IT systems do not function in isolation, data transfer or information propagation is vital for sharing data across multiple platforms. It is also imperative that the data is sent and received with minimum delay, and that the mode of transportation must guarantee data accuracy, reliability, privacy, and security[1, 2].

A common data transfer system to transfer data from one system's database backend to another is the data replicating tool [2]. This is popular among large organizations for transferring their data between IT systems. Some data transfers may be direct, while others require additional manipulation and processing before, they can be accepted. A data replicating requirement is to have the data at Near Real-Time (NRT) where a certain record, which has been changed on one system, can be seen almost instantaneously on the receiving ends at other target IT systems. This NRT data replicating capability is dependent on several factors such as the features of a replicating tool, the fast-processing capability of the databases, and the amount of network bandwidth available. Everything must work together harmoniously to support the data replication process [3].

The number of resources required to keep the IT systems and infrastructure in full operation is usually high. It incurs high commitment in terms of human resources, labour, time, expertise, and investment. Apart from the operational and administrative

support, another labour-intensive and complex task that IT administrators face is detecting and fixing problems that arise in the data replication environment. Therefore, a major part of their work revolves around monitoring, detecting faults, diagnosing the faults, and resolving them to keep the systems operational. However, there is a limit to the period of their IT administration's working hours as well as their operational scale and size. Another further consideration is the risk of human error which is common especially in an environment where work fatigue sets in[1].

It will be desirable to have systems or support that can complement human labour in the domain of fault detection and resolution to maintain the continued operation of data replication services. Such a system should be able to monitor the suite of heterogeneous software participating in the data replicating service for faults, identifies and fixes them before they cause further delays and service outages. Additionally, the (semi-)autonomous system should be able to work with IT administrators, absorbing more information and bolster its knowledgebase much like a human trying to learn a trade until it reaches a level of competency that it can function independently.

Commercial software vendors and academic researchers have attempted to develop new ways to automate the administration works using statistical and heuristic algorithms, creating an expert system to reduce the load on human administrators. These implementations are specific to individual software that remains proprietary to the software vendors. Heuristics, rule-based or decision trees are the most common algorithms that are used in these systems. However, these supporting software tools are both restrictive and well-defined, thus they are unable to adapt to a fast-changing environment. As such, their capacity to grow and cater to new changes is heavily restricted.

Current research in artificial intelligence application toward fault diagnosis and resolution is geared towards root cause fault analysis as applying fixes to software poses a significant risk to the system's stability and carries liability and litigation risk. The Data Replicating Environment (DRE) is an enterprise software setup that is common among large IT setup, and it comprises of specialised software that read data changes in databases before forwarding them to other designated databases at (NRT) speeds. It interacts with a series of other software and network environments such as

the Operating system and network protocol. Despite its critical importance to many organisations, most developments in DRE have been kept exclusive to software vendors, and scientific literature on the topic is scarce. At the time of writing, the use of artificial intelligence to manage DREs has not been reported in industry or the literature.

## **1.2. Problem Statement**

This thesis attempts to address the challenges as listed in the previous section of diagnosing the various DRE software's faults and correct them intelligently using Reinforcement learning by proposing a novel autonomous, self-learning Fault Diagnosis and Resolution (FDR) approach. The approach is based on Reinforcement Learning which has gained significant popularity in recent times where researchers have applied it to solve complex problems with high permutations of choices. This novel reinforcement learning-based approach is designed to 1) monitor the events in the data replicating environment and detect for any anomalies, 2) analyse if the anomalies are temporary or a genuine defect, 3) do root cause investigation and fault diagnosis, 4) apply the appropriate actions to resolve the faults and 5) retain the knowledge plus re-action for future faults.

Identifying the root cause of the software's potential faults and defects is a complex task that requires a series of investigative actions and intrusive probing before the real fault can be identified. While a set of prior knowledge can help to prepare the baseline for the defects-to-faults identification, there is still a multitude of other potential causes that may not be able to surface until they can be investigated more thoroughly. A *fault diagnosis* system should complement the humans where troubleshooting to find the faults can be a team effort. Once the faults have been identified, the next logical step is to resolve these errors in the least interruptive manner in an IT system. A *fault resolution* system should have the basis of intelligence to learn by interacting with the data replicating environment to rectify the faults with actions. It must be intelligent enough to know what action works and do not work and rate those actions accordingly based on their level of usefulness. It must be able to initiate the learning phase by assessing its level of competency first; if it does not have much knowledge of fixing the faults, then it should learn by trial-and-error to build up its

core competency. And when it reaches a level of skill, it should be able to predict the action and potential score from its knowledge base for a fault.

*A fault diagnosis and resolution (FDR) system is expected to identify the faults as well as not just resolve the faults correctly initially, but to learn adaptively from past experiences, to derive the optimum series of actions to maximize the rewards, reaching eventually the goal of an intelligent agent at an expert level that is competent enough to manage the DRE independently. It should be intuitive to interact through trial-and-errors for its initial learning and then slowly gain more expertise in applying the appropriate actions to find and resolve the detected and diagnosed software faults.*

The objective of the research is to develop a novel fault diagnosis and resolution method, based on Reinforcement Learning, that is similar to an IT system apprentice where its skills to deduce and fix the potential software faults is derived from learning to interact with the DRE through trials-errors and feedbacks from guiding diagnostics-resolving libraries that have been built to assist as its training wheels [4-6].

### **1.3. Research Question**

This research proposes an intelligent and adaptive FDR system that can diagnose the faults that arise in each Data Replicating Environment (DRE) and able to resolve them interactively through the guidance of supporting software modules that have been custom-built to assist in its learning process. The proposed system must start its learning journey with little or no a-prior knowledge against the environment, so it relies on a series of trials and errors to learn to differentiate the course of actions that are relevant to the DRE's fault scenarios.

This thesis proposes to answer the following questions:

1. What type of software anomalies be detected in DRE and under what circumstances do these anomalies are considered as defects?
  - a. What type of approach can be used to detect software errors or anomalies? What are the methods that can be used to ascertain anomalies from a multi-faceted software service's perspective, and the various techniques to conduct the identification of the detection processes?

- b. What type of strategies should the DRE software's logs and statistics be analysed once the software anomalies are detected to support both domains of faults diagnosis and resolutions within the proposed Fault Diagnosis and Resolution (FDR) system?
2. What is the strategy for the DRE's software anomalies to be presented from a system-wide holistic point of view? There are tens and thousands of components and configurations that work based on their inter-connectivity and operation to deliver a common service. The report of the fault diagnosis should be consolidated at the service level instead of on the detailed component.
  - a. What are the methods required for the system diagnostics to identify these software's subcomponent and interoperative errors, and extend them to predict the overall service outage diagnostics, instead of an individual specific fault?
3. What is the approach that the proposed FDR system needs to take the appropriate fault resolving actions to mitigate the defects?
  - a. What type of techniques can the FDR system be designed to be intelligent enough to learn by itself to discover the appropriate actions to resolve specific faults successfully?
  - b. What type of techniques can it use to act intelligently and select the optimum corrective action for any given identified fault? What kind of algorithms can fix the faults across different software groups?
  - c. What type of initial problem-solving routine can allow Deep Reinforcement Learning (DRL) to build up the experience before it can be used to prescribe accurate remedial actions?
4. How effect the proposed FDR system is as compared to the other methods of fault diagnosis and resolutions? How can its performance be measured quantitatively and qualitatively? What type of metrics can be used to assess its performance and effectiveness on its usage?

## 1.4. Research Objectives

From the research questions listed above, the research's main objectives have been identified as followed.

1. Develop a strategy to understand the DRE software and acquire their operational statistics. This is for the **first** research question 1a, the anomalies encountered across the series of DRE's software are explicit, with each error or exception being reported or shown by the operating system, replicating software and databases. In-depth research is required into these software technologies and familiarized with their operation, including the usual type of exceptions and errors that can be encountered. So, a series of different DRE's configurations is set up and evaluated in the test environment. For question 2, a list of external OS batch scripts is to be developed to read or acquire the DRE software's statistics via their log files and by the command-line interface.
2. Develop a *fault diagnosis* system that can learn the anomalous status of the DRE and diagnose the data replication environment's faults intelligently, with assistance from a comprehensive system-wide statistic querying module. Addressing question 3, the current fault diagnosis methods are commonly rule-based methods that are restrictive and confined within the designed specification. Those that are based on machine learning approaches require a substantial amount of a-prior knowledge for model training. This proposed method seeks to overcome the limitation of rule-based approaches and the prerequisite of large a-prior information for supervised machine learning-based approaches. The expected task is to guide the agent to learn via trial-and-error with the assistance of a guiding module that acts as a teacher, starting by depending on it for the initial phase of the environment's interaction, then learn to predict the best action based on the accumulated experiences to maximise the rewards.
3. Create a *fault resolution* system that learns by interacting with the DRE to find out what actions work well to resolve the faults. For research question 3a, the intelligent agent to be used here follows a similar approach in research question 2, using reinforcement learning with a guiding module to teach it how to resolve the faults initially, trying out all combinations of correcting paths and

their scores through trial and errors. The knowledge of fault, actions and rewards are then accumulated into the knowledgebase until it reached the final threshold of the training set. By this stage, it will have a knowledgebase that is comprehensive enough to determine what action is most appropriate and what is not relevant. This satisfies question 3b. For question 3c, the knowledgebase is used to train the agent's RL model to know what the best action is to take for a given environment's state and fault encountered. The fault resolutions approach available in current research for DRE faces a similar limitation as described in the previous goal which is to overcome the rule-based processes' limitation in scopes, and the machine-learning models' needs for a large a-prior dataset.

4. Establish the list of benchmarking and measurement techniques for the FDR to assess the models' effectiveness and efficiency in identifying the DRE's faults and resolutions. The efficiency on the model's training and the provision of the knowledge base to support it is also measured. There are two groups of measurement: quantitative and qualitative. Each group of benchmarking is based on the type of outputs from the model that is to be used, either in categorical form or in numerical metrics. The research will compare the FDR against the other research in fault resolution in software's fault management domain.

## **1.5. Research Contribution**

There are several contributions that this thesis makes. The first is to introduce a fault diagnosis and resolution system in the domain of near real-time data replication setup which involves multiple software such as databases, data replication and other IT services operating in a complex interconnecting and interdependent configuration. The second contribution is to overcome the current limitations of rule-based faults' diagnosis and resolutions approaches which are pre-designed, and the constraints of the need for a large a-prior training dataset for machine-learning-based approaches.

Reinforcement learning is branch of machine learning in the area of semi-supervised learning [7]. In recent times, there is a surge in the number of research that use it to manage complex problems that have exceptional high computational complexity where there is a large permutation of system states, choice of actions and

scores on the results. The domain of fault diagnosis and resolution for complex IT systems has similar complexity that most conventional rule-based expert systems may not have the capability to manage [8]. So, this thesis utilizes the current research of reinforcement learning to develop an intelligent and adaptive system to meet this challenge. From the initial literature survey, there has yet to have any publication that utilizes model-free reinforcement learning for fault resolution on real-time data replication systems on a multi-tier data replicating setup environment.

## **1.6. Research Significance**

The proposed research has the potential to support IT administrators in managing the data replication environment, providing fast fault detection, diagnosis, and resolution support around the clock and can scale to manage more systems than a human can. Another advantage of the proposed fault diagnosis and resolution system is that it is non-intrusive and complements any existing IT applications. It will not require modification of their codes, change in any products, or invoke new licenses or options. It will run in parallel on top of the existing systems, complementing the human administrators and apply the learnings that can troubleshoot and resolve the system's problem. This will free up the human administrators from the laborious work of overseeing the IT application so that they can focus on the different areas of work. It will also increase the human administrators' ability to manage even more IT systems, be it heterogeneous or homogeneous systems.

The research in the domain of fault detection and diagnosis has been focused on the use of rule-based and heuristic algorithms, together with machine learning models for a wide range of applications toward machinery and software but few have embarked on the use of reinforcement learning for software fault management. Both the rule-based approaches and deep learning models have some constraints in terms of the extent of their usage which is confined to the scope they are intended for, and the presence of available dataset or knowledge to support the models training. Such constraints are not acceptable especially in the field of complex software operation where both the boundary of faults' considerations is highly dynamic that cannot be confined and, the a-prior information may not be readily available. The proposed system in this thesis takes into consideration and mitigate it with an adaptive method in acquiring knowledge dynamically and be aware of the choices for optimum actions



selection as it interacts with the data replication software. This is synonymous with the analogy of how a junior IT administrator will learn on the job and acquire the knowledge to manage the system progressively over time through trial-and-errors, where certain events of fault occurrence will require a series of corresponding corrective actions to be taken. Another paper that showed similar intelligent fault diagnosis and repair is the research made to implement fault repair for the network use reinforcement learning which served as one of the inspirations [9].

## **1.7. Outline of The Thesis**

This section provides an overview and structure of the thesis. Chapter 2 covers the literature review on the essential knowledge that is required for the thesis. They are software technology, faults and anomalies, various type of reinforcement learning models, neural network, optimization, and control, including the real-world setup of an enterprise IT environment in a Queensland energy utility company (called Energex).

Chapter 3 describes the research methodology and design of the proposed Fault Diagnostics and Resolution system (FDR) that comprises of two mains modules: Fault Diagnostic Module (FD) and Fault Resolution Module (FR). They require a set of knowledge such as software technology, machine learning algorithms and faults management. For the software technology part which is the DRE that has both database and replicating tools, there is a need to research in detail their properties, characteristics, operation, and services.

Chapter 4 describes the construct and implementation of the Fault Diagnosis Module (FD). This is where the scripts, modules construction, cost functions and algorithms are set. Tests have been conducted to ascertain the validity and accuracy of the models.

Chapter 5 describes the design of the Faults Resolution Module (FR), its methodology and setup, action scripts, library builds, design, and test strategies, together with the test strategies and results.

Chapter 6 described the integration of both Diagnostics and Resolution modules together and the testing conducted to validate their outcome. It is the final aggregation where both FD and FR need to perform the task as per the thesis' objective.

Chapter 7 concludes the research of this thesis. It discusses the potential shortcoming that has been discovered on the proposed FDR design as well as future enhancement that have been identified.

# CHAPTER 2: LITERATURE REVIEW

---

This chapter presents the literature review that has been guided by the research objectives on the knowledge acquisition of anomaly detection, fault diagnosis and resolution in the contemporary field, covering the techniques and models used by the researchers. This is followed by the survey on Energy Queensland Limit's Data Replication Environment (DRE)'s setup and their respective software which include Oracle relational database management system and Shareplex data replicating tool.

## 2.1. Fault Detection and Diagnosis

Referring to the series of papers on fault detection and diagnosis by Venkatasubramanian et al [4-6], a fault can be defined as a deviation from the acceptable state or value that is associated with a process/function/system that doesn't perform as what it is designed for [4]. The cause of this abnormality is called a basic event or root cause which is referred to as malfunction or failure. Different types of faults can be classified according to their ability of recoverability, the number of damages caused, the severity of the faults and their impact. They can also be considered as both structured and unstructured uncertainties [4, 5, 10]. The fault diagnosis approach which is used to identify the various causes or sources of failures can generally be grouped into several classes:

1. Gross parameter changes in the model. Failure occurs when a disturbance enters the process and disrupts its work, causing it to deviate from its parameter of operations and thereby malfunctioned[11].
2. Structural changes. The process fails when the supporting entities that the process depends on have altered or become unavailable. That results in disruption to the resources available to the process which eventually halt its function[11].
3. Malfunctioning sensors or detectors. Systems that interact with other systems require transmission and reception of signals, especially in software where signals of transmission and acceptance form the basis of communication between points. When there is a fault that hinders the process from transmitting

or receiving the signals properly, communication will be impeded which eventuates malfunction in the process[11].

The process of fault detection transiting to the final decision-making process can be generally summarized in the following flow[11-13];

*Measurement Space → Feature Space → Decision Space → Class Space*

In the Measurement Space, raw data are gathered from sensors, logs, meters, and other data reading devices. This data is acquired without prior knowledge. At the Feature Space, the function of measurement is obtained or derived with the usage of a-prior problem knowledge to obtain features that can be used to support decisions. At the Decision Space, the obtained features are mapped to the decisions based on an objective function such as indiscriminate or threshold functions. In the Class space, the categorization of faults is executed based on the results produced at Decision Space [4, 5, 10].

Numerous fault diagnosis techniques have been developed. They are generally grouped into three categories: quantitative-based methods, qualitative based methods, and process history methods [4, 5, 14]. For the quantitative-based methods, analytical processes are used to generate results or residuals that can be used to isolate the fault; this is where all the sensors or log data must be present. The method will derive a decision concerning a-prior problem knowledge [4, 10, 14]. For the qualitative based methods, they focus on contexts, topographic or symptomatic searches to determine the fault with a priori problem knowledge [4, 10]. For the process history-based methods, it is assumed that a large amount of historical data is present and can be used to extract features or build models using machine learning or statistical algorithms. It can also be a combination of quantitative and qualitative approaches[4].

While the three approaches defer in their techniques and field of use, from a global perspective, fault diagnosis processes can be decomposed into a series of feature extraction and classification stages before submitting to a decision stage. Under the classification stage, there are three subgroups; pattern recognition, model-based reasoning, and model-matching, in which all three are key factors under the three different fault diagnosis models mentioned above [15].

So, whenever a fault is detected in a process, the diagnostic model will have a classifier that can draw up a set of hypotheses or assumptions that can explain the

reason behind the anomaly. It is questionable how complete is the diagnostic classifier and how accurate is the hypothesized faults versus the real fault? The difference will need to be as small as possible. Venkatasubramanian et al [4-6] proposed a list of desirable features that a good fault diagnosis system should have.

1. Quick detection and diagnosis. A fast and accurate response is required for mission-critical machines and people cannot afford to wait around for a resolution when services are unavailable [4, 5, 14].
2. Ability to Isolate. It can distinguish the different failures that occur in the process and can detect symptoms that are specific to specific faults without getting confused with other faults that are not occurring [4, 5, 14].
3. Robustness. It must be strong enough to withstand and accommodate different environments that the process is working in. and able to withstand a range of external environment factors and have some resilience to hardware faults [4, 5, 14].
4. Novelty identifiability. It should be able to distinguish the process conditions and determine whether it is running normally or abnormally [4, 5, 14].
5. Classification error estimate. All classification models will give a certain degree of error, but this must be accurate enough to be of use and able to give the user confidence in the detection system [4, 5, 14].
6. Adaptability. It must be able to adapt whenever the process or environment changes and persist in its function while exempting its sensors from the external disturbance, from single to heterogeneous multi-site setup [4, 5, 14].
7. Explanation facility. Besides detecting the fault, it should be able to explain the fault's origin including the cause/effect symptoms [4, 5, 14].
8. Modelling requirements. The classifiers and other feature mining tools require time to train their models. So, it is desired that the time needed for this effort should be kept to a minimum [4, 5, 14].
9. Computational requirement. It should be computationally cheap to maintain with average hardware consumption needs [4, 5, 14].
10. Multiple fault identifiability. It should have the ability to detect multiple faults simultaneously. Most of the systems in use are non-linear so the interaction will be almost spontaneous and ad-hoc, so the system must be able to combine

those single fault detecting models to form a more comprehensive multi-fault detecting one [4, 5, 14].

## 2.2. Anomaly Detection (AD)

The fault diagnosis methods discussed in the previous section need to extract meaningful features to establish the decision of anomaly detection concerning a-prior knowledge. The information produced by the feature extraction algorithm can contribute to posterior knowledge. Anomaly refers to the data that do not conform to a typical expected behaviour or pattern. It can be considered as outliers, exceptions, abnormal or deviation from the norm [16].

Referring to figure 1, the type of anomaly detection methods is determined by several criteria, namely the type of data, anomalies, models and software area that the AD is intended for [16]. There are three types of anomalies: point, contextual and collective anomalies. Point anomalies refer to an individual data point that is anomalous concerning the rest of the data and this can be detected by using a threshold setting. The contextual anomalies refer to the data point that is anomalous to the specific context of the data, but it is not anomalous if it occurs in a different time, region, or group. This can be detected by focusing on the data set by segments. Collective anomalies refer to the group of data points' relationship that is anomalous concerning the rest but not to individual values. This type of anomaly has two variants: the occurrence of an event in an unexpected order or, unexpected combination of data values. It requires the use of collective anomaly techniques on a specific segment of data individually [17].

There are various forms of anomaly detection; for a given dataset  $D$ , the method will find those data points of  $x \in D$  with a score higher than a threshold  $t$  or score within the extreme top or bottom- $n$  of the dataset. The other form tests the data point of its anomaly score concerning the dataset  $D$  that contains normal data such as distance or score [16].

A typical anomaly detection method depends on the type of data, the anomaly type (point, contextual or collective), the type of anomaly detection used, the presence of administrators' supervision and the detection results [16]. Relating the knowledge learned from the literature review on intrusion detection based on network research

[18], the source of input data comes from various avenues. The attributes of the input data can be described as binary, categorical, continuous or hybrid. A typical example of such an attribute in this research project can be the software's service status, group IP addresses, system values or a combination of both. For complex data types, their relationship can be extended to sequential, spatial, spatial-temporal or graph. Examples of these data in this research environment can be substation time series, geographical map, time-space of field crew deployment and power distribution network graph map [17].

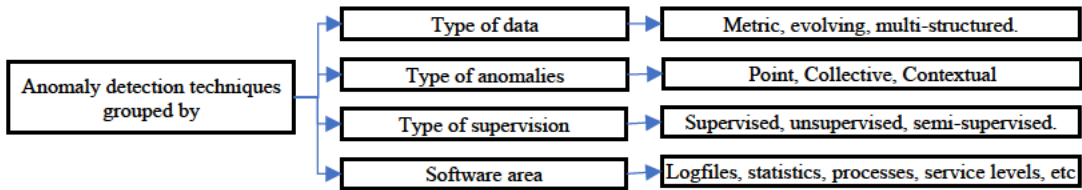


Figure 1 - Classifications of anomaly detection techniques

### 2.2.1. Anomaly detection techniques

Many machine learning methods can be utilized for the task of anomaly detection. There are two modes of learning available for this task, supervised and unsupervised. The nature of the data and their attributes determine the modes of learning that can be used. If both normal and abnormal data can be labelled explicitly, then supervised learning should be considered. But in the event where the frequency of anomalous data occurrence is sparse, but no labelling is available, then unsupervised learning can be applied. A hybrid solution is considered semi-supervised if labelling is available only for the normal data or a subset of data [19].

For the supervised and semi-supervised learning methods, the common approach of anomaly detection is to build a classification model using the label information of the training data; the test instances are scored to define their state whether they are normal or anomalous based on the prediction of the trained classifier. One of the advantages of a classification-based method is that it can be clearly understood as the data has been segregated into the normal and abnormal groups by administrators with domain knowledge, as well as it can detect anomalies with a high level of accuracy. However, the downside of this method is the requirement for labelled data which may not be readily available. Also, it will not be able to mitigate new events that are either unknown or emerging [12].

For unsupervised methods where labels are not readily available, an assumption is made that the abnormal test instance is very extraordinary as compared to the mass normal data and their occurrence is rare. The approach here is to build a set of data (i.e. clusters) that are deemed as normal and this normal model is used to compare with the sample data to test whether they are normal or not [20]. For a clustering-based method, the assumption is that most of the normal data tend to group in clusters whereas anomalous ones will either be out of the group or form their unique group. Methods may require post-processing to determine the data points' distance from the cluster and that in turn determines the degree of normality or anomaly [16]. One of the strengths of clustering-based methods is that no supervision is required and can be adapted to near real-time or incremental mode of data changes. The downside is the amount of computation needed to compute the model. Also, it heavily relies on the ability of normal data to form a cluster or else it will fail, especially in high dimension data where the concept of distance to distinguish data points diminishes, thus obfuscating the separation of normal data from anomalous one [16].

In a point anomaly detection method, the process is simple. It determines an anomaly by measuring a single data point concerning the rest of the data using a threshold or deterministic rule. Should the data exceed the threshold, it is considered anomalous. For a contextual anomaly detection method, it assumes that all data instances, within a context, will exhibit similar patterns and attributes whereas anomalous data behaviour will be different. It will use a set of contextual attributes to validate a new data instance to see if it is normal or anomalous. The advantage of this method is that it can detect anomalies that may not appear as a point anomaly, but it is anomalous when it is detected against a higher plane of perspective [16] [19]. However, this method requires pre-processing; the contextual attributes will need to be defined as well as the context in which these attributes will relate to. The attributes are used to segment data and then apply a point outlier concerning the attributes to that specific context [21].

A collective anomaly detection method tests the relationship among the given set of data points for anomalies within a segment from a global perspective. The set of data points may not be anomalous if they stand alone, but in comparison to a wider



perspective of the data series, this specific set's behaviour may not comply with the wider dataset's pattern.

Both the context and collective detection methods cover context and collection of data instances that are of the spatial, graph, sequential or profile nature. They rely on the correlation to determine the nature of the abnormal operation or symptom for the given segment. But comparing these methods to the point detection approach, it will not be able to yield much information to determine the real root cause or fault [22].

Data in the present world is produced at an enormous pace and volume. The definition of data normality here is not static and it changes through time. The challenge here is to detect anomalies over a large volume of data and update the definition of what is considered as normal data constantly at frequent intervals. Therefore, the data that is streamed into the system are examined within a certain time segment to derive the normal profile, which in turn will be used in the next time segment for data detection. Incremental Local Outlier Factor (LOF) algorithm is commonly used in this context [22].

In distributed anomaly detection, data come from numerous sources, and they come in the various form of speed, volume, and variety. They form the most difficult challenge where detection must be performed not only across the various sources but also correlating them to detect anomalies from a global perspective. Multiple aspects such as timing, the relationship of specific data's outliers to other data sources can result in different outcomes and anomaly categorization. Because the data sources come in multiple forms and high volume, higher computational throughput and quicker turnaround time are needed.

There are several approaches for distributed anomaly detection. The first approach is the simple data exchange where all the data instances from multiple sources are merged into a single location and processed. The second approach is to use distributed nearest neighbours such as  $k$ -nearest neighbour algorithm to find one data instance per distance computation. The third method relies on the exchange of data mining or statistical models for the near data source of a certain level of similarity and then combine to form an over-arching detecting process to find anomalies from a global perspective [19]. Finding the root cause problems that occur within a distributed

data replicating system is complex as well as tedious; there is usually more than one origin where multiple sources of events and logs are present. To make things complicated, these data are generated from their sources based on their functions or services, e.g., time or event triggered. It makes it more difficult to consolidate all this widely scattered information to extract the necessary features for the analysis. There are numerous methods available for anomaly detection, each developed to meet specific field's needs across different but not limited to scientific, engineering, business, or financial sectors, and they are described as follows.

### **2.2.2. Clustering-based techniques**

Clustering-based techniques can be segregated into the following groups.

1. Grid-based techniques. These techniques use a graph technique of hypercube or cells across the set of data points and group them based on specifications of the domain of interest [20].
2. Centroid-based techniques. These techniques assume that anomalous data points will not be part of the clusters of normal data. They measure the distance of the data point to the centres of all cluster centroids. If the distance is longer than what the other data points' have, then it is considered anomalous [20].
3. Density-based techniques. The data points are grouped based on their proximity to one another while forming density within regions. These techniques compute the regions' density across the dataset and those that are in low-density regions are considered as anomalies whereas those in high-density regions are considered as normal [16, 20].
4. Nearest neighbour-based techniques. This type of method assumes that all normal data points should be close to one another whereas the anomalous points will be a certain distance away [16, 20]. It finds the distance of each data point to its k-th nearest neighbour, sort the data points based on the computed distance and find the top-n groups of data points that have the largest distance. These are then considered outliers or anomalies. However, the nearest neighbour approach has some limitations as it cannot detect anomalies from both contextual or collective contexts [23].

### 2.2.3. Classification based techniques

There are numerous classification techniques such as the following.

1. Deviation detection is where the amount of data that belongs to a certain class is over or under-represented so that in the class distribution, the ratio of normal data will exceed those anomalous, then it uses the misclassification that is based on that cost ratio. Synthetic Minority over-sampling technique is one example [16].
2. Rule-based such as Association rules like A-priori or Frequency Pattern (FP) growth algorithm. It creates association rules among the items or data points that have higher support higher occurring normal data with higher support as compared to rare but anomalous ones [16].
3. Cost-sensitive classification uses the method of misclassification on a data point that must be labelled with the use of a cost matrix and then works out a classification based on the data point's cost. So when the cost of the data is derived against the cost matrix and fall below the threshold, it is regarded as anomalous [16].
4. The use of machine learning models such as support vector machines, decision trees, random forest, neural networks, and many others.

Support Vector Machine (SVM) is a supervised machine learning model that can be used for both classification and regression. [24] The concept is to find a hyperplane that can divide a given dataset into classes. So data points that are closest to the dividing hyperplane are called support vectors, which are considered as important elements as they can affect the hyperplane's position [24, 25]. It has been used in research for anomaly detection across a broad sector [26]. A decision tree model builds a classification tree in a hierarchical tree structure form that splits a dataset into smaller groupings. Each split of the leaves is branched into a decision node that represents a classification or decision. A benefit of using a decision tree is that it can manage both categorical and numerical data [27]. A neural network comprises multiple units called neurons and they are interconnected and arranged in multiple layers, with each subsequent layer taking inputs from the previous one as a vector. For each neuron, it takes in all the inputs, applies a nonlinear function, and transfer the output to the next layers of neurons in a cascading flow [10]. Weights are applied to the signals which

pass among the neurons and these weights are adjusted by the training phase. This model will be covered in greater detail as it will play great importance in the candidate's thesis [27]. K-nearest-neighbours takes a set of labelled points to learn how to label new data points by assessing the surrounding labelled points that are closest to the new point and take voting from the neighbouring point. Whichever label has the most significant presence will influence the new points and transfer their labels to them [27].

#### **2.2.4. Other techniques**

There are several other techniques inherited from areas other than machine learning like Information Theory which assume that anomalies have high information content due to the irregularities. They perform an investigation to find the subset of data points that have the highest irregularities [16]. Dimension reduction assumes that normal data can be expressed clearly in a lower dimension after the dimension reduction technique is applied, where anomalous data will be difficult to express [16]. Graph analysis depicts the interaction among the data points as a relationship graph, which is used to verify the data to determine the normality [16].

### **2.3. Organization IT Infrastructure**

The next section is the study conducted against the organization's data replication environment and its setup. The organization is Energy Queensland (EQ) and it is Queensland's state-wide power distribution utility. With an asset of more than \$2.4 billion and 4.8 million customers, it is one of the largest utility companies in Australia. The database team of EQ's digital Office provided a study ground with access to its non-production system together with its system operating procedures and software documentation. The software that this research focuses on will be on the data replicating tools and relational databases.

#### **2.3.1. Data Replication System - Shareplex**

Shareplex is a common data replicating tool developed by Quest software for both commercial and open-source databases [28]. Data replication is an essential business requirement where data are copied to other systems to maintain high availability, data reporting, business consolidation, workload sharing as well as

support disaster recovery standby nodes with redundancy in data sources. High availability refers to the continuity of the application services should the hardware or software that provide the service accidentally fail. Another server that is a duplicate of the primary server and has the latest copies of the data will start up and take over the services. The most widely used database platform that Shareplex support is Oracle databases. However, Oracle database technology has its data replicating technology such as Oracle Streams, Materialized Views, Data Guard or GoldenGate. Both solutions have their strength and weakness and it becomes a customers' choice to use a solution-driven by cost, complexity or license options [29].

Shareplex runs in the background without interrupting the business processes that occur in the database and it replicates only the changes as they occur by the means of reading the Oracle's redo log and archive logs constantly [28]. Oracle redo logs record all the committed transactions that occur in the database and Shareplex reads these redo logs periodically to acquire all the commit SQL statements. It then performs the data capture and replication in near real-time, sending changes to targets that are specified. The Shareplex framework comprises several components apart from the source and target databases that they run against with the processes of data capture, read, export, import, and post. Figure 2 shows the various Shareplex components and illustrates how the changes are captured and transported from the source to the target databases [29].

Capture process – it runs against the source database, constantly reading Oracle's redo logs and sometimes archived logs for changes, then sending the change to the capture queue. The capture process is called `sp_ocap` [30].

Queues – all the queues are dynamic data repositories that hold the temporary data for the duration of data capture, transmission, and reception through the process of data replication. The order of the queue's relationship follows from the capture queue to the reader's queue, to the export, then to the target side's import which connects to the post queue.

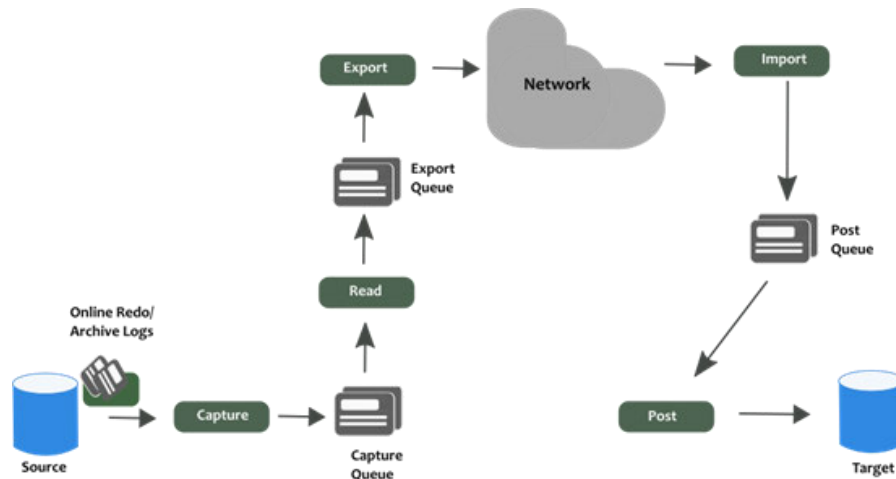


Figure 2 - Shareplex data replication flow [28]

The major components and subsystems of Shareplex are as follows:

**Read process** – this runs only at the source side; it reads the data from the capture queues and processes it by repackaging them with information for network transmission. The processed data is then stored in the export queue.

**Export process** – this runs at the source side, and it reads the processed replicated data from the export queue and transfers it to the target across the network. The process name is `sp_xport` and it can send over the data changes information to single or multiple target systems.

**Import process** – this runs at the target side, and it intercepts all transported replicated data sent out by the export process and stores them in the import queue. The process is named `sp_import`.

**Post process** – this process runs at the target side and transforms the data read from the import queue into relevant SQL statements before they can be executed against the target database [29].

**Replication configuration:** The replication can be set up or controlled by a configuration file that defines the list of tables that need to participate in the data replication and the information is split into two sections, the source, and the target. For the source side, there are the schema and table names, while on the target side, the intended schema and object names to which the data is replicated to. This is followed by the routing information which defines how the relationship between the source and the target are linked in the following format of `(target_system:named_queue@o.Target_oracle_sid)` [31].

Shareplex filesystem: Shareplex operates from two main directories; product directories where all the binaries are stored and executed from as well as a variable data directory where template files, licenses, parameters, logs and trace files are stored. It also stores all the temporary data files that are used by the queues as well as the associated network routing information that is associated with them [29].

Shareplex operation: Shareplex system runs under a specific UNIX user account that shares the same admin group as the Oracle database group which was used to install the Oracle binaries. There are several Unix's environment shell parameters; \$ORACLE\_SID, \$ORACLE\_HOME. \$SP\_SYS\_VARDIR, \$SP\_SYS\_HOST\_NAME, \$SP\_COP\_TPORT and \$SP\_COP\_UPORT. Once Shareplex is installed on both the source and target systems, the administrator will activate a configuration file to initiate the data replication. All information, including debug and errors, are captured and stored into event\_logs under \$SP\_SYS\_VARDIR/logs directory. In the event should some tables are out of sync, Shareplex has a compare/repair feature that allows the administrator to fix the replication tables and bring them back into synchronization [29]. Figure 3 is a screen log of how the status of shareplex's processes show in within the SP\_CTRL console, plus a copy of the Shareplex's event\_log with the indication of operational anomalies via warning and error messages.

```

sp_ctrl (linux:2100)> status

Brief Status for linux
Process      State          PID    Running  Since
-----
Cop          Running       4408   07-Apr-22 11:32:31
Capture     Running       4409   07-Apr-22 11:32:31
Read        Running       4412   07-Apr-22 11:32:31
Export      Running       4411   07-Apr-22 11:32:31
Import      Running       4414   07-Apr-22 11:32:31
Post        Running       4410   07-Apr-22 11:32:31
Cmd & Ctrl   Running       4488   07-Apr-22 11:32:37

Warning 2022-04-07 11:36:45.416558 5243 4000511936 Import cannot connect to export on linux:
Info 2022-04-07 11:36:45.417203 4408 1437173696 Export exited normally, pid = 5242 (exporting to linux queue Q0)
Info 2022-04-07 11:36:45.437226 4408 1437173696 Import exited with code=1, pid = 5243 (importing from linux queue Q0)
Notice 2022-04-07 11:36:54.066617 5521 2039494592 User command: splx start export (from linux)
Info 2022-04-07 11:36:54.068437 5846 1437173696 Export launched, pid = 5846 (exporting to linux queue Q0)
Info 2022-04-07 11:36:54.217853 5846 7727040 Export connected to host on linux
Info 2022-04-07 11:36:54.218547 5846 7727040 Export connected to import on linux
Info 2022-04-07 11:36:54.229552 5848 1437173696 Import launched, pid = 5848 (importing from linux queue Q0)
Info 2022-04-07 11:36:54.328264 5848 820037568 Import connected to export on linux
Notice 2022-04-07 11:36:54.329323 5848 820037568 Import: Queue write recovery started, qrw_srcseq=2031903000 msg.mpseq=2026292000 (importing from linux queue Q0) [module que]
Notice 2022-04-07 11:36:54.338286 5848 820037568 Import: Queue write recovery complete, 2436 duplicate messages skipped , 896520 bytes total (importing from linux queue Q0) [module que]
Error 2022-04-07 11:37:12.636665 4408 1437173696 Reader killed due to SIGKILL, pid = 4412 (from o.DB1)
Info 2022-04-07 11:37:12.660896 5965 1437173696 Reader launched, pid = 5965 (from o.DB1)
Notice 2022-04-07 11:37:12.786507 5965 953239488 Reader: Replicating according to target compatibility of "9.0.1" (from o.DB1) [module sys]
Notice 2022-04-07 11:37:14.081914 5965 953239488 Reader: Oracle Available (from o.DB1) [module ord]
Error 2022-04-07 11:37:46.148842 4408 1437173696 Import killed due to SIGKILL, pid = 5848 (importing from linux queue Q0)

```

Figure 3 – Shareplex's SP\_CTRL status and EVENT\_LOG output with error messages

### 2.3.2. Relational database management system (RDBMS) - Oracle

Oracle database is an object-relational database management system developed by Oracle Corporation. It consists of an instance and data storage. The instance

comprises a set of processes and memory structures that interact with the data storage. The primary processes are processed monitor, system monitor, database writer, log writer with other secondary processes that support it. Oracle instance is also a shared memory domain that has several areas like System Global Area (SGA) that holds information on data, users, programs or SQL statements, dictionaries, data, and others. Figure 4 shows an overview of an oracle RDBMS architecture depicting the memory structure and interaction with its various data storage [32].

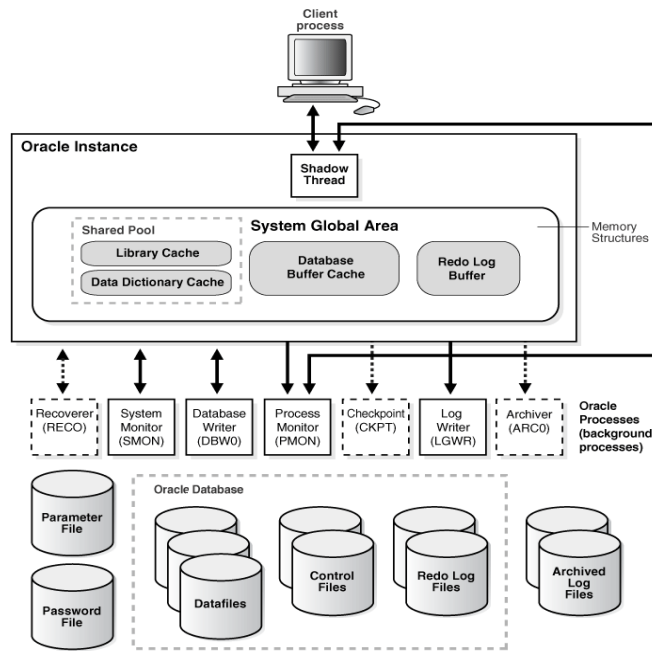


Figure 4 - Oracle RDBMS architecture [33]

The smallest entities in the Oracle storage system are called a block which corresponds to a specific number of bytes of database physical space on the hard disk storage. The next level is an extent which is a collection of contiguous data blocks. The next storage group is called a segment that holds a set of extents, and they are used to store both data and index. The RDBMS consists of multiple logical storage units called tablespaces which store the entire database's data. Each tablespace consists of data files which are physical structures that are actual files on the operating system, and they are made up of segments. The RDBMS has a series of primary files that are essential to its functionality and they are described as followed [32];

1. Data files – physical files that hold both user and system data permanently as well as other system information.



2. Redo logfiles – stores logs of transaction activities that have been written out from the redo log buffer memory.
3. Archived logfiles – these are archives of redo logfiles that will be used in data recovery.
4. Temp-files – this file holds the temporary transactions that occur in the database as well as sorting made by both the users and the system.
5. Control-files – these are files that hold the essential background information that is critical to the database system operation and it stores change control sequence, the location of the files, parameter setting plus other fundamental details [34].

Memory structures are.

1. System Global Area (SGA) – the primary memory structure that comprises the redo log buffer, shared pool, large pool, buffer cache, java pool, and stream pool
2. Library Cache – stored the shared SQL, execution plans for all the executed SQL statements.
3. Data Dictionary Cache – stored the information about the database’s logical and physical attributes such as file location, user details, and various database objects information.
4. Program Global Area (PGA) - keeps track of all the Oracle server and background processes’ information [34].

The above are the primary components that all Oracle databases have. However, the list of services listed above are the main core to the database’s operation, and there is a wide range of other value-added enterprise options such as partitioning, encryption, data analytics, spatial, Java and XML support. However, these options are beyond the research scope and will not be discussed here.

