Adaptive Fault Diagnosis for Data Replication Systems

Chee Keong Wee[1(✉)] and Nathan Wee[2(✉)]

[1] Database Team, Energy Queensland Limited, Townsville, QLD, Australia
ck@outlook.com
[2] Faculty of Science, University of Queensland, Brisbane, QLD, Australia
nathan.wee@uq.net.au

Abstract. Data replication among multiple IT systems is ubiquitous among large organizations and keeping them running is a critical success factor for their IT departments. When services are disrupted, IT administrators must be able to find the faults and rectify them quickly. Due to the scale and complexity of the data replication environment, the fault diagnostic effort is both tedious and laborious. This paper proposes an approach to fault diagnosis of the data replication software through deep reinforcement learning. Empirical results show that the new method can identify and deduce the software faults quickly with high accuracy.

1 Introduction

Data replication is one of the essential IT services in a large organization where data is distributed and shared to service various business needs, and these replicating services are conducted by numerous commercial software which had been built for this purpose at very low latency. Their uptime and service are critical to the business' functions, so ensuring that they are operating at an optimum level is important [1]. However, software in a complex IT environment will face operational issues and faults that can be attributed to various reasons such as issues arise from underlying operating systems, network connectivity, permission, and many other causes [2]. The amount of effort to troubleshoot and resolve any IT fault consume the bulk of any IT administrator's work time and it varied from a short period if the fault is easy to fix to long-duration where the fault may require software vendors to develop bug fixes plus the cycle of product acceptance testing. The extent of this will exacerbate when multiple faults are occurring concurrently and there is a limited number of IT administrators available to handle the job. The IT administrators' manual hands-on effort is not well known for scalability for a larger number of IT systems, coverage in the period of support throughout the period on 24×365. In addition to that, they are prone to fatigue, human errors, and slowness. The capability of any IT administrator is also limited to their skills and experience including other social or human conditions too. As the data replicating environment comprise of a multitude of software and hardware technology, any single IT administrator will find it difficult to maintain a certain expert level of expertise across multiple software domains [3].

M. Qiao et al. (Eds.): ADC 2021, LNCS 12610, pp. 125–138, 2021.
https://doi.org/10.1007/978-3-030-69377-0_11

We propose a novel method to conduct fault detection and diagnosis for the data replicating setup using deep reinforcement learning. To our best knowledge, there is no prior research on applying Machine learning for fault diagnostics in the field of data replication for databases. The computational search space to align the combination of information from the Data Replicating Environment (DRE) versus the possible root cause is high and difficult. For any single fault detected, there may be other various forms of causes. Techniques like decision trees, Bayesian networks, or statistical methods are commonly used in the industry to meet this need. The diagnostic models are customized to the solution that they are intended for, and they are not readily modifiable to fit in new changes nor interoperable with other software domains. We want our new solution to be general-purpose enough to adapt to any software domain and with greater flexibility and ability to expand its knowledgebase of diagnosing faults with ease.

2 Literature Review and Background

A common implementation of fault diagnosis with machine learning involves the acquisition of signals or data, the perform feature extraction, or information infusion before using machine learning models to perform pattern recognition to derive fault diagnosis [4]. Another approach is the use of a convolutional neural network for better accuracy in fault diagnosis for industrial's permanent magnet synchronous motors in conjunction with the analysis of generalized frequency response function for [5].

Deep neural networks are used prevalently in both academia and industry for the system's fault diagnosis across different domains. But there are some shortcomings that researchers have identified in the use of NN for this approach and they are; 1) the complexity of mapping relationship between data and outcome of faults for complex, non-linear systems, 2) the availability of labelled data and high quality extracted features for the model training, 3) the configuration of the neural networks models need frequent retraining, reconfiguration and optimization to keep them relevant [6].

The authors use Reinforcement Learning (RL) to diagnose the ball bearing faults in a motor, by acquiring the signals from monitoring meters set against various components within the motor and use Neural Network (NN) to predict the outcome [7]. The outcome is compared against a set of labelled data that has been predetermined to gauge the quality of their predictions. A q-table is built between the detected motor's conditions and the outcome of the predictions during training, which is used by the agent during its knowledge exploitation phase. Another research [8] followed a similar approach with a signal band filter in its fault determination criteria on signals data gained from rotary machinery motors. These are initial research conducted in using deep reinforcement learning for motor machine fault diagnosis and we observed that the application has several features; it is an enclosed system with minimum or no direct interaction with external entities. All the data input are signals from monitoring meters set around the motor, and the data acquired are explicit homogeneous. There is a set of prelabelled data that the RL's NN can refer to. However, in a complex IT system environment, it is an open system with many interfacing components and the data received for fault diagnosis are heterogeneous in data type, categories, frequency of occurrence and status usage. It requires an additional set of data manipulation and a new design before RL can be used