### **2.3.3. Energy Queensland’s Data Replication System**

The following section reviewed Energy Queensland’s (EQ) data replication setup between Shareplex and Oracle, which will form the basis and test subject for this research [35].

Shareplex is a data replicating system that is used by EQ to copy changing data that occur in IT systems over to a different location or system [18]. It is common to

replicate data from online processing systems (OLTP) such as order processing systems to a backend reporting system like a data warehouse. There are two types of data replication: Data direct languages (DDL) or Data manipulating language (DML). The DDL refers to the SQL language that alters the structure of database objects like adding or dropping a column whereas the DML refers to the data level manipulation such as insert, delete, or update.

The data replication process functions in near-real time with very low latency between the databases to meet the expectation of sharing data among the various IT system with minimum delay [30]. A Shareplex system usually runs in parallel with a relational database and reads the database's redo logfiles continuously for data changes. It has an active configuration list that contains the sets of tables that specify the replication services; e.g. what schema and table that need to be read and where the data should be replicated to [30].

Figure 5 shows the Shareplex deployment and architecture that had been set up in the Enterprise Performance Management (EPM) Exadata appliance and its replication connectivity to other EQ's IT systems, from the design overview down to the details of deployment and operations. The EPM is another instance of EQ's enterprise data warehouse (EDW) that uses Shareplex technology extensively. Each source to the EDW is handled by an individual Shareplex instance operating at a specific port as the list above. This is to isolate and mitigate the instance failures or human errors, plus segregating their roles to minimize overlapping replicating responsibilities. Table 2 lists the technical configuration of the Shareplex instance setup in the corporate environment while table 2 shows the configuration file that is used to control the replication setup and routing.

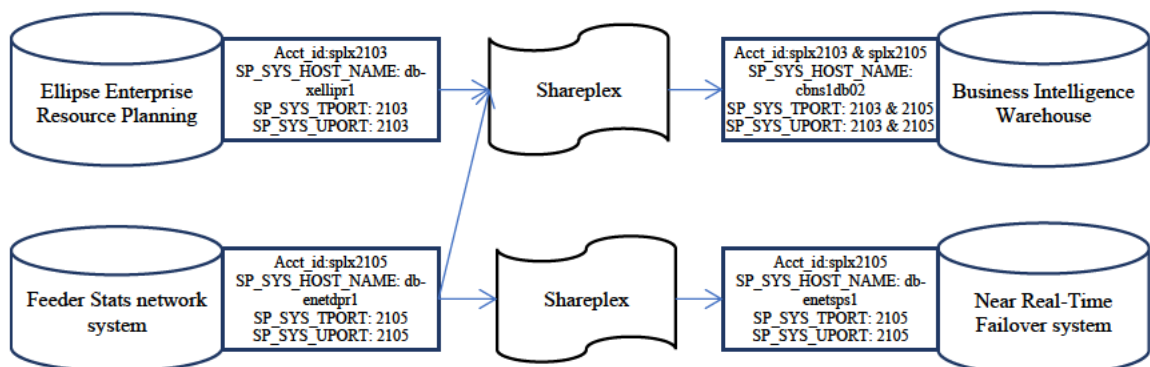


Figure 5 - Energy Queensland’s Shareplex and oracle integration overview

For this setup, all the instances at the target side will be administered and launched from the same Unix user account, e.g., qxedw1ut, with a menu option to choose and set up the environment that corresponds to the specific Shareplex instance. The environment variables that need to be set up are SP\_SYS\_TPORT, SP\_SYS\_UMPORT and SP\_SYS\_HOST\_NAME. Each of the Shareplex instances has its unique product directory and variable directories which store their binary, configuration, parameters, and queue data. Figure 5 depicts the two such Shareplex instances set up on the EDW’s system and their inter-connectivity relationship and Table 1 contains the details of their setup including the port that they operate on and their designated home directories [29]. Table 2 is an example of the configuration file that Shareplex uses to set up the replication for individual tables from the source to the target and that includes the routing map which has the source/target hostnames well as the desired queue plus the databases.

Table 1 - Shareplex configuration table on EPM data warehouse

Source Unix user	Source	Qxedw1pr’s MDIR	Qxedw1pr’s VDIR	Source		Target	
				Server	port	Server	port
qxellipr, qxel2ipr	ellipse	/db/SharePlex/XEDW1PR/splxprod/oh1/8.5	/db/SharePlex/XEDW1PR/splxvardir1/oh1/8.5/XEL LIPR1/2210	cds12a	2104, 2310	cbns1db01, cbnf1db02	2104, 2310
qxnetwpr, qxne2wpr	netsys	/db/SharePlex/XEDW1PR/splxprod/oh2/8.5	/db/SharePlex/XEDW1PR/splxvardir1/oh2/8.5/XNE TWPR1/2220	cds14a/14b	2100, 2320	cbns1db01, cbnf1db02	2100, 2320
qxfl1hpr	ffah	/db/SharePlex/XEDW1PR/splxprod/oh3/8.5	/db/SharePlex/XEDW1PR/splxvardir1/oh3/8.5/XFF AHPR1/2230	xbneuv03	2330	cbns1db01, cbnf1db02	2330
qxmk1spr	peace	/db/SharePlex/XEDW1PR/splxprod/oh4/8.5	/db/SharePlex/XEDW1PR/splxvardir1/oh4/8.5/XM KTSPR1/2240	xbneuv40	2340	cbns1db01, cbnf1db02	2340
qxes1fpr	esafe	/db/SharePlex/XEDW1PR/splxprod/oh5/8.5	/db/SharePlex/XEDW1PR/splxvardir1/oh5/8.5/XES AFPR1/2250	cds12a	2350	cbns1db01, cbnf1db02	2350

Table 2 - Typical Shareplex configuration file sample

```

datasource:o.XFFAHPR2
#source tables target tables routing map
## E X A D A T A P R O D C O N F I G
QLAG_OWNER.Q30 (QNAME,QTIME1) QLAG_OWNER.Q30 (QNAME,QTIME1) db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
EDW_LDG_OWNER.AREA_NODE SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.ASN_ASSIGNMENT EDW_LDG_OWNER.ASN_ASSIGNMENT_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.ASN_VISIT_T EDW_LDG_OWNER.ASN_VISIT_T_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.LAB_BUSINESS EDW_LDG_OWNER.LAB_BUSINESS_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.LAB_RESOURCE EDW_LDG_OWNER.LAB_RESOURCE_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.LAB_USER EDW_LDG_OWNER.LAB_USER_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.ORD_ACTIVITY EDW_LDG_OWNER.ORD_ACTIVITY_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.ORD_ACTIVITY_CUST_DATA EDW_LDG_OWNER.ORD_ACTIVITY_CUST_DATA_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.ORD_JOB_CODE EDW_LDG_OWNER.ORD_JOB_CODE_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.ORD_ORDER EDW_LDG_OWNER.ORD_ORDER_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.ORD_ORDER_CUST_DATA EDW_LDG_OWNER.ORD_ORDER_CUST_DATA_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.SR_REPORT EDW_LDG_OWNER.SR_REPORT_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR
VSS_HDB.ORD_ORDER_STATE EDW_LDG_OWNER.ORD_ORDER_STATE_SV db-xffahpr2:exa30*cbns1db01-vip@o.XEDW1PR

```

In EQ's IT landscape, there are generally about 350+ oracle databases and 96 of them are managed by Shareplex replication to transfer data to the in-house data warehouse systems as shown in Figure 6. Database administrators must ensure that all the databases and replication services are operational at optimum level 24x7. with a service level agreement of 99.5% uptime, a very strong emphasis must be placed on identifying and fixing the problem with absolute minimum downtime. Here is a typical scenario of troubleshooting that is commonly practised by their database administrators.

1. Suppose the enterprise system and network monitoring team reported that one of the Shareplex on backlog has reached counts of over 30,000 in the queues. The queue's lag time has exceeded 30 mins threshold.
2. The sysadmin logs in and checks the Shareplex's event\_logs and associated logs for details.
3. If the administrators cannot determine the error, they would check the /var/admin/syslog to verify if the error has been captured by the OS. At the same time, they will check Oracle's alert log in its trace folder for errors.
4. A further check is conducted on the status of the Shareplex processes internally using the sp\_ctrl console and externally using `ps -ef|grep plx|grep -i` import commands.`
5. Based on the information obtained running checks #1 to 4, the sysadmin may or may not be able to recollect from his experience the best possible solution to resolve this issue.
6. If they can resolve this problem, then the replication service will be restored and resumed.
7. If they are not able to resolve it; then they must open a request ticket with the Quest technical support for the resolution. That may involve a lengthy bilateral collaboration between the sysadmin and the Tech support before a final solution can be derived.
8. Assume the sysadmin applies the newfound solution and resolves the issue.
9. The cycle is repeated for the next incident or fault.

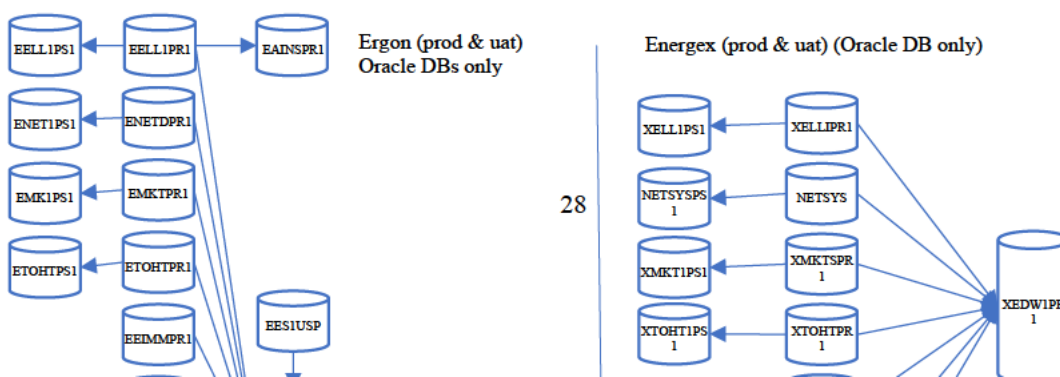


Figure 6 - Current architecture of EQ's enterprise data replication

After the fault has been diagnosed, the next stage is problem resolution. The problem resolution or treatment requires an iterative task of applying recommended actions and observing the effects or change state. While most recommendations work in a deterministic environment with assured controlled parameters settings, certain abnormal events do occur during the data replication operation which may be caused by unforeseen direct or indirect circumstances which impact the IT infrastructures. The resolution task must be adaptable to interact readily with the environment on the fault issue. It should be able to try all possible solutions that are closely related to the existing fault and its recommended fixes and find the optimum solution. As such, this domain of problem resolution is considered stochastic as the real world is complex and dynamic, which makes the treatment of the problems difficult to identify. Because of this complexity, state-space must be abstracted for a problem-solving attempt that replaces the real state with the abstract state, the complex real actions with the abstract action and a real path to a solution in the real world with an abstract solution.

Data replication problems can be grouped into the following three categories: (1) Deterministic where agent knows what the data replication's state is in and the solution to restore its functionality is a sequence; (2) non-observable which is a conformant problem where the agent doesn't know what state the data replication state is in or if any solution is available or not, and (3) Non-deterministic which is a

contingency problem that may be partially observable and the agent only can receive new information about the current state and solution is another new contingency policy that needs interleave of search and execution process. The unknown state-space of the data replication where the problem to solution mapping are not defined created an exploration challenge since nothing is known about the state [36-38] [39].

#### **2.3.4. System Anomalies detection in DRE**

This section describes the general type of anomalies that can be encountered in the DRE software which occur in their configuration and operation. The anomalies relating to the DRE can be regarded as explicit and implicit. The explicit DRE anomalies can show that there is an immediate and present problem among the DRE software, and this is generally on configuration or absent of services [32]. While the implicit DRE anomalies are more related to performance issues which are caused by numerous factors such as slow connection or services or impeding system resources constraint. These DRE anomalies and errors can be classified into quantitative and qualitative groups [30, 32].

For the Quantitative related anomalies these are related to statistical information that are generated by the various DRE to indicate issues that are developing, and they involve metric readings of software's operation and configuration parameters [28]. Some values are direct indication of functional and configuration anomalies while others have values that exceed certain acceptable operational thresholds. The first group is on explicit error that indicate a clear and present outage, such as software of the database and data replication services are either not running or unavailable, while the other group may show the queue of the data replication processes have some performance bottleneck or operational contention problems [29] [30]. The following are some examples from the extensive list of possible DRE explicit quantitative anomalies.

- Absence or failure of process id in the OS environment, Operating system's resource issue such as disk storage or memory full.- Network's ping value return error value [28].

- Backlogs of records that are present in the queue for the capture, export, read, import, write queue, abnormal value can indicate some issue with the overall data replication transfer process and will require some attention [28].

This is quite significant within the DRE as the Oracle database and Shareplex must remain in peak condition to support an efficient replication process [30].

For the Qualitative group of DRE anomalies, there are human-readable texts that describe certain software issues that pertain to the DRE's Shareplex, oracle or the OS [30, 32]. Each software has its own specific error messages that have been developed by their vendors and they offer concise and accurate description on their anomalies encountered. These messages are delivered in two forms; the first is the list of associated event and error log files which the software constantly update them. The other form is only available when the IT admin initiate a console to explicit interact with the software and extract the error messages. Some examples of the qualitative anomaly are follows:

- User's accounts are denied or password is invalid.
- Services such as Oracle database, Shareplex instance, network connectivity is not available.
- Certain software processes that have been stopped due to specific errors that arise.

There is an extensive list of DRE anomalies which shows whether the issue is isolated or specific to each software's domain or they are related to the interoperability among the various software [29, 30, 32]. Their detection is done via consistent scanning of the software's logs and probing on their statistics via their console programs. For this research, the challenge is to develop all the routines that can perform these specific tasks of acquire the necessary information about the DRE' software anomalies so that they can be used for the training of the models.

## **2.4. Reinforcement learning**

In the context of artificial intelligence, one of the approaches to resolving a complex problem task is the use of dynamic programming. In dynamic programming, the method to solve a complex problem is to break it down into a group of simple sub-

problems. Each sub-problem is solved one at a time and the solution is stored in a data structure or repository. So, when a similar sub-problem is encountered, the program will look upon this repository for the solution which is faster instead of recomputing it again, so it saves time and resources [40]. Reinforcement learning is a variant of dynamic programming that creates an agent that learns by interacting with its environment and receives rewards and penalties for the correct and incorrect tasks performed. The agent learns to be autonomous by maximizing its reward and minimizing its penalty [40]. There are two types of reinforcement learning: model-based and model-free. For model-based, the agent refers to a previously learned model to do the current task, whereas, for model-free, the agent simply depends on a trial-and-error method to find the action [39, 41-44].

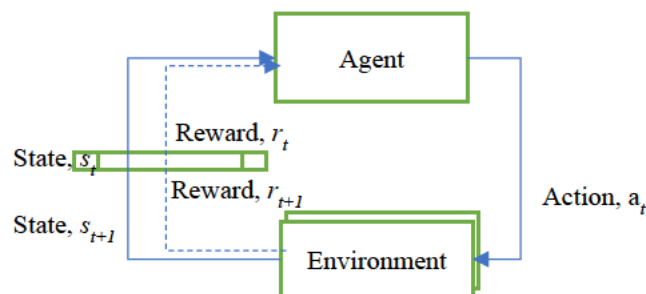


Figure 7 - Reinforcement learning's Agent processes

Generally, the agent interacts with the environment and perceives the state of the environment. It then takes action and receives rewards. The goal is to choose actions to maximize rewards. Refer to figure 7, for time  $t$ , the agent executes the action,  $a_t$ , which will receive an observation of  $s_t$ , and a reward,  $r_t$ . The environment changes when it receives the action,  $a_t$ , and gives out observation,  $s_{t+1}$ , together with reward  $r_{t+1}$ . The cycle repeats until the goal is achieved. The optimal behaviour,  $\pi$ , is based on past actions and it tries to maximize the expected cumulative rewards over time [39, 41-44].

However, this model has some challenges as the agent is not able to reason about the long-term effects of the actions it takes. Also, the agent cannot respond immediately as the feedback is delayed. So, in this situation, the agent must learn about its current situation and try to maximize the chance to score more rewards. To do this, it must do some trial-and-error through exploration of other actions. This is to try out all sorts of actions and the rewards that they will produce against the environment.



The agent accepts actions that are more favourable in yielding better rewards. All the encountered events and actions, including the calculated rewards, are stored in a knowledgebase. So eventually when the agent reached a matured state with high interaction of encounters with the environment, it would have accumulated the experiences and knowledge on the environment's states versus relevant actions and which of them can yield positive outcomes. By this stage where the agent has reached certain advancement or maturity, it will start to exploit or use the knowledgebase as compared to performing more exploration of trial-n-errors. So it is said that the agent balance the changes of choosing exploration of alternate actions versus exploitation of its knowledgebase is hinged on an  $\epsilon$ -greedy action-selection algorithm with a random number between 0 and 1. So for a given  $n$  number of interactions, the initial phase of the RL, the greedy algorithm favours exploration over exploitation since there is little or no knowledgebase. But when the number of environment's interactions increases, the greedy algorithm will decrease its preference on choosing exploitation and emphasize more toward exploitation of the knowledgebase as it has accumulated more as the number of interactions accumulates [39, 41-44].

#### **2.4.1. Markov Decision Process (MDP) – model-based**

Markov Decision Process (MDP) is model-based reinforcement learning. It models the environment in which the agent operates as a sequential decision-making problem. It has a tuple,  $(s, a, r, p)$ , which comprises of state,  $s$ , action  $a$ , reward  $r$ , and transition possibility  $p$ . An MDP must meet the Markov property, that is, the effect of an action taken in a state is dependent only on that state and not influenced by its history. The entities of MDP are described as followed [39, 44].

State,  $s_t$ , is the state of the environment at time  $t$  which may be observable or hidden to the agent. Action,  $a_t$ , is the action that the agent performs against environment, which is at state,  $s_t$ , at time  $t$  and then the environment yielded a new state environment,  $s_{t+1}$ . State transition model,  $p(s_{t+1}/s_t, a_t)$ , describes how the environment changes from a current state,  $s_t$ , to a new state,  $s_{t+1}$ , with the action  $a$ . The reward model,  $p(r_{t+1}/s_t, a_t)$  is a reward that the agent receives from the environment after it receives the action,  $a_t$ , when it reached time,  $t+1$ . Discount factor,  $\gamma$ , which controls the importance and influences of future rewards to the current reward [45].

The policy defines the behaviour of the agent's action to achieve the maximum cumulative reward over time. The learning task is to run actions in the environment to see the reward. Not to be influenced by its history, the learning policy is denoted as;  $\pi: s \rightarrow a$  where  $s$  is the state and  $a$  is the action [41].

There are two types of learning environments; the first one is the deterministic environment where both the transition and reward models are deterministic functions. When the agent repeats a given action for a given state, the new state and new reward are the same each time. This type of environment is easier to solve as the agent knows how to plan its activities with great certainty. The other environment is the stochastic environment where there is some uncertainty about the action effect. When the action repeats the same action for the given state, the new state and new reward may not be the same each time. This environment is both dynamic and volatile, making it harder to solve [41].

### Value Function

The state value function  $v^\pi(s)$ , shows how good is a state for the agent to be in. It is equal to the expected total reward starting from the state,  $s$ , and it is dependent on the policy that the agent picks the action to do. The function of this value function is denoted as followed [41];

$$V^\pi(s) = \sum_{i=1}^T \gamma^{i-1} r_i$$

Where,  $s$  is the state,  $\gamma$  is the discount factor,  $\pi$  is the policy,  $r$  is the reward,  $i$  is the iteration to all steps  $T$ .

For all the value functions, there is an optimal value function. The function is denoted with an asterisk,  $*$ .

$$V^*(s) = \max_{\pi} V^\pi(s)$$

To achieve this optimal value function, there will be an optimal policy  $\pi^*$ .

$$\pi^* = \arg \max_{\pi} V^\pi(s)$$

Apart from the state value function, there is another function called Q-function which pairs both the state and the actions as  $Q^*(s,a)$ . It is a normalized value that signifies the preferential in weights for the algorithm to select. So, when the agent starts from the state,  $s$ , and pick an appropriate action,  $a$ , based on the Q value and behave

optimally afterwards. It also indicates how good it is for an agent to pick action  $a$  while in the state,  $s$ . As  $V^*(s)$  is the maximum expected reward that the agent expects to start from the state,  $s$ , it will be the maximum of  $Q^*(s,a)$  for all possible actions. Therefore, both  $Q^*(s,a)$  and  $V^*(s)$  relationship is expressed as follows [42, 45] in eq(1);

$$V^*(s) = \max_a Q^*(s, a) \quad (1)$$

The optimal policy can be expressed after the optimal Q-function,  $Q^*(s,a)$  is known in eq(2).

$$\pi^*(s) = \arg \max_{\pi} Q^{\pi}(s, a) \quad (2)$$

The next important equation is Bellman's equation, eq (3), which is used in reinforcement learning especially in the dynamic programming domain, provides a recursive definition for an optimal Q-function.  $Q^*(s,a)$  is equal to the summation of reward after the agent performs an action,  $a$ , while in state  $s$  and the discounted expected future reward after moving to the next state,  $s'$  [45].

$$V^*(s) = \max_a [R(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) V^*(s')] \quad (3)$$

where,

$R(s,a)$  is the reward from the state,  $s$ , after receiving action  $a$ .

$P(s'|s,a)$  is the transition model for all states.

$V^*(s')$  is the optimal state value function.

$\gamma$  is the discounted factor.

Model-based learning such as MDP requires the agent to interact with the environment and attempt to approximate the environment state transition and rewards based on its history of interactions. When it has learned the model, the agent uses value-iteration or policy-iteration to find the optimal policy. However, there several challenges within the model-based reinforcement learning. One of them is that it requires the real-world environment to be modelled clearly. This task is difficult and prior domain knowledge may be required. Furthermore, transition state models or probability value is not readily available or difficult to define; therefore, they are hard to control. The next learning that will be explored is model-free training. Q-learning is one of the most popular algorithms [45]. Unlike model-based approaches, the agent in model-free learning will not try to learn the models of the environment state

transition and reward functions. It gets its optimal policy by interacting with the environment instead [39, 41-45].

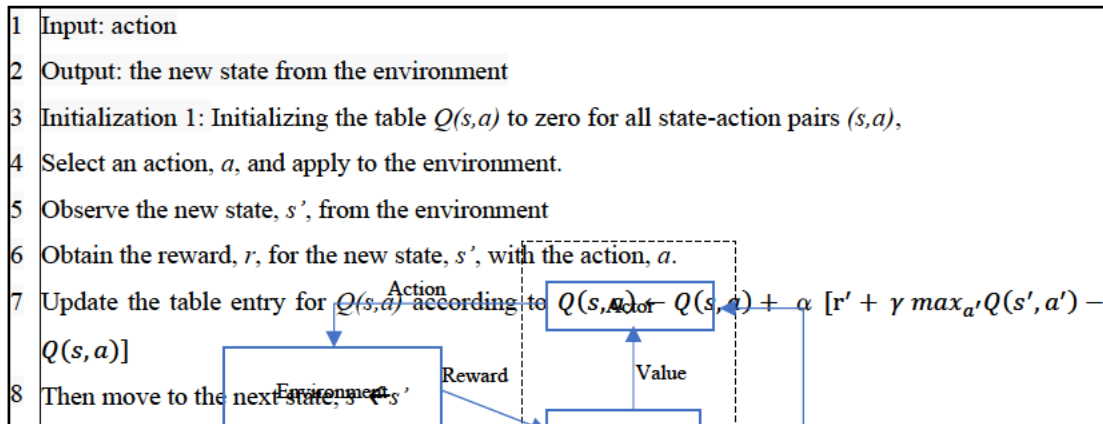
### 2.4.2. Q-learning – Model-free

Q-learning is a model-free learning algorithm that explores the environment and exploits the current knowledge at the same time through trial-and-error to find both good and bad actions. At each step, it looks forward to the next state and observes the best possible reward for all available action in that state. It uses the knowledge to update the action-value of the corresponding action in the current state with the learning rate  $\alpha$  ( $0 \leq \alpha \leq 1$ ). Therefore,  $Q(s,a)$  becomes the immediate reward and the discounted future reward. It is expressed in eq(4) [45].

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r' + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (4)$$

Where  $\alpha$  is the learning rate,  $\gamma$  is the discount factor,  $r'$  is the reward,  $\max_{a'} Q(s', a')$  is the expected optimal value and  $Q(s,a)$  is the old value. The algorithm is listed below.

Algorithm 1 – RL’s iterative process



However, the agent will need to choose between exploration and exploitation which has been mentioned earlier here it uses the  $\epsilon$  greedy algorithm and randomly chooses the action whether to explore or to exploit. The  $\epsilon$  value can decrease over time when the agent becomes more confident with its estimate of  $q$ -values [39, 41-44].

### 2.4.3. Actor-Critic Reinforcement Learning

Q-learning is a value-based RL and it has several drawbacks [46]; the first is that Q learning tends to have poor convergences if the value space is large. Any small change in the value estimate will harm the policy space [47]. For policy-based, it

derives the optimal policy through policy improvement from the evaluation of policies that are based on previous value functions. It is suitable for continuous and stochastic environments as it can give a good approximation of the value function. Its strength is that it works directly in the policy space, and it has better learning performance with new updates. For its drawback, it is susceptible to high variance and sample inefficiency, plus it tends to converge to local maxima. The Actor RL is policy-based, and it decides what action to take for the state. The Critic RL method is value-based and it tells the actor how good its action taken was and how it should adjust [47]. The Actor-Critic (AC) algorithm as shown in figure 8, combines the best of both methods, the actor's policy gradient method and the critic's value function method. So now, both the actor and critic complement each other; the actor produces the action for a given state from the environment, and the critic gives feedback and criticizes the actor's action.

Figure 8- Actor-Critic Reinforcement Learning

The critic can reduce the variability of approximation and provide an update to the actor's policy. The actors learn from policy and use it to apply actions continuously. Given that the policies are parametric on the state-action features, the AC model can work well in continuous state-action spaces that have feature vectors representation and they are used with Neural Network (NN) for their action and value function approximations [47, 48]. But Reinforcement learning by itself will not be adequate to tackle our research questions. Another method is required to perform the function approximation between the various state inputs from the environment to the anticipated actions and the preferred algorithm for this is a Neural network (NN) [49].

#### **2.4.4. Artificial Neural Network**

Supervised learning is another form of machine learning task where the learning maps inputs to outputs based on pairs of input-output dataset [50]. NN is supervised

learning that uses a large set of training data that comprise both normal and abnormal types with each data entry labelled accordingly. The neural network model can be visualized as a network of points or neurons that are organized together in different layers with data or predictor fed as inputs from one input end and the predicted results are produced at the output end which are outcome or labelled prior to the training. The layers of connection points or neurons in the centre are known as the hidden layers [51]. Figure 9 shows typical neural networks with hidden layers of neurons, also known as a feed-forward network [50].

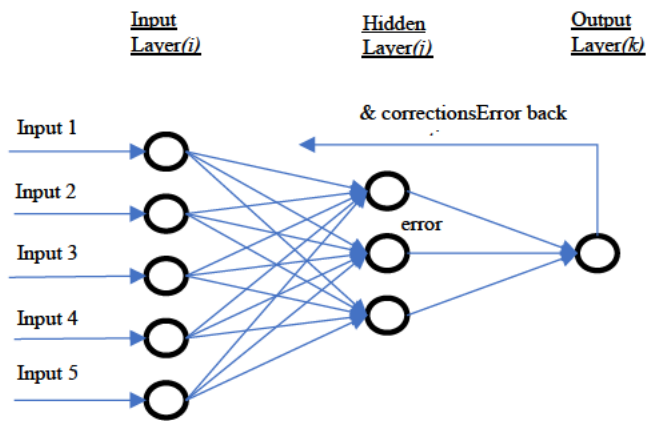


Figure 9- Diagram of a neural network with 5 inputs, 1 hidden layer and one output

The notions for the NN's components are.

$x_j^l$  is the input to node  $j$  of layer  $l$ .

$W_{ij}^l$  is the weight from layer  $(l-1)$  node  $i$  to layer  $(l)$  node  $j$ .

$\sigma(x) = \frac{1}{1+e^{-x}}$  is the sigmoid transfer function.

$\theta_j^l$  is the bias of node  $j$  of layer  $l$ .

$O_j^l$  is the output of node  $j$  of layer  $l$ .

$t_k$  is the target value of node  $j$  of the output layer.

The repetitive operation of each neuron takes inputs from the previous layers, each multiplied with a weighted value, then combined and modified by a nonlinear activation function and sent out as output [50]. So, for a hidden neuron in figure 10 below,

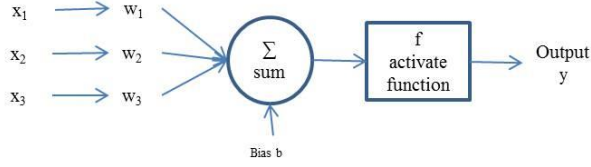


Figure 10 – Neuron's transfer and activating function diagram.

The combined function is in eq (5).

$$Z_j = b_j + \sum_{i=1}^l w_{i,j} X_i \quad (5)$$

Where  $z$  is the combined value,  $w$  is the weight,  $x$  is the input,  $i$  is the input number,  $j$  is the layer,  $b$  is the bias. The combined value is then changed by a non-linear function like a sigmoid as in eq (6) to reduce and prep it as input for the next layer[50].

$$s(z) = \frac{1}{1+e^{-z}} \quad (6)$$

The final output of  $y$  will be the trained result. However, the trained result will not be the same as the actual data as the NN guess with the current weights, therefore the weights will have to be adjusted to minimize the error. This is done by calculating the error followed by propagating the adjustment of the weights in the opposite direction as in eq (7). To do this; First, the error is calculated for a given set of training data  $t_j$  and output layer of  $O_j$  [24],

$$E = \frac{1}{2} \sum_{k \in K} (O_k - t_k)^2 \quad (7)$$

Then calculate the rate of change on the connective weight to minimize it in eq (8);

$$\frac{\partial E}{\partial w_{jk}^l}$$

So, for the output layer node  $k \in K$

$$\frac{\partial E}{\partial w_{jk}^l} = O_k \delta_k \quad (8)$$

Where  $\delta_k = O_k(1 - O_k)(O_k - t_k)$

And at the hidden layer node, eq (9); node  $j \in J$

$$\frac{\partial E}{\partial w_{ij}^l} = O_i \delta_j \quad (9)$$

Where  $\delta_j = O_j(1 - O_j) \sum_{k \in K} \delta_k W_{jk}$

So, in summary, the back-propagation algorithm is as followed [24]:

1. First, run the network forward with input data to get network output
2. Each output node computes  $\delta_k = O_k(1 - O_k)(O_k - t_k)$

3. Each hidden node computes  $\delta_j = O_j(1 - O_j) \sum_{k \in K} \delta_k W_{jk}$
4. Update the weights and biases as followed.
  - a. Given  $\Delta W = -\eta \delta_l O_{l-1}$  and  $\Delta \theta = \eta \delta_l$
  - b. Apply  $W + \Delta W \rightarrow W$  and  $\theta + \Delta \theta = \theta$

A typical implementation of the neural network is to train the NN with enough data so that it can recognize the new dataset whether they are normal or anomalous. one example of such an application is Brotherton and Johnson [52] who proposed the use of NN in fault detection, diagnosis and prognosis in military aircraft maintenance. Another popular use of NN was proposed by Mukkamala *et. al.* [25] for intrusion detection in networks by checking the users' activities for abnormal behaviour and usage patterns.

### 2.4.5. Deep Reinforcement Learning

For environments that have limited environment's states and actions, common reinforcement learnings, both model-free and model-based, can be used with the tabular function to define the Q (s, a) and estimate the return. These functions are usually tabular mapping of inputs and outputs for environment with discrete or well-defined states. But in complex environments where there is an infinite number of states, it is difficult to have a clear mapping function. Instead, generalization approach is used for this instance where function approximation is used for this type of large state space to approximate the Q or action values [40, 53]. Referring to figure 11, in DRL, such function approximation task is handled by neural networks. The neural network is trained with the accumulation of knowledgebase of (state, action) and then used to predict the Q-value where,

$$Q = \text{neural\_network.predict}(\text{state}, \text{action})$$

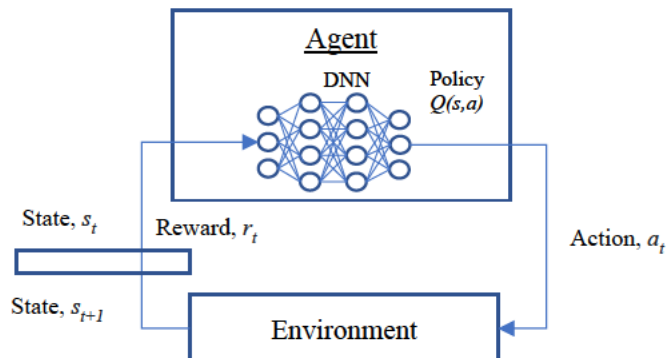




Figure 11- Deep Reinforcement Learning

The neural network used in the DRL will be based on deep learning with multiple hidden layers to capture the intricacy of the complex states to action relationship, and it can generalise the complex problems [40, 53, 54].

There does not exist a large amount of work on automated problem resolution. Farivar and Admadabadi [43] proposed the use of RL for controlling fault-tolerant on non-linear systems to ensure stability. The neural network model was used to perform Lyapunov's function to prove the stability of the equilibrium for the ordinary differential equations that are developed for the Fault tolerance control system, then use the RL to interact and learn to optimize the value functions to control the situation like a Cart pole game. This research is an excellent foundation to which this thesis referred [43]. Another paper was from Cao [55] who proposed the use of RL to identify the faults in the network and diagnose the root cause. The agent is comprised of a fault diagnosis module, a learning module, and a diagnosis knowledge base. The RL is used in the learning and fault diagnosis module which interacts with the network to identify the fault and reference the fault back to the knowledgebase to extract the root cause problems which in turn are delivered to the users via the user interface.

## **2.5. Research Gap and Summary**

There is a substantial body of knowledge in the domain of anomaly detection, fault diagnosis as well as other machine learning models that have been briefly covered in the literature review. Methods such as SVM and ANN have been used to detect intrusion (or anomaly) in a complex environment such as computer networks, crowd movement and IT security. These machine learning models have been used to develop intelligent expert systems in the engineering and medical field in anomaly detection and prognosis. However, this research has applied the machine learning models to very specialized and complex environments, be it medical, IT or engineering. Most of them achieved great in-depth analysis in the subject matter and develop bespoke algorithms to meet the challenge; however, major changes and customization are required to adopt these anomaly detecting algorithms and processes to a multi-tier data replicating environment (DRE) such as this.

The DRE itself has several software modules and each of them have their own unique operating functionalities, operations, and parameters. To establish a proper operating setup, each of the modules has specific expectations or dependency on the adjacent ones. This complexity has introduced a level of challenges that most of the literature on anomaly detection has described multi-source events, but not for software such as DRE.

Another challenge lies in the task of pinpointing the exact root cause of the defects that may occur in the DRE. In its course of operation, there will be anomalies appear among the events and finding the real defect among the noise of anomalies prove to be difficult. While the literature on multi-view data may be compatible with this requirement, it is still difficult to define all the conditions which can pre-mediate the default cause and the fault associated with it. the gap here is to bridge the complexity between the DRE's potential anomalies, defects, and faults. That is a task which this thesis needs to undertake, to formulate an aggregating routine that can acquire the anomalies and deduce the defects, followed by faults systematically. The whole routine will have to be assessed and tested rigorously to affirm its operating expectations. One potential model that can assist in this setup is deep learning. Recently neural network-based deep learning methods have become popular to learn from a wide variety of data and it is a powerful and popular model to be used for fault diagnosis in many fields such as transformers, oil refineries [5, 10, 56].

This study is intended to model after organizations such as Deepmind [57] and OpenAI [58], where they build an intelligent system based on deep reinforcement learning (DRL) to tackle games with extremely high computational complexity such as Go and Multi-player battle arena. These projects only surface in recent times and although DRL theory is not new, such implementations are. As such, in the space of FDR system and DRL, there is currently limited research or project in development. While the candidate is confident that this type of project will take off eventually and reach the mainstream, it is the data replicating environment (DRE) which he has an immense interest in to support and therefore, take this opportunity to apply this domain of RDL to the tackle the fault diagnosis, detecting, and resolution of the DRE.

So how can this thesis go about defining the state and action for the data replicating environment? There is limited research conducted to date in the usage of

DRL for software fault diagnosis and resolution particularly in interconnectivity among enterprise relational databases. Furthermore, there are not many types of research made in the domain of data replication, particularly around ETL. While some research papers discuss the method of optimizing workflow, there is no precedence on software such as near-real-time data replicating tools like Shareplex.

Some research can place the causal relationship between defects and faults through decision trees, but this is feasible if the system's scope is simple and well-defined. If we have an environment with extreme high complexity that cannot be defined, then a more dynamic approach is required. The relationship of defects to faults and to resolving actions are complicated and may not be accurate in the first version, and constant revisions or corrections are required to recalibrate them.

To define the state of the DRE model, the initial design set the model to accept hundreds of system-wide defining inputs as variables in the DRE, and with these inputs, a strategy is required to build up function approximations on them to deduce the potential defects. Likewise, for the defects to the hundreds or thousands of potential actions that may be required for both statistical information query and real system corrective actions. It is important to note that some of the DRE's defects are usually inter-related, some are compound, or hierarchical whereas others are isolated. This is the type of complex condition that the candidate faces and there is no existing research available for him to refer to. A new strategy must be methodically designed to address this gap. Furthermore, resolutions for the faults are not straightforward and it will require multiple cycles before it can reach the final resolution. The proposed system must be able to reiterate to its best ability to search for a solution before giving up to the IT administrator for help.

To the best of our knowledge, this is the first research to utilise deep reinforcement learning for fault diagnosis and resolution for database replication technology as there is no precedence of similar work in this fields.

# CHAPTER 3: RESEARCH METHODOLOGY & DESIGN

This chapter presents the research methodology with the proposed phases, and the design of the FDR system.

## 3.1. Overview

This section detailed the research design and methodology which is illustrated in the different stages as shown in figure 12. The researcher has completed the phase of data analysis and the exploration of using the reinforcement learning model for fault resolution by the time of the confirmation. The datasets used are obtained from the real-world Shareplex and Oracle's setup within Energy Queensland Limited's IT systems. In the next phase of this research, we propose to develop a novel methodology built on model-free reinforcement learning to learn from the IT operational information and interact with the system to identify the faults and find the best possible actions to resolve them.

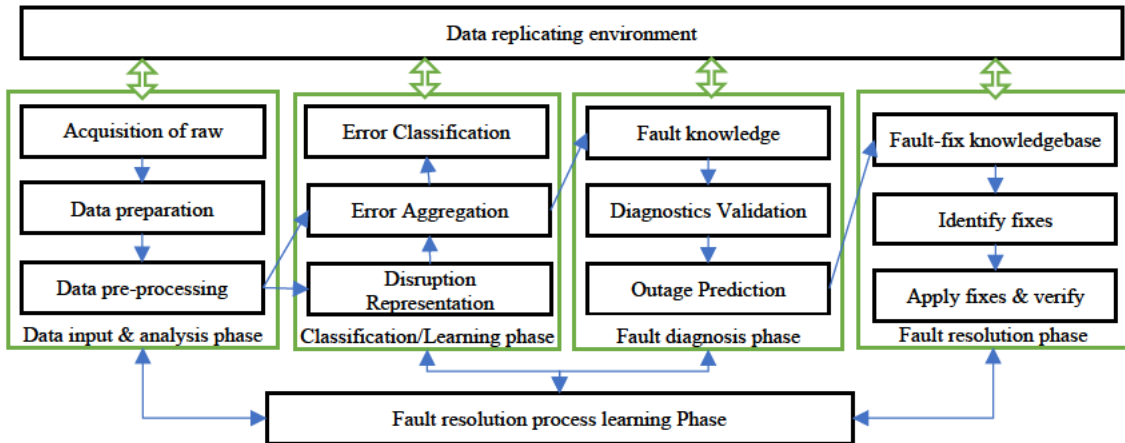


Figure 12 - Research plan

This research starts with the concept planning and design phase of the theoretical self-diagnosis/resolution model. Knowledge about fault diagnosis and detection will be referenced to the research in the field of process fault diagnosis made by Venkatasubramanian et al, 2003 [4] and anomaly detection made by Chandola et al, 2009 [16]. A set of expected features will be listed defining what is expected from this proposed system. This list of features will serve as the guide to establish a series

of key success factors. These factors will design the goals and deliverables that each following phase in the research process must be met.

### **3.2. Data Input and Analysis Phase**

Followed by the establishment of the architectural design of the proposed model, the next stage is to analyse the data that need to be managed. This is an important phase as all the sources of data pertaining to the various DRE software's operation, integration, configuration and inter-operability requirement are discovered. This is in preparation to meet the research question 2 and 3. In this phase, only analysis of the target systems that need to be managed will be conducted. This is to gain an insight into the intended managed system's modes of operation; their configuration, activity and status information output, interaction as well as a series of operational functions that normal IT administrators would do regularly. A review of EQ's database team's operational procedures and manuals will be conducted to gain an insight into their IT administration work, together with the documented faults that had been encountered and as well as the resolution techniques that had been used. This will give the research an initial application on the landscape of knowledge before formulating the attack plans.

Each domain of IT systems will generate a wide variety of data that have different functions and attributes. There are challenges in how to manage the data and the types of expected outcomes that should be required. While the data cleansing process here will be limited, they will be done to clarify those data that will impact the research. The next stage is to analyse the data meaning and their implication to the impending system that this proposed model will manage. Referencing the fault diagnosis literature [6], some faults can be easily identified through simple univariate anomaly detection, whereas others will require multitudes of data to support the diagnosis. In other cases, there will be incomplete data that require further investigation against the managed system via upstream, downstream, or adjacent in the fault-finding process. On some occasions, shifting to a different domain to gather more information will be required but that will require the learning phase to conduct trial-and-error information gathering. The information gathered will be categorized following their groups and hierarchies; what is classified as system, functions, operational process, expected outputs, potential faults, root cause, and resolutions

technique. An initial graph database and a spreadsheet will be used to map this knowledge. This forms the basis on which the next phase of classification and learning will refer to.

Question.

1. What privileges does our agent require to access the relevant files under various directories that belong to different software owners; Shareplex and Oracle?
2. What are the output files that are required and how will the agent go about finding them repeatedly with minimum error?
3. What kind of content do each of the log files have and how can we gather features from them?
4. What type of method can we use to ensure that the patterns of information recorded in these files are consistent?
5. What other information can we gather from the OS environment and how can we correlate them with the details gathered from the logs?

### **3.2.1. Environment Dataset**

The required data will be provided by Energy Queensland's database department. There will be a mixture of raw data; from systems' log files to the streams of time series data that will be obtained from the production IT systems. There are different varieties of data in this raw output and logs, so there will be an ample set of testing opportunities to be practised on them. The researcher has obtained the approval of using this data. There are two avenues of obtaining information about the target software; Shareplex and Oracle, and they are in explicit and implicit forms. For the explicit form, it is by reading through the event and alert logs that are generated by the software progressively, logging every event, information, error, or alert. The implicit form is done via a persistent search for information through interaction with the software to acquire their statistics. Such activities involved here are by the usage of OS commands or through the software's console such as SQLPLUS or SP\_CTRL. One of the challenges here is that we need to know the exact commands to invoke so that it can query the appropriate information or statistics that the software will respond to. The information or statistics that are returned cannot be interpreted readily and require in-depth background knowledge before their meaning can be derived. For the Oracle

database, the common alert log is named as alert\_<Oracle\_SID>.log and they are located at \$ORACLE\_HOME/diag/RDBMS/ /PROD/trace. For the Shareplex, the log file is called Event\_log and it is located at \$VARDIR/log.

### **3.2.2. Evaluation Procedure**

This is progressive research, and each phase has significant importance to the next one, so accuracy in building and testing the proposed methods in each phase will be imperative. The evaluation process covers the breadth of the research project in terms of independent and inter-operational process capability, as well as the depth of the specific ML models. While the research wants to mimic the entire fault management process that the IT engineers have, it is open to the research on redesigning or optimizing the process flow should the need arises. For all the ML models that will be used, the evaluation process will cover the following aspects.

Test harness occurs at the initial stage, and it involves the acquisition of data that represents the problem which will be used as both training and testing set for the ML algorithms. it prepares the foundation of each ML formulation by assessing how learnable the problem is through analysing the data, and whether the data structures are suitable for the models and if not, how much transformation will be required to alter them.

Performance measurement covers the evaluation of the solution to the problem by calculating the prediction made by a trained model against a test dataset. There is a list of performance measures available, and they are generally relevant based on the type of problem and solution required, be it a classification, regressing or clustering. Some give a generic score; others can give a more meaningful answer to the problem's solution.

Cross-validation is also part of the test harness, and its approach is to split a dataset that presents the problem environment. The training data set is used to train the model whereas the testing data set is for evaluating the performance of the model.

Evaluating algorithms. Once the problem has been defined and the test harness is prepared, the next stage is to shortlist a series of ML algorithms that will be relevant to the problem; most of the ideal models have been mentioned in the literature review. These models will be trained and tested against the dataset prepared at the test harness

stage and their results will be measured and compared with one another. An assessment will be made on the chosen model against the problem environment to determine if they are relevant or practical. Each one of them will be also measured in terms of performance using the ratio of both loss and cost functions.

The next section describes the evaluation methods that need to be performed against each phase that has been designed in this research.

Evaluation of data input and analysis phase - The requirement is to assess the ability of the pre-processing routines are in removing data inconsistencies and redundant content as well as the quality of the final processed data. Each log and data input, e.g., event\_logs, alert\_log, syslog and system stats will have their unique features that need special consideration to clean out the unwanted or erroneous information in preparation for the analysis phase, so the measurement here is to gauge the completeness of data cleansing that this phase can cover. The routines will be developed against the sample data from the IT environment and once the logic has been established, testing data set from another batch will be used to validate against the routine to test its efficiency as well as its effectiveness. Each batch will comprise information from the enterprise combination of the software systems, e.g. Shareplex, UNIX, Oracle DB, with different varieties. This will be tested in several batches to ascertain the thoroughness of the pre-processing routines whether they can handle a bigger load of data with a wider range of errors.

Evaluation of classification and learning phase - in this phase, the test is to evaluate how relevant the result from the phase of data re-working via reduction, conversion and symbolically altered, is to the DRL downstream. the data from various DRE's sub-systems (OracleDB, Shareplex, Unix, network) must be evaluated and tested via the DRL agent and validate the result. Normalization is included in this process which changes all the respective hashed value of the error message and helps to speed up the DRL's NN convergence process. This phase is tightly connected to the next phase. The measurement of the NN's accuracy is also related to this.

Evaluation of fault diagnosis phase - the next phase is to test the fault diagnosis routine on its ability to use the features and anomalies acquired to identify the root cause. This will be achieved by training the fault diagnosis model by using a labelled dataset to train and test like the classification phase above. the model's accuracy can



be measured using a ROC curve and confusion matrix, both will show the performance of the ML models against the set of test data. Also, in this phase, the routine will need to work with the knowledgebase which will be stored in a database. The fault diagnosis routine will be evaluated on its ability to interact with it in a timely and accurate manner, matching the classified faults that have been identified by the model to the knowledgebase and extract the root cause information.

Evaluation of faults resolution phase - In this phase, the reinforcement learning model works with the fault diagnosis model; the latter will feed the information to the RL model and that in turn will interact with the system or known as the environment. The RL model will be assessed on its ability to find the optimum solution; the number of iteration and the results that the applied actions from the RL model can yield. While the evaluation here does not focus on speed and accuracy at the beginning, it is still a major factor that needs to be observed in the long run. The main critical success factor here is to interact with the next process learning phase when a certain number of iterations has been exhausted and the optimum solution is not available. at the same time, new knowledge will be acquired for the knowledgebase from the external interface, so it is expected that once there is an accumulation of new knowledge, the RL model should have a greater chance of resolving the faults through less and less iteration. So, the measurement here is gradually, and the expectation level will increase over each epoch of trial-and-error-and-learn.

Integration/Process learning phase - This is the governing phase that controls the fault resolution phase with the users. Once the RL model has exhausted the moves in the knowledgebase, the IT administrator or user will have to assess the problems and input their expertise into the system, thus enhancing the knowledgebase and its capability. The measurement here is on the workflow that will be built into this phase model, which can allow the interaction between the system and the user. As each knowledge that has been entered by the IT administrator, this phase's model must transfer the updates accurately back to the knowledgebase so the RL model can refer to it without any hindrance. Both the evaluation of this phase and the fault resolution phase are interleaved.

### **3.3. Classification And Learning Phase**

After establishing the knowledge about the managed systems' behaviour, the next task is to identify the features that will be required from the raw data to support the fault diagnosis and define what is considered normal or anomalous. Referring to research paper on the survey of anomaly detection [16], some anomalies can be regarded as point anomalies that can be detected easily, others will be more difficult to acquire namely the contextual and collective anomalies. So, based on the faults listed in the survey from the previous phase, the raw data feature to support the diagnosis will be decided. For the classification and learning phase, the impact is to acquire the knowledge of the DRE's software service services from different aspects of each software' sub-systems; DB's component listed in the DB architecture diagram must maintain a certain level of error-free activity to serve other adjacent DB's sub-systems to serve the overall DB's function. The DB's function is, in turn, serve as a sub-system, integrated closely to other software and IT technology to support the DRE's mission, in a hierarchical arrangement of software and technology dependencies. To know the different state of the DRE's software, there are various sources that the information can be acquired; through 1) event, trace, or diagnostic logs, 2) statistics acquired from software's support console, commands or utility, 3) extracting features from other sub-grouping of statistics. They reflect both point and group anomalies across hundreds of parameters and variables. They are then aggregated into logical grouping at the service level of the individual software's sub-system such as connectivity, security, operation status, process control etc. This derives a unique matrix that represents the DRE's entire service level structure, indicating holistically the logical group and their respective service level so an IT administrator can briefly know the nature of the anomaly and where they have been affected. Most of the anomaly that occurs within the tightly integrated software in DRE will not be isolated but have a ripple effect that impacts others. This matrix of service anomalies can capture and illustrate the landscape of service anomalies encountered. This information will be logged into the research which will then be used to support the next phase.

### **3.4. Fault Diagnosis Phase**

Using the information and results acquired in the classification/learning phase, the next phase is to map the different knowledge acquired to the identified faults that are listed in the survey conducted beforehand. This will enable the research to establish a hierarchy of managed system's operation, faults encountered, root cause, and source of faults. Referencing Chen and Patton's model based on fault diagnosis [59], the research here will build a fault diagnosis model based on the information and knowledge acquired through the previous few stages. The proposed fault diagnosis sub-model will be expected to support some basic fault diagnosis initially and it will be augmented progressively to handle a wider range of other faults. A series of validation tests will be conducted on this model to assess its accuracy and the logic used to diagnose the problem. The result will be studied, and corrections will be made to adjust or change the structure. Once the fault diagnosis sub-section is complete, the next step is to develop the fault resolution part. This phase covers the decision to the class space under fault diagnosis's hierarchy method, and the challenge here is to handle different types of faults. Therefore, the main challenge is to customize the concept of DRL in deriving both the DRE's environment state and action in conjunction with the concept of DRL. However, there are some customizations required with regards to the policy-based approach of the model-free DRL, using an alternate set of sub-algorithms to complement it.

#### **3.4.1. Defining Data Source for System Anomalies**

The initial phase learnt the entire setup of all the software modules that operate inter-dependently; the configuration that is required to be set up in each of them before they can start operation. Keeping the data replication process going between the databases and the Shareplex requires an in-depth understanding of the construct of Shareplex's requirement and its demand for the database's operation so that its data replication can function at an optimum level. Apart from getting the entire environment to work, there are several aspects of these software modules that give the candidate insight into the internal operation which includes the information, statistics, and errors that they encounter throughout their services. These specific tasks have been identified as followed.

1. From the text logs and system statistics of both Shareplex and Oracle, machine algorithms such as text mining and anomaly detection are applied to detect faults and errors.
2. Classification of text between normal and erroneous.
3. Anomaly detection of erroneous sessions and process among the client sessions' and system backgrounds' attributes.
4. Classify each type of detected error into different categories of IT subsystem in a hierarchical order.
5. Data involved are mainly text, time series and scalar.
6. Error detected can be point and contextual. Point anomalies are easy to detect as they are an outlier. Contextual errors are harder to find.
7. Time series data in utility companies are ubiquitous among all business operations and functions in Energy Queensland, its fundamentals and must be understood including current practices.

One of the main challenges in this thesis is to define the environment of the complex multiple software systems that not only need to operate cohesively but can function at the optimum level with a minimum adverse effect on the overall data replicating functionality as each software has its own set of parameters, operating commands and control, performance statistics, processes, output logs. It is needed to determine the list of events that are important to this software from which they are also dependent on by the other software. Event logs' outputs can be determined explicitly and implicitly through constant mining of the logs or run certain commands against the software to acquire the statistics. However, not all events from a system can be correlated to the events on another system. A complex mesh of relationships will have to be determined in categorical form and in a hierarchical structure which will be covered in the next phase.

So, for each event, there will be a threshold or limits in which their values are regarded as either normal or anomalous. If the specific event is deemed anomalous, it must have an equivalent diagnostic process that can acquire the necessary statistics and then based on them to decide and define the real cause of the problem. Based on causal models, some of the simple root-cause problems can be determined straight away, others require inputs from multiple sources before the root cause can be

determined. Once the root-cause problem is determined, the next step is to find the possible action that has the highest chance of resolving the problem.

There are several approaches and one is to build an expert knowledge-base of problem-actions so that the system can refer and apply the appropriate actions, but this is not the case in reality as there are numerous environmental factors that increase the difficulty of this research, namely; 1) The constant changes in all the dependent software in term of a software upgrade, control parameters, command syntax, statistics value alteration and, 2) the list of problem-action combinations is extremely large and dynamic.

Therefore, this thesis builds the system to be adaptive so that it can learn to observe what actions is the most appropriate for a given state through trial-and-error, plus it should also learn in collaboration with a human IT administrator. His/her knowledge and experience will help the system to build up its knowledgebase and at the same time, a source for it to seek external help if it has exhausted its entire means. This begs the following definition questions.

1. The environment of the setup.
2. Environment's state.
3. Action for the environment.
4. The reward for the action is applied to the environment.
5. Is it a single goal, multiple goals or a hierarchical set of goals that need to be accomplished before the final goal is achieved?

### **3.5. Faults Resolution Phase**

Referring to figure 13, fault resolution is a process where the solution is applied to fix a problem and this phase covers the intelligent agent that will interact with the environment to find and apply the optimal solution. Based on the fault diagnosis' results, the proposed series of solutions can be identified. The agent will reference the diagnosed faults and prescribe a series of solutions to rectify the faults. The process does not end there as the model will check the actual results and compare them with its knowledge of the expected outcomes. If there is a discrepancy between the actual and the expected results, this is classified as a miss. The model will be expected to route back to the starting phase and work through a different route of data analysis;

feature extraction, classification, fault diagnosis and resolution phase again in the attempt to resolve the problem. There will be intelligence to control the different phases and it will learn through trial and error, the optimum strategy to solve a specific problem. Reinforcement learning has been identified as the superior model as compared to the rule-based expert system. Once the agent has achieved in identifying the optimum solution and fix the problem, it will update its knowledgebase for future reference. There will be a need for user interaction in the event if the model is unable to resolve the problem, then external human expertise is required. The IT administrator will intervene and work through a new way of resolving the fault, then teach the proposed system about the new set of troubleshooting information which in turn will re-strategize its diagnosis/resolution capability.

Like the previous chapter, the main emphasis of the algorithm in use is DRL which are supported by contributions from paper 4 and 9 that dwell highly on the use of DRL for database and data replication problem. The research and experiences gained are critical to the success of this phase as well as the previous chapter. While the papers use the DRL as standalone agents to meet the research challenge, in this chapter, it is a double DRL agent implementation so knowledge in deploying the DRL for different scenarios is important. More experiences have been acquired through paper 5 and 6, each dealing with a different research problem scenario, and DRL has been modified to meet the need.



Figure 13 – Faults Diagnosis and Resolution (FDR) workflow

### 3.5.1. Development of Fault Diagnosis and Resolution Process

The process of resolving problems and faults in the data replicating environment (DRE) can be described in the following Figure 14. Throughout the system operation, there will be events that occur and while most of them can be non-critical or erroneous based events, there are occasions where anomalies that may occur. However, not all anomalies are defects, and most may occur that could be out of normal behaviour either on a temporary or semi-permanent basis. For example, the month-end processing will see a spike in OS resource usage and will return to normal. Others will have unusual massive batch updates that can cause some backlogs in data replication and that will

clear up within a short period. But there are occasions where anomalies are genuine defects that have surfaced in the environment and that sort of defects are hindering or preventing it from performing its designed function. Therefore, identifying the actual faults to the defects in the next stage which requires intrinsic analysis to determine the exact reason why the default occurs and where the fault lies. The next stage will be the resolution to the fault. Finding the best possible method to resolve the fault is another domain of knowledge that requires a vast amount of product familiarity and systematic logic of problem-solving skills, which are at times, difficult to pinpoint and identify unless a very skilled IT administrator that has years of experience in this area of specialisation can resolve it.

Here is one example; an event has surfaced where the backlogs of one of the post queues are building up. This event is highly unusual, and it is defined as an anomaly from the usual system functionality. It is now classified as a defect and requires a more detailed investigation. The defect found in the DRE is the inability to insert new data, and the fault that lies with this defect is with the permission issue on the database. Each entity bears the many-to-many relationship to the next and this adds more complexity to the thesis' scope of the challenge.

Applying this knowledge to the proposed DRE fault resolution model in combination with reinforcement learning, two streams of decision-making workflow have been defined. The first one is based on a simple or direct goal where the a priori faults-resolution knowledge has been well defined. The next one is of the complex structure where 1) there is no a priori knowledge, 2) need to perform some trial-and-error tests to assess the environment's state versus the action, 3) start to learn from past mistakes and refine the learning path towards an optimum resolution. The following figure 13 describes the detailed steps involved in the simple and complex goal approaches.

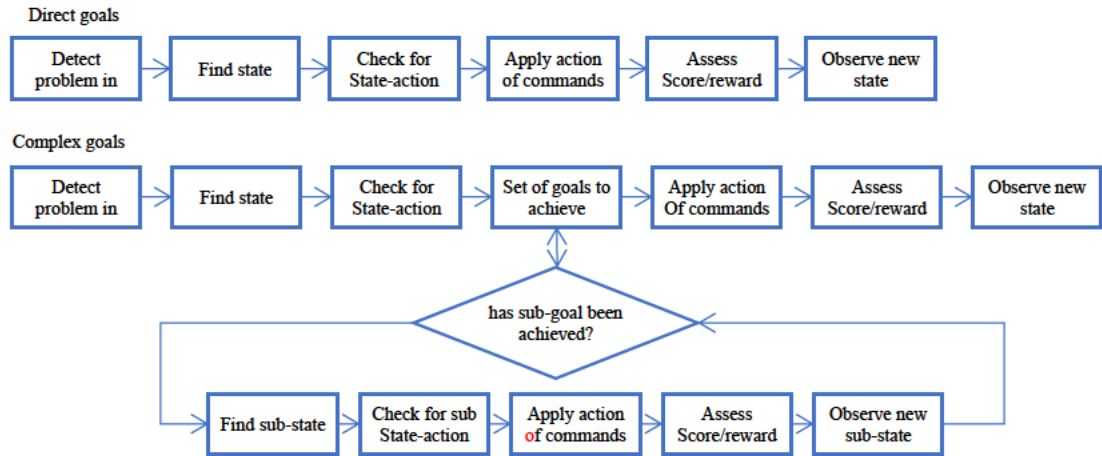


Figure 14 – Direct and complex fault goals’ resolution workflow

The complex fault goals workflow requires an iterative loop to determine which possible actions can be used as a suitable method to resolve a problem. But it requires grouping and subgrouping of actions under similar categories to follow a logical path of action that is relevant. For example, if the group is under OS’ memory, then it is appropriate to have OS commands to acquire the shared memory, virtual memory and utilization within the OS instead of checking permission for the filesystem. It is required to define the possible classes and sub-classes of items. Figure 15 below outlines the entities involved in the DRE, the respective software systems and their underlying parameters, statistics, objects, and information that are required. From there, a map of possible actions can be devised for each item to query their statistics for more information or, to change the configuration to resolve the possible faults. Below is an example of the rules of actions in place to find the presence of an oracle.exe process, and if there is no such process present, another action is executed to bring it back online.

Example.

Rule1 - Presence of oracle.exe process -> `ps -ef|grep smon|wc -l` → 0 (good) or 1 (bad)

Rule2 - For the bad state, corrective action will be → `restart process ($ORACLE_HOME/bin/dbstart.sh`

Rule3 - Performs action 1 again and set the flag of action 2 to “flag#1”

When rule 2 runs rule 1 and achieves 1, the problem is resolved, else the problem persists. Else, rule 2 cannot re-run rule1 since fault persists, so it should choose another rule or alerts the administrator. Another point to note is the use of forecasting, it is



important to predict system's resources time series over a period is required to determine if there is any probability of time series' trend, season and cycle of the Unix-Shareplex-Oracle behaviour through an extended period; 1) Is there a trend that it will surge or dip toward a certain period? (Weekend has fewer activities and will see a dip. Whereas month end's financial processing will see a surge). 2) Is the sudden change in the time series normal or anomalous? DB utilities such as Automatic Workload Repository (AWR) can produce similar outcomes. Other knowledge such as Symbolic Aggregate Approximation (SAX) is important for comparing time series on a contextual basis. Point values are neither efficient nor effective. The contextual representation allows quicker detection and able to view that specific range of time series from a holistic point of view. Four papers have been written concerning time series; two of them use various forecasting algorithms on time series against database technology while another two works with the time series directly.

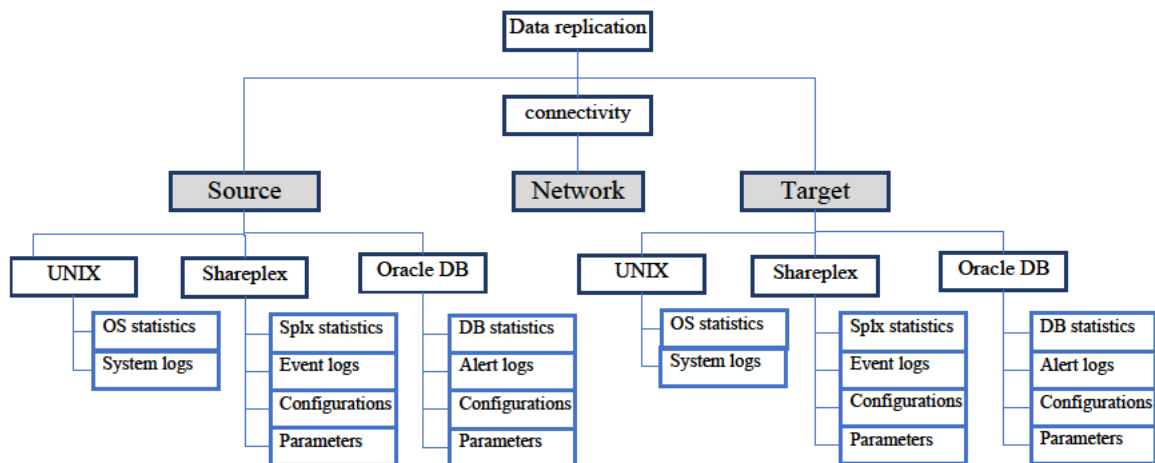


Figure 15 – Hierarchical relationship of DRE's software systems and their components

### 3.6. Integration/Process Learning Phase

In this phase, the research is focused on the module that will perform the intelligent action to evaluate and redesign the process of fault diagnosis and resolution for specific incidents. It is known that the best set of troubleshooting rules do not fit perfectly into a real-world complex environment. Experienced IT administrators adapt and change their strategy when they face difficult problems and especially exhaust the recommended ways of dealing with them. The radical decision is required to re-appreciate the situation and use references from other troubleshooting techniques to build on new solutions. So in this phase, the research will have to devise a routine

using RL [44] that can be flexible enough for the proposed module so that it can use its learning capability to re-strategize the fault resolution techniques. Such techniques will require the reactivation of new tasks of data acquisition and knowledge acquisition to build up a different perspective on the global context. It will require the module to backtrack, reiterate, side-step, switch steps or mix-match across multiple steps among different troubleshooting stages. There will be a human interacting part when the proposed system has exhausted all the means within its knowledge base, external human intervention will be required. When the IT administrators step in and resolve the previously unfixable problem, they will update the knowledge base so that it will know the new fixing methods and be able to perform a similar task when it encounters the same problems soon autonomously.

All the research and experiences gained from the papers have a direct contribution to this chapter. So apart from the main knowledge that has been acquired; DRL, system faults, NN, databases and data replications, other supportive areas such as experimentation setup, test result analysis and development strategies are equally important, and they contribute much to the success of this thesis.

### 3.7. Develop an Intelligent Fault Resolution System

So, based on the previous learnings from research and other publications, a new autonomous fault detection, diagnosis, and resolution (FDR) model for the data replicating environment (DRE) is proposed and depicted in figure 16 below. As an overview, the environment in the diagram is the complete setup where the individual software systems' component generates out their information and that in turn is consumed by the anomaly detection modules which have been specifically designed to handle the respective software systems' events. Once the anomalies are detected, they will be checked to verify if they are genuine faults or not. This, in turn, is fed into the RL agent where there are several series of deep neural networks (DNN) to decipher the state, diagnose the root cause of the faults and identify the possible resolving action for it. that, in turn, will be translated into actions that are meant for specific software systems.

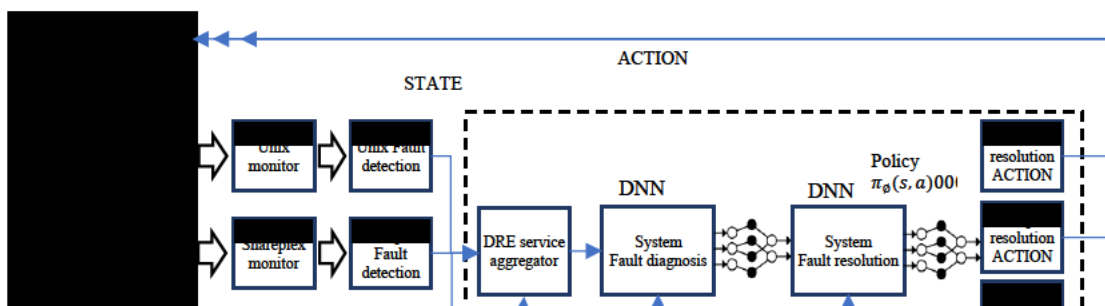


Figure 16 – DRE’s Fault Diagnosis and Resolution (FDR) System model

The actions are applied to the environment and a rewarding channel will send the feedback signal back to the RL agent. If the action can solve the problem, the reward will be positive. But if the action is ineffective, then there will be no reward granted and the RL agent will have to explore other possible actions. The overview is now described in more detail as shown in figure 17. the entire model can be segregated into six groupings, and they are.

Environment – this is the data replicating environment (DRE) that comprise the software systems; Unix OS, Shareplex and Oracle database.

Raw data groups – these are the information that the software produces explicitly, statistics that can be mined implicitly or they can be obtained externally from other sources like user inputs. They comprised a series of different datasets that span from time series, exceptions, log entries and statistics that have been acquired through processes that run continuously to mine the results from the system through the console or by command programs.

DRE service aggregating module – this is where the data obtained from the environment are aggregated and parsed into the service level group. A routine that comprises many custom-built scripts that query each software to get their statistics and read their various logs for anomalies. They are then transformed, changed, summarised, and reduced into values of presentable value ranges, and this is in turn used in a matrix that can represent the entire DRE’s environment state.

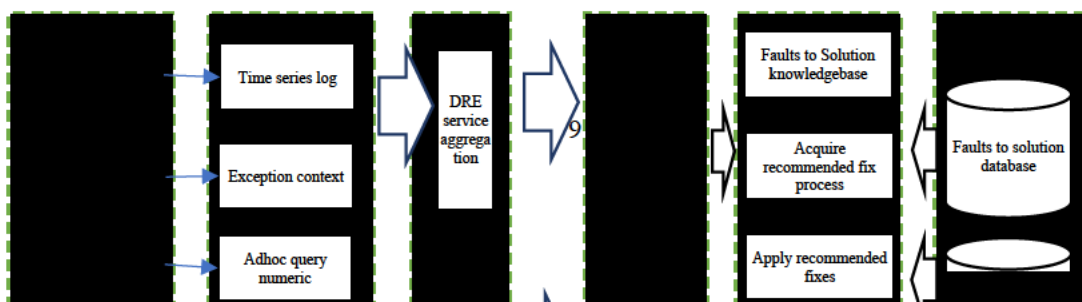


Figure 17 - The detailed process of the FDR model

Fault diagnosis module – from the inputs that were derived by the previous stage, this module will compare it through its knowledgebase to identify the specific possible faults that are associated with the errors. It has several diagnosis models to ascertain the exact faults based on the anomalies discovered: be it a point, contextual or collective.

Fault resolution module – this module finds the best possible actions to mitigate the faults and what it will do is to refer to the knowledgebase of faults-solution for guidance. If there is no a-prior knowledge, it will invoke the RL agent of FDR for trial and test to find another solution. At the same time, it keeps the score of its decisions to manage the processes. Should the number of tests exceed a threshold, it will notify the IT administrator for assistance.

Integration of Fault diagnosis and Resolution module – this combines both the FD and FR to form a complete FDR system. It is expected that the system can function autonomously to detect and resolve any system-related problems on the DRE, building up its knowledge, learn what possible actions can be used to resolve certain faults and establish a feedback loop from it to decide if the actions are effective or not in solving the fault.

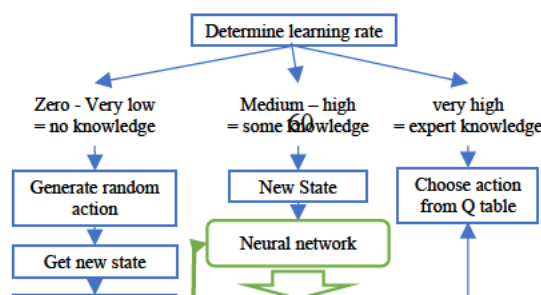


Figure 18 - Different learning stages of RL agent

### **3.8. FDR's RL Agent Learning Process**

The RL agent starts with little or no knowledge about the environment and it has only equipped with a basic reference of a list of actions that are grouped into their respective categories under the software; OS, Shareplex and Oracle. As shown in figure 18, the fault resolution process is a progression of learning that is grouped into 3 stages in which the agent functions according to the availability of knowledge about the environment. The following describes the stages in greater detail.

Non-to-low knowledge: The RL agent starts with the premise that it has no prior knowledge of the data replicating environment (DRE). What it has is a basic guideline of general faults' categories and the associated actions of software's console or commands to operate. What it must do first is learn through trial-and-error with random actions to test if the actions can be able to resolve the faults and the feedback that it receives. Through the initial learning state, the RL agent builds up its knowledge base about the cause-and-effect between the states, actions, and rewards. It builds up the Q-table that sets the Q-score among the state, actions, and rewards. learning and exploration rates are two control that decides the probability of performing actions that are within the agent's knowledge base versus the chance of trying something random. For example, the learning rate, it is set at a very high value of 1.0 at the start and force the RL agent to explore all means and ways. But this rate will be decreased slowly

over time using a decay factor of 0.995 until a threshold of 0.4 is reached. By the time it has reached the 0.4 value, it also means that the RL agent has gained some experience to do more exploitation of its build-up knowledgebase and can rely less on exploration. But it still leaves some room for exploration though, but the chances will be much lesser.

Medium-to-high knowledge: Once it builds up an initial knowledge on the environment's states to various actions, the candidate applies the Q-learning algorithm to fault resolution technique by building a deep network neural (NN) model that predicts the possible score and actions for a given state. The NN model is trained with the already built-up knowledgebase to find the best reward and action for a given state from the environment. At this stage, there may not be enough past data to train the NN model accurately and so the trained NN model may not give accurate predictions in its first few attempts, but the agent will run the NN model repeatedly over several iterations internally until the predicted rewards reach an acceptable threshold, after which the best NN prediction is sent to back to the RL agent. The agent will validate the prediction against the data replicating environment; if the predicted action achieves positive feedback or reward, it will log it as a Q's tuple of state, action, rewards in its Q-table as well as adding the new information to its knowledgebase, enriching the dataset for the next round of NN training. But if the prediction is not accurate, the RL agent will correct the information and do the knowledgebase's enrichment. The agent will go back to repeat predictions and validation until the reward meets or exceeds the desired threshold, thus finding the  $argmax(Q(s_n, a_n))$  of rewards [7]. This process is repeated until the learning rate has decayed to a value such that it can be ascertained that the agent has acquired sufficient knowledge about the environment. It is now supposed to be competent to apply the action to a state that the environment can produce.

Very high-to-expert knowledge: Toward the end of RL's iteration, the RL agent has accumulated enough information for it to exploit any states that it encounters from the data replicating environment. In its knowledgebase, it will have a sizable volume of states, actions, and Q-value. from there, it can be considered that the RL agent has attained a high proficiency in solving all the faults with accurate and optimum actions. But the algorithm will not allow the RL agent to be conceded and be closed off to other

possibilities. Instead, it will allow a small room of chance that it will attempt to balance the choice of exploring for other new actions through randomization of chance. Algorithm 2 described the process in detail.

Algorithm 2 – FRD’s agent training process based on different learning phases

```

routine 1 - apply a preconfigured fault initiating procedure.
routine 2 - reset to DR environment to baseline
Input: The state of DRE statistics and computed rewards
Output: The action of new parameters’ value for the database
Initialization 1: set value for learning, reward preference and exploration rate, threshold for exploration,
learning, and exploitation.
Initialization 2: initialize memory, Q-table collection, and respective counters

Apply a baseline reset of the DR environment
Acquire the state of the DRE from stats report
Set exploration rate to 1
Set the learning rate to zero, med_learning to 30%, high_learning to 90%
Loop the iteration process
    Check the learning rate.
    /* low learning phase */
    If learning <= med_learning, do the exploration phase
        /* exploration phase */
        initiate random fault initiation routing
        Apply the action to the DRE environment to get a new state
        ***
        Find the reward/penalty =  $\Delta$  new state vs current state
        Add the knowledge of state, action, reward-penalty, and new_state to minibatch’s training data

    /* medium learning phase */
    If learning is > med_learning and < high_learning, then do
        /* learning phase */
        If run=1, apply a baseline reset of the DRE environment
        Get the current state of the DRE environment.
        Find best future reward and action based on new_state;
        Loop until the reward is higher than 0.05
            Train the NN model using knowledgebase as minibatch, with state as input and reward plus action
            as an output.
            Call the NN model to predict the possible new reward and action.
        Validate the action against the environment and get a new reward
        Add the information to the memory and minibatch
        Find the q-value for the state and action with consideration from gamma, then add them to Q-table
    /* high learning phase */
    If learning > high_learning, do exploitation phase.

```

```

If random(exploration) > exploration_rate, do exploration phase
Acquire the current state from the environment
Get the best action from the Q-table based on Q-value
Apply action to the environment
Decay learning rate & exploration rate

```

### 3.9. Approximation Between State and Actions

The numbers of statistics that are gathered from the DRE range in the hundreds and reflect the states and conditions of the software's operating conditions and status. Shareplex, Oracle DB, OS, network, and their interoperability states generate their unique statistics, and their values are combinations of numeric and text values. So pre-processing is required before they are used in the training. For the actions toward the environment, each software has its own set of commands that are suited for its setup. they are used for a variety of intent and changes. the actions for specific conditions may require one or more series of commands to achieve the goal. The setup for this action arrangement will be a vector of size,  $e+f+g+h$ , where  $e$ ,  $f$ ,  $g$ , and  $h$  are the number of commands for the software; Shareplex, Oracle DB, OS and interconnectivity respectively involved in the DRE. It is difficult to approximate between the numbers of combinations the software commands need to be actioned against the environment, and the yielded a large volume of performance statistics from it. This is a typical problem of curse dimensionality. To mitigate this problem, we use a Neural Network [26] which takes in the input of states,  $s$ , which are aggregated values of DRE statistics,  $n$ , to predict a scalar reward,  $r$ , and the possible  $m$  number of parameter values of actions,  $a$ , of that attribute to the final Q-value derivation in Eq (10).

$$\text{Predicted action} = f^{\theta}(s_1, \dots, s_n)_t \tag{10}$$

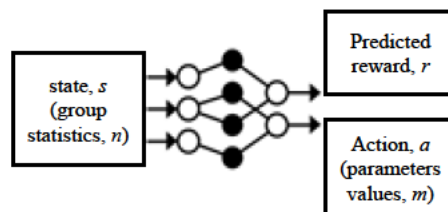


Figure 19 - NN function approximation between states versus predicted reward and actions



Referring to figure 19, the data set used for NN training is from the knowledge base that the RL agent builds up at the start with its trial-and-error testing. To simplify our approach, we focus on recalculating the current reward to become the Q-value. The reward here is the summarised absolute difference between the action of DRE’s service outage that is produced by the NN, and the actual values generated from the DRE’s system diagnostics procedures of,  $a_{predict} - a_{actual}$ .

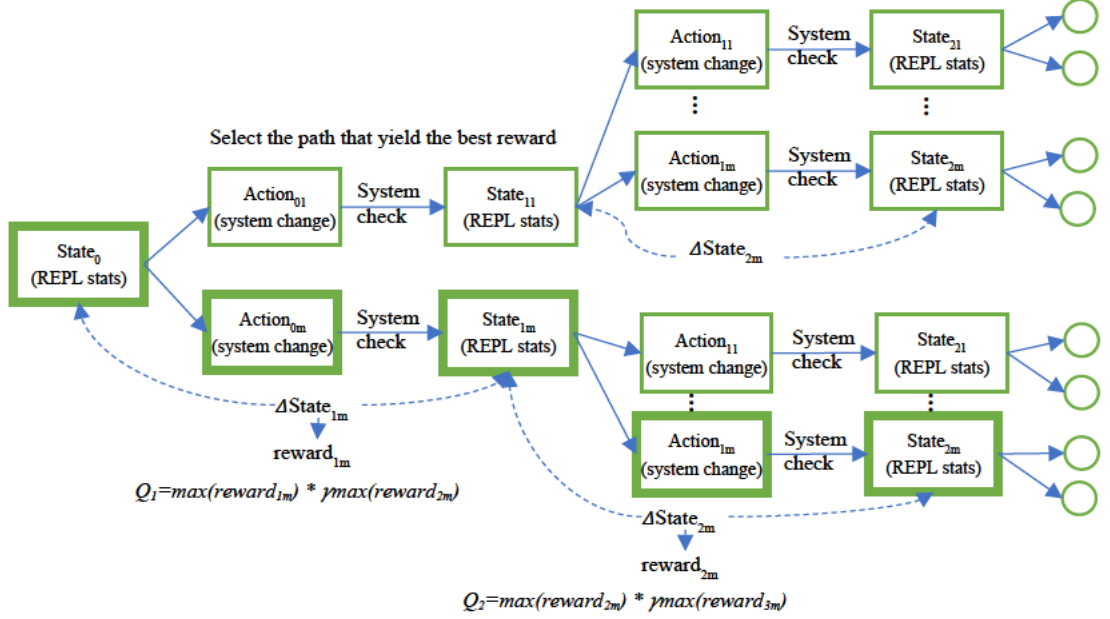


Figure 20 – RL agent’s action state-action flow

The summary of the difference will form the reward. This, in turn, is used as the Q-value that relates between the state and the diagnostic action, then associated with the validated information as  $Q(s, a, r)$ . This is then added to the knowledge base for the next round of NN training. Figure 20 shows the flow of the RL agent in finding the optimum path along with the DRE’s states and best actions that yield the optimum reward. The Q value is the computed normalized value that takes into consideration the current and future rewards.  $\gamma$  is set to 0.1 for some preference on future events but the emphasis is still on the immediate event.

### 3.10. Scoring the Environment’s State

The state of the DRE’s environment is represented by a vector that summarizes the information of the various software operation which is acquired after the stats gathering routine is executed as shown in Eq (11). The variables for software

operations' statistics of Shareplex, OracleDB, OS and interconnectivity are denoted here as  $sp$ ,  $db$ ,  $nw$  and  $os$ . Under each group, there are three summarized grouped variables; the first indicates the operational status of the software's attributes, e.g. processes are running or not. The second covers the performance statistics related to each software's attributes, e.g. CPU load at 60%. The third is the difference or variation between the current and previous values for the software's attributes, e.g. CPU load difference between previous and current periods. The iteration of the training period is denoted as  $t$ . But some statistics such as  $sp$  are more important and need more emphasis. So, weight is associated with them to augment their importance.

$$s_t = [\sum_{i=1}^k sp_i w_k, \sum_{i=1}^l sp_i w_l, \sum_{i=1}^m sp_i w_m, \sum_{i=1}^k db_i w_k, \sum_{i=1}^l db_i w_l, \sum_{i=1}^m db_i w_m, \sum_{i=1}^k nw_i w_k, \sum_{i=1}^l nw_i w_l, \sum_{i=1}^m nw_i w_m, \sum_{i=1}^k os_i w_k, \sum_{i=1}^l os_i w_l, \sum_{i=1}^m os_i w_m] \quad (11)$$

Table 3 – DRE's software stats and logs queries

DRE's software	Status(k) – Task's purpose	Perfstat(l) - information source
Shareplex (sp)	Query for process up or running Test if sp_cop is responsive Check if queue is stopped Check if config file is active Check response is proper Check if capture process is running Check if read process is running Check if export process is running Check if import process is running Check if post process is running	Shareplex's Parameter setting Shareplex's Process's Status Shareplex's Queue status via sp_ctrl Shareplex's directories information Shareplex's \$Pdir and v\$vdirdirectories
OracleDB (db)	Query for Oracle instance, DB1, is up Query for Oracle instance, DB2, is up Test if sqlplus can connect to DB1 Test if sqlplus can connect to DB2 Check if DB1 is in open mode Check if DB2 is in open mode Check if listener1 is running check if theres any invalid SPLX's objects on DB1 check if theres any invalid SPLX's objects on DB2 check if tablespace is full on DB1 (100%) check if tablespace is full on DB2 (100%)	Oracle DBs' Processes DB Operational stats DB's tablespaces Oracle's command console Oracle DB's system views OracleDB's trace logs Oracle's AWR reporting tool
network (nw)	Check if DB1 can connect to DB2 Check if DB2 can connect to DB1 Check if network is available between DB1 and DB2 Check if DB1 can be resolved via tnsnames Check if DB2 can be resolved via tnsnames Check network card	Listener.ora file Tnsnames.ora file Lsnrctl command console Listener's stats Listener logs
Operating system(os)	Check if the user account SPLX exists Check if the user SPLX belong to SPLX group Check if user account splx's permission is correct Check if user account oracle's permission is correct Check if server name is in /etc/hosts Check if Oracle unix account is in /etc/passwd Check if SPLX unix account is in /etc/passwd Check if Oracle is in Dba group in /etc/groups Check if Splx is in dba group in /etc/groups	Disk space availability via df command System based file; /etc/passwd, /etc/shadow, /etc/hosts, /etc/group Network card is up Vmstat's for cpu, memory

Where software components' critical status is  $k$ , Performance statistics is  $l$ , statistics variation between previous iteration and current is  $m$ , the time is  $t$ . the weights for  $w_k$ ,  $w_l$  and  $w_m$  are 1. The rationality for the default value of 1 is that each state representation has equal importance to their purpose in the DRE and require similar attention. Table 3 shows the list of the status purpose's description, their corresponding tasks and the result that are derived from the tasks. All the commands, scripts and utilities used for the tasks are available in the appendix.

Table 4 – list of system commands for DRE's software and functions

DRE's software	System Remedial actions	Query action
Shareplex (sp)	Start Capture on splx1 Stop Capture on splx1 Start Read on splx1 Stop Read on splx1 Start Export on splx1 Stop Export on splx1 Start Import on splx2 Stop Import on splx2 Start Post on splx2 Stop Post on splx2 Startup sp_cop on host1, 2 Shutdown sp_cop on host1, 2	Query for process up or running Test if sp_cop is responsive Check if queue is stopped Check if config file is active Check response is proper Check if capture process is running Check if read process is running Check if export process is running Check if import process is running Check if post process is running
OracleDB(db)	Startup Oracle instance, DB1 Shutdown oracle instance DB1 Startup Oracle instance, DB2 Shutdown oracle instance DB2 Startup listener on host1,2 Shutdown listener on host1,2 Change database, DB1, to open mode Change database, DB2, to open mode Compile invalid objects on DB1 Compile invalid objects on DB2 Re-grant specific privileges to SPLX on DB1 Re-grant specific privileges to SPLX on DB2 Increase tablespace size for SPLX on DB1 Increase tablespace size for SPLX on DB2	Query for Oracle instance, DB1, is up Query for Oracle instance, DB2, is up Test if sqlplus can connect to DB1 Test if sqlplus can connect to DB2 Check if DB1 is in open mode Check if DB2 is in open mode Check if listener1 is running check if theres any invalid SPLX's objects on DB1 check if theres any invalid SPLX's objects on DB2 check if tablespace is full on DB1 (100%) check if tablespace is full on DB2 (100%)
Network (nw)	Enable network card status Disable network card status Restore \$oracle_home/network/tnsnames.ora on host1,2 Restore \$oracle_home/network/sqlnet.ora on host1,2 Restore \$oracle_home/network/listener.ora on host1,2	Check if DB1 can connect to DB2 Check if DB2 can connect to DB1 Check if network is available between DB1 and DB2 Check if DB1 can be resolved via tnsnames Check if DB2 can be resolved via tnsnames Check network card
Operating system(os)	Unlock user account SPLX Change permission on oracle's directories Change permission on splx's directories Restore /etc/host.orig Restore /etc/groups.orig	Check if the user account SPLX exists Check if the user SPLX belong to SPLX group Check if user account splx's permission is correct Check if user account oracle's permission is correct Check if server name is in /etc/hosts Check if Oracle unix account is in /etc/passwd Check if SPLX unix account is in /etc/passwd Check if Oracle is in Dba group in /etc/groups Check if Splx is in dba group in /etc/groups

### 3.11. Action for the Environment

The FDR system needs to carry out a series of remedial actions in response to the faults that have been detected to resolve them. the actions comprise of a series of commands and custom-built scripts that run on the OS, DB and Shareplex console to

change their parameters, configuration, manipulate UNIX based files and alter OS' components. These actions are referred to as actions under the FDR's Fault Resolution (FR) module and to select and activate the required action, they are referenced via a vector that is mapped to the entire list of commands, segregated into their respective software and functions as shown in table 4.

# CHAPTER 4: DESIGNING THE FAULT DIAGNOSTIC (FD) MODULE

---

This chapter presents the Fault Diagnostic (FD) module with its design architecture and operations, testing, and results analysis.

## 4.1. The Current Approach Toward DRE's Fault Diagnosis

The current common method of implementing fault diagnosis for complex IT systems for both academia and industry is to use machine learning models such as Random Forests or Bayesian Network [60, 61]. Both require well-designed models that are specifically tailored to the intended IT systems where the fault detection and diagnosis procedures need to be performed. The premise for the design of such complex and well-defined fault detection and diagnosis (FDD) model has complete knowledge of every sub-system, component, relationship, and operation including data exchange in the IT system. The limitation with this approach is that every implementation of these complex IT systems is not generic and are tailored to specific business IT requirement. So, having a rigid and well-defined FDD agent will not have the adaptiveness nor flexibility to meet the range of different system setups. It will require numerous customization which is time-consuming and laborious. Another downside with this approach is the coverage of the FDD models, as they are designed based on the IT human's expert knowledge. If there are new anomalies occur in the systems and the diagnosis model in the FDD agent may not have the information to mitigate it, then these new anomalies will be set as blind spots for the agent permanently unless the IT administrators take note of this and provide corrective actions to the FDD model like updates or upgrades. This is also applicable to situations where the IT system's setup must be altered to meet new business requirements, and that may render the static FDD model invalid. Therefore, the FDD models must be constantly updated to keep up with the changing environment.

One of the most important components for proactive fault detection is highly competent monitoring software that can monitor all the software used in a complex setup. However, most of the monitoring systems available are built to monitor specific

software which is commonly used in the industry. They usually include other common supporting technology and software such as OS, but most require additional add-on or patches to enable them. But they do not cater for uncommon products such as Data replicating or ETL software. The only option is to either use the replicating software vendor's customized monitoring tool or the IT administrator must write scripts to function as basic monitoring. But all of them fill the monitoring needs on a vertical basis, meaning they are primarily focus on the overall functionality of the software alone, but rarely monitor the inter-operability or relationship from a holistic perspective of a multitude of software functioning as an integrated system. Another shortcoming with the existing FDD approach is the need for every iteration of the diagnosis task, it will require an exhaustive execution of detailed checks on every component and attribute of the IT systems, followed by passing statistics into the FDD model and deducing the faults. While some checks may be instantaneous, others will take a longer time to validate and acquire their statistics. This is time-consuming and computationally intensive which may significantly delay the overall fault diagnosis. This delay is not desirable especially when the IT system is mission-critical and requires a very fast turnaround time in its fault diagnosis and resolution process.

What is required here is a new approach where the FDD model can be made general-purpose enough to suit any combination of software for the IT systems; be it database, web application, firewall or network. It should minimize unnecessary steps of detailed check procedures and be able to deduce the diagnosis quickly simply by looking at the symptoms and refer to its knowledge just an experienced IT administrator. It should be flexible to extend or correct its existing model to cover any new alteration that occurs in the IT system's environment. In other words, we consider the new FDD model as a new mechanic apprentice that needs to learn on the job to perform the checks and deduce the faults from the gathered information under the guidance of his supervisor. We expect it to learn in both detecting and diagnosing adaptively, starting from an early stage where it will do extensive checks on every aspect of the IT system, but once it reaches a certain level of maturity, it should be able to determine from its expert knowledge that the certain symptoms or events exhibited in the IT system can be related to certain sub-domain of the system's setup with great confidence, similar to the skill difference between an inexperienced and an expert IT administrator.

There are several common contemporary practices to the approach of Data replication's Fault Diagnosis in academia and industry and they are listed in Table 5 with their complexity and effort of implementation rated, and this is based on feedback from experienced IT administrators from the industry using Delphi method[62].

Manual approach- this is the basic task where all IT administrators need to do especially for the junior staff that is not familiar with the system. They must go through a checklist that stipulates the attributes of sub-systems that need to be checked, including the corresponding script commands or shell's interface that they need to interact with. information or statistics are collected slowly from each sub-system and then tally them to determine the cause of the faults. This is a time consuming and laborious manual process that is unproductive, but it is still the baseline of any software fault finding procedures.

Best practice - another option is following a well-defined checklist that was well-prepared by experts. **By** associating with the symptoms and service errors found, the checklist can narrow down the possibility and guide the IT administrator to perform only those checks that have the closest relation to the potential faults group. This can cut down a significant amount of labour and save time, but it is still a tedious manual task, and it is difficult to scale.

Decision rule-based scripting - another option is to script the process of the above checks and run it whenever the faults surfaced. the IT administrator will run the specific scripts to check the sub-systems' components, thus automating some of the parts of the manual query processes and increase the turnaround time. This approach has higher scalability, but it still requires human intervention and expert knowledge to decide the exact type of scripts to run and be able to interpret the results to deduce the faults.

Machine learning models (SVM, decision-tree, deep-learning) - researchers have used this approach to implement fault diagnosis on various types of machinery and other hardware with success[63-66]. It can automate the detection and diagnosis of the machinery's faults process, giving a fast and accurate response. However, the pre-requisite for implemented these supervised learning is that the environment must be stable with limited variability. Besides this, the models' pre-requisite is a large set of labelled data for its training. Another approach is to engineer the features from

unlabelled samples to lead the training which is a significant amount of preparation works. on top of that, their designs are usually meant for well-defined systems and should thereby any variation in the system's structure or operation, then they must be redesigned and recalculated which is laborious.

Table 5 – Fault Diagnosis Method (low 1 to high 10)

Method	Complexity	Effort	Remark
Manual[67]	3	9	Require medium IT knowledge and skill. Go through a prescribed checklist. Manual, inefficient, time consuming, laborious, not scalable. low productivity, cannot cater for unforeseen or bespoke setup.
Best practice, decision rule-based[67]	4	8	Require medium level skill, a checklist has a decision flowgraph. Improve troubleshooting time. still manual, inefficient laborious and time-consuming but superior to a manual approach.
Bayesian network model	7	4	Require a med-high level of knowledge and skill. The target system's model must be pre-known. tightly customized. Rigid design. Difficult to enhance or correct. Need redesign. Less time consuming, limited scalability, higher productivity. Require IT admin intervention to correct. No research available for DRE
Machine learning model	5	4	Require a higher level of knowledge and skill. The target system's model must be pre-known. Tightly customized. Less rigid design. Difficult to enhance or correct. Need redesign. Less time consuming, scalable, higher productivity. No research available for DRE

## 4.2. Problem Formulation

DRE has several systems working together and each system has a list of subsystems that comprise of other individual components or smaller sub-system that serve the function of the higher sub-system. The diagnosis of the fault is linked closely to each of the systems' hierarchically related sub-systems, and the finer the sub-set, the more accurate the diagnosis will be. The diagnosis process and their granularity are determined or limited by several factors such as the 1) availability of the data, 2) resources to collect them, 3) additional requirements by the IT administrators. Referring to the AC reinforcement learning model in section 2.4.3 under chapter 2, the set of all the software systems in DRE is represented as,  $S$ , and the individual software system and their sub-system are represented as  $s_n$  and  $s_{nm}$ , respectively.  $n$  is the number of software and  $m$  is the number of sub-systems that software,  $s_n$ , has.

The sub-systems have metrics associated with them, where  $M$  refers to this set of all the metrics is. Each of the metrics is linked to one sub-system's attribute,  $m \in M$ . The set of all service models is represented by  $C$ , and each model,  $c$ , is a logical model that group those sub-systems together. They support similar services and can have multiple metrics. The metric has  $n$ -to-1 mapping to a sub-system  $\alpha: M \rightarrow S$  and the set of all their mapping is  $A$ . The metric has  $n$ -to-1 mapping to the models;  $\beta: M \rightarrow C$  and the set of all their mapping is  $B$ . The model has  $n$ -to- $n$  mapping to the sub-



systems,  $\sigma: S \rightarrow C$ , and is represented as  $D$ . The relationship among the model, metric and sub-systems can be represented by a tuple  $(S, M, C, A, B)$  and their  $n$ -to- $n$  relationship of  $D$  is represented using an association matrix,  $X$ . Each element,  $X_{ij}$ , of  $X$  has a value of 1 if,  $m$ , is mapped to  $\alpha(m)$  and  $\beta(m)$ , where  $\alpha(m)=S_i$  and  $\beta(m)=C_j$ . Otherwise, it is 0. We use a detecting service vector,  $O$ , to represent the state of the services and it is linked to  $C$ ; where  $O(t) = [o1(t), o2(t), \dots, on(t)]^T$ . If  $oi(t) > 0$ , a fault has occurred in the service model. Therefore, both the matrix,  $X$ , and the detecting vector,  $O(t)$ , formulates an anomaly-scoring vector,  $r = [c1, c2, \dots, cn]^T$ , which indicates if subsystem  $S_{it}$  has faults and whether it is degrading or improving at  $S_{it+1}$ .

The software modules (DB, replicating tool, networking, OS etc.) are sub-systems and the group functions as service models as shown in figure 21. The diagnosis localizes the fault of the software towards the group functions. The tuple  $(S, M, C, A, B)$  can be enhanced to accommodate the diagnosis of other models to sub-system with ease.

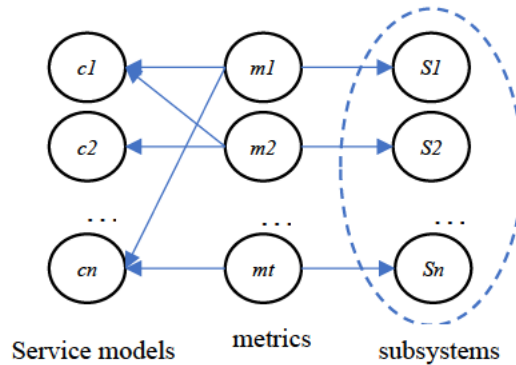


Figure 21 – Relationship between Services, metrics and DRE’s sub-systems

### 4.3. Adaptive Fault Diagnosis (FD) Module Design

Deep Reinforcement Learning (DRL) is used to perform the fault diagnosis against the data replication environment (DRE) to integrate an intelligent system into the FD module. The FD Module can therefore emulate the learning and work process of a junior IT administrator when managing a system related problem.

Figure 22 shows an overview of the FDR model. As the module is trained, it will learn to analyse every aspect of the DRE and gather all the related information. It also starts with little or no a-prior knowledge on the relationship between information and diagnostic conclusions. By interacting with its supervisor or someone with expert domain knowledge, it receives guidance and information to derive the diagnosis. This iteration repeats and the FDR will gradually build up its knowledge. Eventually, it will

possess a critical level of know-how on the environment, allowing it to derive an accurate deduction of the fault based on the environment's symptoms without going through unnecessary system checks.

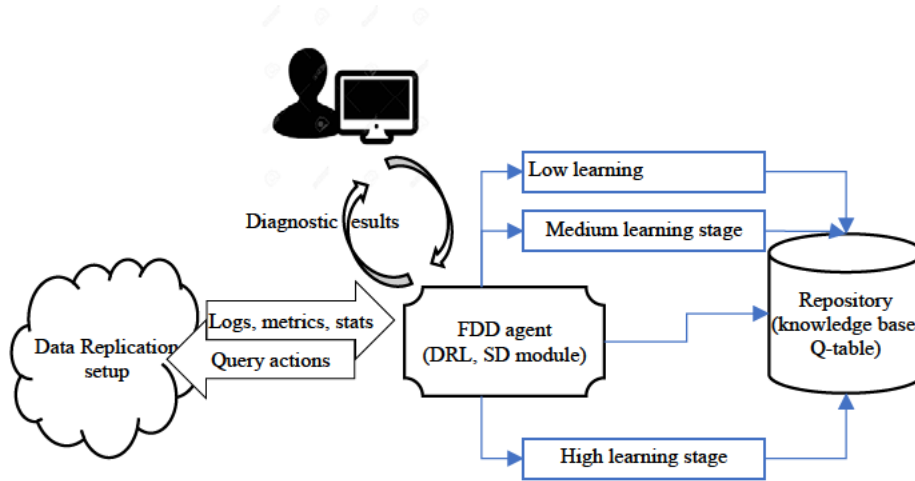


Figure 22 – Adaptive Faults Diagnosis overview

The FD has three objectives; 1) it must be able to interact with the environment and identify the fault promptly without any error, 2) the system is adaptive and learns by interacting with both the environment and input from the IT administrator constantly to improve in its diagnostic skill and knowledge-base and 3) it must have a flexible architecture that allows it to extend its capability vertically or horizontally in its diagnostic capability. While these routines can be easily performed by hardcoded, well-defined models, the scope for these groups of software fault diagnostic systems cannot be fixed. they must be flexible to cater to different fault scenarios and grow to cover other forms of software systems that are subjected to the FD's management.

A series of data collection points is established to ingest and process the information from the DRE before sending it to the FDR's DRL unit. Once the agent has determined that there is a need to investigate, it will launch a series of queries to acquire more details from the environment for its fault analysis and diagnostic routine until it can either deduce the root cause or list it as an unknown error. An unknown error triggers a separate routine to notify the IT administrator for assistance and input. The FD model can be split into 3 modules: Information Acquisition (IA), Diagnostic Reinforcement Learning (DRL) and System Diagnostic (SD) (figure 22).

### 4.3.1. Information Acquisition (IA) module

The availability of timely and accurate information from various software subsystems of the DRE is important to the diagnostic analysis process and they come in three forms: logs, metrics, and events. All the DRE's software; Oracle DB, Shareplex and Operating System, constantly and proactively records information about their states as log files under the software's respective product directories. They are available in a well-defined format and provide enough information to support a system diagnosis effort. For the metrics part, these are system or software statistics that can only be obtained through explicit command queries via OS' shells or their respective utility tools. The third form is the events that logically describe the users' experience depending on the system while interacting with the DRE. It is a brief description of service anomalies, like login failure, slow replication, and missing data for example, that are encountered for the FD to refer to during investigations. Once the inputs have been processed with all the mandatory details extracted, they are used to represent the system environment's state to the DRL and as input into both the SD modules.

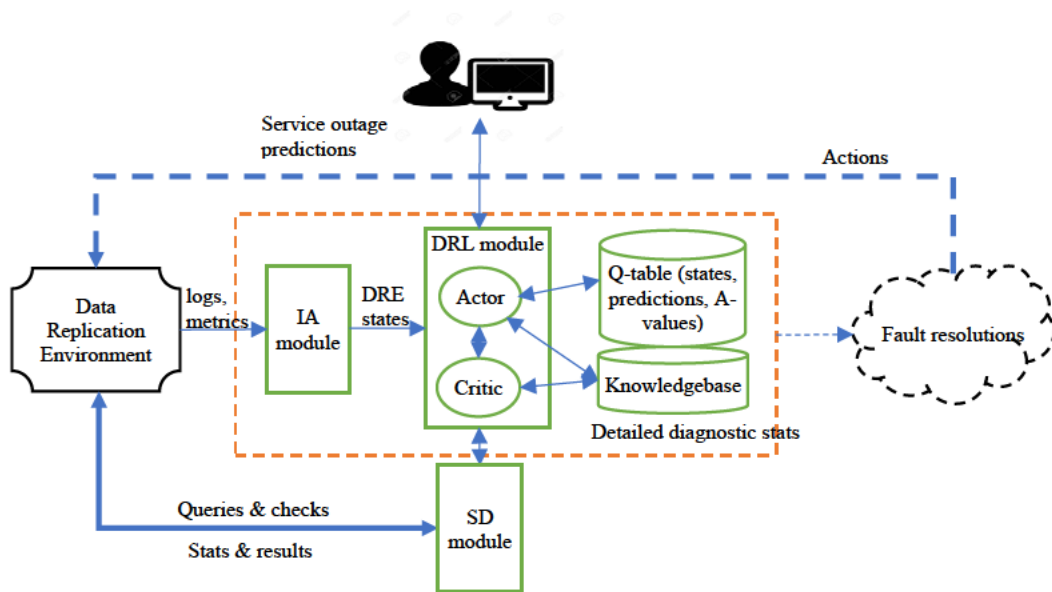


Figure 23 – Faults diagnosis agent's architecture and workflow

### 4.3.2. Diagnostic Reinforcement Learning (DRL) for FD Module

The FD module uses the Actor-Critic Deep Reinforcement Learning algorithm which is described in section 2.4.3 under chapter 2. Referring to figure 23, in this setup, the DRL's Actor performs as a function approximator that tries to predict the best

action, the best diagnosis in this case, for a given state. The DRL's Critic takes in the DRE's state-input plus the Actor's action, combines them, and output the action's maximum future reward, Q-value, for the given state-action. The Critic uses the SD module to validate and score the Actor's action. There are 3 phases of learnings for the DRL as shown in figure 24.

1) Early learning phase: where the agent has little a-prior knowledge about the DRE initially, so it needs to perform exploration by interacting with it through trial-and-error. It passes the DRE's states into the SD module as symptoms and the SD module runs the diagnostics routine against the DRE's software to gather information about their attributes and service status. The SD module will then be processed and the acquired information summarised to formulate a service diagnosis matrix as shown in eq (1).

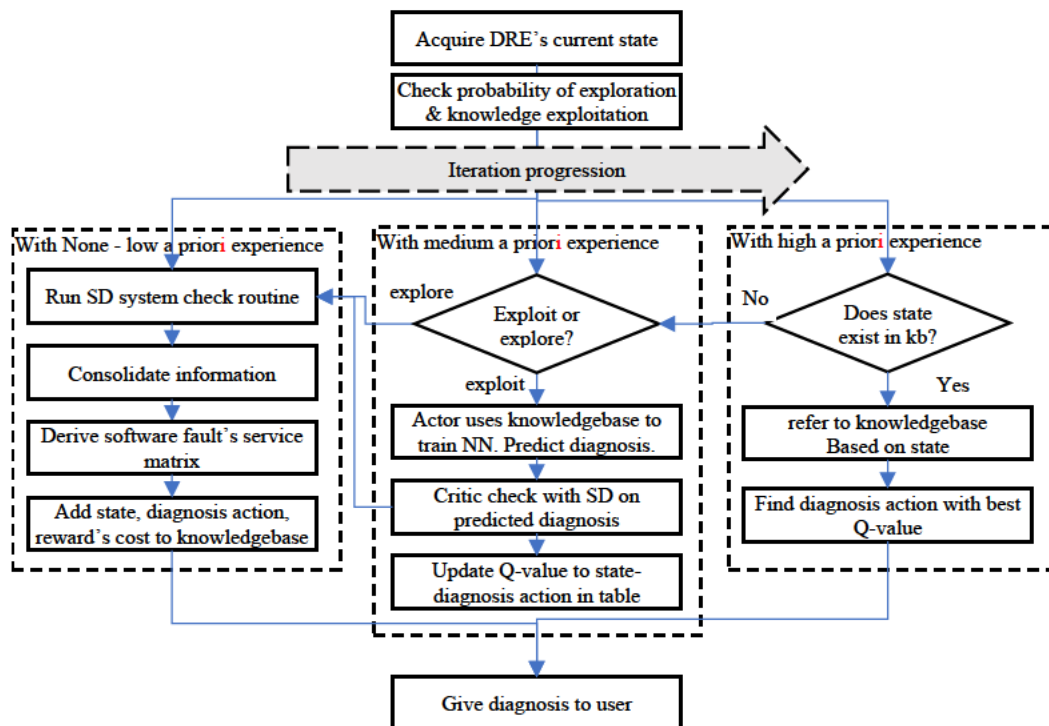


Figure 24 - Different phases of reinforcement learning in the RL agent.

2) Middle learning phase: the DRL's Actor learns to predict the best action-diagnosis against the environment's symptoms-states using its Neural network which has been

trained by using the DRE knowledgebase gathered from the earlier learning phase as its minibatch. There is a high chance that the neural network will make predictions. In such an event, the RL's Critic will run the validation process through the SD module which corrects the Q-value and assigns it to the state-action pairs stored in the Q-table, as well as updating the knowledgebase. The gradual build-up of the knowledgebase will improve the accuracy of DRL's NN prediction.

3) High learning phase: By this stage, the DRL agent will have learned all the states-symptoms that is associated with the faults in the DRE and can predict the best actions-diagnosis with high accuracy. This is regarded as the exploitation of the DRL's rich build-up of knowledge where it can provide a very quick turnaround time by identifying the faults' matrix without performing excessive checks or validation through the SD module. However, the agent also performs a probability-based decision between exploitation versus exploration at this time; Exploitation where the DRL decides to refer to its knowledgebase to respond to the best action for the DRE's state, and Exploration where the DRL decides to run detailed checks through the SD module to get the diagnosis instead of relying on the NN's prediction. At the low learning phase, the probability of exploration will be high. However, this diminishes over time when it reaches the high learning phase, where the exploration rate has decayed over iterations and the preference shifts towards knowledge exploitation.

#### **4.4. System Diagnostic (SD) Module**

The DRE comprises of different software and technology working together to provide the service. Each software and technology have a unique list of configurations, checks, operations, and attributes. Therefore, the SD Module has several groups of check routines that target this software, and within each group are sub-routines that query specific areas in the software like privileges, permission, process status, usage statistics, for example. Referring to figure 22, the DRL agent gives instructions to the SD module to perform the checks against the DRE's environment, ranging from comprehensive top-down checks to selective ones. This is analogous to junior workers who need to perform every check to make sure, compared with a senior worker who can deduce the exact areas to verify before deducing the root cause of the error. The SD module then performs the detailed checks by running a long list of command queries and scripts against the DRE's software. Some of the details

collected are the 1) states of their system processes, 2) space availability of directories in which the systems' binary files reside and their information are processed, 3) current privileges of the system's process, files, accounts that they operate from, 4) details in their configurations and parameters that they are using, operating or initializing from, 5) network connectivity that is required for their operation, 6) statistics of specific operations like process backlogs, connectivity delays, abnormal system values. Others contain a summary of software-wide statistics which range in the thousands. The result is then consolidated and sent back to the RL agent. It is a matrix that presents the multiple sub-areas under the DRE across different software about their functional status from a high-level perspective. Further details can be made available from the diagnostic module upon request, but the vast number of details will be too overwhelming for its administrators to go through. The following is a tabulation of the output in which each command performing the specific information extraction from the various software.

The SD provides its diagnosed results of the DRE's software status on the participating server hosts,  $n$ , at their service group level instead of the technical attributes. This is to give an overview of the DRE software's availability from a general administrative perspective, taking into consideration their 1) process availability, 2) filesystem's attributes and permissions, 3) responsiveness to administrative interaction, 4) communication functionality, 5) data transfer and input-output capability, and 6) software's function and operation status. The specific software details can be made available, and they will be connected to the future Fault Resolution agent. The four diagnosed service groups in Eq (12) are as follows: Database service,  $sdb_s_n$ , Shareplex replication service,  $srpl_n$ , Network and communication services,  $snet_n$ , supporting the OS environment,  $sose_n$

$$DRE's\ diagnosed\ service\ matrix = \begin{bmatrix} sdb_s_1 & srpl_1 & snet_1 & sose_1 \\ sdb_s_2 & srpl_2 & snet_2 & sose_2 \\ \dots & \dots & \dots & \dots \\ sdb_s_n & srpl_n & snet_n & sose_n \end{bmatrix} \quad (12)$$

#### 4.5. Data Replication Environment (DRE)'s State Representation

It is challenging to define the DRE's state due to its complex multi-tier software setup and the characteristics of the IT applications under its service. Therefore, a direct

method is needed to identify a state in a database without time properties, where each software's operation information is mined continuously for anomalies and errors. In this research, the use of a matrix is proposed to capture a list of events and process statuses of the DRE's software across multiple sources and target instances,  $n$ . Therefore, the two sections in the state's matrix contain information from both their logs and process statuses. For the logs, the attributes are a numerical representation of the encountered error messages in their respective logs, which are concatenated to 10 characters long and hashed using Secure Hash Algorithm 1 (SHA1). The following is the list of the software's logs location and their respective variables assigned.

1. Oracle database's alert logs with the prefix of ORA-XXX, files exist in the location;  $\$ORACLE\_BASE/diag/rdbms/DB1/trace/alert\_DB1.log$ , as  $oralog_n$ .
2. Shareplex replication's event\_logs with the initial string of "Error", files available in the location at;  $\$VARDIR/log/event\_log$ , as  $splxlog_n$ .
3. Network-related Listener's logs with the prefix of LSNR-XXX, available in  $\$ORACLE\_HOME/diag/network/log/.log$  as  $nwlog_n$ .
4. OS's error with the string, err, in  $/var/log/syslog$ , as  $oslog_n$ .

For the process's status, the status shows the presence of the DRE's software main processes in the VM host's background as well as the reachability of remote VM from the current VM. The representation is; 1) Oracle DB's primary process, smon, as  $orastat_n$ . 2) Shareplex replication's main process, sp\_cop, as  $splxstat_n$ . 3) Oracle's listener's processes and network, lsnrctl, as  $nwstat_n$ . 4) Ping status from both UNIX nodes to one another, as  $osstat_n$ .

The services under the different software are represented as; 1) Oracle DB's as  $orasvc_n$ . 2) Shareplex replication as  $splxsvc_n$ . 3) Oracle's listener and network, as  $nwsvc_n$ . 4) Operating system and host's, as  $ossvc_n$ .

Therefore, the final matrix represent the DRE's state in eq (13).

$$DRE's\ state = \begin{bmatrix} oralog_1 \\ oralog_2 \\ splxlog_1 \\ splxlog_2 \\ nwlog_1 \\ nwlog_2 \\ oslog_1 \\ oslog_2 \end{bmatrix} \begin{bmatrix} orastat_1 \\ orastat_2 \\ splxstat_1 \\ splxstat_2 \\ nwstat_1 \\ nwstat_2 \\ osstat_1 \\ osstat_2 \end{bmatrix} \begin{bmatrix} orasvc_1 \\ orasvc_2 \\ splxsvc_1 \\ splxsvc_2 \\ nwsvc_1 \\ nwsvc_2 \\ ossvc_1 \\ ossvc_2 \end{bmatrix} \quad (13)$$

Table 6 describes the specific software validation and checks that need to be performed to acquire the DRE's collective status together with the associated details that depict their respective software components including the checks are performed against them. Each of the software is checked by different OS scripts which have encapsulated commands to interrogate them on their respective service groups of logs, processes, and services. For various software logs check, the scripts are check\_alert\_log\_err.sh, check\_event\_log\_err.sh, check\_os\_log\_err.sh on OracleDB with listener, Shareplex and OS. As for the DRE's software processes checks, check\_all\_processes.sh is used. The last group check is done by check\_all\_services.sh which validates their specific services.

Table 6 - Memory and logs checks

DRE's software	Service Group checks	Software attribute. Area of focus	Detail checks/description	Name of Process, logs, system views
OracleDB	Process check	Process' stats	DBs' memory process in the OS	Smon, pmon
	Process check	Operation's stats	DB's mode of operation	V\$instance
	Service check	Tablespace's stats	Tablespaces have enough space on DBs	V\$freepace, v\$tablespace
	Service check	Session's stats	No existing blocking session on both DBs	V\$locked_session
	Service check	Users' stats	User accounts are open and accessible on DBs	V\$users
	Service check	Object's stats	objects' validity on DBs	Db*_objects
	Alert Log error check	Alert logs	Check for exception and errors in both alert logs	Alert_<oracle_sid>.log
Shareplex	Service check	Parameter setting	parameters are valid in Shareplex instances	Show param
	Process check Service check	Process's Status and operation	Shareplex's memory process in the OS	Sp_cop, sp_<processes>
	Process check Service check	Queue's status and statistics	Shareplex's various queues status on both nodes	Capture, export, post ....
	Service check	Directory's stats	Both directories, \$pdir and \$vdir, have enough free spaces	\$PDIR, \$VDIR
	Log error check	Shareplex logs	Shareplex exception in both event logs on two nodes	\$VDIR/log/Event_log
Network	Process check	Listener Process stats	Oracle's listener process on both nodes	Lsnrctl
	Process check Service check	Listener.ora availability, and stats	Listeners' availability for service on both nodes	Lsnrctl
	Process check Service check	Listener's stats – error or available	Listeners' operations are valid and not in error	Lsnrctl status
	Log error check	Listeners' logs	Exception and error in logs for both listeners	Listener.log
	Service check	DBs reachability	Able to connect to each DBs from opposite side	Tnsping <DB1>
	Service check	Hosts reachability	Able to reach each VM host from adjacent node	Ping <host1>
OS	Service check	Disk space	Free space availability on OS for both nodes	Df -m
	Service check	Primary conf files	Validate /etc/passwd, /etc/shadow, /etc/hosts, /etc/group files	Os: ls -lt /etc
	Process check	Network card operation	Network card status and availability	Os: ifconfig
	Service check	CPU resource	CPU usage not full	Os: sar



	Service check	Memory resource	memory usage not high	Os: free -m
	Log error check	System log	Check for error on system logs	Os: check /etc/log/syslog

#### 4.6. DRE's Action of Diagnostic Prediction

The DRE's state information in 5.5 are the summarized raw inputs that the FDR takes in, and part of its diagnostic routine is to show its ability to predict or estimate the possible faults with the DRE's software. Part of the outcome of the FDR is to produce the diagnostics report that shows the status of the DRE's operation at a high service level which indicates the software's respective sub-group and level of errors it has, concerning the DRE's environment state. The outcome is a series of tuples that signify the status or condition of the software group and their sub-group services in the arrangement of *<software\_typ>* and *<software\_sub-service\_grp>*. Their statuses are derived from a custom-built script which contains a list of OS commands that extract and aggregate all the statistics from the various DRE software into their respective sub-system service groups, to show the service outage based on the state's matrix from the environment in the previous chapter. The process of showing the service-level exceptions will be later handled by the DRL's NN.

DRE's service level diagnosis = {*dba, dbb, dbc, dbd, spa, spb, spc, spd, spe, spf, nwa, nwb, nwc, osa, osb, osc, osd*}

where,

1. for OracleDB, *dba* = DB's memory process, *dbb* = DB's Status, *dbc* = DB's Account security, *dbd* = DB's storage space.
2. for Shareplex, *spa* = Splx's main processes, *spb* = splx's console availability, *spc* = splx's queues operation, *spd* = splx's configuration validity, *spe* = splx's queues' backlogs, *spf* = Splx's DB accessibility.
3. for the networks, *nwa* = Network connectivity of Databases' listeners, *nwb* = Splx's network connectivity, *nwc* = VM hosts interconnectivities.
4. for the OS, *osa* = hosts' OS unix account status, *osb* = hosts' file storage space, *osc* = hosts' network card status, *osd* = hosts' resource availability.

#### 4.7. Approximation Between DRE's Symptoms-States and Diagnosis-Actions

It is not practical for the optimal policy to be used due to the large problem and solution space of both the DRE's states and the actions for diagnosis prediction. This

is regarded as a typical curse dimensionality problem and to mitigate this issue, Neural Network is used by the DRL-Actor to the DRE's states as input and then predict the best possible action of diagnosis [7]. Although the dataset used for the NN training comes from the knowledge base, there is not be enough data initially to train the NN model competently as it starts with little or no a-prior knowledge about the DRE's environment. Therefore, knowledge must be accumulated by investing in the initial phase of RL through multiple iterations of trial-and-error. The predicted action,  $a_{predict}$ , from the DRL-Actor NN is compared with the actual action,  $a_{actual}$ , that the DRL-Critic has validated with the SD module.

In a typical DRL model, the agent will try to find the best values based on the policy, including predicting the potential rewards and actions for the input states. Whereas in the Actor-Critic RL model, the Critic has a separate NN to validate the Actor's predicted action toward the environment and get the real score, before correcting the actor's policy. In this thesis, it is proposed that the SD module acts as the optimal policy for the DRL which the Critic uses directly. The difference between the predicted versus Actual actions forms the mean square error function for the NN for optimization. The Critic also assigns the Q-value, which is the maximized reward for the state-action pairs and is a normalized value of the MSE value. Section 3.15 has described this in detail.

#### **4.8. FD's Algorithm**

Algorithm 3 describes the process of how the FD's SD unit works both as a procedure to gather and process information from the DRE's state, plus the execution of the external script to derive the detailed system diagnostics and convert them into service outage information. The procedure, *p\_complete\_diagnosis*, uses the same process but against a list of faults to build up the knowledgebase required for the FD's NN training.

Algorithm 4 describes the FD's process from a holistic point of view. This Fault Diagnostics is inspired by reinforcement learning but with some customization. The DRE is a highly complex software integrated with many parameters and configurations that can affect the stability of the system. Therefore, the implementation for the research has some deviation from the typical reinforcement learning method. However, the principle of learning from experience to build up the knowledge and its

exploitation at the latter phase remains. It is a policy-based reinforcement learning and the SD's output is a definitive certain goal for each state, taking the SD's service outage and diagnostics as the product of its immediate rewards with no consideration for any future rewards. The Q-value that defines the state and action are certain with no room for any alternate consideration. The usual reinforcement learning's parameters that control exploitation-exploration will not be used. This follows the work rules where the IT administrator trainee is not allowed to perform the diagnosis independently until the Senior IT supervisor has watched over all their tests before allowing them to perform the first duty.

The initial, early and middle learning phases of the FD uses the SD unit aggressively to build up the knowledge to a sufficient level for the NN to competently predict them. The SD runs the procedure, *p\_complete\_system\_diagnosis*, which runs through the simulated faults of both true positive and negative scenarios that were planned in the previous section, followed by running a list of external routines which contain a series of system-related commands to gather all the information from various sources, including the software's log, processes, and internal operation statistics. It then aggregates the information and represents the DRE's state for that period after the fault is injected in. Another external file, *system\_matrix.txt*, is then executed against the DRE. This file contains all the system commands that interact with various DRE's software elements and service their statistics. This, in turn, is processed and aggregated to form the logical representation of the DRE's service outage. At this point, there are three types of information: (i) DRE's state, (ii) service outage and (iii) system diagnostics. These are considered knowledge and are stored in an external file. Once this phase of learning is complete, the next phase of using the NN of handling the diagnostics process takes over. This occurs while looping through the simulated fault list: inject each fault, acquire the DRE's state, use the NN to predict the service outage and lookup from the knowledge on the closest matching system diagnostics information based on the service outage details. The loop ends with the execution of the fixes to restore the services.

Algorithm 3 – System diagnostics, for individual and batch run

###this System Diagnostics routine is for single run Input: The state of the DRE Output: Service outage and system diagnostics
--

```

def p_single_system_diagnosis():
    all_process_err = execute(check_all_process.sh) #gather stats on dre software processes
    all_services_err= execute(check_all_services.sh) #gather stats on dre software services
    ora_log= execute(check_alert_log.sh)          #scan oracle log for anomaly
    splx_log= execute(chec_event_log_err.sh)      #scan shareplex log for anomaly
    lsnr_log= execute(check_listener_log_err.sh)  #scan listener log for anomaly
    os_log= execute(check_os_log_err.sh)         #scan os log for anomaly
    all_log_err=aggregate(ora_log, splx_log, lsnr_log, os_log) #combine all scanned log anomalies
    system_diagnostics=execute(system_matrix.txt) #this run a list of external sys cmds #run list of syscmds to
gather all DRE stats
    dre_state=join(all_log_err,all_process_err,all_service_err) #join up all the errors/anomalie to form dre's
current state
    service_outage=aggregate(system_diagnostics)
    knowledgebase.write(dre_state,service_outage,system_diagnostics)
    return(dre_state, service_outage, system_diagnostics)

###this run through the whole scenario of pre-planned faults to buildup knowledgebase
Initialization1: learning, Knowledgebase, breakfix_file
Set the learning rate to zero, med_learning to 80%
breakfix_file =fault_breakfix.txt

def p_complete_system_diagnosis():
    Loop the iteration,i, of reading(breakfix_file):
        break_cmd = f<breaki>
        fix_cmd = f<fixi>
        Check the learning rate.
        execute(break_cmd) #inject fault
        all_process_err = execute(check_all_process.sh) #gather stats on dre software processes
        all_services_err= execute(check_all_services.sh) #gather stats on dre software services
        ora_log= execute(check_alert_log.sh)          #scan oracle log for anomaly
        splx_log= execute(chec_event_log_err.sh)      #scan shareplex log for anomaly
        lsnr_log= execute(check_listener_log_err.sh)  #scan listener log for anomaly
        os_log= execute(check_os_log_err.sh)         #scan os log for anomaly
        all_log_err=aggregate(ora_log, splx_log, lsnr_log, os_log) #combine all scanned log anomalies
        system_diagnostics=execute(system_matrix.txt) #this run a list of external sys cmds #run list of syscmds to
gather all DRE stats
        dre_state=join(all_log_err,all_process_err,all_service_err) #join up all the errors/anomalie to form dre's
current state
        service_outage=aggregate(system_diagnostics)
        knowledgebase.write(dre_state,service_outage,system_diagnostics)
    end loop

```

Algorithm 4 – Fault diagnosis, using SD and NN units

```

Input: The state of the DRE
Output: Service outage and system diagnostics
Initialization1: learning, Knowledgebase, breakfix_file
Set the learning rate to zero, med_learning to 80%
breakfix_file= fault_breakfix.txt
Loop the iteration,i, of reading(breakfix_file):
    break_cmd = f<breaki>
    fix_cmd = f<fixi>
    Check the learning rate.
    execute(break_cmd) #inject fault

    ###exploration phase - use SD unit
    If learning < med_learning, do the exploration phase
        dre_state, service_outage, system_diagnostics= p_system_diagnosis()

    ##Knowledge exploitation phase - use NN unit
    If learning is > med_learning then do
        ##Train NN with data from knowledgebase
        NN_model=NN_build(knowledgebase)
        service_outage=round(NN_model.predict(DRE_state))
        system_diagnosis=search_knowledgebase(DRE_state,service_outage)

    print(dre_state, service_outage, system_diagnostics)
    learning rate +=1
    execute(fix_cmd) #fix fault
loop

```

## 4.9. Empirical Analysis

This section describes the tests conducted for the FD module. The purpose of experiments is to determine the effectiveness of the proposed FD method in producing the best diagnosis for the DRE under simulated faults situations. Before each experiment's iteration, the testing environment DRE's services are restored to the baseline where all the DRE's services are functioning normally. Not all errors introduced can result in a service's disruption. The goal is to ascertain the diagnosis of those faults that can disrupt the services and less toward those that are either too minor or ineffective to cause major issues to the replication services. However, the test scope is limited to faults that are recoverable and not catastrophic failures. In the event of a catastrophic failure, the DRE service is irrecoverable and can only be solved by an entire system rebuild.

#### **4.9.1. Software used for FDR tests**

This section describes the setup, arrangement, and configurations of the respective software in the DRE test environment.

**Database:** two Oracle DBs are installed on two VMs and each of them houses the Test schema that owns 10 tables. One of the DB serves as the source site where SQL updates are applied against the Test Schema's tables. The other DB on the other VM serves as the target where it receives the data changes from the source via Shareplex. The Test schema tables' structure is identical between the DBs for the replication to occur. While the target DB requires no special setup, the source DB must have several features enabled, including both archive log mode and supplemental log data. The archive log mode enables the DB to archive their redo log files, which contain the information of all the DML and DDL operations that had occurred in the DB, into a specific file location. The supplemental log data feature enforces the DB to log additional information about the DML and DDL which the Shareplex needs for its reading operation. The Shareplex requires superuser level privileges on both DBs to work properly, with the privilege not only to read system views but also to make changes to the table structures.

**Replication tool:** The Shareplex were installed on each VM host using the common accounts on both the UNIX and Oracle DB. They share the same names for their installation directories, as well as privileges in the OS and DB. There are three main directories to store the relevant files and attributes: 1) product directory for the software binaries and libraries, 2) variable directory for the dynamic or volatile data, logs and operational information and, 3) maintenance directory for the administration scripts. Both the source and target Shareplex instance operates on TCP port 2100 and under the specific hostname set as an environment variable in the start-up scripts. A configuration file has been created and activated at the source Shareplex to support the replication of the 10 tables from the source DB to the target via one queue.

**Network:** the two guests VMs reside on the common host, and they are configured to communicate via TCPIP protocol. The addresses used by the VMs are static IP addresses that belong to the 192.168.1.X subnets, and they can reach each other on the internal network. The next network connectivity required is between the DBs and via TNS resolution and managed by the listener. For each of the installed Oracle's

binaries, two network configurations need to be set. The first binary is the listener.ora which resolves incoming connection requests to the listener for the Oracle DB. The second binary is the tnsnames.ora which informs the oracle client of the address where the DB's listener is at. On each VM, the Shareplex instance uses the Oracle client to connect to the local Oracle DB to perform its required services and tasks. It does not directly connect to the remote DBs using the TNS connectivity, but it is also important that both DBs can contact one another through the Oracle network in this experiment.

Operating system: Both VMs are running on Oracle Linux and their internal activities, such as CPU, memory, and storage, are monitored. While hardware failure is considered catastrophic, therefore it falls beyond the scope of this thesis, some other aspects like excessive usage of the CPU, memory or storage volume are of significant concern. This is because they cause both software to seize up or suffer significant delays. Furthermore, part of the OS's security involves the management of user and group accounts, and each software requires access to the OS to operate their service. Therefore, they need to be assessed in the event of locked user accounts or changes to permissions.

Table 7- Faults induction and restoration on DRE software's component services (service status flag: 0 – good, 1 – faults)

Software	Component/Services	Target for Faults	Fault inducing action	Service restoring action
Databases	Memory process	PMON, SMON processes availability	Kill off PMON process Kill off SMON process	Start oracle instance (which start both PMON and SMON)
	Status	DB operational and service status	Shutdown and start in mount mode	Open DB for use
	Account security	DB's System and splx accounts' status. Splx has quota on splx tablespace	Lock up system and splx DB account. Splx user has no quota on tablespace to write	Unlock system and splx DB user account Splx has quota to write on tablespace
	DB storage space	Amount of free space in system and splx tablespaces.	Shrink tablespace to 100% full	Increase tablespace space to have 20% of free space
Shareplex replication	Mmain processes	Shareplex main processes availability. Sp_cop, Capture, Read, Exp, Imp, Post processes	Kill off individual processes	Restart sp_cop to resume all processes
	Queues' operation	Capture, Export, Import, Post and Read's queues	Stop the queues' operations	Start the queues' operations
	DB accessibility	DB connection using splx Unix account from current and opposite VM hosts	Lock DB user account	Unlock DB user account
Network connectivity	Oracle listeners	Source & target Listeners Source & target host connect to target DB via sqlplus	Stop the listener process to stop user from connecting to on-site DBs	Start the listener process to allow user to connect to on-site DBs
	Oracle network files	Essential files availability; tnsnames.ora, listener.ora	Delete off network files	Restore network files
	VM hosts	Each VM host can reach the opposite node	Disable sshd service	Enable sshd service
Host OS	Unix account status	splx and oracle's Unix accounts	Lock the Unix user accounts	Unlock the Unix user accounts
	Essential OS system files	Essential Unix files like /etc/hosts	Delete the /etc/hosts file	Restore /etc/hosts file
	Network card status	Network service on enps03network cards on both hosts	Disable network card	Enable network card

## 4.9.2. The experimental set-up

The objective of the test is to ascertain the accuracy of the FD module in detecting and diagnosing the DRE fault. This is done by simulating a series of software component faults that impact the DRE's systemic service. The outage of a specific software service has repercussions for other software component functionality, leading to a partial or complete service outage among the software in the DRE. Each of the specific simulated faults is induced by running one or more commands either against the OS or through their interactive utility. The FD module will interact with the DRE under these fault simulations to build up its competency and its diagnosing competency will be assessed after its training routines. Table 7 below lists the various DRE software's core services or components, the intended functionalities to target, together with the corresponding actions to induce and restore their specific faults. The experiments are run on two VM running on Linux OS and both have Oracle DB and Shareplex installed on them. Each VM has 4GB of RAM and 100GB of hard disk storage. The OS of the Shareplex is Oracle 12 Enterprise edition, version 9.1. The network protocol that both VMs use is TCPIP.

Table 8 - Detailed Test for Fault Diagnostic module, with break-fix routines

Software	Function	Attributes	Commands	Break	fix
Databases	memory process	PMON SMON	Ps -ef grep smon grep DB1 grep -v grep Ps -ef grep smon grep DB2 grep -v grep	export ORACLE_SID=DB1 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba" export ORACLE_SID=DB2 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	export ORACLE_SID=DB1 && echo "startup;" sqlplus -s "sys/password as sysdba" export ORACLE_SID=DB2 && echo "startup;" sqlplus -s "sys/password as sysdba"
	Status	DB's mode is open, not restricted DB available for use	Select open_mode from v\$database;		
	Account security	System, splx account not locked Splx has quota on splx tablespace	Select username, account_status from dba_users where account_status='OPEN'; Select username, tablespace_name, max_bytes from dba_ts_quotas where username='SPLX';	echo "Alter user splx account lock;" sqlplus -s "system/password@DB1" echo "alter user splx quota 0 on USERS;" sqlplus -s "system/password@DB1"	echo "Alter user splx account unlock;" sqlplus -s "system/password@DB1" echo "alter user splx quota unlimited on USERS;" sqlplus -s "system/password@DB1"
	storage space	System Tablespace's free space >20% Splx tablespae's free > 20%	--refer to script	??	Alter tablespace USERS datafile 'XXX' autoextend on 100m; (??)
Shareplex	main processes	Sp_cop is running Exp, imp,post are running	ps -ef grep sp_cop grep -v grep ps -ef grep sp_ocap grep -v grep ps -ef grep sp_opst_mtl grep -v grep ps -ef grep sp_xport grep -v grep ps -ef grep sp_ordr grep -v grep ps -ef grep sp_import grep -v grep	echo password su - splx -c \$MMDIR/shutdown.sh	echo password su - splx -c \$MMDIR/startup.sh
	queues operation	Capture, export, import, post, read status = Running	echo "show capture" sp_ctrl echo "show export" sp_ctrl echo "show import" sp_ctrl echo "show post" sp_ctrl echo "show read" sp_ctrl	echo "stop capture" sp_ctrl echo "stop export" sp_ctrl echo "stop import" sp_ctrl echo "stop post" sp_ctrl echo "stop read" sp_ctrl	echo "start capture" sp_ctrl echo "start export" sp_ctrl echo "start import" sp_ctrl echo "start post" sp_ctrl echo "start read" sp_ctrl
	DB accessibility	Source Splx can connect to source DB Target Splx can connect to target DB	splx@host1> echo "select 1 from dual;" sqlplus splx/splx@DB1 splx@host2> echo "select 1 from dual;" sqlplus splx/splx@DB2	export ORACLE_SID=DB1 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba" export ORACLE_SID=DB2 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	export ORACLE_SID=DB1 && echo "startup;" sqlplus -s "sys/password as sysdba" export ORACLE_SID=DB2 && echo "startup;" sqlplus -s "sys/password as sysdba"
Network connectivity	Databases listeners	Source Listener is running Target listener is running Source host connect to target DB via sqlplus Target host connect to source DB via sqlplus	oracle@host1> ps -ef grep lsnr grep -v grep oracle@host1> tnsping DB2 oracle@host2> tnsping DB1	\$ORACLE_HOME/bin/lsnrctl stop #break listener without proper db entries Rm \$ORACLE_HOME/network/admin/listener.ora_bak mv \$ORACLE_HOME/network/admin/listener.ora listener.ora bak	\$ORACLE_HOME/bin/lsnrctl start #restore listener services cp \$ORACLE_HOME/network/admin/listener.ora.orig listener.ora lsnrctl restart



				<pre>Lsnrctl restart #break tnsnames.ora Rm \$ORACLE_HOME/network/admin/tnsnames.ora_bak mv \$ORACLE_HOME/network/admin/tnsnames.ora tnsnames.ora_bak</pre>	<pre>#restore tnsnames.ora mv \$ORACLE_HOME/network/admin/tnsnames.ora_orig tnsnames.ora</pre>
	Replication tool	Socket_test from source to target Socket_test from target to source	<pre>splx@host1&gt; ssh host2 splx@host2&gt; ssh host1</pre>	<pre>Host1&gt;Echo "password"   sudo systemctl stop ssh.service Host2&gt;Echo "password"   sudo systemctl stop ssh.service</pre>	<pre>Host1&gt; Echo "password"   sudo systemctl start ssh.service Host2&gt; Echo "password"   sudo systemctl start ssh.service</pre>
	VM hosts	Host 1 can ping host 2 Host 2 can ping host 1	<pre>oracle@host1&gt; ping host2 oracle@host1&gt; ping host1</pre>	<pre>Host1&gt; ifconfig enpsp03 down Host2&gt; ifconfig enpsp03 down Host1&gt; echo password  su -c "shutdown -h now" Host2&gt; echo password  su -c "shutdown -h now"</pre>	<pre>Host1&gt; ifconfig enpsp03 up Host2&gt; ifconfig enpsp03 up Manual power on host1 or host2</pre>
Host OS	unix account status	Unix account are open & not locked on host1 Unix account are open & not locked on host2	<pre>root@host1&gt; passwd --status splx root@host1&gt; passwd --status oracle root@host2&gt; passwd --status splx root@host2&gt; passwd --status oracle</pre>	<pre>root@host1&gt; passwd -l splx root@host1&gt; passwd -l oracle root@host2&gt; passwd -l splx root@host2&gt; passwd -l oracle</pre>	<pre>root@host1&gt; passwd -u splx root@host1&gt; passwd -u oracle root@host2&gt; passwd -u splx root@host2&gt; passwd -u oracle</pre>
	storage space	/ has free space > 10% on host1 / has free space > 10% on host2	<pre>oracle@host1&gt; df -h  grep /u01 awk '{print \$5}' oracle@host2&gt; df -h  grep /u01 awk '{print \$5}'</pre>	<pre>--not tested</pre>	<pre>--not tested</pre>
	network card status	Ifconfig enps03 is up on host1 Ifconfig enps03 is up on host2	<pre>oracle@host1&gt;ifconfig enps03 grep -i up oracle@host2&gt;ifconfig enps03 grep -i up</pre>	<pre>Host1&gt; ifconfig enpsp03 down Host2&gt; ifconfig enpsp03 down</pre>	<pre>Host1&gt; ifconfig enpsp03 up Host2&gt; ifconfig enpsp03 up</pre>
	memory utilization	Vmstat < 95% used on host1 Vmstat < 95% used on host2	<pre>oracle@host1&gt; free  grep Mem  awk '{print \$4/\$2 * 100.0}' oracle@host2&gt; free  grep Mem  awk '{print \$4/\$2 * 100.0}'</pre>		<pre>???</pre>

Table 8 lists the details for the fault inducing and resolving routines for all the DRE's software various services and components. For the DBs, the simulated faults will impact Oracle's primary memory process such as SMON and PMON. A failure in one of these processes will cause the DB service to stop. The script will then perform a root level kill to simulate the DB outage and a start-up command via DB's admin level is required to restore it.

Another feature is the operational status of the DB. If the database is open, then users can log in and interact with it. However, if it is brought down to mount mode, then the DB service will no longer be available. A script will start the DB in mount mode, and another will change it to open. The user account, SPLX, that the Shareplex requires for interacting with the DB is critical to the overall operation. If the account is not open or locked, then Shareplex's replication service can no longer function. The fault-inducing and correcting scripts will modify the user account status to be in open or locked mode.

The Shareplex also require a user account to have a list of DB level privileges to function, so scripts that simulate the absence and presence of these privileges were prepared. Likewise, for the schema objects that the user account owns and access, the Shareplex creates a list of DB objects under the user account during installation and continues to use them for its operation. Should there be any changes to their

accessibility to the user account or the validity of the object, the Shareplex will malfunction. Scripts were written to simulate this error. Another factor to note is the availability of free space within DB for the Shareplex to operate on. If there is insufficient space, the Shareplex will not be able to write data into the DB, resulting in the suspension of its service. Scripts that constrict and free up the storage space were written.

For the Shareplex's fault simulation, it follows a similar pattern to the DB, with the focus on their instance's primary processes that run on the OS. Their service disruption and restoration are done by scripts that execute system-level commands against their console. Similar actions are performed against the Shareplex queues in altering their status and operations for inducing and reverting the faults. The Shareplex also needs to be able to connect to the DBs from different nodes in the DRE setup, and this is done via network and oracle's essential network files setup. The fictitious faults on the setup are simulated with scripts to disable and enable the network cards, remove, and reinstate the tnsnames.ora and listener.ora, as well as shutting down and starting up the listener processes.

For the network inter-connectivity, there are two main areas in which the fault can be induced for this setup: 1) the connection via the TCPIP protocol at the OS level between the two VM hosts and, 2) the ability of the software's client to connect to the current and remote DBs through the oracle's network grid which comprises of listener services, OCI library, and oracle-related network files setup. The scripts that perform the opposing functions of fault induction and restoration target the network card's status, the listener process availability and status, the presence and validity of the network configuration files, as well as the OS' network files under the /etc folders.

Finally, in the OS, the emphasis is on 1) the Unix user accounts that Oracle and Shareplex need to use throughout their services, 2) the availability of free space on the disk partitions that their home and operational directories are installed on, 3) the resource availability in the OS which both Oracle and Shareplex can operate under and 4) status of the network card. In the first group, their scripts that can lock up and revert the Unix account's status were written. In the second group, a script to simulate an error by changing the permissions of the Unix accounts, thereby suspending their ability to write, was written. Finally, in the third group, a script was written to shut

down or restore the network connectivity services by disabling and re-enabling the network card. For each of the software's core functionalities, two of its attributes will be assessed and a metric measuring their service normality is associated with them. A value of 0 indicates a normal state whereas  $>0$  indicates an abnormality. The following tables 9 and 10 lists all the software components and the respective commands that can simulate and restore their faults.

These tests do not include malicious or terminal faults to the software as they are either irreversible or require a substantial amount of effort to restore. Examples of such faults are the corruption or deletion of the software's binaries or libraries, deletion of DB's repository, file-based data store and erasure of OS' disk mount-point. The neural network that the RL used for its rewards-action prediction is made up of 3 hidden layers of 30 nodes. It is trained with data in 50 batches and 500 epochs. Different configurations and combinations of neural networks have been tested, and this setup was selected based on the better results with the least fluctuations.

### **4.9.3. True Negative test results**

Apart from the data obtained from the faults-inducing scripts in the previous section, another group of scripts have been created to induce software faults that have no impact on their DRE's software functionalities and services. This is to form a set of true negative data to support and enrich the dataset for the NN's training so that the NN can be competent enough to recognize the environment's state data that can cause service disruption.

For the script to induce this group of faults, research has been made across the DRE's software to identify faults that have a high chance of occurring, but do not have a direct consequential effect on the entire software's stability or create an outage on the DRE's functionalities. This is verified by the SD module which confirms the presence of a service disruption. For this group, the service disruption matrix values should all be zero. Once these faults are induced, the software will capture their exceptions and events in their event or trace logs, which in turn are detected by the FD module. Table 9 lists the faults on the DRE software that are considered to have no direct impact on the DRE's services.

#### 4.10. Evaluation Criteria and Benchmarking

This section describes how the FD module is evaluated and the criteria used in its assessment. The faults statistics cover the four main DRE's software: Database, Replication, Network, operating system, and service level. They are represented by a vector with each element representing the service. Within each element is a scalar value from 0 to 1. Values  $>0$  indicate the presence of a fault, with greater values related to greater severity of the fault, while a value of 0 means that all components are operating normally.

The vectors form the basis for the primary evaluation criteria. The statistical differences among fault diagnosis of DRE's states can indicate the progress of the DRE's overall service. Each diagnosis is correlated to the detailed diagnostic statistics that were generated by the FD module which will be vital for the next module of fault resolution. There are two groups of evaluation criteria for the FD's diagnosis: the quantitative and qualitative criteria. The quantitative criteria measure the level of the severity value of the faults under the software's service group within a normalized range, between the prediction and the actuals. The mean square error test applies to this group. The qualitative criteria are on the group classification of the software's service's faults. It is regarded as a binary classification too; with zero indicating normal operation, and values greater than 0 to indicating the presence of faults. The binary classification test applies to this group, and it measures the performance in terms of 1) sensitivity: the measure of how good the model is in detecting the positives, 2) Specificity: if it can avoid the false positives, and 3) Precision: the number of True Positive it can find that are relevant. A receiver operating characteristic (ROC) graph is plotted between the sensitivity and specificity to evaluate the quality or performance of the diagnostic tests. The formula for the statistical measurement of the FD's classification test is listed in eq (14). The prediction results are also summarized into a Confusion matrix.

$$\text{Sensitivity/recall} = \text{sum (TP)} / [\text{sum (TP)} + \text{sum (FN)}] \quad (14)$$

$$\text{Specificity} = \text{sum (TN)} / [\text{sum (TN)} + \text{sum (FP)}]$$

$$\text{Precision} = \text{sum (TP)} / [\text{sum (TP)} + \text{sum (FP)}]$$

where TP is True-Positive, FP is False-Positive, TN is True-Negative, FN is False-Negative.

For the quantitative assessment, the accuracy of the FD module’s DRL-NN prediction is measured against the output from the SD module. Both vectors’ difference is calculated using the Mean Average Square Error (MASE) to assess their accuracy. This is used to evaluate if the NN needed to be re-organized and optimized for improved performance.

#### 4.10.1. Test results

This section describes the results obtained from the FD module after it has completed the training and is subjected to the evaluation test processes. By this stage, the FD module has been trained thoroughly and it is regarded to have achieved the expert level of fault diagnostic capability. The minimum expectation of its prediction accuracy internally is expected to reach 85% accuracy or greater. A sample of the DRE’s states, including both the predicted and actual service outage results, is shown in Table 9.

1) The DRE state data is derived from the information gathered against the DRE’s software components from their logs, internal system statistics, and monitoring after a fault is simulated.

2) The FD module predicted the service outage results after it has received the DRE input based on its learned NN.

3) The SD module produced detailed results by running a list of diagnostic routines against the DRE environment to derive and aggregate the actual statistics.

4) The classification of the outage results is derived by comparing the sum of the predicted results’ values against the actual service outage results.

5) The MASE score is calculated based on the difference in the vectors’ values between the predicted and actual results. Table 10 shows the raw output from the simulated test.

Table 9 - Results of service outage prediction & scores against DRE’s state

DRE State	Service outage Predicted	Service outage Actual with rounding	Classes	MASE
[0,64655058,76223968,0,0,0,0,64351381] [1,0,1,0,0,0,0] [1,0,0,0,0,0,0]	[[6,1,0,1,3,1],[2,2,1,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[6,1,0,1,3,1],[2,2,1,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.6

[46968001,0,0,0,0,0,64351381] [0,1,0,0,0,0,0] [0,1,0,0,0,0,0]	[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.3
[46968001,0,0,0,0,0,64351381] [0,1,0,0,0,0,0] [0,1,0,0,0,0,0]	[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.2
[46968001,0,0,0,0,0,0] [0,0,1,0,0,0,0] [0,0,0,0,0,0,0]	[[6,1,0,1,3,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[6,1,0,1,3,0],[0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.1
[46968001,0,0,0,0,0,64351381] [0,1,0,0,0,0,0] [0,1,0,0,0,0,0]	[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.6
[0,0,0,0,0,0,0,64351381] [0,1,0,0,0,0,0] [0,1,1,0,0,0,0]	[[2,0,3,0,0,1],[2,2,1,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[2,0,3,0,0,1],[2,2,1,1,0,0],[1,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.0
[46968001,0,0,0,0,0,0] [0,0,0,0,0,0,0] [0,0,1,0,0,0,0]	[[4,0,4,1,0,1],[0,0,1,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[4,0,4,1,0,1],[0,0,1,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.3
[46968001,0,0,0,0,0,0] [0,0,0,0,0,0,0] [0,0,1,0,0,0,0]	[[4,0,4,1,0,1],[0,0,1,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[4,0,4,1,0,1],[0,0,1,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.0
[0,0,0,0,0,0,0] [0,0,0,0,0,0,0] [0,0,1,0,0,0,0]	[[3,0,4,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[3,0,4,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.1
[0,0,0,0,0,0,0] [0,0,0,0,1,1,0,0] [1,1,0,1,1,1,0,0]	[[0,0,0,0,0,2],[0,4,2,2,0,0],[3,2,1,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[0,0,0,0,0,2],[0,4,2,2,0,0],[3,2,1,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.2
....	....	....	...	....
[0,0,0,0,0,0,0] [0,0,0,0,1,1,0,0] [1,1,0,0,1,1,0,0]	[[0,0,0,0,0,2],[0,4,2,2,0,0],[2,2,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[0,0,0,0,0,2],[0,4,2,2,0,0],[2,2,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.0
[58729172,0,0,0,0,0,0] [0,0,0,0,0,0,0] [0,0,0,0,0,0,0]	[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0]]	[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TN	0.2
[0,0,0,0,0,0,0,28387490] [0,0,0,0,0,0,0] [0,0,0,0,0,0,0]	[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TN	0.0
[34823972,58729172,0,0,0,0,0] [0,0,0,0,0,0,0] [0,0,0,0,0,0,0]	[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TN	0.1
[0,82736461,0,0,0,0,0] [0,0,0,0,0,0,0] [0,0,0,0,0,0,0]	[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	[[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TN	0.1

Table 10 – Outputs from SD’s simulated tests

```

*** break **** export ORACLE_SID=DB1 && echo "shutdown immediate;"|sqlplus -s "sys/password as sysdba"
DRE log stats,DRE proc stats,DRE srvc stats = [0,64655058,76223968,0,0,0,0,0] [1,0,0,0,0,0,0,0] [1,0,0,0,0,0,0,0]
Diag service fault= [[0,0,0,1,0,1],[2,2,1,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,1,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0]
*** break **** export ORACLE_SID=DB2 && echo "shutdown immediate;"|sqlplus -s "sys/password as sysdba"
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,1,0,0,0,0,0,0] [0,1,0,0,0,0,0,0]
Diag service fault= [[0,0,0,0,0,1],[2,2,1,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0]
*** break **** echo "Alter user splx account lock;"|sqlplus -s "system/password@DB1"
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0]
Diag service fault= [[0,0,0,1,0,1],[0,0,1,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]
*** break **** echo "alter user splx quota 0 on USERS;"|sqlplus -s "system/password@DB1"
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0]
Diag service fault= [[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
*** break **** echo password|su - splx -c $MDIR/shutdown.sh
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0] [0,0,0,0,0,0,0,0]
Diag service fault= [[6,1,0,1,3,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
*** break **** echo "stop capture"|sp_ctrl && echo "stop export"|sp_ctrl
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0]
Diag service fault= [[3,0,2,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
*** break **** echo "stop capture"|sp_ctrl && echo "stop import"|sp_ctrl
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0]
Diag service fault= [[3,0,2,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
*** break **** echo "stop capture"|sp_ctrl && echo "stop post"|sp_ctrl
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0]
Diag service fault= [[2,0,2,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,1,1,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
*** break **** echo "stop capture"|sp_ctrl && echo "stop read"|sp_ctrl
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0]
Diag service fault= [[2,0,3,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,1,0,0,1,0,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
*** break **** echo "stop export"|sp_ctrl && echo "stop import"|sp_ctrl
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0]
Diag service fault= [[2,0,2,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,0,0,1,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

```

```

*** break **** echo "stop export"|sp_ctrl && echo "stop post"|sp_ctrl
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0]
Diag service fault= [[3,0,2,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,0,1,1,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
*** break **** echo "stop capture"|sp_ctrl && echo "stop export"|sp_ctrl && echo "stop import"|sp_ctrl
DRE log stats,DRE proc stats,DRE srvc stats = [0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0]
Diag service fault= [[3,0,3,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE diagnose metrics= [0,1,0,1,0,1,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
.....

```

**4.10.2. Service outage classification results**

The test is conducted with 80 fault inducing scripts. 30 of them have a direct effect on the software’s functionalities which impact the DRE’s software services, and 50 of them do not. It is expected that the FD module can predict accurately for both groups. The results are split into qualitative and quantitative groups. Table 11 is the tabulation of the prediction’s result classes in a confusion matrix. The results showed that the SD module can predict the group of service outages concerning the information received from the DRE’s environment. While it has a high capability in recognizing most of the induced faults that can affect the DRE’s software functionalities, it fares less well when it comes to the detection of those in the other groups. Based on the result, the FD’s sensitivity is 0.87, specificity is 0.98, precision is 0.871. The SD module has been shown to be accurate enough that its prediction can produce the correct category of service outage for the given environment state’s data input. Compared to the other published research works, the FD’s results are comparative in acceptable term for the respective domain of application [68-70]. It has the competency to differentiate if the inputs are related to DRE’s service functionalities. In the next section, the accuracy of true-positive predictions is discussed.

Table 11- Confusion matrix of the classification of the service outage’s prediction

N=80	Predicted: Yes	Predicted: No
Actual: Yes	27(TP)	4(FN)
Actual: No	1(FP)	49(TN)

**4.10.3. Service outage prediction accuracy**

For this test, The SD module forms the baseline against which the FD’s predictions are measured against. Each value in the service outage results produced by both the FD and SD is calculated using the MSE approach, and they are summed up

to form the total overall degree of accuracy for the SD. The results are shown in the chart in figure 25. Based on the results, the accuracy is  $< 0.3$  except for one entry that scored 0.6. This can be attributed to the Fault diagnosis being DRL based and that it learns adaptively with the environment. While it is experienced to recognize the fault scenario that it was trained for, it will have some deviations for new and unfamiliar ones. One solution is to enable more iterations of exposure for the SD module's DRL to learn more about the true positive and negative of DRE's scenario, although the list of potential faults that can affect DRE is controlled and limited. Another possible solution is to expose the NN to the more true-negative class scenario which does not impact the DRE. This has more potential to be generated in greater volumes and can assist in enriching NN's training dataset.

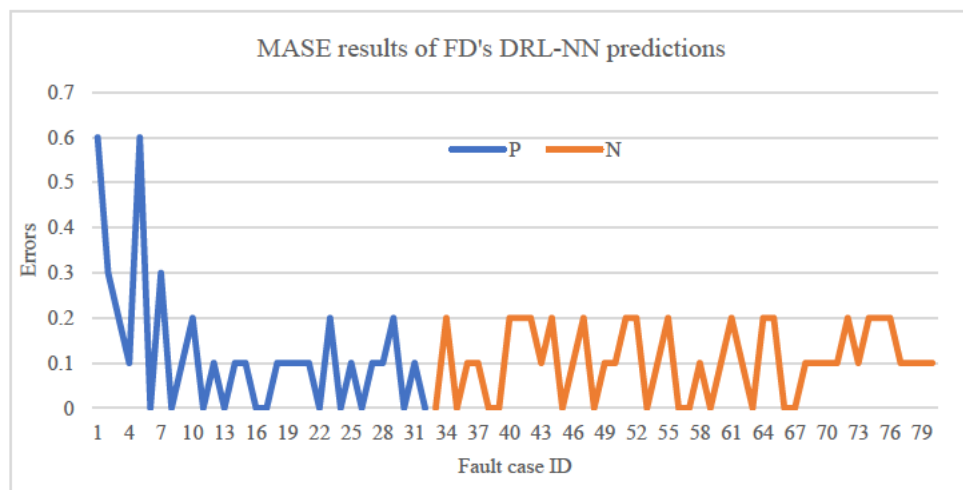


Figure 25 – MASE score of True and positive predicted results

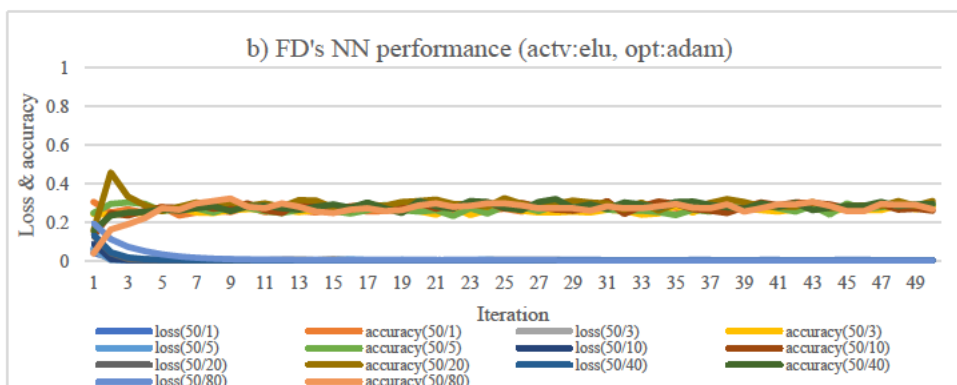
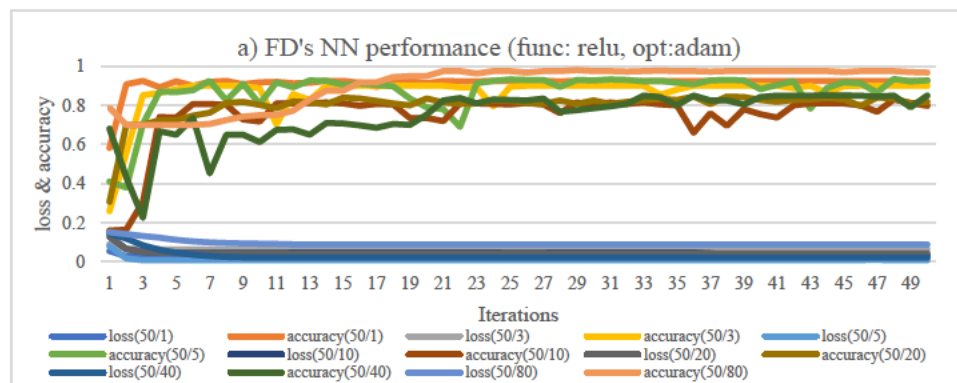
#### 4.10.4. FD's DRL-NN performance results

The FD's learning phase stretched through 3 phases and during the first two, it relies on the SD to do most of the work in gathering and aggregating the DRE information. Once it has obtained enough of it, it will rely on the NN to carry on the work through the NN's prediction. The advantage of this approach lies in the decrease in the turnaround time of the service outage derivation for any DRE's faulty state, including producing associated system diagnostic information from the knowledgebase without going through the laborious SD's routine.

But before using the NN within the DRL, a short exercise is required to ascertain and find the optimum combination of the NN's hyperparameters with the given dataset



from the knowledgebase. The library to be used is python's neural network library, Keras, and it is a requirement that it should have a productive model that can perform well against the current dataset with accuracy  $> 80\%$ . Each of its hyperparameters has a variety of options, and their combination can have a big influence on the NN's accuracy. Starting with the optimizer which is the iterative learning algorithm that governs and controls how the internal parameters of the NN are being optimized against a performance measure such as MSE, using the dataset to train and update the model. Optimizers used are stochastic gradient descent (SGD) and Adaptive moment estimation (Adam). The next hyperparameter is the batch size and this controls the volume of training sample data to be used in conjunction with the internal parameters updated within the model. Epoch is another important hyperparameter of gradient descent that regulates the number of passes that the training dataset is used. The next one is the activation function which is an internal mechanism of the NN's neurons and decides what to activate and the range of values it should do based on the data input. There are many types of activation functions but for this test, the more common ones such as ReLU, sigmoid and Elu are used.



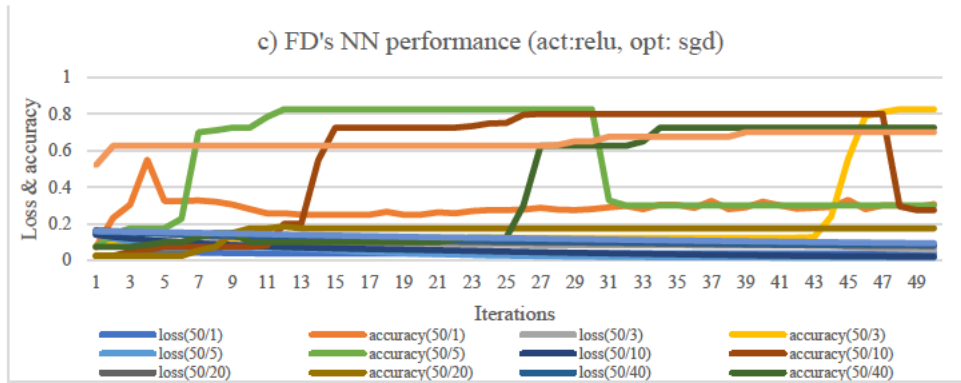


Figure 26 - Accuracy results of the SD's DRL-NN with different epoch/batch size

The combination of the hyperparameters used on the NN against the dataset which are acquired from the SD running against a set of simulated faults, mixing different options of epoch, batch size, optimizers and activation function to derive the following results are plotted in the three charts in figure 26. Other minor hyperparameters, dropout rate, initializing factor, the learning rate of the optimizers and others, that can contribute to the NN's performance will not be considered at this point. Figure 26b showed that the combined hyperparameter of using Elu for activation function and Adam for optimizer did not fare well, yielding results with low accuracy. The other two that used ReLU for activation function and Adam or SGD for optimizer did better, although the accuracy only improved from the 50-epoch cycle with the batch size smaller than 10. Figure 26c has similar achievement, but the results are not as stable and adjustments to both batch size and epoch did not contribute much to the NN's performance stability. The number of hidden layers used is 3 with each hidden layer having 100 neurons inside. Both the number of neurons handling the input and outputs are set at 24 and 30, respectively. Based on the results and chart, it is decided that the optimal hyperparameters' settings for the NN model in the FR module are Exponential Linear Unit (ELU) activation function, Adam optimizer, batch size of 30 and 20 for epochs.

Figure 27 shows the test result from the simulated faults against the DRE. For every fault test case, a series of faults are executed to break some of the functionality of the software element and followed by the SD to gather all the data against the DRE. Once this is completed, the injected faults are reversed out. For the DRL-NN test, the SD part must be completed to build up the necessary knowledgebase which in turn is used as the dataset for the NN training. The faults are simulated but once the test is

detected, the input from the DRE is received by the DRL-NN and the service outage is predicted, bypassing all the diagnostic routines which the SD had to do. This shortens the turnaround time in deriving the service outage information. Figure 25 charted both group tests together to show the difference in time savings against a list of fault test cases. The stacked bar on the left for each case is the time taken by the SD to break and fix the faults with the diagnostics routines included. The bar on the right is the NN's results where the initial fixed are required to be executed but the timing taken by the NN is significantly lower. The fault reversal runs in the background after the DRL-NN diagnosis phase is complete. For the SD, it must validate all the checks after the fixes are applied, which explain the additional overhead in time duration.

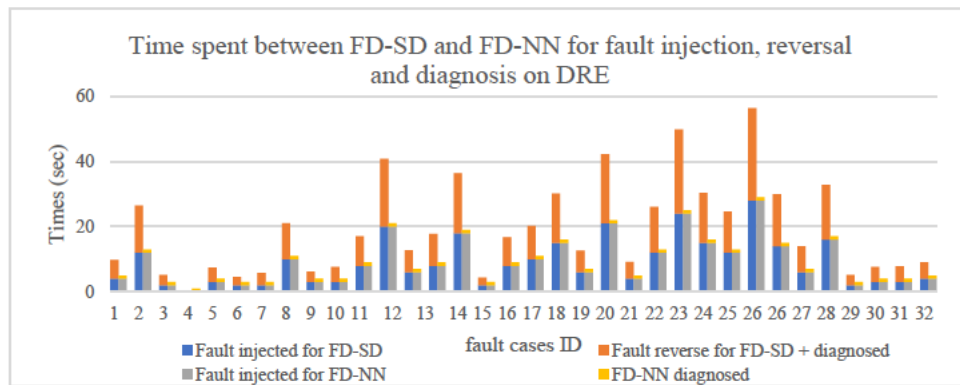


Figure 27 - Time spent between FD-SD and FD-NN for fault injection, reversal and diagnosis on DRE

#### 4.11. FD's conclusion

The FD module has been proven that it can produce the outcome of the service outage based on the DRE's state information with good accuracy. It made use of the model-free actor-critic Deep Reinforcement learning to learn against the Data replication setup to predict the outage gradually as it interacts with it and learns with the help of the SD module that corrects its prediction. This is to train the FD incrementally over time with minimum or no a-prior knowledge of the environment. The training was done with a set of scripts that induce non-fatal faults into the software in the Data replication environment and these faults are picked by the SD's module. Not all faults induced have direct impacts on the data replication operations, and this is to train the FD module to recognize and differentiate among those faults that can and cannot cause a direct impact on the DRE. With each induced fault, there is another script that resolves them and restores the state to its functioning baseline. The FD's DRL initially made a lot of mistakes at the beginning of the learning phase, but it gets

better as it progresses with the iterations of interactions produce more data for its NN's training minibatch and thereby, improves its prediction accuracy rate. By the end of its learning phase, it has achieved an expert level and is able to recognize the state input to predict the service outage. The results from the experiments had proven its capability. The entire content of this chapter has been published with the Australasian Database Conference 2020[71]. In the next chapter, the next module called fault restoration is discussed.

# CHAPTER 5: DESIGNING THE FAULT RESOLUTION (FR) MODULE

---

Chapter 5 presents the Fault Resolution (FR) module with its design architecture and operations, testing, and results analysis.

## 5.1. Adaptive Fault Resolution (FR) Module Design

The Fault Resolution (FR) module performs the act of resolving the faults against DRE after it receives the inputs from the SD module on the service outages, and it uses the same architecture as the FD module which has the objective of creating an expert medium that can able to decipher the possible software's service outages based on the DRE input without the need of running through a time-consuming, resource-intensive and fine-grained system diagnostics every time throughout its production's operation. It also serves as a repository of correlating the specific list of outages to the respective group of diagnostic information that has been gathered and curated, for the FR module to follow up. The FR module has a similar setup but serves another function, which is to decipher the service outage matrix to the required actions that can resolve the faults to restore the service. Once the FR module is trained fully, it can prescribe a series of corrective actions for the DRE's service outages that are obtained via the SD module. The DRE responds to the SD module with its new state. if the SD's analysis on the DRE and decides that it no longer acknowledge any DRE's related outages, it completes the fault diagnostics and resolution cycle. Else it will keep to the FR module for the next course of action.

The FR is based on the following outlines: 1) there is a finite number of corrective actions that can be taken to resolve a finite number of faults within the DRE. The DRE has several software require configuration setting perfectly tuned to interoperate harmoniously. So, a single fault can not only cause one direct outage, but it will affect other and create a cascading effect. Fixing the cascaded service faults requires some insights into the software element's attributes and function, where a series of appropriate actions is chosen by the FR module to restore the elements' function. 2) it may take one or more correction iteration before the faults can be resolved. it is

anticipated that not one single fault can be resolved with one corrective action and more may be required if there are generally multiple occurrences across different software elements. Some may or may not be interdependent. 3) in the event if the faults cannot be resolved, then the system should be notifying the IT administrators for assistance. This is for cases that have been regarded as outside the DRE's fault diagnostics and resolution scope or, the faults that are too critical that is beyond the capability of the FDR to manage which requires external expertise and intervention. 4) the FR module's architecture is adaptive and flexible for new changes and unfamiliar events that the FDR has no encounter before. While most of the scenario has been anticipated, some have not been covered. Therefore, in the event when an unplanned scenario occurs, the FR module can adapt to the new challenges. It learns to overcome them and update its knowledgebase, thus increase in its expertise. In its infancy stage of learning, this cycle will keep repeating until the FR module can match the appropriate set of corrective actions for the service outages that maps to the faults detected. It takes numerous cycles until it can find the best course of action. When the FR module reaches its maturity or expert level, it knows the exact actions to take for the DRE's fault and outages. Figure 28 shows an overview of the FR within the FDR system.

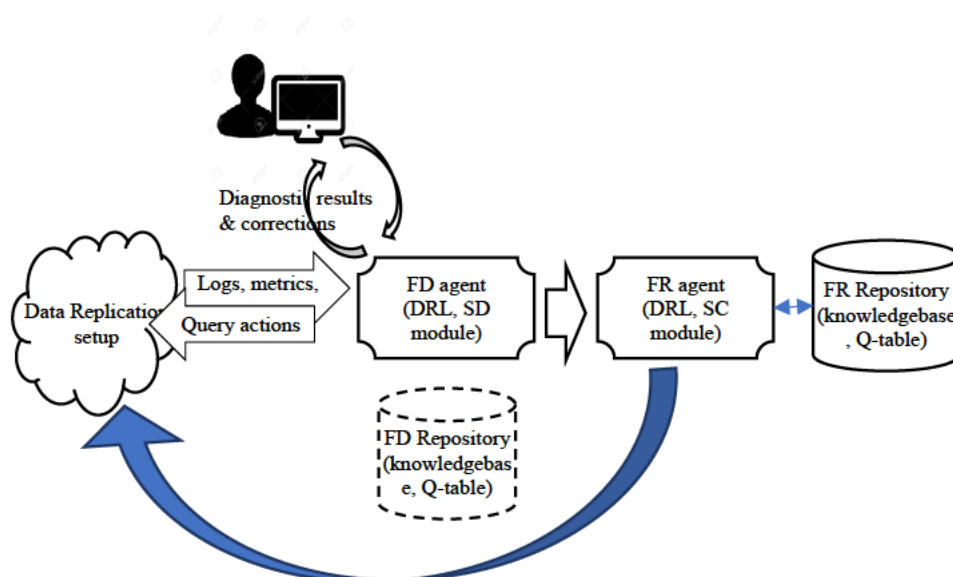


Figure 28– Adaptive Faults Resolution overview

### 5.1.1. Diagnostic Reinforcement Learning (DRL) for FR Module

Figure 28 shows the workflow of the FR module and how it works with the FD module on the DRE. It is like the FD design and it uses the Actor-Critic reinforcement learning model which is described in section 2.4.3 under chapter 2. After the FD module passed the service outage information as input to the FR which in turn perform a function approximator to predict the best corrective action based on equation 1. this forms the Actor part of the DRL model. Besides the FR module is another module called the System Correction (SC) module which looks up detailed system diagnostics information based on the service outage information and tries to determine the exact course of corrective actions to take. these actions are UNIX scripts that contain commands for each of the elements within the specific software that perform some changes, be it altering the configuration, modifying the parameters, reconstitute some files or alter the state of the processes. The SC module forms the DRL's critic which is used to validate the action of the FR as shown in figure 29. There are 3 phases of learnings for the DRL as depicted in figure 30.

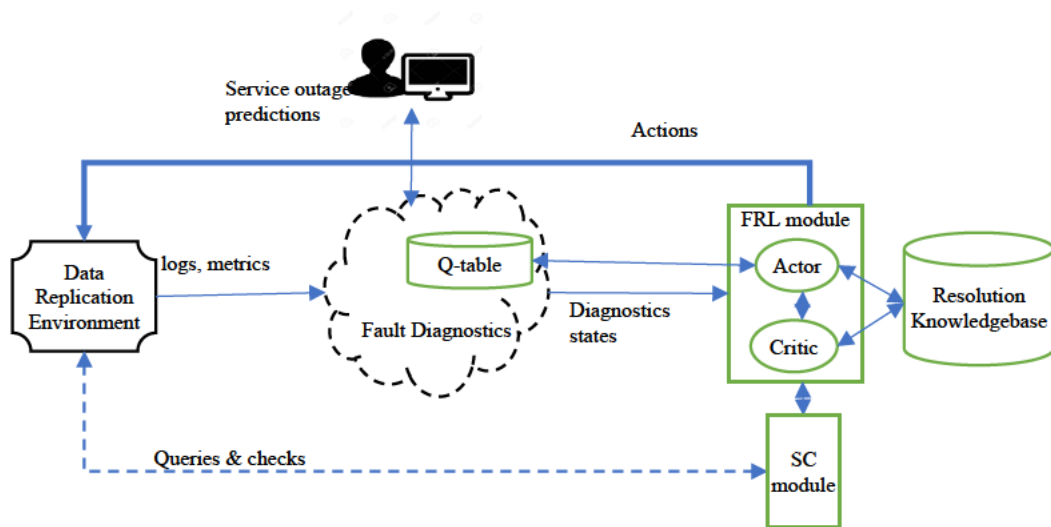


Figure 29 – Faults Resolution agent's architecture and workflow

1) Early learning phase; It starts with little or no a-prior knowledge on what actions to take for the given FD's service outage input. At this stage, the FR cannot predict anything useful, so it relies on the SC module to guide it. the SC took in the correcting diagnostics information in response to the service outage inputs and derive the list of possible corrective actions. The FR module then applies these actions onto the faulted DRE and receive a new state. if the fault or error persists, this routine will repeat until the problem is either resolved or the number of fault resolution attempts

has failed a limit of retries, then it alerts the IT administrator for assistance. Both the IT administrators' input and the SC's information are used for the correction and the details are added into the minibatch which is used in the next phase of learning.

2) Middle learning phase; After it has gathered enough knowledge between the service outage input and the fault resolution actions, the DRL's actor will use this information to train its NN and tries to predict the corrective actions for new service outages input. There is a high possibility that the predicted outcomes will have a high level of error, so the validation process will gauge between the predicted results and those from the SC module. The state-action relationship of both the service outage versus the corrective actions is assigned with a Q-value, and this determines the quality of the actions for a given state. As the knowledgebase grows, it is expected the minibatch for the NN training will be bigger and thus helps in its prediction accuracy.

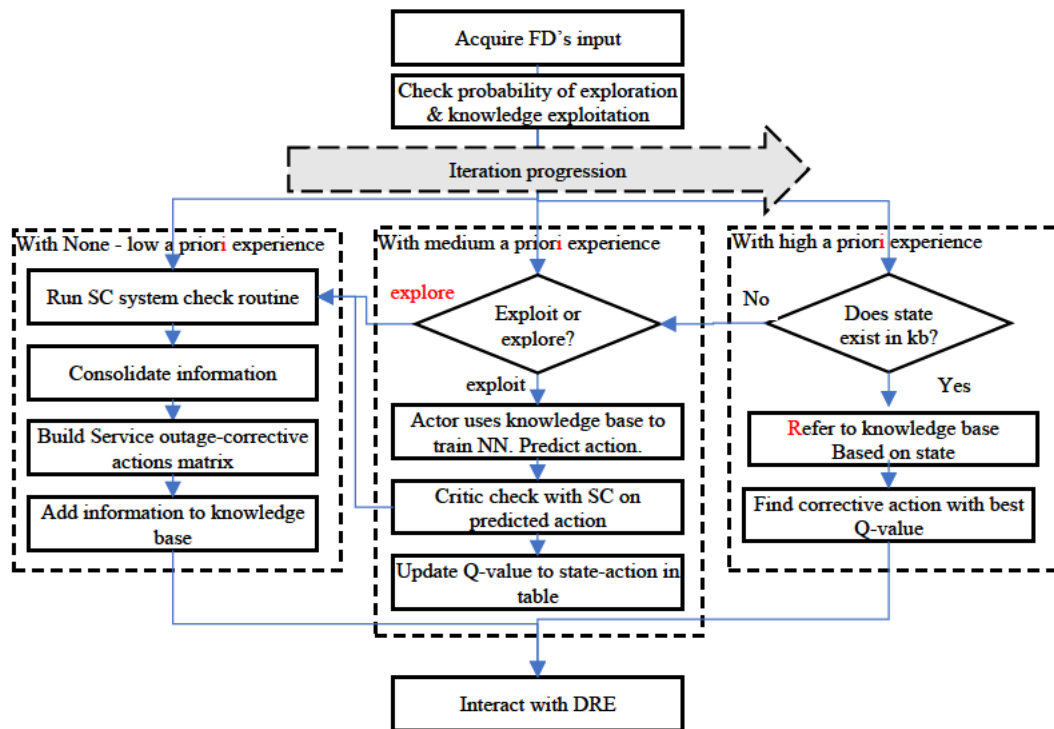


Figure 30 – DRL different phases of learning for the FR module

3) High learning phase; when it reaches this phase, the DRL within the FR module would have considered reaching an expert level. it would have known all the state-actions relations that are associated with the DRE and be competent enough to predict accurate corrective actions for it. This is regarded as the exploitation of the DRL's rich build-up of knowledge where it can provide a very quick turnaround time



in providing solutions quickly without invoking any action on the SC module. this is considered exploitation of its rich expert knowledgebase. However, it also performs a probability calculation to determine whether it should continue to rely on its knowledgebase or take a chance to explore for potential newer actions.

Like the FD module's DRL, the balance between exploitation vs exploration is mutually inverse. For the exploration phase, in this DRL setup, it is referred to as the use of FR and SC module to ascertain the corrective actions. for the exploitation phase, the module relies on its build-up knowledgebase as a reference. this is to minimize the chance of getting stuck in local optima. Through the learning phase from low, medium to expert, the probability of the exploration starts high whereas the exploitation rate is low. As the learning interaction goes, both phases decrease and increase respectively till they reach the end of the expert phase. By the end of the phase, the module favours the use of its knowledgebase instead of exploring new ones.

### **5.1.2. System Correction (SC) module**

To guide the FR's DRL in its path to find the appropriate action to respond to the information generated by the SD module, the system correction (SC) module serves as the guide. The SC's purpose is to search through its repository for the appropriate actions to correct the SD module's predicted service outages, using the service outage related system diagnostics statistics. The SC module serves only as the passive reference to validate the FR module's output and it doesn't play the proactive role as the intent of the FR module is to be trained up to a level that it knows the corrective actions to take for any given service outages event. For a certain software element's faults, there are multiple corrective solutions to use and at times, it requires more than one corrective action before their faults can be resolved. However, instead of randomly running through every combination of the corrective actions, the SC module chooses the appropriate actions, much like the equivalent of having an expert IT administrator guiding the junior on the appropriate action to take for a given identified fault.

Referring to figure 28, the SC module receives the FD module's service outage matrix, and it can use the knowledgebase to lookup for the corresponding system diagnostics detailed statistics. each entity in the system diagnostics statistics is related to a vector of corrective action (CA) for that software element's attributes and function. the correlations have pre-determined like an IT troubleshooting guide except that the

guide recommends a series of checks to determine the actions whereas the fault-correction matrix bypasses the checks and prescribe the actions. This is like an experienced IT administrator who knows what to do when he sees a certain faulty situation.

The SC module comprises a multitude of libraries of external OS-based commands that interact with the software elements and make changes. These libraries are comprehensive and are maintained in accordance with each software's groups domain such as Oracle DB, Shareplex, network and OS. each of the scripts that are intended to make the corrective changes have been crafted as a response specifically for each unique software element's functions and they are indexed for reference. The rationality of mapping the diagnosed faults to specific actions is derived from the fact that in any typical fault resolving scenario, the troubleshooting workflow passes the system information through various conditions and checks to decide whether a certain course of actions is to be taken and what specific commands or changes are needed. But when such system faults arise, the end goal is to use one or more appropriate corrective actions against the software element in the hope to rectify them or restore their function, it is a 1:n relationship between faults and corrective actions.

Examples of some of the actions incurred are; 1) altering the state of the system processes through start-up or shutdown, 2) increase space availability for the system's directories, 3) correcting the setting of the privileges of the system's process, files, accounts that they operate from, 4) setting the values of the configurations and parameters that they are using, operating or initialize from, 5) ensuring network card operations' status for network connectivity, 6) unlocking the user accounts or regranting the appropriate privileges, 7) restoring the original baseline copy of the system and network files onto the Unix's /etc folders and, 8) enabling the replication queues back to operation. This holds the flags of activation to match the library of corrective action scripts in a 1:m relationship as shown in the matrices in eq (15). This will be discussed in greater detail in the next section.

$$\begin{aligned}
 \text{Diagnostics matrix} & \begin{bmatrix} \text{oraf}_1 & \text{spxf}_1 & \text{nwf}_1 & \text{osf}_1 \\ \text{oraf}_2 & \text{spxf}_2 & \text{nwf}_2 & \text{osf}_2 \\ \dots & \dots & \dots & \dots \\ \text{oraf}_n & \text{spxf}_n & \text{nwf}_n & \text{osf}_n \end{bmatrix}, \\
 \text{corretive matrix} & = \begin{bmatrix} \text{oraV}_1 & \text{spxV}_1 & \text{nwV}_1 & \text{osV}_1 \\ \text{oraV}_2 & \text{spxV}_2 & \text{nwV}_2 & \text{osV}_2 \\ \dots & \dots & \dots & \dots \\ \text{oraV}_n & \text{spxV}_n & \text{nwV}_n & \text{osV}_n \end{bmatrix}
 \end{aligned} \tag{15}$$

Where, the *ora*, *spx*, *nw*, and *os* identify the software groups as Oracle DB, Shareplex, network, and operating system. The suffix, *f*, identifies the specific software's diagnosed faults and, *V*, refers to the corrective actions vectors that have *m* dimensions. Following the Actor-Critic Reinforcement learning model outline which is described in section 2.4.3 under chapter 2, the state, *s*, is the diagnostics matrix while the corrective matrix is the action, *a*. The reward, *r*, is the number of faults that the action, *a*, can resolve.

### **5.1.3. Representation and correlation of diagnosed faults to corrective actions**

For the FDR design, a software element can have multiple types of faults and there is a list of corresponding corrective actions. Starting from the FD module, it produced the predicted outcome of the service outage (SO) information of the DRE's state for both the users and the FR module. Each SO has its corresponding System Diagnostic (SD) statistics which has all the specific errors found. The FR module is then based on the SO information to predict the course of corrective actions for the troubled DRE, and the corrective actions are obtained from the external library that has a list of pre-built system commands and OS scripts for the various software elements. It is important to map each specific software elements' fault to those corrective actions that have been predetermined to restore their function. For example, the Oracle DB user account may have been locked or lack the system privilege, so the appropriate list of actions is a multitude of commands that range from unlocking the account, granting additional space quota, granting system privilege, to recreating the account. Table 12 illustrates the relationship between the two state-space of diagnosed faults and corrective actions.

Sometimes, a single software entity outage can cause multiple faults. For example, Oracle instance outage can attribute to other problems such as loss of database to the Shareplex, inability to read Oracle DB's logfiles by the Capture process, and Oracle DB's account checks. The table below illustrates this complex relationship hypothetically. The level of inter-correlation is high but for this thesis, the scope is narrowed down to the major and more significant form of changes that the corrective actions are developed for, which involves in services' start-stop, parameters, and configuration changes, plus privilege and resources allocation. The reduction in

the scope is to improve the manageability of the proposed system’s complexity as well as prediction’s reliability.

However, a future enhancement to this thesis can include a more complex route to enhance its capability. All these corrective actions’ scripts have been crafted through numerous consultations with experienced IT experts and technical research.

Table 12 – Example of diagnosed faults correlation to corrective actions

Corrective actions Diagnosed faults		Service fault#	OracleDB			Shareplex			Network			Linux OS		
			Action1 (script1.cm.d)	Action2 (script2.cm.d)	Action3 (script3.cm.d)	Action1 (script4.cm.d)	Action2 (script5.cm.d)	Action3 (script6.cm.d)	Action1 (script7.cm.d)	Action2 (script8.cm.d)	Action3 (script9.cm.d)	Action1 (script10.c.m.d)	Action2 (script11.c.m.d)	Action3 (script12.c.m.d)
OracleDB	Fault1(process, services)	2	1	0	0	0	0	0	0	0	0	0	0	0
	Fault2(operation, privileges)	3	0	0	1	0	0	0	0	0	0	0	0	0
	Fault3(configuration, parameters)	4	0	0	0	0	0	0	0	0	0	0	0	0
Shareplex	Fault1(process, services)	2	0	0	0	1	1	0	0	0	0	0	0	0
	Fault2(operation, privileges)	1	0	0	0	0	0	1	0	0	0	0	0	0
	Fault3(configuration, parameters)	3	0	0	0	0	1	0	0	0	0	0	0	0
Network	Fault1(process, services)	6	0	0	0	0	0	0	0	0	1	0	0	0
	Fault2(operation, privileges)	1	1	0	0	0	0	0	0	0	0	0	0	0
	Fault3(configuration, parameters)	2	0	0	0	0	0	0	1	0	0	0	0	0
Linux OS	Fault1(process, services)	4	0	0	0	0	0	0	0	0	0	1	0	0
	Fault2(operation, privileges)	2	0	0	0	0	0	0	0	0	0	0	0	1
	Fault3(configuration, parameters)	1	0	0	0	0	0	0	0	0	0	0	0	0

The software DRE is segregated into 4 tiers of importance; OS being the highest, followed by the network, database, and lastly Shareplex. This is because one software technology provides a more foundational service to the others which are dependent on it. Because of this, the DRL can dictate the choice of actions to take, especially when it comes to multiple types of actions available for one fault. So, like a typical troubleshooting workflow where the conditions direct the tacit consideration to the most appropriate course of action. The DRL uses the Q-value is used to define the best course of corrective action matrix.


Table 13 – Association of software corrective actions to diagnosed faults for specific software elements.

Oracle’s diagnosed faults ID and description	Oracle DB corrective actions flag array	Corresponding action vector to external commands
1= locked user account	[1,0,0,0,0]	Unlock user account.
2= incorrect password	[0,0,0,1,0]	Grant space quota.
3= not enough space	[0,1,0,0,0]	Grant more privilege.

4= insufficient privilege	[0,0,1,0,0]	Reset password.
5 = user does not exist	[0,0,0,0,1]	Create the user account.

Referring to Table 13, for an example of resolving a common fault like a locked DB account, only the action of unlocking will be required. Therefore, the action for the fault with the vector of actions is required to enable the necessary activation of the commands to unlock it. the following table illustrates this relationship. Each array has a tuple of <action flag, script id> where action flag stipulates for activation, and the script id identifies the commands for the software element.

Therefore, the initial representation of the detailed diagnosed information to the corrective actions can be depicted as followed where the diagnosed matrix is changed to a vector to show the relationship to the respective vector of arrays that hold the tuples of corrective action information.

Detailed diagnostic stats	corrective actions for each software element's configurations and functions		
$\begin{bmatrix} ora_1 \\ ora_2 \\ \dots \\ ora_n \\ spx_1 \\ spx_2 \\ \dots \\ spx_n \\ nw_1 \\ nw_2 \\ \dots \\ nw_n \\ os_1 \\ os_2 \\ \dots \\ os_n \end{bmatrix}$		$\begin{aligned} & [(ora_{a11}, oras_{11}), (ora_{a12}, oras_{12}), \dots, (ora_{a1m}, oras_{1m})] \\ & [(ora_{a21}, oras_{21}), (ora_{a22}, oras_{22}), \dots, (ora_{a2m}, oras_{2m})] \\ & \dots \\ & [(ora_{an1}, oras_{n1}), (ora_{an2}, oras_{n2}), \dots, (ora_{anm}, oras_{nm})] \\ & [(spxa_{11}, spxs_{11}), (spxa_{12}, spxs_{12}), \dots, (spxa_{1m}, spxs_{1m})] \\ & [(spxa_{21}, spxs_{21}), (spxa_{22}, spxs_{22}), \dots, (spxa_{2m}, spxs_{2m})] \\ & \dots \\ & [(nwa_{11}, nws_{11}), (nwa_{12}, nws_{12}), \dots, (nwa_{1m}, nws_{1m})] \\ & [(nwa_{21}, nws_{21}), (nwa_{22}, nws_{22}), \dots, (nwa_{2m}, nws_{2m})] \\ & \dots \\ & [(nwa_{n1}, nws_{n1}), (nwa_{n2}, nws_{n2}), \dots, (nwa_{nm}, nws_{nm})] \\ & [(osa_{11}, oss_{11}), (osa_{12}, oss_{12}), \dots, (osa_{2m}, oss_{2m})] \\ & [(osa_{21}, oss_{21}), (osa_{22}, oss_{22}), \dots, (osa_{2m}, oss_{nm})] \\ & \dots \\ & [(osa_{n1}, oss_{n1}), (osa_{n2}, oss_{n2}), \dots, (osa_{nm}, oss_{nm})] \end{aligned}$	(16)

This can be summarised as;  $d_m = a_{tmn}$

Where  $d$  is the diagnosed faults,  $t$  is type software group,  $n$  is the number of software element faults,  $a$  is the corrective action array,  $m$  is the number of array's action flags position,  $d \in D$  and  $a \in A$ , where  $d$  is the element of all diagnosed faults of set  $D$ ,  $a$  is the element of all corrective actions of set  $A$ . the array of corrective action,  $a$ , is a list of tuples, each with an identification and a numerical reference to the specific entries in the correction external libraries of scripts and OS commands.

#### 5.1.4. Prioritization of the software groups' action

Not all the software in the DRE is regarded equally. Some can function independently without the need of others while others depend heavily on others to conduct their purpose and service. There is different level of dependencies stacked

hierarchically, starting from the top where one Shareplex operates on top of all the software and IT infrastructure, followed by the oracle DB that requires both networks and operating systems to support its service, but it is not dependent on Shareplex. The Operating system is considered as a platform on which all the other software depends, but it can function without any of the above software. The next in line is the network connectivity which sees the connectivity and communication among the hosts plus software elements that require this feature to talk to one another. Therefore, any faults that occur in the software that has the higher importance in this hierarchical order of dependencies, will cascade down the errors down to the other software, causing multiple faults among the other software groups. Likewise, when the fault is resolved at the top, the other faults that developed due to the faults may get fixed subsequently without any intervention. With this in the plan, the prescription of corrective actions to resolve faults in the DRE should start with the most important software in the hierarchical order first, then observe the cascading of the problem fixed across other software groups.

In the DRE's fault resolution process, there are two extreme scenarios. In the best-case scenario, the fault is a minor and isolated incident that can be resolved by single action. Such as the database account for Shareplex is locked the only corrective action needed is to unlock it. The service outage information to represent this will be a straight vector that contains no errors except with one value to depict the specific error, e.g. service outage information =  $[0,1,0,0,\dots,0,0,0,0]$ . So, the action required is the corresponding array of software elements' corrective actions array which points to the respective external system commands. But in the worst-case scenario, this happens when a major software element in the DRE fails and that impact the rest of the other elements that depend on it. For example, when the network card status is disabled or has the wrong IP configuration at the OS level, this will cause an outage on the connectivity and communication. This has a direct impact on the DRE's network services, which in turn affect the Oracle DB service and eventually the operation of the Shareplex. In this scenario, the service outage information to represent this DRE's state will be  $[1,2,2,1,0,0,\dots,1,2,3,1,1]$  and the corresponding fault resolution information is a large matrix of corresponding corrective actions.

However, referring to the previous section on how the hierarchical importance of software impact the others, we proposed that we only perform corrective actions only to the most important groups of the software elements from the top level down, and observe the effect on the rest of the other discovered faults. It is a sequence of succession in solving each software group in stages. This problem-solving process is not a single one-off but requires several iterations to assess just how effective the change will be against the problematic DRE before the next course of action is decided. The process can be simplified as followed.

1. It will require the FD module to sample and diagnose the level of service outages if there is any that arise from the above set of corrections. In the event, if the entire service outages which had been previously reported, have turned up all negatives, then the fault resolution process is then considered to be successful.
2. If the service outage information is still positive with all sector's faults as active, then the FR module will try to activate the other corrective actions based on their output to resolve them.
3. If the service outage information's section for that specific software group, e.g., OS, is clear, that means the problem-solving routine is a success. The next course of action is to move and focus on the rest of the other software groups' elements such as network, Oracle DB, and Shareplex in a hierarchical sequential order.
4. A counter is used to keep track that in the event where the problem-solving process iterates and there is no success, then it will notify the IT administrator for help.

#### **5.1.5. Cost function and Q-Values for FR module**

The accuracy measurement of each DRL's NN prediction is derived from finding the absolute differences in value between the predicted corrective actions values against those that are produced by the SC modules. Refer to eq (17), if the cost values of the predictions are high, then more work is required in building up the knowledgebase and retrain the NN for the DRL. It also determines the confidence

level in the FR module to predict the corrective action accurately and this reduces the dependency on the SC module.

$$\text{Cost} = |a_{\text{predict}} - a_{\text{actual}}| \quad (17)$$

For a given input state of the service outage and based on the consideration of the software group's priority plus the best number of actions needed, the DRL will select the best course and how it does that is based on the reward that takes in the software group's priority and the number of actions it should take, given the assumption that the corrective actions can indeed resolve the faults as in eq (18).

$$\text{Q-values} = g_s * h_s \quad (18)$$

where  $g$  is the software group's priority in which the faults have occurred,  $h$  is the total actions used,  $s$  is the service outage state's identification.

For example, if there are faults in the SO matrix [ $\langle$ OracleDB $\rangle$ ,  $\langle$ Shareplex $\rangle$ ,  $\langle$ network $\rangle$ ,  $\langle$ OS $\rangle$ ], then the best choice of action is to resolve the OS' fault first based on its priority. Once the action has resolved the OS's fault, the focus is turned onto the Network's, and this keeps repeating onto the OracleDB and eventually the Shareplex's. However, if the applied action for a particular software group managed to resolve one set of faults but the DRE's state return with another fault of the same group, then the FR module's focus will not move and keep trying to resolve. Therefore, the values for  $g$  are 1 = Shareplex, 2 = OracleDB, 3 = Network, 4 = OS. Table 14 illustrated this sequence of considerations. As it resolved them, it calculates the cost or reward for the action to the state and builds up a hierarchical structure of actions vs service outage states like figure 18. Algorithm 6 shows the FR module's process in solving each of the software groups' faults in the proposed hierarchical order and algorithm 7 shows describe the procedure of how the direct corrective actions are obtained for each of the software element's fault from the external libraries of pre-scripted commands that have been duly prepared by the IT administrators.

Table 14 – Sequence of actions' consideration for series of faults

Run	State of Service outage information [ $\langle$ OracleDB $\rangle$ $\langle$ Shareplex $\rangle$ $\langle$ Network $\rangle$ $\langle$ linuxOS $\rangle$	Prescribed corrective actions [ $\langle$ OracleDB $\rangle$ $\langle$ Shareplex $\rangle$ $\langle$ Network $\rangle$ $\langle$ linuxOS $\rangle$ ]	Description
1	[1,2,1,0][1,0,1,2][2,3,1,0][3,0,1,0]	[0,0,0,0][0,0,0,0][0,0,0,0][0,0,A <sub>1</sub> ,0]	Resolve OS fault as priority
2	[1,2,1,0][1,0,1,2][2,3,1,0][0,0,0,0]	[0,0,0,0] [0,0,0,0][A <sub>1</sub> ,A <sub>2</sub> ,A <sub>3</sub> ,0][0,0,0,0]	OS fault resolved, attempt to fix network faults



4	[1,2,1,0][1,0,1,2][0,0,0,0][0,0,0,0]	[A <sub>1</sub> ,0,A <sub>2</sub> ,A <sub>3</sub> ][0,0,0,0][0,0,0,0][0,0,0,0]	Both OS and network faults resolved, try fixing OracleDB
5	[0,0,0,0][1,0,1,0][0,0,0,0][0,0,0,0]	[0,0,0,0] [A <sub>1</sub> ,0,A <sub>2</sub> ,0][0,0,0,0][0,0,0,0]	OS, Network & DB faults resolved, remain Shareplex faults, attempt to fix
6	[0,0,0,0][0,0,2,0][0,0,0,0][0,0,0,0]	[0,0,0,0] [0,0,A <sub>1</sub> ,0][0,0,0,0][0,0,0,0]	2nd faults in Shareplex found, attempt to fix
7	[0,0,0,0][0,0,0,0][0,0,0,0][0,0,0,0]	[0,0,0,0][0,0,0,0][0,0,0,0][0,0,0,0]	All faults have been fixed

## 5.2. FR's Algorithm

The algorithm for Fault Resolution follows the same design as FD's in using defining the amount of iteration for solution exploration to build up the knowledgebase for the latter phase of exploiting them via NN prediction as shown in algorithm 5. In this phase, both the current and new state refers to the DRE as the system diagnostics details, unlike the FD. There is a separate iteration of the fault simulation and correction routine which employ the use of the SC unit to identify and correct the simulated faults with corrective actions that are described in algorithm 6. This routine updates the information and the reward associated with the actions in a knowledgebase file, which will be used in the next phase. The next routine performs against the simulated faults again and it can contain another set of simulated faults. The process is split into two distinct sections between the exploration and the exploitation phase. Similar to the first routine, the exploration phase in the second routine also assesses if the learning rate is in the favour of the exploration. If so, the SC unit is performed, and the information is added to the knowledgebase. If the learning rate favours exploitation, the NN unit is used to predict the corrective action against the state, and then use the *p\_single\_element\_corrective\_action* procedure to execute them one by one. The new state plus all the information is added to the knowledgebase file alongside their rewards. This enriches it and bolsters the NN's training competency. The NN is to map the large state space between the diagnostic information about the DRE's state and the corrective actions that are required to resolve the faults.

Referring to algorithm 5, the System Correction (SC) unit is the foundation of the FR where it takes in the system diagnostics information and device the array of corrective actions. Referring to algorithm 6, the SC unit has its algorithm where the software groups are prioritized following their group's hierarchical level of importance in the order of OS, network, OracleDB and Shareplex, respectively. The SD unit then iterates through each of the element's corrective actions as flagged as active in the subgroup from the start to the end individually, finding the corresponding system

commands from the external library and execute them against the DRE. Using the SD unit's procedure, *p\_single\_system\_diagnostics*, as previously mentioned in chapter 5.8, the SC unit queried and build up the information of the DRE's new state after the corrective action has been applied, together with its diagnostics information which determines the progress of the fault resolution. If the new state is the same as the old state, then that signifies that the corrective action is either not effective or the cause of the fault for that specific software element is not isolated, but a cause from other adjacent software. Therefore, the SC unit moves on to the next corrective actions in line until it completes the whole sub-group of flagged corrective actions. By the end of it, the SC unit achieves two outcomes, either it succeeds or failed in the fault resolution task for the software sub-group. If it succeeds, the SC unit moves to the next software sub-group to resolve its problem. If it fails, then it should stop the troubleshooting effort and notify the IT administrator for attention. The information is captured and stored in the knowledgebase; stipulating the system diagnostics details, versus the corrective action and the outcome which is the equivalent of the result. if there is a change between the current and new state and for the better, the reward is set to 1, else it is zero. So, in the knowledgebase, there is a list of DRE current state, system diagnostics, corrective action, new state, and the respective reward.

Algorithm 5 – Main algorithm of FR

Input: DRE_state, system_diagnostics
Output: corrective_actions
Initialization1: learning, Knowledgebase, breakfix_file
Giveup_limit=10

```

Fault_count=0
breakfix_file =fault_breakfix.txt
kb_file=frkb.txt
#####
# loop through all the simulated fault, fix up, build up kb
#####
Loop the iteration,i, of reading(breakfix_file1):
    break_cmd = f<breaki>
    fix_cmd   = f<fixi>
    execute(break_cmd)          #inject fault
    dre_current_state = p_single_system_diagnostics()
    fault_count+=1

while sum(dre_current_state) >=1 and giveup<giveup_limit:
    kb=[]
    corrective_action = p_system_corrective_action(dre_curent_state)
    execute_single_element_corrective_action(corrective_action) #loop through one by one action
    dre_new_state = p_single_system_diagnostics()
    #append all the info to the kb array
    kb=append_to_kb(dre_current_state,dre_new_state, single_element_corrective_action)

    if dre_current_state = dre_new_state then      #compare current state vs new state
        reward =0
        giveup +=1
        tries+=1      #try next element corrective action in position
    else
        reward=1      #made some progress, +ve reward
        tries =0
        dre_current_state = dre_new_state #new state become current state

    if sum(single_element_corrective_action)==0 and sum(dre_current_state)>=1 then
        print "no more solution. Giveup"
        break
        kb.append_to kb(reward)
        write_to_kb_file(kb_file, kb)
    end loop
#####
# simulate individual fault, using SC and NN
#####
Loop the iteration, j, of reading(breakfix_file2):
    break_cmd = f<breakj>
    fix_cmd   = f<fixj>
    Check the learning rate.
    execute(break_cmd)          #inject fault

```

```

###exploration phase - use SD unit
If learning < med_learning, do the exploration phase
  dre_current_state = p_single_system_diagnostics()
  fault_count+=1

while sum(dre_current_state) >=1 and giveup<giveup_limit:
  kb=[]
  corrective_action = p_system_corrective_action(dre_curent_state)
  #loop through one by one action
  execute_single_element_corrective_action(corrective_action)
  dre_new_state = p_single_system_diagnostics()

  #append all the info to the kb array
  kb=append_to_kb(dre_current_state,dre_new_state, single_element_corrective_action)

  if dre_current_state = dre_new_state then    #compare current state vs new state
    reward =0
    giveup +=1
    tries+=1    #try next element corrective action in position
  else
    reward=1    #made some progress, +ve reward
    tries =0
    dre_current_state = dre_new_state #new state become current state

  if sum(single_element_corrective_action)==0 and sum(dre_current_state)>=1 then
    print "no more solution. Giveup"
    break
  kb.append_to kb(reward)
  write_to_kb_file(kb_file, kb)

##Knowledge exploitation phase - use NN unit
If learning is > med_learning then do
  NN_model = NN_build(knowledgebase) ##Train NN with data from knowledgebase
  corrective_actions = round(NN_model.predict(dre_current_state))
  p_execute_corrections(corrective_action)
  dre_new_state = p_system_diagnostics()

  if sum(system_diagnostics) = 0 then
    print("all faults are fixed")
  else
    print("unable to fix faults. notify IT admin")
    email "iteadmin@domain" < dre_state,system_outage
  learning rate +=1
  execute(fix_cmd) #fix fault

```

```
end loop
```

Algorithm 6 – FR module finding corrective actions for DRE based on system diagnostics input and software group's priorities

```
Input: system diagnostics input S
initialize1: software element's corrective action array, A, external libraries X
Initialize2: learning rate, l, predict_threshold, p=0.8, retry_count, r
initialize3: input from FD module, S, 1st element in the software group, g,
            last element in the group, e
Result: corrective actions, A

Def p_system_corrective_action:
g =1, r = 0, e = g+3
l +=0.01
if sum[S[*]] != 0 then                #Check for any faults
#loop through software groups based on hierarchy via index
/* Attempt to fix OS faults */
if sum(S[g:e])>0 then:                #check on faults between index range for group
if r > 2 then:                        #if retries exceed 2, call for IT admin
sp_alert_IT_administrator (S[g:e], S)
r:=0
break
if l < p then:                        #check if l has exceeded p threshold
for i in range (g..e):                #loop through all faults that belong to g
corrective_action_OS += sp_find_solution(S[i]) #accumulate actions from SC module
else:
corrective_action_OS += NN_predict(S[i]) #accumulate actions from DRL module
r +=1
Apply_action_on_DRE(corrective_action_OS) #apply actions to DRE
g=e+1, e=g+3                          #shift to next group
/* Attempt to fix Network faults */
if sum(S[g-4:e-4])=0 and sum(S[g:e])>0 then: #check on faults between index range for group
if r > 2 then:                        #if retries exceed 2, call for IT admin
sp_alert_IT_administrator (S[g:e], S)
r:=0
break
if l < p then:                        #check if l has exceeded p threshold
for i in range (g..e):                #loop through all faults that belong to g
corrective_action_Nw += sp_find_solution(S[i]) #accumulate actions from SC module
else:
corrective_action_Nw += NN_predict(S[i]) #accumulate actions from DRL module
r +=1
Apply_action_on_DRE(corrective_action_Nw) #apply actions to DRE
```

```

g=e+1, e=g+3                                #shift to next group
/* Attempt to fix OracleDB faults */
if sum(S[g-8:e-8])=0 and sum(S[g-4:e-4])=0 and sum(S[g:e]) > 0 then: #check on faults between index
range for group
  if r > 2 then:                             #if retries exceed 2, call for IT admin
    sp_alert_IT_administrator (S[g:e], S)
    r:=0
    break
  if l < p then:                             #check if l has exceeded p threshold
    for i in range (g..e):                   #loop through all faults that belong to g
      corrective_action_DB += sp_find_solution(S[i]) #accumulate actions from SC module
    else:
      corrective_action_DB += NN_predict(S[i])   #accumulate actions from DRL module
    r +=1
  Apply_action_on_DRE(corrective_action_DB)   #apply actions to DRE
g=e+1, e=g+3                                #shift to next group
/* Attempt to fix Shareplex faults */
if sum(S[g-12:e-12])=0 and sum(S[g-8:e-8])=0 and sum(S[g-4:e-4])=0 and sum(S[g:e]) > 0 then:
# Check if all the previous faults have been resolved before proceeding
  if r > 2 then:                             #if retries exceed 2, call for IT admin
    sp_alert_IT_administrator (S[g:e], S)
    r:=0
    break
  if l < p then:                             #check if l has exceeded p threshold
    for i in range (g..e):                   #loop through all faults that belong to g
      corrective_action_SP += sp_find_solution(S[i]) #accumulate actions from SC module
    else:
      corrective_action_SP += NN_predict(S[i])   #accumulate actions from DRL module
    r +=1
  Apply_action_on_DRE(corrective_action_SP)   #apply actions to DRE

```

Algorithm 7 – the procedure to find the corresponding action for each software element’s fault from the external library of pre-scripted commands

**I**nput: section of service outage input, s  
initialize1: External corrective action library, X, fault index, f, action index, a.  
Result: corrective action, c

```

Def Sp_find_solution (s):
    Count:=1
    with open(X) as f:           #search external library for corresponding corrective actions to faults
        for line in f:
            if found s[count] in line[f,]:
                c += line[, a]   #where line[,a] return a reference to external scripts and commands
    Return c

```

From this knowledgebase, the FR's NN unit routine picks out all the entries with the positive rewards and use them to train the NN. Once it has achieved a good performance, the NN unit can take in any known DRE state-system diagnostics and provide the best corrective action with the sub-group that has the priority to fix, meaning if the system diagnostics contains faults in the OS group, then the most appropriate corrective actions will be from the knowledgebase that applied the relevant actions to the OS sub-group. when the OS group's fault is resolved, the system diagnostics information will not contain any faults for that sub-group. the future rewards hyperparameter that the common reinforcement learning is not included here as the current reward is sufficient for the algorithm to determine the best choice. also, there is no probability to contemplate between exploration versus exploitation as this FR's routine has defined it clearly.

### 5.3. Empirical Analysis

This section describes the test for the FR module against the DRE. The purpose is to determine the effectiveness of the proposed FR module in prescribing the best corrective actions for the DRE based on the service outage information that the FD module has determined about the DRE under simulated faults. For this to work, the FD module should have achieved sufficient accuracy in its service outage predictions in the previous phase where it has been properly trained and tested to a certain expectation. Like the FD module testing phase, the test starts with the DRE's baseline where all its software is functioning normally. Then each fault is injected into the DRE, causing abnormalities across the DRE's software elements, which are in turn acquired by the FD module and the service outages information is released. The FR module received the SO details and predicted the course of actions, which are then applied to the DRE and incur a new state. if the state is fixed, then the fault resolution process is considered a success. If the faults are not clear, the FR module iterates for the next

round of prescribing corrective actions for the new DRE's state. This repeats for a specific number of loops before it can be regarded as an unsuccessful attempt. Before each iteration test can start, the DRE must be reinitialized to the baseline to ensure a clean slate where all the software is fully functional. The goal is to assess the capability and accuracy of the FR module in prescribing the relevant corrective actions for the detected service outage in the DRE. The scope of the test is confined to those recoverable failures and those catastrophic ones are excluded.

### **5.3.1. Software setup**

The setup for the DRE software is the same as what was described in section 5.8.1. The test environment which had been used for the FD module testing is now used for the FR module test. There is no deviation on the underlying DRE setup and operation. Only the FR module is added together with the external libraries of resolution.

### **5.3.2. Experiment setup and goal**

For the FR module experiment setup, the DRE test environment is used with the same DB and Shareplex configuration as well as the DB schema and objects in place, all of which have been described in section 5.8.2. There is no deviation nor alteration required. The only pre-requisite here is the FD module that needs to be trained and tested before proceeding on with the FR module test.

The DRE is set to the baseline where all the software work as per norm. Each fault that belongs to the specific software is introduced in the DRE which will then yield information that the FD module will detect and predict the possible service outage, which in turn, is input into the FR module that will predict the best possible corrective actions based on its trained knowledge to score the best rewards. The FR module is expected to prescribe action for the following software element's faults in response to the service outage input concerning its associated system diagnostics information. Table 15 contains the test case with a list of common faults, and the corresponding action to rectify them. The priority for the FR module is to resolve the OS, followed by the network, OracleDB, and Shareplex in a hierarchical inter-dependent arrangement.

Table 15 – List of DRE's software groups service outage fault and their corrective actions



Function	Attributes	Commands	Reason and Corrective actions	Corrective actions	Diagnosed Fault vectors	Corrective action vectors
<b>OracleDB</b> = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]						
memory process 0~0~0	Check for DB1's smon Check for DB2's smon Check for DB1's pmon Check for DB2's pmon	if [ \$(ps aux grep -i ora_smon_DB1 grep -v grep wc -l) -eq 1 ]; then echo 0; else echo 1; fi if [ \$(ps aux grep -i ora_pmon_DB1 grep -v grep wc -l) -eq 1 ]; then echo 0; else echo 1; fi if [ \$(ps aux grep -i ora_smon_DB2 grep -v grep wc -l) -eq 1 ]; then echo 0; else echo 1; fi if [ \$(ps aux grep -i ora_pmon_DB2 grep -v grep wc -l) -eq 1 ]; then echo 0; else echo 1; fi	Oracle instance is not active, start it up Pmon, smon are together. Either both are up or both are down DB1 and 2	export ORACLE_SID=DB1 && echo "startup;" sqlplus -s sys/password as sysdba  export ORACLE_SID=DB2 && echo "startup;" sqlplus -s sys/password as sysdba	[1,1,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,1,1,0,0,0,0,0,0,0,0,0,0]	[1,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0,0,0,0,0,0,0]
Status 0~1~0	if DB1 is open if DB2 is open  (DB's mode is open, not restricted DB available for use)  Check any session blocking on DB1  Check any session blocking on DB2	if [ \$(echo -e "SET PAGESIZE 0 FEEDBACK OFF; \n select instance_name from v\$instance;" sqlplus -s system/password@DB1 head -n1) == 'DB1' ]; then echo 0; else echo 1;fi  if [ \$(echo -e "SET PAGESIZE 0 FEEDBACK OFF; \n select instance_name from v\$instance;" sqlplus -s system/password@DB2 head -n1) == 'DB2' ]; then echo 0; else echo 1;fi  if [ \$(echo -e "SET PAGESIZE 0 FEEDBACK OFF; \n select ltrim(count(*) from v\$session where blocking_session is not NULL;" sqlplus -s system/password@DB1 head -n1) 2>/dev/null -eq 0 ]; then echo 0; else echo 1;fi  if [ \$(echo -e "SET PAGESIZE 0 FEEDBACK OFF; \n select ltrim(count(*) from v\$session where blocking_session is not NULL;" sqlplus -s system/password@DB2 head -n1) 2>/dev/null -eq 0 ]; then echo 0; else echo 1;fi	Oracle database could have been mounted but not available; open the DB for use  Presence of blocking sessions in DB; kill all blocking session	export ORACLE_SID=DB1 && echo "alter database open;" sqlplus -s sys/password as sysdba  export ORACLE_SID=DB2 && echo "alter database open;" sqlplus -s sys/password as sysdba  ./kill_blocked_sess.sh DB1 ./kill_blocked_sess.sh DB2	[0,0,0,0,1,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,1,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,1,0,0,0,0,0,0,0] [0,0,0,0,0,0,1,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,1,0,0,0,0,0]	[0,0,0,0,1,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,1,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,1,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,1,0,0,0,0,0]
Account security 0~2~0	~SPLX account is available on DB1  ~SPLX account is available on DB2  System, splx account not locked Splx has quota on splx tablespace	if [ \$(echo -e "SET PAGESIZE 0 FEEDBACK OFF; \n select ltrim(decode(account_status,'OPEN',0,1)) from dba_users where username='SPLX';" sqlplus -s system/password@DB1 head -n1) 2>/dev/null -eq 0 ]; then echo 0; else echo 1;fi  if [ \$(echo -e "SET PAGESIZE 0 FEEDBACK OFF; \n select ltrim(decode(account_status,'OPEN',0,1)) from dba_users where username='SPLX';" sqlplus -s system/password@DB2 head -n1) 2>/dev/null -eq 0 ]; then echo 0; else echo 1;fi	DB account for SPLX is locked; unlock the account  SPLX's DB account does not have enough quota space, increase its quota	echo "Alter user splx account unlock;" sqlplus -s "system/password@DB1"  echo "Alter user splx account unlock;" sqlplus -s "system/password@DB2"  echo "alter user splx quota unlimited on USERS;" sqlplus -s "system/password@DB1"  echo "alter user splx quota unlimited on USERS;" sqlplus -s "system/password@DB2"	[0,0,0,0,0,0,0,0,1,0,0,0,0,0] [0,0,0,0,0,0,0,0,1,0,0,0,0,0]	[0,0,0,0,0,0,0,0,1,0,0,0,0,0] [0,0,0,1,0,0,0,0,0,0,0,0,0,0]
storage space 0~3~0	~check USERS TBLSP free space on DB1  ~check USERS TBLSP free space on DB2	if [ \$(echo -e "SET PAGESIZE 0 FEEDBACK OFF; \n select case when (1-used_percent) > 0.1 then ltrim(0) else ltrim(1) end from dba_tablespace_usage_metrics where tablespace_name='USERS';" sqlplus -s system/password@DB1 head -n1) 2>/dev/null -eq 0 ]; then echo 0; else echo 1;fi  if [ \$(echo -e "SET PAGESIZE 0 FEEDBACK OFF; \n select case when (1-used_percent) > 0.1 then ltrim(0) else ltrim(1) end from dba_tablespace_usage_metrics where tablespace_name='USERS';" sqlplus -s system/password@DB2 head -n1) 2>/dev/null -eq 0 ]; then echo 0; else echo 1;fi	User Tablespace has reached max size; extend the datafile size;	Alter tablespace USERS datafile 'u01/oradata/DB1/user01.dbf' autoextend on 100m; Alter tablespace USERS datafile 'u01/oradata/DB2/user01.dbf' autoextend on 100m;	[0,0,0,0,0,0,0,0,0,0,1,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,1,0,0]	[0,0,0,0,0,0,0,0,0,0,1,0,0,0] [0,0,0,0,1,0,0,0,0,0,0,0,0,0]

		2>/dev/null -eq 0 ]; then echo 0; else echo 1;fi				
<b>Shareplex = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]</b>						
main processes 1~0~0	Check if sp_cop process is up Check if sp_ocap process is up Check if sp_opst_mt process is up Check if sp_xport process is up Check if sp_ordr process is up Check if sp_mport process is up	if [ \$(pidof -s sp_cop) > 1]; then echo 0; else echo 1; fi if [ \$(pidof -s sp_ocap) > 1]; then echo 0; else echo 1; fi if [ \$(pidof -s sp_opst_mt) > 1]; then echo 0; else echo 1; fi if [ \$(pidof -s sp_xport) > 1]; then echo 0; else echo 1; fi if [ \$(pidof -s sp_ordr) > 1]; then echo 0; else echo 1; fi if [ \$(pidof -s sp_mport) > 1]; then echo 0; else echo 1; fi	When sp_cop is down, all the rest will be done too; start up sp_cop	echo password su - splx -c \$MDIR/startup.sh	[1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0]	[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Splx console 1~1~0	Check if sp_cop is responsive	if sp_ctrl status 2>/dev/null grep -q "Running" ; then echo 0; else echo 1; fi	If sp_ctrl is not response; restart the sp_cop	echo password su - splx -c \$MDIR/startup.sh	[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]	[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
queues operation 1~2~0	Check if capture state is running Check if read state is running Check if export state is running ~Check if import state is running ~Check if post state is running Capture, export, import, post, read status = Running	if [ \$(sp_ctrl status 2>/dev/null tail -n+5 head -n-5 grep -i capture awk '{ print \$2 }' grep -i stopped wc -l) 2>/dev/null -eq 0 ]; then echo 0; else echo 1; fi if [ \$(sp_ctrl status 2>/dev/null tail -n+5 head -n-5 grep -i read awk '{ print \$2 }' grep -i stopped wc -l) 2>/dev/null -eq 0 ]; then echo 0; else echo 1; fi if [ \$(sp_ctrl status 2>/dev/null tail -n+5 head -n-5 grep -i export awk '{ print \$2 }' grep -i stopped wc -l) 2>/dev/null -eq 0 ]; then echo 0; else echo 1; fi if [ \$(sp_ctrl status 2>/dev/null tail -n+5 head -n-5 grep -i import awk '{ print \$2 }' grep -i stopped wc -l) 2>/dev/null -eq 0 ]; then echo 0; else echo 1; fi if [ \$(sp_ctrl status 2>/dev/null tail -n+5 head -n-5 grep -i post awk '{ print \$2 }' grep -i stopped wc -l) 2>/dev/null -eq 0 ]; then echo 0; else echo 1; fi	If the capture, read, export and post status are stopped; start them up	echo "start capture" sp_ctrl echo "start export" sp_ctrl echo "start import" sp_ctrl echo "start post" sp_ctrl echo "start read" sp_ctrl	[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Config file 1~3~0	Query if config file is active	1~3~1~if sp_ctrl show config 2>/dev/null grep -q 'Actid' ; then echo 0; else echo 1; fi		Echo "activate config ck.cfg"  sp_ctrl	[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]	[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]
Queues' backlogs 1~4~0	~ Check for post backlog ~ Check for capture ~ Check for export backlog	if sp_ctrl status 2>/dev/null grep -q "Running" ; then if [ \$(sp_ctrl qstatus grep -A 3 -i post grep "Backlog" awk '{ print \$3 }' awk '{s+=\$3} END {print s}') 2>/dev/null -eq 0 ]; then echo 0;else echo 1;fi else echo 1; fi if sp_ctrl status 2>/dev/null grep -q "Running" ; then if [ \$(sp_ctrl qstatus grep -A 3 -i post grep "Backlog" awk '{ print \$3 }' awk '{s+=\$3} END {print s}') 2>/dev/null -eq 0 ]; then echo 0;else echo 1;fi else echo 1; fi if sp_ctrl status 2>/dev/null grep -q "Running" ; then if [ \$(sp_ctrl qstatus grep -A 3 -i post grep "Backlog" awk '{ print \$3 }' awk '{s+=\$3} END {print s}') 2>/dev/null -eq 0 ]; then echo 0;else echo 1;fi else echo 1; fi	Substantial backlogs discovered, send email to IT admin requesting for attention	mail -s 'Attention required for DRE – backlog detected' ITAdmin@company.com	[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

For the OS, the common faults that lie with this group are related to the system resources, configurations, and privileges. System resources refer to various facilities

that the rest of the other software groups need to perform, such as CPU, memory, disk storage space, and communication facility. The configurations' part is mainly on the underlying settings that are present in the system directories that controlled various aspects of the OS' functionality. For this thesis, the main emphasis on the OS' configuration is on the security and network portion. The third area is where the security and privileges govern most of the OS operations.

1. The system resource for CPU and memory are commonly allocated more than what the overall DRE will needs but, in the event, should there be events that they are reaching the maximum threshold and the FD module has flagged them as a potential risk, then the corrective action to take is to turn off some of the adjacent non-critical processes, removing log files and compressing the DB's archived log files to free up the resources. The network cards' status must be brought to active status if they are found to be inactive, plus activating the IP configuration if they are not accurate.
2. There are several critical system configuration files that the OS depends on, like networks, hosts lookup, DNS setting, user accounts securities and they reside in the /etc folder. Faults may arise in the DRE especially when these files are changed or removed illegally, causing massive outages across the different software groups. Therefore, to resolve this, the original backed-up copies of these configuration files must be restored usually from a backup device or off-site storage. To simplify this task of restoration, a set of backups has been made on this system configuration files and the FR module will initiate their restoration to overcome these faults.
3. For the OS' privileges, these refer to the User accounts and their privileges to specific folders on which they are designated to operate on. if the FD module finds that they are not available for use or the privileges have some mismatched permission, then the actions need to rectify them. System commands such as altering the file and directory privileges, plus unlocking the locked users' accounts, or resetting their passwords to the original baseline are some of the actions that can restore their service for the DRE. when there are faults with the OS, it has a cascading effect on other software groups, therefore, it is important for the FR module to resolve the OS' fault first.

Network service is the next group in line for the FR module to attend to when the OS service outages are resolved. The network supports the database listeners to facilitate the network connectivity from sqlplus client connection, and to do that, it requires the OS' network infrastructure and TCPIP protocol to be active. It operates on Oracle's network-related process which needs its own set of configuration files.

1. The common faults identified with this software element related to its processes and configuration's information, so the corrective actions for it usually involve restarting the listener process, enabling the listener service through the lsnrctl command console, including restoring the various oracle network-related files like listener.ora, tnsnames.ora and sqlnet.ora.
2. The network service is also required for connection among the OracleDB and Shareplex, so this involves the underlying OS' system configuration files under /etc and network card configuration, both fall under the scope of OS' fault management. it is expected the corrective actions should cover as part of the OS' fault validation and resolution process too.

Once the infrastructure issue is resolved, the next area to focus on is the software service that operates on them such as the OracleDB and Shareplex.

The Oracle DB's faults cover several areas such as the processes that support the DB services, the configuration that controls the services, and the security and privileges that allow both the internal and external clients to operate on. The corrective actions that the FR module is expected to prescribe are like those that are carried out for the above software groups, and that is to restart processes, change the configuration, and restore files.

1. If the essential DB processes are not found, it means that the Oracle instance is not operational. The solution is to restart the Oracle instance to bring back all the relevant oracle's essential processes. This is the first and most fundamental approach which solves most of the current hung operation in any environment. the restart also cleared out orphaned client processes that can impose complications that hinder the server's operations too.
2. As for the configurations part, it comprises different parameter settings that direct and controls the OracleDB's various services. So, any faults with these

configurations can arise when the parameters are changed incorrectly or illegally. The course of action is to correct them in accordance with their pre-set values to restore their services' purposes.

3. The OracleDB's security concerns with accounts & privileges including internal operations and other internal DB resourcing that can hamper its ongoing services. Common issues such as locked user accounts, lack of proper system or objects privileges and execution rights, availability of free spaces, and even session that are causing deadlocks. These are some of the most common and major issues that have a direct impact on the overall DB services on which the Shareplex is dependent. The FR module must resolve them by providing solutions that can rectify them via actions such as resetting their passwords, unlocking the accounts, regrating the permission and privileges, allocating more space quota to their accounts, or removing blocking sessions to name a few. The last area for the FR module to focus on is the Shareplex, once all the rest of the other software groups have been resolved.

The fault categories for the Shareplex are like the OracleDB, covering the processes and internal configurations. One exception is the account and privileges that Shareplex needs on the OracleDB, but this should be resolved at the OracleDB's stage of fault resolution instead.

1. Shareplex has several important memory processes, and all are controlled from the main process called sp\_cop. In the event should any of the other processes, e.g. capture, read, export, import, and post, died or hung, the recommended course of action is to shut down the main sp\_cop process, kill off any remnant orphaned processes and restart it. The FR module is expected to prescribe such a course of action should it receive such service outages.
2. For the configuration, Shareplex depends on one main configuration file to operate, if the FR module finds that this is either not available or in an erroneous stage, then it will invoke the action of reconfiguring or reactivating with a backed-up copy of the config file. This action will cause the Shareplex to flush out all the active cache and reinitialize new queues under the VARDIR or variable directory.

3. For Internal Shareplex operations where the different queues that belong to the various processes must remain clear with minimum blockage or backlog. One of the common corrective actions is to quiesce and restart them. This is the first level of support intervention required. But in the event, if this action cannot resolve the fault, then other actions may involve external intervention that needs more in-depth analysis, which requires IT administrator intervention.

#### **5.4. Evaluation Criteria**

This section describes how the FR module is evaluated and the criteria used in its assessment. For any instance of the DRE's faults, it can occur in either one or all software groups concurrently. The FR module exploits its knowledgebase and remediates the DRE with a series of corrective actions. Two evaluation criteria, both quantitative and qualitative that have been used in section 5.9, are utilized to assess the FR module's capability. The qualitative outcome of the FR module is referred to the effectiveness is on its ability to achieve the goal of resolving the faults. While the FR module has the flexibility to address a wider range of software faults and service outages. In this thesis, it is restricted to handle those issues that only impact the DRE. Also, the control environment set the faults that are to be used for the test, have a corresponding corrective action that can resolve them. Therefore, the test is to assess just how effective the FR module can be when it comes to handle various known faults and recall the relevant actions from its knowledgebase.

There cannot be two outcomes from the FR module that can help the DRE to reach the objective, whether the corrective actions that have been applied can resolve the faults or not. The feedback loop from the DRE's response to the action generates the new state where the FD module can ascertain if there are more errors to resolve or not. It is a progressive stepped resolution process that can take several sub-iterations in the attempt to resolve the faults, be it the outcome is successful or not. There is a limit on the number of resolution iterations that the FR module can action before the entire resolution effort is considered unsuccessful.

The next test is the accuracy of the FR's module DRL-NN. The output from the SC module serves as the reference where the accuracy of the DRL-NN's output is derived using the MASE formula. This is used to validate the degree of errors that the

DRL-NN has, and it is useful in optimizing the NN design, including their parameters and other NN's related attribute to improve the overall accuracy.

#### **5.4.1. Test results**

There are two groups of results that are generated by the FR. One is from the SC module where it responds to the diagnostics information with a series of corrective actions and the outcomes of the DRE states. The other is the prediction performance and outcome of the DRL-NN in response to various diagnostics information from the DRE. These results are obtained through the process of introducing a series of known faults into the DRE and the FR module is executed against it.

#### **5.4.2. FR module - SC's results**

The SC is used prevalently when the entire FR's learning phase is at the early and middle stages. Table 15 showed the outcomes from the SC when a list of the known faults is injected into the DRE. The FR module used the diagnosed information from FD and derive the appropriate corrective actions based on the hierarchical order of the software group and action planned to solve that software of higher importance first and work its way down. the result showed the state of the DRE once the faults are executed and followed by the new state when the corrective action is executed. The single element corrective action is the information in which the corresponding single action that the SC refer to run the relevant system commands in the attempt to rectify the fault. If the first action for that specific software element within the group is successful, the result will render the software service back to the norm, giving the DRE's new state zero through the diagnostics information. Fault run #1 and 6 are such example. However, if the first action failed, the SC will iterate to the next element's action. This goes on until either the fault for the software group is resolved such as in fault run #8 or it reached the end with no positive outcome, which signifies a failed attempt of giving up for the troubleshooting effort. This is translated into the routine where an alert is sent to the IT administrator. Referring to table 16, fault run #1 is a single fault occurring at the OracleDB which terminated the DB services but that in turn sets off multiple error flags across the diagnostics information. By default, if the troubleshooting effort starts from each flag, it will not be practical since the software elements among the four groups are interdependent. So, once the OracleDB's fault is

fixed, the rest of the error flags that were detected previously have been cleared too, indicating positive cascading effects from one successful fault resolution to the rest of the other depending on software elements.

But for fault run #5, the diagnostics information did specify multiple outages across the field for Shareplex's process, but in fact, the main sp\_cop process has been shut down, rendering all vital Shareplex process inactive. Therefore, the SC module tried to match and execute specific actions for detected down service without much success until it reached the action where the system command restarted the Shareplex's main process. These are the controlled exploitative and explorative phases of SC running multiple tests within its knowledgebase and external libraries of system commands to resolve the faults. These encounters produced useful information of DRE's varying states with different end-goal of corrective actions and rewards. These are then used for the DRL-NN training and prediction, and subsequently, the FR's DRL to exploit and find the optimum course of corrective actions for any known faults.

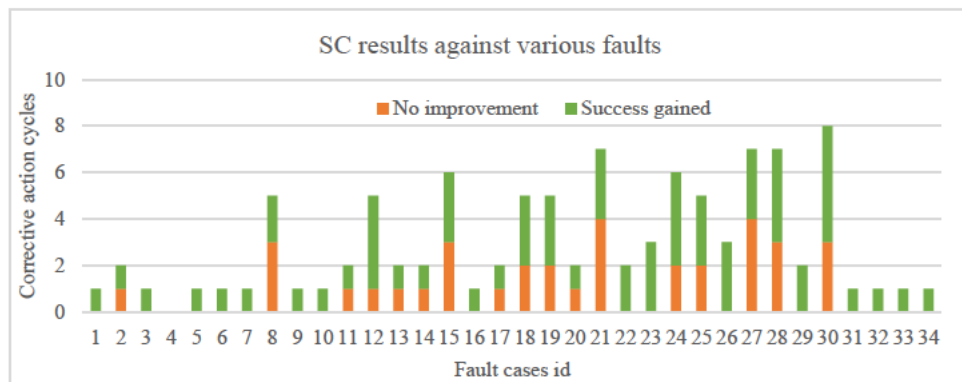


Figure 31 – FR-SC progress results against various faults

Table 16 – SC's results in response to injected faults

```
#####
fault count= 0 fault cmd= export ORACLE_SID=DB1 && echo "shutdown immediate;"|sqlplus -s "sys/password as sysdba"
#####
DRE_current_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0]
index= 19 correct cmd= export ORACLE_SID=DB1 && echo "startup;"|sqlplus -s "sys/password as sysdba"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
resolved
#####
fault count= 1 fault cmd= export ORACLE_SID=DB2 && echo "shutdown immediate;"|sqlplus -s "sys/password as sysdba"
#####
DRE_current_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0]
index= 21 correct cmd= export ORACLE_SID=DB1 && echo "startup;"|sqlplus -s "sys/password as sysdba"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0]
SAME reward = 0
DRE_current_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0]
index= 22 correct cmd= export ORACLE_SID=DB2 && echo "startup;"|sqlplus -s "sys/password as sysdba"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
resolved
#####
fault count= 2 fault cmd= echo "Alter user splx account lock;"|sqlplus -s "system/password@DB1"
#####
```



```
DRE_current_state = [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 27 correct cmd= echo "Alter user splx account unlock;"sqlplus -s "system/password@DB1"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
Resolved
#####
fault count= 11 fault cmd= echo "stop capture"|sp_ctrl && echo "stop import"|sp_ctrl
#####
DRE_current_state = [0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_current_state = [0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 1 correct cmd= echo "start capture"|sp_ctrl
single_element_corrective_action= [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
DRE_current_state = [0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 5 correct cmd= echo "start import"|sp_ctrl
single_element_corrective_action= [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
DRE_current_state = [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 3 correct cmd= echo "start export"|sp_ctrl
single_element_corrective_action= [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
DRE_current_state = [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 25 correct cmd= /home/oracle/fddr/kill_blocked_sess.sh DB1
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
resolved
#####
fault count= 12 fault cmd= echo "stop capture"|sp_ctrl && echo "stop post"|sp_ctrl
#####
DRE_current_state = [0,1,1,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 1 correct cmd= echo "start capture"|sp_ctrl
single_element_corrective_action= [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
DRE_current_state = [0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 2 correct cmd= echo "start post"|sp_ctrl
single_element_corrective_action= [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
resolved
```

Figure 31 shows the progress of SC using its algorithms and workflow to resolve the faults. Each of the faults has a different level of complexity and some can be resolved by one cycle of applying the appropriate corrective actions. However, some need more than 1 cycle to resolve them and these are the more complicated faults like id #8, #15, #21 and #30. The SC did have several unsuccessful attempts to rectify the faults which are highlighted in orange. And those actions have yielded some positive changes in the DRE's diagnostics information are represented in green. The SC progress is considered rule-based which most decision-based system management system is based on and it showed the inefficiency involved. But this step is important to the overall FDR system as it forms the explorative phase where SC experience the different combination of ordered random actions against the environment's states, mapping what actions works against each type of DRE's state. These experiences are then stored into the knowledgebase which is then used to build the intelligent phase of the FR's DRL where the NN is trained with those datasets from the knowledgebases that have proven to have positive rewards.

### 5.4.3. FR module – NN performance and result

Once the FR's learning phase reached the advance or mature stage, it starts to rely on its NN to predict the corrective action. The advantage of using this is to improve the overall turnaround time of deriving corrective actions that are effective for a given DRE state based on its corresponding diagnostics information. While the SC can iterate through multiple permutations of controlled random actions to states, it incurs a substantial number of trials and errors before arriving at the best actions. Whereas in the mature phase where the NN is trained against minibatch that comprised of only those effective actions against different DRE states, it can cut short of the turnaround time needed to find the optimum solution.

Prior to the use of the NN, a test is required to determine the optimum hyperparameters needed against the dataset. The choice of neural network library is Keras, and it has many hyperparameters to tune but only the major ones are focused on for this experiment. They are 1) optimizer – it is the iterating learning algorithm that optimizes the internal parameters within the NN against a performance measure like MSE based on the use of datasets to train and update the NN model. The most common optimizers used are stochastic gradient descent (SGD) and Adaptive moment estimation (Adam). 2) batch size is the hyperparameter that controls the number of training samples or rows to use before the model's internal parameters are updated. 3) epoch is the hyperparameter of gradient descent which controls the number of passes through the training dataset, and finally 4) activation function is part of neural network and it determines what is deemed to be activated based on the neuron's input. There are several types of activation functions available but only a few of them are used in this test.

Figure 32 shows the NN's performance results from the tests using a combination of different types of activation functions (RELU, SIGMOID, TANH) against two optimizers (Adam, Stochastic Gradient Descent (SGD)) against varying values of epochs and batch-size settings. The legends listed in the charts depict both the loss and accuracy readings with different epoch-batch size values. There are many other types of optimizers and activation functions which they have been tested too, including adjustment of other minor hyperparameters such as dropout rate, model's initialization factor, optimizer's learning rate etc. but the gains or losses are not as

significant as the four hyperparameters mentioned previously. So, only four combinations of results are discussed in this section. Figure 32d showed that the NN's performance fared the worst when using SGD and sigmoid as both the optimizer and activation function. the accuracy did not reach above 0.3 regardless of the epoch and batch size. Figure 32a has some mixed results when the NN is configured with RELU and ADAM as activation function and optimizer, reaching toward 82% while other fair badly. the optimal value batch size for this combination is 20-50 with a value greater than 10 for the epoch. But by far the combinations of using either ELU or TANH as activation function in conjunction with Adam as optimizer yielded the best result as shown in fig 29b and c. The NN model that uses ELU managed to achieve high accuracy of 100% under the epoch of 10 regardless of the batch size, whereas the model that used TANH function need slightly more epoch to achieve the same goal. The batch size has some more visual influence on the convergence but they are minor in comparison to the others. Therefore, the optimal hyperparameters' settings for the NN model in the FR module are ELU activation function, Adam optimizer, batch size of 30 and 20 for epochs. From the figure 32, it also showed that the NN converge before the 35<sup>th</sup> epoch moving from none-low experience to medium-high level.

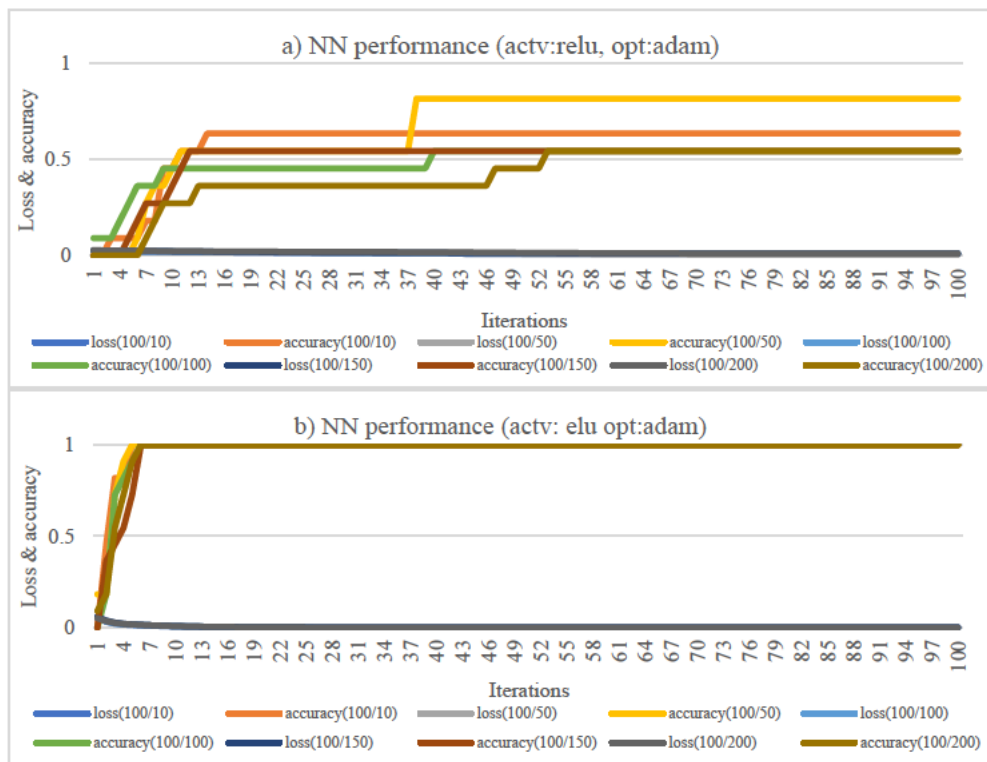




Figure 32 – NN’s performance with varying epoch and batch size against different activation functions and optimizers

#### 5.4.4. FR’s efficacy test results

The outcome is to validate how effective is the FR’s final stage of prescribing the optimal corrective actions based on its NN model after it has learned via using the SC to decipher and execute corrective actions iteratively to resolve the detected faults. Two groups of faults are to be tested: known and unknown. For known faults, these have been predetermined and through the series of faults-corrective actions matrix relationship, scripted system commands that are present in the external libraries and detectable in the FD-SD’s detecting routine. The performance measure is the percentage of fault correcting success that the FR can achieve over the step that it needs to resolve. The FR-SC may take numerous iterations of applying corrective actions that may be ineffective for a given DRE’s state before arriving at one final effective one. The FR-NN should be able to predict the conclusive and effective one for a similar state directly. Table 16 showed the results using FR-NN to predict the best corrective actions that can yield positive rewards against the same series of fault injections. Comparing with the SC’s results in table 17 against the fault case #11 and #12, note

that the NN did not choose those corrective actions that yield zero rewards, whereas the SC did go through iterative as its algorithm dictates it. Therefore, the outcomes from the NN are more productive and efficient as compared to those from the SC.

Table 17 – SC’s results in response to injected faults

```
#####
fault count= 0 fault cmd= export ORACLE_SID=DB1 && echo "shutdown immediate;"|sqlplus -s "sys/password as sysdba"
#####
DRE_current_state      = [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,1,0,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 19 correct cmd= export ORACLE_SID=DB1 && echo "startup;"|sqlplus -s "sys/password as sysdba"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state          = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
resolved
#####
fault count= 1 fault cmd= export ORACLE_SID=DB2 && echo "shutdown immediate;"|sqlplus -s "sys/password as sysdba"
#####
DRE_current_state      = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0]
index= 22 correct cmd= export ORACLE_SID=DB2 && echo "startup;"|sqlplus -s "sys/password as sysdba"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state          = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
resolved
#####
fault count= 2 fault cmd= echo "Alter user splx account lock;"|sqlplus -s "system/password@DB1"
#####
DRE_current_state      = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 27 correct cmd= echo "Alter user splx account unlock;"|sqlplus -s "system/password@DB1"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state          = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
resolved
#####
fault count= 11 fault cmd= echo "stop capture"|sp_ctrl && echo "stop import"|sp_ctrl
#####
DRE_current_state      = [0,1,0,1,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 1 correct cmd= echo "start capture"|sp_ctrl
single_element_corrective_action= [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state          = [0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
DRE_current_state      = [0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 5 correct cmd= echo "start import"|sp_ctrl
single_element_corrective_action= [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state          = [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
DRE_current_state      = [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 3 correct cmd= echo "start export"|sp_ctrl
single_element_corrective_action= [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state          = [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
DRE_current_state      = [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 25 correct cmd= /home/oracle/fddr/kill_blocked_sess.sh DB1
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state          = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
resolved
#####
fault count= 12 fault cmd= echo "stop capture"|sp_ctrl && echo "stop post"|sp_ctrl
#####
DRE_current_state      = [0,1,1,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 1 correct cmd= echo "start capture"|sp_ctrl
single_element_corrective_action= [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state          = [0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
DRE_current_state      = [0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 2 correct cmd= echo "start post"|sp_ctrl
single_element_corrective_action= [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state          = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
CHANGE reward = 1
resolved
```

Table 18 showed the results from the various efficacy tests, scoring 100% for all the registered or known faults. While the plan is to anticipate all possible faults that

can and will happen in any complex environment, there is always a chance that some unplanned and unknown faults that can occur and the FR's fault-corrective action matrix has no provision for. In this scenario, a feedback process is anticipated so that in such an event where the FR is unable to handle the fault since it is beyond its knowledgebase, the even must be sent to the IT administrator to request additional assistance. This process of acquiring external intervention is necessary so that the FR module can readapt its SC and reoptimize its knowledgebase against the newly found knowledge, thus expanding its capacity to handle more faults soon.

Table 18 – FR module's efficacy test results

Software groups	Occurrences/combination/ service specifics	Total	outcomes
network	Network files, listeners	2	2 resolved
oracledb	privileges, accounts	4	4 resolved
oracledb	Oracle processes	4	4 resolved
Shareplex+ oracledb	Access, privileges	2	2 resolved
Shareplex	Sp_cop processes	5	5 resolved
Shareplex	Queues and services	5	5 resolved
shareplex	Accounts	2	2 resolved

While the above results have indicated the efficacy of the FR's SC and DRL-NN components, the next step is to differentiate the efficiency between the two components. Figure 33 showed the number of corrective action cycles that each of them took against the series of faults. For easier faults such as id #1, #3, #5 and #9, both SC and DRL-NN performed the same number of corrective action cycles. However, for more complex ones like fault id #8, #18, 21 or #30, the SC had to perform more cycles following its internal logic before resolving them. The NN, however, can pick those actions that can yield a positive outcome for the faults and managed to resolve them under few cycles. Thus, proving that having a deep NN that can learn from its knowledgebase to pick only those relevant corrective actions for the different DRE states.

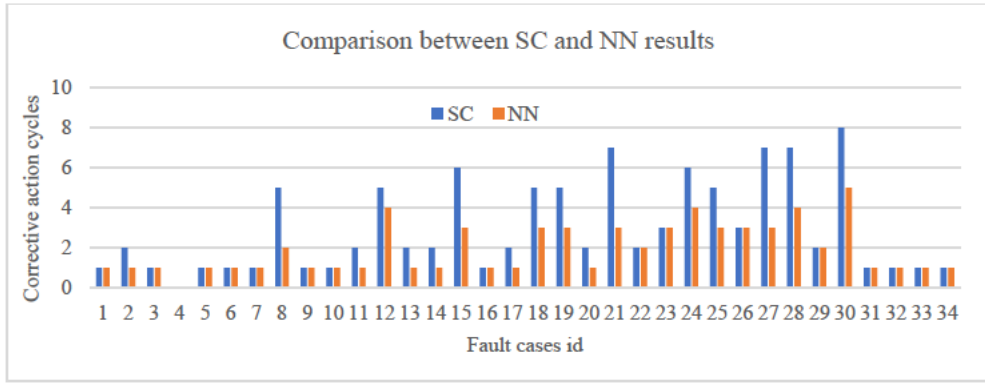


Figure 33 – Corrective action cycles results between SC and NN

Another downside of using SC to resolve the fault is the time needed to iterate through various paths of matching specific elements’ fault with its libraries of corrective actions which are equivalent to a trial-n-error approach. For simpler faults, the turnaround time that SC needs may be less than a minute but for more complicated ones, it will take several mins to iterate through the actions, including wait times for system response and allowance for software element to take effect on the changes. The time taken by the NN to predict actions lies with the trained algorithm and it can produce the answers of corrective actions for any given faults within a few secs. That is far more efficient as compared to SC’s process. Figure 34 shows the difference in the time taken by both SC and DRL in deriving the solutions for each of the encountered faults.

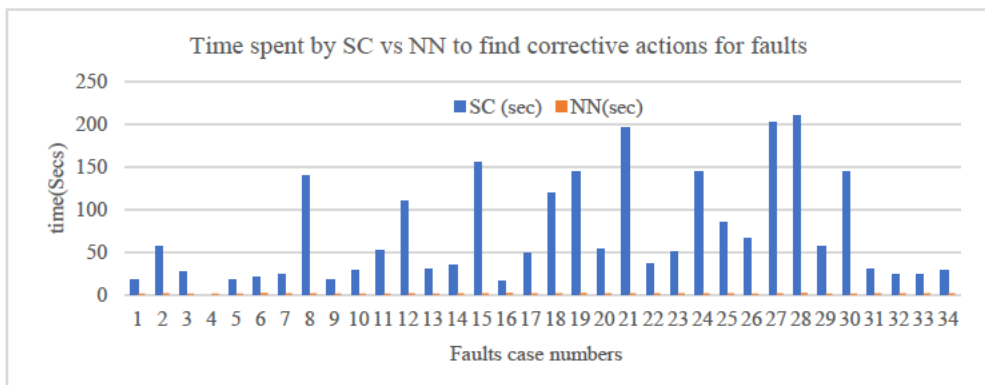


Figure 34 – Comparison of time taken to find corrective actions between SC and DRL-NN

Some other experience has been gained particularly in the conduct of the test. It is found that there must be sufficient time delay at each junction between the execution of the faults and the next corrective action activity as the DRE software cannot respond within a short turnaround time. This can mislead the diagnosis of the DRE which has

been experienced in the trials. So, delaying the diagnostic query with some waits is needed.

## **5.5. FR's Conclusion**

The FR module has been tested and can predict the corrective actions based on the service outage information that is provided by the FD module with good accuracy. It makes use of the DRL that is based on the Actor-Critic approach where the SC module plays the role of the critic and forms as a reference for the DRL to develop and hone in its prediction capability while enriching its knowledgebase and build up the information for DRL's NN minibatch repository. This effort trains up the FR's DRL-NN slowly over time with minimum or no a-prior knowledge of the environment. By the end of its learning phase, it has achieved an expert level and is able to recognize the state input to predict the service outage. But not all DRE faults can be resolved by the FR module, it should alert the IT administrator about the failed outcome of its fault resolution.

There are two categories of faults vs actions that the FR module can encounter: known-known and known-unknown. For those known faults with known solutions, if the FR module is unable to dispense the proper corrective actions, then the solution is to optimize and retrain its NN to improve its prediction. For those known-unknown where there are no known solutions to identified faults, then the IT administrator is required to assist to evaluate other new possible methods of changes and then write a new system command routine for the action libraries and then retrain the NN to include the new routine. So, these are two paths of recourses for failed resolution outcomes.

The concept of using the FR module instead of the SC module is in its transfer learning where copies of the FR modules can be replicated to other FDR systems to provide the capacity of an intelligent Fault resolving agent but protect the proprietary knowledge of the fault-to-resolution expertise. The results from the FR experiments had proven its capability. In the next chapter, the combined FD and FR module for the FDR system and their future are discussed.

The entire content of this chapter has been accepted and published with the 17th International Conference on Advanced Data Mining and Applications (ADMA) [72].



# CHAPTER 6: FDR – FD AND FR INTEGRATION

---

Both FD and FR modules have been designed, developed, and tested against various simulated DRE faults in the previous two chapters. The next step is to consolidate them together and run them as a single FDR system. Now the complete FDR has both modules working together to serve the goal that this thesis has set out, which is to detect errors within the DRE, predict the specific type of software service's outage to the user that these faults can affect. It refers to the system diagnostics information that it has previously captured and stored, then repeatedly searches within its knowledge to interact with the DRE until the faults are resolved or until it has exhausted all tries before notifying the IT administrator for assistance.

## 6.1. Background of integration testing

In software and machine learning testing, there are three types of testing: unit, regression and integration. The unit testing focus on individual specialised functionality, the regression testing is to test the reliability of the modules to discover errors and resolve them. The integration testing combine all the rest of the modules and test their higher-level operational behaviour of all the various units from the codebase [73]. The difference between software and ML testing is the nature of inputs and outputs. For Software testing, data and logic are the inputs into the software module while the output's desired behaviour is measured. But in ML testing, the inputs are comprised of the data and desired behaviour, while the output is the logic. The goal is to observe the operation from a higher perspective when the modules of FD and FR are integrated, including the addition of other supporting programs and functions such as OS, networking, Oracle and Shareplex libraries. This is to ensure that there are no contention or integration issues that can impact the new FDR system's functionality.

For the common integration testing of ML models, the common approaches are offline and online testing [74]. In Offline ML testing, the model has not been deployed for any use or it is still undergoing training and tuning. For online testing, the ML

modules may be currently active in use and the test is conducted to compare and improve the ML model competency. These methods are common for supervised and unsupervised learning models deployment. However, the FD and FR are based on Reinforcement learning which requires iterative interaction with the DRE with their respective guiding SD and SC modules, and it requires a different approach such as continuous integration (CI) testing [75]. The basis for the CI is due to the constant and continuous changes of test cases with the expectation of quick feedback into the model's adaptation and time execution constraints [75]. Therefore, the RL-specific CI testing is conducted at the unit level for FD and FR modules, whereas the integration testing employed the finished trained models to ascertain their performance and functionality with no CI process for the FD and FR modules. Should there be new cases of DRE events, they will be introduced for the modules at their unit testing level. The CI testing for the FDR as an integrated system will be regarded as part of future enhancements.

In the FD and FR integration testing process, the objective is to simulate the ability of the trained FD and FR modules operating in the DRE and observe their behaviour. its operating capability is measured by the metrics of its outputs to ascertain if it meets the criteria and checks for errors. However, the FR module played a more significant role in the combined FDR as it holds the critical function of resolving the faults, as compared to FD which hold the role of the less critical but still important service outage reporting. the integration test also covers the performance metrics of the ML modules using RL to resolve the DRE faults as compared to the rule-based intelligence that the SD and SR function.

## **6.2. FDR Test Analysis**

This section discussed the test results from the integration testing, together with the findings.

### **6.2.1. Usage of Software**

There are no changes to the test environment for the integration-test phase and it is the same setup that we used for chapters 5 and 6. Only the python functions and other shell scripts that supported the FD and FR are integrated into the common libraries, bearing labels to differentiate them and the modules that they serve.

### **6.2.2. Experiment setup and goal**

The DRE setup for the integration testing remains the same without any changes from the setup that both FD and FR modules tests were conducted. Configuration of Shareplex, OracleDB, network and OS retains the exact configurations, parameter settings and patch level to ensure consistency and stability in the test environment. The test procedure follows the same protocol which introduces the simulated fault for the test and is followed by restoration of service through the execution of applying rollback scripts to restore the various DRE's functions.

The goal is to observe how the integrated FDR can react under the fault simulation and derive their outcome plus observing their behaviour when they integrated to work together. One of the main aspects to observe is the overlapping of specific software operations between FD and FR against the various software components in the DRE that potentially can cause conflicts or contention that may aggravate the fault states to the level where the service restoration scripts may not be able to perform correctly. The outcome of the integration testing should not have lesser accuracy as compared to the FR's unit testing outcomes as the integration testing is merely to ensure that the two modules can work together seamless without errors. Table 16 lists the simulated test faults that are injected into the DRE and the corresponding scripts that restore or revert them for Shareplex, OracleDB, network and OS. While the faults can be injected at random, the restoration of their services will require a proper sequence that is based on the hierarchy of the DRE's software inter-dependency, where one software's restoration of service takes precedence of another that depends on it.

### **6.2.3. Test analysis procedures**

The focus now is to validate the efficiency and effectiveness of the FDR toward the DRE. Most of the tests have been done at the FR stage and that has concluded most of the criteria that have been laid out for the FDR, such as the time taken between the FR's SC unit versus the FR's NN unit which significantly shortens the turnaround time in producing corrective actions as shown in figure 32. This is the goal of this thesis which is to develop a new approach in resolving faults in a fast and efficient manner while able to expand its learning ability and adaptive to an ever-changing DRE.

The next area is on the FDR's effectiveness. The faults that are used for both the training and testing are controlled within the tolerance of the DRE's functionalities. The slightest deviation of parameters, values, or configuration, including the syntax on the system commands, can cause chaos to the overall FDR's process and render it ineffective. The faults that the FDR referred to for training and testing are anticipated ahead with minimum guesswork available. Its effectiveness is tested against the same batch of known faults but with a different combination of various software element's faults to provide a more complex test scenario that can test every combination of system diagnostics and fault resolution processes available in the FDR are validated. The FDR's FR module has been proven to handle faults from various elements of OracleDB, Shareplex, network and OS individually. This current set of tests will combine them to ascertain how effective the FDR can be after both the FR and FD modules are merged. Table 19 list out all the test cases with the various combination of faults tests that span across the various software groups and their respective elements' units with the combination that range from 2 to 9, together with the validation test results. With each test, the FD modules' outputs are included in and their responses are tracked in the diagnosed status columns. The FDR's FR module is using the DRL-NN unit to predict the corrective actions but the number of iterations that the FR's SC unit performs against similar faults are recorded to ascertain the efficiency of the FDR's FR modules, comparing the number of iterations it must perform using both SC and DRL-NN units, as well as the time, is taken.

Table 19 - Test scenario and increase the number of simultaneous occurring faults for each case

Fault #	Software	Software groups' units impacted	No. Of Faults	Faults' System Commands	Diag Status	Resolved Status	FR's SC runs	FR's NN runs	Time Taken FD (S)	Time Taken FR (S)	Time Taken SC (S)
---------	----------	---------------------------------	---------------	-------------------------	-------------	-----------------	--------------	--------------	-------------------	-------------------	-------------------

1	OracleDb	Main Processes	2	export ORACLE_SID=DB1 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba" && echo password su - splx -c \$MDIR/shutdown.sh	passed	resolved	2	2	10.8	17.2	27
2	OracleDb	Process & Privileges	2	export ORACLE_SID=DB2 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba" && echo "Alter user splx account lock;" sqlplus -s "system/password@DB1"	passed	resolved	3	2	11.8	23.2	51.2
3	OracleDb	Privileges	1	echo "Alter user splx account lock;" sqlplus -s "system/password@DB1"	passed	resolved	1	1	10.2	11.2	17.2
4	OracleDb	Security	1	echo "alter user splx quota 0 on USERS;" sqlplus -s "system/password@DB1"	passed	resolved	0	0	11	0	5
5	OracleDb	Main Processes	1	echo password su - splx -c \$MDIR/shutdown.sh	passed	resolved	1	1	10	12.4	15.2
6	OracleDb	Privileges, Operation	2	echo "Alter user splx account lock;" sqlplus -s "system/password@DB1" && echo "stop capture" sp_ctrl	passed	resolved	2	2	11.8	31.2	36.6
7	OracleDb	Privilege, Process, Operation	2	echo "Alter user splx account lock;" sqlplus -s "system/password@DB1" && echo password su - splx -c \$MDIR/shutdown.sh	passed	resolved	2	2	11.4	29.2	25.8
8	OracleDb	Privilege, Operation	2	echo "Alter user splx account lock;" sqlplus -s "system/password@DB1" && export ORACLE_SID=DB2 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	3	2	10.6	18.8	53
9	Shareplx, OracleDb	Operation, Main Process	5	echo "Alter user splx account lock;" sqlplus -s "system/password@DB1" && echo "stop post" sp_ctrl && echo "stop capture" sp_ctrl && echo "stop export" sp_ctrl && echo "stop import" sp_ctrl	passed	resolved	9	6	11.2	64.8	127.4
10	Shareplx, OracleDb	Operation, Main Process	4	echo "Alter user splx account lock;" sqlplus -s "system/password@DB1" && echo "stop read" sp_ctrl && echo "stop import" sp_ctrl && echo "stop post" sp_ctrl && echo "stop read" sp_ctrl	passed	resolved	9	6	10	64.8	109.4
11	Shareplx, OracleDb	Operation, Main Process	3	echo "stop capture" sp_ctrl && echo "stop export" sp_ctrl && export ORACLE_SID=DB1 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	2	2	11.2	26.8	34.6
12	Shareplx, OracleDb	Operation, Main Process	3	echo "stop capture" sp_ctrl && echo "stop import" sp_ctrl && export ORACLE_SID=DB2 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	9	5	10	48	116.6
13	Shareplx, OracleDb	Operation, Main Process	3	echo "stop capture" sp_ctrl && echo "stop post" sp_ctrl && export ORACLE_SID=DB1 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	3	3	11.4	36.6	41.6
14	Shareplx, OracleDb	Operation, Main Process	3	echo "stop capture" sp_ctrl && echo "stop read" sp_ctrl && export ORACLE_SID=DB2 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	4	3	10	36	57.8
15	Shareplx, OracleDb	Operation, Main Process	3	echo "stop export" sp_ctrl && echo "stop import" sp_ctrl && export ORACLE_SID=DB1 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	7	4	11	58.4	91.8
16	Shareplx, OracleDb	Operation, Main Process	3	echo "stop export" sp_ctrl && echo "stop post" sp_ctrl && export ORACLE_SID=DB2 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	4	3	10	39.6	63.4
17	Shareplx, OracleDb	Operation, Main Process	3	echo "stop export" sp_ctrl && echo "stop read" sp_ctrl && export ORACLE_SID=DB1 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	3	3	11.6	24	39.2
18	Shareplx, OracleDb	Operation, Main Process	3	echo "stop import" sp_ctrl && echo "stop post" sp_ctrl && export ORACLE_SID=DB2 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	8	3	11	37.2	118.6
19	Shareplx, OracleDb	Operation, Main Process	3	echo "stop import" sp_ctrl && echo "stop read" sp_ctrl && export ORACLE_SID=DB1 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	8	4	12	35.2	99.4
20	Shareplx, OracleDb	Operation, Main Process	3	echo "stop post" sp_ctrl && echo "stop read" sp_ctrl && export ORACLE_SID=DB2 && echo "shutdown immediate;" sqlplus -s "sys/password as sysdba"	passed	resolved	4	3	10.8	25.2	65.8
21	Shareplx, OracleDb	Operation, Privileges	4	echo "stop capture" sp_ctrl && echo "stop export" sp_ctrl && echo "stop import" sp_ctrl && echo "Alter user splx account lock;" sqlplus -s "system/password@DB1"	passed	resolved	8	5	10.2	68	112.2
22	Shareplx, OracleDb	Operation, Privileges	4	echo "stop capture" sp_ctrl && echo "stop export" sp_ctrl && echo "stop post" sp_ctrl && echo "Alter user splx account lock;" sqlplus -s "system/password@DB1"	passed	resolved	4	4	11.6	54.4	59.4
23	Shareplx, OracleDb	Operation, Privileges	4	echo "stop capture" sp_ctrl && echo "stop export" sp_ctrl && echo "stop read" sp_ctrl && echo "Alter user splx account lock;" sqlplus -s "system/password@DB1"	passed	resolved	4	4	11	63.2	63.4
24	Shareplx, OracleDb	Operation, Privileges	4	echo "stop capture" sp_ctrl && echo "stop import" sp_ctrl && echo "stop post" sp_ctrl && echo "Alter user splx account lock;" sqlplus -s "system/password@DB1"	passed	resolved	5	4	11	48	58
25	Shareplx, OracleDb	Operation, Privileges	4	echo "stop capture" sp_ctrl && echo "stop import" sp_ctrl && echo "stop read" sp_ctrl && echo "Alter user splx account lock;" sqlplus -s "system/password@DB1"	passed	resolved	9	5	11.2	47	109.4
26	Shareplx, OracleDb	Operation, Privileges	4	echo "stop capture" sp_ctrl && echo "stop post" sp_ctrl && echo "stop read" sp_ctrl && echo "Alter user splx account lock;" sqlplus -s "system/password@DB1"	passed	resolved	4	4	10	33.6	62.6
27	Shareplx	Operation	3	echo "stop export" sp_ctrl && echo "stop import" sp_ctrl && echo "stop post" sp_ctrl	passed	resolved	7	4	12	36	103
28	Shareplx	Operation	3	echo "stop export" sp_ctrl && echo "stop import" sp_ctrl && echo "stop read" sp_ctrl	passed	resolved	6	4	11.4	37.6	99.8
29	Shareplx	Operation	3	echo "stop export" sp_ctrl && echo "stop post" sp_ctrl && echo "stop read" sp_ctrl	passed	resolved	3	3	10	43.8	39.8
30	Shareplx	Operation	3	echo "stop import" sp_ctrl && echo "stop post" sp_ctrl && echo "stop read" sp_ctrl	passed	resolved	7	5	12	69	112.8
31	Network	Configuration	3	mv -f \$ORACLE_HOME/network/admin/listener.ora \$ORACLE_HOME/network/admin/listener.ora.orig && \$ORACLE_HOME/bin/lsnrctl stop && \$ORACLE_HOME/bin/lsnrctl start	passed	resolved	1	1	10.8	12.4	18.2
32	Network	Main Processes	1	\$ORACLE_HOME/bin/lsnrctl stop	passed	resolved	1	1	10.8	11	16.2
33	Network	Configuration	1	mv -f \$ORACLE_HOME/network/admin/tnsnames.ora \$ORACLE_HOME/network/admin/tnsnames.ora.orig	passed	resolved	1	1	10	11.4	18.6
34	OS	Security	1	echo password su - root -c "passwd -l splx"	passed	resolved	1	1	10.2	14.6	18.8
35	OS	Security	1	echo password su - root -c "passwd -l oracle"	passed	resolved	1	1	12	9.4	18

### 6.3. Test results and analysis

This section discusses the test results from the integration testing, together with the findings.

#### 6.3.1. FDR modules performance results

Figure 35 shows the time used among the two main FD and FR modules but FR's details are split into two using both its SC and DRL-NN unit to form a visual comparison. Note that the time taken by the FD to produce the service outage report is fast, occurring in less than 3 secs. For the FR, the time spent is significantly higher as they need to interact with the DRE's software as some of them require some time to start up or enable their services. Figure 34 shows that the time spent using the SC takes a much longer time to reach the end goal of resolving all the faults as it followed its algorithm to handle the faults one at a time. The method of using the DRL-NN proves to be much more efficient and time-saving since it predicts only effective actions that have been learnt from its knowledge.

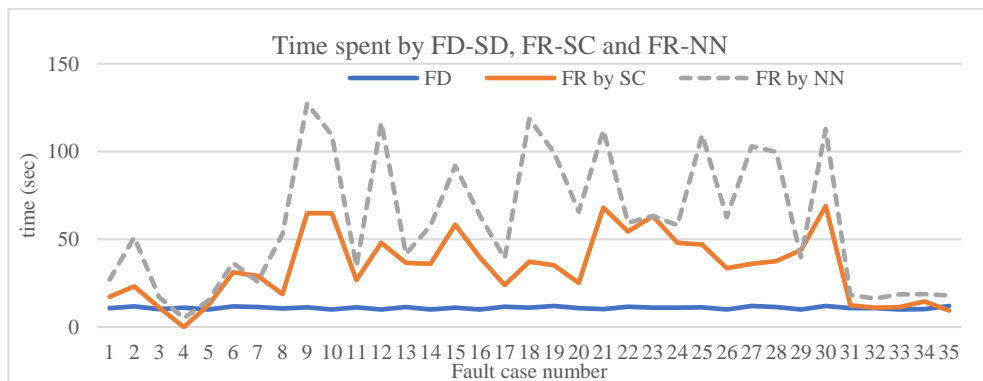


Figure 35 – Time taken by FD to diagnose, FR to resolve faults using SC and NN unit

#### 6.3.2. FDR integration test outputs and findings

Table 20 shows some of the results from the validation test using the test cases from table 18. Each of the faults is printed with the system commands to induce the faults. The FD performance the query against the DRE retrieving and compiling all the information and statistics, then predict the service outage information of the DRE based on the software's services logical grouping. The diagnostics information is passed to the FR module, and it is depicted as the current starting state. The FR module predicted the corrective action, depicted in the table as the single element correction

action to tackle the faults based on its knowledgebase. Then the FR retrieves a new state from the DRE after it executed the action. If there are some positive changes, some of the faults that have been previously captured will no longer be there and their diagnostics will not flag any error. The FR repeats the next iteration and focuses on the next faults to fix. The positive changes have been indicated with a reward of 1. But this plays only as a visual indicator unlike in the previous chapter. This keeps on going until the new state of the DRE return zeros for all the fault indication. Note that some of the more complex faults with the higher number of faults combination require a much longer iteration to repair, whereas those with 2 or fewer faults, get fixed up rather quickly.

Table 20 - Results from FR integrated testing

```
#####
fault count= 0 fault cmd= export ORACLE_SID=DB1 && echo "shutdown immediate;"|sqlplus -s "sys/password as sysdba" && echo
password|su - splx -c $MDIR/shutdown.sh
#####
DRE log.process & service stats= [0,64655058,76223968,0,0,0,64351381] [1,0,1,0,0,0,0] [1,0,0,0,0,0,0], SD Service Outage =
[[6,1,0,1,3,1],[2,2,1,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE_current_state = [1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0]
index= 19 correct cmd= export ORACLE_SID=DB1 && echo "startup;"|sqlplus -s "sys/password as sysdba"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], CHANGE reward = 1
DRE log.process & service stats= [0,0,0,0,0,0,64351381] [0,0,1,0,0,0,0] [0,0,0,0,0,0,0], SD Service Outage =
[[6,1,0,1,3,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE_current_state = [1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 0 correct cmd= echo password|su - splx -c $MDIR/startup.sh
single_element_corrective_action= [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], CHANGE reward = 1
resolved
#####
fault count= 1 fault cmd= export ORACLE_SID=DB2 && echo "shutdown immediate;"|sqlplus -s "sys/password as sysdba" && echo "Alter
user splx account lock;"|sqlplus -s "system/password@DB1"
#####
DRE log.process & service stats= [46968001,0,0,0,0,0,64351381] [0,1,0,0,0,0,0] [0,1,0,0,0,0,0], SD Service Outage =
[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE_current_state = [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0]
index= 22 correct cmd= export ORACLE_SID=DB2 && echo "startup;"|sqlplus -s "sys/password as sysdba"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], CHANGE reward = 1
DRE log.process & service stats= [46968001,0,0,0,0,0,64351381] [0,0,0,0,0,0,0] [0,0,0,0,0,0,0], SD Service Outage =
[[0,0,0,1,0,1],[0,0,1,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE_current_state = [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 27 correct cmd= echo "Alter user splx account unlock;"|sqlplus -s "system/password@DB1"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], CHANGE reward = 1
resolved
#####
fault count= 18 fault cmd= echo "stop import"|sp_ctrl && echo "stop read"|sp_ctrl && export ORACLE_SID=DB1 && echo "shutdown
immediate;"|sqlplus -s "sys/password as sysdba"
#####
DRE log.process & service stats= [0,64655058,0,0,0,0,64351381] [1,0,0,0,0,0,0] [1,0,1,0,0,0,0], SD Service Outage =
[[3,0,3,1,0,1],[2,2,1,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE_current_state = [0,0,0,1,1,1,0,0,1,0,1,0,1,1,0,0,0,1,0,1,1,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0]
index= 19 correct cmd= export ORACLE_SID=DB1 && echo "startup;"|sqlplus -s "sys/password as sysdba"
single_element_corrective_action= [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,1,1,1,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], CHANGE reward = 1
DRE log.process & service stats= [0,0,0,0,0,0,0] [0,0,0,0,0,0,0] [0,0,1,0,0,0,0], SD Service Outage =
[[3,0,3,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE_current_state = [0,0,0,1,1,1,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 4 correct cmd= echo "start read"|sp_ctrl
single_element_corrective_action= [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], CHANGE reward = 1
DRE log.process & service stats= [0,0,0,0,0,0,0] [0,0,0,0,0,0,0] [0,0,1,0,0,0,0], SD Service Outage =
[[2,0,1,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]
DRE_current_state = [0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
index= 5 correct cmd= echo "start import"|sp_ctrl
single_element_corrective_action= [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
DRE_new_state = [0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], CHANGE reward = 1
```





Therefore, care is taken to segregate both to ensure that they do not impact one another or to cause conflict and deadlocks. The results that are received must be integrated into the mini-batch knowledgebase which subsequently can be reused for continuous module training and testing. The steps in which the series of actions that the FDR need to restore the services for any given faults must adhere to the hierarchical importance and reliance among the software within the DRE's configuration. However, the FDR has shown that it can navigate and learn this order through its trial-and-error iterative RL routines, which fulfil the primary feature of what RL is, and that is to maximize the rewards by using the best actions for each fault situation that it encounters.

# CHAPTER 7: CONCLUSIONS

---

The rapid growth in the use of complex, multi-tiered IT systems across many industries has posed a unique challenge to the IT personnel that support them. The number of different software functioning with thousands of configurations and parameters to serve other software and technology to meet the needs of the business is increasing exponentially and it is stretching the pool of system administrators' resources to the limit. Not only do they face the daily stress of supporting these systems but the expectation to have a fast turnaround time to resolve any faults in the production system environment grows daily. The Service Level Agreement (SLA) that most mission-critical systems commanded are expected to be at least at 99.5% uptime and above. Therefore, it is expected that the IT administrator must always be present and focus on monitoring them closely. In addition to this, organisations have difficulty in increasing their cost and talent pool to increase their resources. This research is an attempt to use machine learning to propose a novel way to complement the IT administrator in monitoring and resolving any systems faults encountered in the production systems.

This thesis has contributed to the research of a better fault resolution approach toward complex, multi-tier software systems via a novel faults identification and mapping to solution method that this thesis has developed, including the use of a customized version of the Deep reinforcement learning model. This chapter list the answers to the research questions that were listed at the beginning of this thesis based on the knowledge and experiences gained in this research. It also summarized the research contributions and findings together with potential future works and enhancement.

## **7.1. Research Contributions**

The research made in this thesis has some positive contributions to the space of fault diagnosis and resolution in the domain of enterprise data replication using machine learning models such as Deep reinforcement learning. Enterprise data replication environment across large enterprise IT environments are getting complicated and this new FDR introduces an alternate option to the academia and

industry in the use of an intelligent agent that can learn adaptively to resolve detected faults that arise from the multitude of participating software in the replication setup. For any normal expert system that uses supervised learnings or rule-based, they serve the current model with the provision of an available large dataset for training or, they need a substantial number of resources and time to compute the optimal feedback for the environment input, including the requirement for a large proven dataset of faults and actions to support deep learning. The FDR, with its DRL model, doesn't have this pre-requisite for the a-prior knowledge base. The FDR can reduce the amount of unnecessary computation required as it stored past known actions and results against various environment states as part of its knowledge base. It learns by self-playing and self-testing to validate all the potential faults that can occur among the participating software across the DRE. This saves not only time and resources but improves the speed of response. It can handle unknown states by using part of its routine to explore better new solutions and thereby adding more information into its knowledge base.

If the DRL's knowledgebase has all the known states with their optimum actions to resolve the faults, the DRL can also be structured to handle problems within the environment states in a hierarchical form as shown from the FR module, by increasing the value in the rewards to steer the algorithm to favour those with higher reward values. the newer states from the environment are treated equally as another instantiation of state-action tuples and the DRL will exploit its knowledgebase to provide another optimal action for it. This can be regarded as a two planes level between existing and new states where the best corrective actions can steer the environment to a better goal direction based on their state-action's rewards and Q-value.

The research also develops a novel method in using the details from the system diagnostics to develop a systematic hierarchical approach to resolve software faults based on their importance, element by element, based on a relationship matrix between diagnosed faults to the list of corresponding correcting system commands and scripts. The approach is not to use this as the primary mode of fault resolution but to learn to fix them following the SC's algorithms and build up the knowledgebase sufficiently enough before the next phase, which is the Neural network, to learn predict the best course of corrective actions for any given DRE's state regarding the knowledge that is

gained from the SC phase. This is more efficient than the brute force method of exhaustive, high iteration of randomly selecting all related actions to apply for any given fault. The FDR can be able to provide high accuracy while applying the right actions for all sorts of DRE's states and their errors encountered.

This system also allows cross-platform inter-operations, not restricted to any specific technology, unlike the current commercial proprietary system that is limited to the software brands. The software vendors that develop their system usually rule or statistics-based, and they do not integrate with competitor's software due to privacy and copyrights. This method of expressing various faults to corrective actions and exploit the external libraries set the initiative for better and greater autonomous and adaptive fault resolving systems that are based on machine learning.

The rise of enterprise cloud service is posing a potential big challenge to the administrative work responsibility for IT administrators as it increases the landscape of data replication and ingestion requirements especially the multitude of data marts and enterprise data warehousing projects increase in multiple folds. They require even more technology and software functions to manage a variety of ETL and ELT processes with heterogenous data sources that are based in data lakes, blob storage, IoT streams and databases. The data that they manage can be in structured, semi-structured or unstructured form. Therefore, this ETL/ELT software require intensive IT administrators' attention to keep them working at 99.95% Service Level Agreement. Therefore, this approach of FDR with the DRL can be set up to mitigate and manage this class of complicated environments, able to interact and learn the various technique and derive the correct flow of troubleshooting steps in hierarchical order to restore the faults if they are detected. The anticipated contribution of the FDR to the IT industry is huge and there is a big potential that it can be applied to other IT sectors and the environment as well.

## **7.2. Comparison of FDR to other Diagnostic and Resolution Methods**

There are several methods of fault diagnosis and resolving them as listed in Table 21 and they range from the most common method of having IT personnel performing hands-on work to rectify the faults when the faults are detected. The fault detection can be employed in numerous ways; from the common manual method where IT administrators must run multiple queries against different software systems, to write

bash scripts that can execute similar commands but through an OS' task scheduler, and the final option of having a commercial software package that can do the monitoring intelligently. However, most of this software does not handle fault resolution and it is up to the IT administrators to handle that. How efficient and effective each of the manual intervention against the detected faults depend strictly on the competency of the IT administrator. And it is not cost-effective nor practical to employ many IT personnel to cover the operation of the DRE and their maintenance around the clock. The landscape of the DRE grows exponentially and the limited number of resources, including manpower, cannot scale efficiently to meet the demand.

For fault resolution, one of the most common methods for any IT administrator to fix any system-related fault is to do it hand-on with the DRE software directly. Usually, they must refer to some knowledgebase that comes in vendor's recommended best practices or support websites that the organizations have to pay a premium to access the software vendors' knowledgebase or more if they want additional professional support. This is preferred but it is not feasible for every IT administrator to have the necessary skill nor experience to handle the overall troubleshooting tasks and with outsourcing becoming prevalent in the IT industries, getting someone to respond to DRE faults across the globe at different time zone is tedious and cumbersome.

Therefore, some of the experienced IT administrators took to the task to write a substantial number of custom scripts to do some of the basic troubleshooting and correction using simple conditions and rule-based. But this is limited on most occasions as not all of them can cross-skilled to other software or limited by their jurisdiction and ring-fencing in terms of job responsibility. The rule or condition-based method of their fault resolution are fixed and have limited adaptability, so if they are going to be deployed to support other forms of DRE with heterogeneous platform and technology, the entire scripts need to be revamped and rebuild. This is both laborious and maintenance, in the long run, is difficult. Based on the Delphi method, we present our method as well as the others to a group of experienced IT administrators from Energy Queensland Limited and gather their feedback. Each method has their strength and weakness described as shown in table 21 below.

Table 21 - Benchmarking FDR against other methods of fault diagnosis/resolution

<b>Methods</b>	<b>Strength</b>	<b>Weakness</b>
Manual/ Hand-on	Slow, error-prone, passive, limited in feature and coverage, can only handle limited complex issues, not scalable	Fault detection, limited diagnosis, low resolution capability
Vendors' best practice and knowledgebase	Slow, error-prone, passive, limited in feature and coverage, can only handle more complex issues, not scalable, enhance IT admin's knowledge and capability.	Fault detection, medium diagnosis, medium resolution capability
Professional support	Very slow, passive, expensive, able to mitigate high complex problem, not scalable, enhance organisation's IT competency, not scale	Only for fault resolution
Small shell script with task scheduler	Fast, proactive, more reliable, better feature and coverage, scalable, improve monitoring and diagnosis reliability	Fault resolution is very limited
Program with condition or Rule-based	Faster, proactive, better reliability, more feature and coverage, scalable to handle more system, improve monitoring and diagnostic ability including limited fault resolution	Not adaptive, limited in scope, require major rework if configuration changes, not flexible to adapt
FDR	Faster, proactive, reliable, handle complex, multi-tier software system, able to do fault detection, diagnosis, and resolution, adaptive,	Need to predefine all the potential faults and corresponding action to remediate them. Require massive amount of pre-scripted commands to handle all fault scenarios. May be labour intensive

Currently, there is limited automated fault diagnostic method in academia that explicitly applies to diagnose a complex multi-software Data replication integrated system. Even in the industry, the fault diagnostic for each of the software is limited to the individual vendors that produce it. Therefore, there isn't a large base of available methods for troubleshooting DRE that can be used for empirical analysis and comparison.

The consensus on the usability of FDR is promising as compared to all other methods. For system troubleshooting to be done manually or with their software utilities, the level of fault diagnosis that can be achieved is strongly correlated depending on skillset of the DBAs. Commercial fault diagnostic tools can expedite and reduce the tedious tuning effort, but they are also limited to software's specificity and the human factor. Also, they do present a potential risk to the production environment as they require direct interaction with the production DRE.

The other methods that are rule, heuristic and NN based, provide a higher level of autonomy and greater coverage throughout the day without being dependent on humans, and they do provide a high probability of detecting and diagnosing the faults. Moreover, they can be duplicated to cover other instances of DREs. However, for the NN-based model, it has a pre-requisite of an existing large dataset to start with. Other methods such as rule-based may be static and limited by its internal knowledge base of rule settings. All of them also have a significant risk impact on the production of DRE if diagnosing efforts are made against them directly.

### **7.3. Future Works and Enhancement**

While the FDR has been proven to be an intelligent fault diagnosis and resolution approach for the enterprise set up of tools that replicate data, it is a **proof of concept** with plenty of room for improvement. One of the main attributes with FDR is the amount of information from DRE software, versus the libraries of system commands for each software element, the relationship matrices of errors to faults, outages and the respective fault-remediating actions are both huge and static. To make any amendment either to accommodate new software or feature to monitor, to diagnose or to be corrected, need updates among the various FDR's configuration files from end to end which is both laborious and time-consuming. so because of this, it is desirable that greater flexibility and information storage, including retrieval, can be improved either by creating a higher dimensional matrix that accommodates multiple software integration relationships with greater ease or allow a more dynamic method of ingesting a set of system commands and store shell scripts internally into a central repository and update their relationship to the associated software's list of faults dynamically with a procedure. A backend database repository to facilitate all these would-be new enhancements will be better too, as it allows easy information query and storage with more security and performance.

Another issue that has been encountered in this research is the set of system commands that are used to alter the software element's configuration. All of them are hardcoded but upon deeper analysis, it is found that they tend to follow repetitions of statements except for values used in their parameters. Ideally, this can be replaced with a new unit that can perform NLP specifically to each software environment using some pre-set configuration or parameter values. This proposed NLP system command and

control module can also formulate new query languages that scan and gather all the real-time statistics and variables from the software. This will certainly give the FDR greater flexibility in adapting to a more complex integrated software environment. This can overcome the current cumbersome method of hardcoding all the fault correcting actions into a large list of system commands which may be difficult to maintain and prone to errors.

For future enhancement, the FDR can benefit if it can be ensembled with other models such as Monte Carlo Tree Search (MCTS) to strategize the solution paths among the large state space of combinations for the fault resolution actions, similar to Alpha-Go Zero's strategies [76]. MCTS is used to build a local policy to save the subsequent move for fault resolution, by searching for moves and record the results in a search tree which enrich a knowledge base of a hierarchical tree structure and from it, local policies are made and used to support the subsequent solution search. The current FDR has been designed to respond to the move in a hierarchical manner which limits the possibility of the model to search linearly across other potential action-space that can potentially resolve the fault situation. This is a limiting factor and MCTS may overcome this.

The performance of the FDR design can also be improved by rewriting the software libraries that support the DRE's software which inject the simulated faults and restore their services in the attempt to support DRL's learning process. The preparation for these libraries and their functions is time-consuming and proprietary to each software vendor, especially on the service restoration where the DRE's state is required to restore to the baseline for the iterative learning trial. The scripts used in this FDR research have been hard-coded and stored in the libraries to support the various DRE's software. It is laborious and to maintain them, with multiple repetitions. It is advisable to execute them via meta-data, configurations and parameters which allow greater flexibility in code readabilities, and thus reduce the number of scripts required. This can be replaced with a database that contains all the combinations of parameters, variables, account id, details and passwords which can be referred to, and construct the required working scripts to serve different functions with ease [77, 78].

One of the potential future work that can extend the FDR's capability to cover other complex IT systems with a matrix that can map the complex multiple inter-



operating dependencies among the software and their services to each other [79]. The current FDR require expert's insights who knows the intricacy of the DRE's software relationship but should this FDR extend to other heterogeneous replication systems that are not part of the current Shareplex/Oracle expert, then it will be difficult to re-establish the FDR for the new setup. A matrix of software's inter-operating relationship can help to mitigate this challenge, and this, in turn, supports a range of stored procedures or programs that can construct the relevant scripts to support each of the software's specific functions together with the adjacent co-dependent software their corresponding functions or services [80].

#### **7.4. Conclusion**

The DRE is a complicated IT system that serves the need for enterprise data replication that span multiple organisations and across geographical regions. It is not cost-effective to employ hundreds of IT administrators to support this DRE around the clock with a near real-time turnaround in fault resolution effort. The research here developed a novel autonomous fault detection and resolution system that is adaptive to the changing technology and business landscapes to provide a responsive interaction that is equivalent to a human level 1 technical support in rectifying the faults within the DRE and it comes in the form of an adaptive intelligent system. Most of the commercial and academic fault resolution system relies on either a decision-based system that is based on rules or conditions or a NN to prescribe the remedial actions. However, these popular approaches have their drawbacks; for rule-based systems, the model of the IT system must be predetermined, and it contains a list of fixed rules which aren't scalable and scripts that are proprietary to the vendors. Plus, they are only designed for specific software, and they do not usually work well across other platforms or technology. Those systems that depend on deep learning can be more flexible and scalable as compared to those rule-based ones, but they require a large amount of labelled dataset for training, something which is not readily available in large IT setups that have different policies and configurations. However, the FDR system is not based on an explicit rule or decision logic, and it does not need to have a model based on DRE that is both explicit and well-defined. In any enterprise IT landscape, it is impossible to impose such control and not all the DRE sites share the same configuration or infrastructure. So, the FDR system is adaptive enough to learn

and serve the data replications setup. There are instances where FDR systems can be trained specifically to support individual systems separately and then combine to serve a higher and more global purpose. for example, one FDR system can be trained to serve a DRE that is tied into a client-server OLTP finance system whereas another can be trained to support another DRE via a web-based internet system. Both can be deployed to various organisations to run as virtual autonomous IT administrators that can work together to support a wide range of heterogeneous IT systems without revealing the knowledge. these FDR systems can have encrypted external libraries of system commands and scripts that the FDR system can exploit. But their secrecy can be protected without revealing to the users.

FDR has overcome these limitations and the research has shown that this novel approach does enable the FDR to be flexible enough to adapt its services of fault diagnosis and resolution to any multi-tier software and learn on the job dynamically with little or no a-prior dataset nor knowledge. While there is room for improvement, it is novel especially in the space of complex software involving enterprise data replication management.

## References

---

1. Hamrouni, T., S. Slimani, and F.B.J.E.A.o.A.I. Charrada, *A survey of dynamic replication and replica selection strategies based on data mining techniques in data grids*. 2016. **48**: p. 140-158.

2. Gill, N.K. and S. Singh, *A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers*. *Future Generation Computer Systems*, 2016. **65**: p. 10-32.
3. Malik, S.U.R., et al., *Performance analysis of data intensive cloud systems based on data management and replication: a survey*. 2016. **34**(2): p. 179-215.
4. Venkatasubramanian, V., et al., *A review of process fault detection and diagnosis: Part I: Quantitative model-based methods*. *Computers & chemical engineering*, 2003. **27**(3): p. 293-311.
5. Venkatasubramanian, V., et al., *A review of process fault detection and diagnosis: Part III: Process history based methods*. *Computers & chemical engineering*, 2003. **27**(3): p. 327-346.
6. Venkatasubramanian, V., et al., *A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies*. *Computers & chemical engineering*, 2003. **27**(3).
7. Van Hasselt, H., A. Guez, and D. Silver. *Deep Reinforcement Learning with Double Q-Learning*. in *AAAI*. 2016.
8. Lei, Y., et al., *Applications of machine learning to machine fault diagnosis: A review and roadmap*. *Mechanical Systems and Signal Processing*, 2020. **138**: p. 106587.
9. Littman, M.L., et al. *Reinforcement learning for autonomic network repair*. in *International Conference on Autonomic Computing, 2004. Proceedings*. 2004. IEEE.
10. Venkatasubramanian, V. and K. Chan, *A neural network methodology for process fault diagnosis*. *AIChE Journal*, 1989. **35**(12): p. 1993-2002.
11. Robinson, W.N., *A requirements monitoring framework for enterprise systems*. *Requirements engineering*, 2006. **11**(1): p. 17-41.
12. Russell, S.J. and P. Norvig, *Artificial intelligence: a modern approach*. 2016: Malaysia; Pearson Education Limited.
13. Žarković, M. and Z. Stojković, *Analysis of artificial intelligence expert systems for power transformer condition monitoring and diagnostics*. *Electric Power Systems Research*, 2017. **149**: p. 125-136.
14. Venkatasubramanian, V., R. Rengaswamy, and S.N. Kavuri, *A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies*. *Computers & chemical engineering*, 2003. **27**(3): p. 313-326.
15. Isermann, R., *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. 2006: Springer Science & Business Media.
16. Chandola, V., A. Banerjee, and V. Kumar, *Anomaly detection: A survey*. *ACM computing surveys (CSUR)*, 2009. **41**(3): p. 15.
17. Oliner, A., A. Ganapathi, and W. Xu, *Advances and challenges in log analysis*. *Communications of the ACM*, 2012. **55**(2): p. 55-61.
18. Garcia-Teodoro, P., et al., *Anomaly-based network intrusion detection: Techniques, systems and challenges*. *computers & security*, 2009. **28**(1): p. 18-28.
19. Xu, W., et al. *Detecting large-scale system problems by mining console logs*. in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 2009. ACM.
20. He, Z., et al. *A frequent pattern discovery method for outlier detection*. in *WAIM*. 2004. Springer.

21. Patcha, A. and J.-M. Park, *An overview of anomaly detection techniques: Existing solutions and latest technological trends*. Computer networks, 2007. **51**(12): p. 3448-3470.
22. Yang, H., F. Xie, and Y. Lu, *Clustering and classification based anomaly detection*. Fuzzy Systems and Knowledge Discovery, 2006: p. 1082-1091.
23. Boriah, S., V. Chandola, and V. Kumar. *Similarity measures for categorical data: A comparative evaluation*. in *Proceedings of the 2008 SIAM International Conference on Data Mining*. 2008. SIAM.
24. Steinwart, I. and A. Christmann, *Support vector machines*. 2008: Springer Science & Business Media.
25. Mukkamala, S., G. Janoski, and A. Sung. *Intrusion detection using neural networks and support vector machines*. in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*. 2002. IEEE.
26. Erfani, S.M., et al., *High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning*. Pattern Recognition, 2016. **58**: p. 121-134.
27. Agrawal, S. and J. Agrawal, *Survey on anomaly detection using data mining techniques*. Procedia Computer Science, 2015. **60**: p. 708-713.
28. Software, Q., *Shareplex for Oracle v9.1.4*. 2018.
29. *Quest Software Releases SharePlex v9*, in *ICT Monitor Worldwide U6 - ctx\_ver=Z39.88-2004&ctx\_enc=info%3Aofi%2Fenc%3AUTF-8&rft\_id=info%3Aasid%2Fsummon.serialssolutions.com&rft\_val\_fmt=info%3Aofi%2Ffmt%3Akev%3Amtx%3Ajournal&rft.genre=article&rft.atitle=Quest+Software+Releases+SharePlex+v9&rft.jtitle=ICT+Monitor+Worldwide&rft.date=2017-06-22&rft.pub=SyndiGate+Media+Inc&paramdict=en-US U7 - Newspaper Article*. 2017, SyndiGate Media Inc: Amman.
30. Brunt, B., *Going for gold: Dell Software's SharePlex database replication offering is a powerful tool with a small footprint*. Computer Reseller News (UK), 2016: p. 23.
31. *Dell Software Extends SharePlex to Optimize Data Integration and Analysis*. Information Technology Newsweekly, 2013: p. 136.
32. Kyte, T. and D. Kuhn, *Expert Oracle Database Architecture*. 2014: Apress.
33. Alapati, S.R. and Books24x7 Inc., *Expert Oracle database 11g administration*. Books for professionals by professionals. 2009, Berkeley, Calif.: Apress. 1 online resource (liii, 1344 p.).
34. Kyte, T., *Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming Techniques and Solutions*. 2010: Apress.
35. team, E.Q.s.D.O.-C.d., *Shareplex/Oracle setup annd operating procedure 2016 rev 2.0*. 2016.
36. Tabet, K., et al., *Data replication in cloud systems: a survey*. International Journal of Information Systems and Social Change (IJSSC), 2017. **8**(3): p. 17-33.
37. Milani, B.A. and N.J. Navimipour, *A systematic literature review of the data replication techniques in the cloud environments*. Big Data Research, 2017. **10**: p. 1-7.
38. Iacob, N., *Data replication in distributed environments*. Annals-Economy Series, 2010. **4**: p. 193-202.

39. Russell, S.J.N., Peter, *Artificial intelligence: a modern approach*. 2016.
40. Van Hasselt, H., A. Guez, and D. Silver. *Deep reinforcement learning with double q-learning*. in *Thirtieth AAAI conference on artificial intelligence*. 2016.
41. Littman, M.L., *Reinforcement learning improves behaviour from evaluative feedback*. *Nature*, 2015. **521**(7553): p. 445.
42. Lillicrap, T.P., et al., *Continuous control with deep reinforcement learning*. arXiv preprint arXiv:1509.02971, 2015.
43. Farivar, F. and M. Nili Ahmadabadi, *Continuous reinforcement learning to robust fault tolerant control for a class of unknown nonlinear systems*. *Applied Soft Computing*, 2015. **37**(Supplement C): p. 702-714.
44. Mellouk, A., *Advances in Reinforcement Learning*. 2011.
45. Szepesvari, C., Brachman, R. & Dietterich, T.G., *Algorithms for Reinforcement Learning*. 2010.
46. Mellouk, A., *Advances in reinforcement learning*. 2011, InTech.
47. Grondman, I., et al., *A survey of actor-critic reinforcement learning: Standard and natural policy gradients*. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2012. **42**(6): p. 1291-1307.
48. Fujimoto, S., H. Van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*. arXiv preprint arXiv:1802.09477, 2018.
49. François-Lavet, V., et al., *An introduction to deep reinforcement learning*. arXiv preprint arXiv:1811.12560, 2018.
50. Hinton, G.E., S. Osindero, and Y.-W. Teh, *A fast learning algorithm for deep belief nets*. *Neural computation*, 2006. **18**(7): p. 1527-1554.
51. Hippert, H., D. Bunn, and R. Souza, *Large neural networks for electricity load forecasting: Are they overfitted?* *International Journal of forecasting*, 2005. **21**(3): p. 425-434.
52. Brotherton, T. and T. Johnson. *Anomaly detection for advanced military aircraft using neural networks*. in *Aerospace Conference, 2001, IEEE Proceedings*. 2001. IEEE.
53. Henderson, P., et al. *Deep reinforcement learning that matters*. in *Proceedings of the AAAI conference on artificial intelligence*. 2018.
54. Li, Y., *Deep reinforcement learning: An overview*. arXiv preprint arXiv:1701.07274, 2017.
55. Jingang, C. *Using reinforcement learning for agent-based network fault diagnosis system*. in *2011 IEEE International Conference on Information and Automation*. 2011.
56. Zhang, Y., et al., *An artificial neural network approach to transformer fault diagnosis*. *IEEE Transactions on Power Delivery*, 1996. **11**(4): p. 1836-1841.
57. Holcomb, S.D., et al. *Overview on DeepMind and Its AlphaGo Zero AI*. in *Proceedings of the 2018 International Conference on Big Data and Education*. 2018. ACM.
58. Brockman, G., et al., *Openai gym*. arXiv preprint arXiv:1606.01540, 2016.
59. Chen, J. and R.J. Patton, *Robust model-based fault diagnosis for dynamic systems*. Vol. 3. 2012: Springer Science & Business Media.
60. Chen, Z., et al., *Random forest based intelligent fault diagnosis for PV arrays using array voltage and string currents*. *Energy conversion and management*, 2018. **178**: p. 250-264.

61. Cai, B., L. Huang, and M. Xie, *Bayesian networks in fault diagnosis*. IEEE Transactions on industrial informatics, 2017. **13**(5): p. 2227-2240.
62. Habibi, A., A. Sarafrazi, and S. Izadyar, *Delphi technique theoretical framework in qualitative research*. The International Journal of Engineering and Science, 2014. **3**(4): p. 8-13.
63. Wen, L., et al., *A new convolutional neural network-based data-driven fault diagnosis method*. IEEE Transactions on Industrial Electronics, 2017. **65**(7): p. 5990-5998.
64. Saimurugan, M., et al., *Multi component fault diagnosis of rotational mechanical system based on decision tree and support vector machine*. Expert Systems with Applications, 2011. **38**(4): p. 3819-3826.
65. Jia, F., et al., *A neural network constructed by deep learning technique and its application to intelligent fault diagnosis of machines*. Neurocomputing, 2018. **272**: p. 619-628.
66. Zhang, W., et al., *A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals*. Sensors, 2017. **17**(2): p. 425.
67. software, Q., *SharePlex 9.0.1 - Administration Guide*. 2019.
68. Dai, W., et al., *Fault Diagnosis of Rotating Machinery Based on Deep Reinforcement Learning and Reciprocal of Smoothness Index*. 2020.
69. Ding, Y., et al., *Intelligent fault diagnosis for rotating machinery using deep Q-network based health state classification: A deep reinforcement learning approach*. 2019. **42**: p. 100977.
70. Xu, T., et al. *Fault Diagnosis for the Virtualized Network in the Cloud Environment using Reinforcement Learning*. in *2019 IEEE International Conference on Smart Cloud (SmartCloud)*. 2019. IEEE.
71. Wee, C.K. and N. Wee. *Adaptive Fault Diagnosis for Data Replication Systems*. in *Australasian Database Conference*. 2021. Springer.
72. Wee, C.K., et al. *Adaptive Fault Resolution for Database Replication Systems*. in *International Conference on Advanced Data Mining and Applications*. 2022. Springer.
73. Zhang, J.M., et al., *Machine learning testing: Survey, landscapes and horizons*. IEEE Transactions on Software Engineering, 2020.
74. Braiek, H.B. and F. Khomh, *On testing machine learning programs*. Journal of Systems and Software, 2020. **164**: p. 110542.
75. Yang, Y., et al., *A systematic study of reward for reinforcement learning based continuous integration testing*. Journal of Systems and Software, 2020. **170**: p. 110787.
76. Wang, Q., Y. Hao, and J. Cao, *Learning to traverse over graphs with a Monte Carlo tree search-based self-play framework*. Engineering Applications of Artificial Intelligence, 2021. **105**: p. 104422.
77. Amorim, R.C., et al., *A comparison of research data management platforms: architecture, flexible metadata and interoperability*. Universal access in the information society, 2017. **16**(4): p. 851-862.
78. Dhawan, U., et al. *Architectural support for software-defined metadata processing*. in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2015.

79. Madni, A.M., et al. *Analyzing Systems Architectures using Inter-Level and Intra-Level Dependency Matrix (I2DM)*. in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. 2019. IEEE.
80. Sangal, N., et al. *Using dependency models to manage complex software architecture*. in *Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. 2005.