for this intent. The authors in [9] used RL for fault diagnosis on a virtualized network in the cloud service, acquiring inputs from a list of subsystems within the Virtual Machine (VM) networks and map to state-attribute vectors. It used an external entity to validate the attributes in response to the state. The attributes are regarded as the diagnosed faults and are manually mapped to external actions to remediate them. This gives rise to the inspiration of a similar approach for this paper, but the main difference is in the setup of the diagnosis validation module where we use a script-based approach as compared to the manual mode in this research [9].

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 [10]. 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, components, relationship, and operations 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 [11]. So, having a rigid and well-defined FDD agent will not have the adaptiveness nor flexibility to meet the range of different system setup. It will require numerous customization which is time-consuming and laborious.

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 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 another word, we relate the new FDD model as a new mechanic apprentice that need 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.

3 Adaptive Fault Diagnosis (FD) Module Design

Deep Reinforcement Learning (DRL) [12] is used in performing the fault diagnosis against the Data Replication Environment (DRE) with the objective of an intelligent system that can emulate the learning and work process that is similar to a junior IT administrator in managing a system related problem. Figure 1 shows an overview of the FDD model. As an individual that is receiving on-the-job training, it will learn initially to analyze every aspect of the DRE and gather all the related information. It also starts with a little or no prior knowledge on the relationship between information and diagnostic

conclusions, it interacts with its supervisor or someone with expert domain knowledge, receiving guidance and information to derive the diagnosis. This iteration carries on and the FDD will build up its knowledge slowly, and when it has sufficient knowhows on the environment, it can derive an accurate deduction of the fault just by looking at the environment's symptoms without going through all the unnecessary and unrelated system checks [12]. The justification for using DRL for supporting the fault diagnosis is mentioned in problem definition, while these routines can be easily performed by hardcoded, well-defined models, the scope for these groups of software fault diagnostic system cannot be fixed. They must be flexible to cater to different fault scenarios and grow to cover other forms of the software system that can be rolled under the FD's management. A series of data collection points are established to ingest and process the information from the DRE before sending it to the FDR's DRL unit [13]. 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 reach a point where it can either deduce the root cause or listed it as an unknown error, which triggers another routine to notify the IT administrator for assistance and input. The FDD model can be split into 3 modules: Information Acquisition (IA), Diagnostic Reinforcement Learning (DRL), and System Diagnostic (SD) as shown in Fig. 2 [12].

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 Database (DB) [14], Shareplex [15], and Operating System (OS) [16], produce information about their states constantly and proactively into 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 experiences from the users or another depending system while interacting with the DRE. It is a brief description of encountered service's anomalies for the FDD can refer to for investigation. An event such as login failure, slow replication, data not found as some examples. Once the inputs have been processed with all the mandatory details extracted out, they are used to represent the system environment's state to the DRL and as input into both the SD modules.

3.2 Diagnostic Reinforcement Learning (DRL) for FD Module

The FDD module uses the Actor-Critic Deep Reinforcement Learning algorithm (DRL) [17, 18]. Referring to Fig. 2, in this setup, the DRL's Actor is performing as a function approximator that tries to predict the best action for a given state, and in this case, the best diagnosis. The DRL's Critic also takes in the DRE' state-input plus the Actor's action, join 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 Fig. 3 [18].

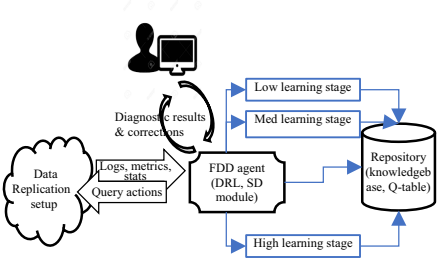

Fig. 1. Adaptive faults diagnosis overview

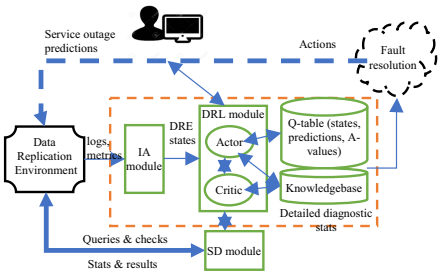

Fig. 2. Faults diagnosis agent's architecture and workflow

1) Early learning phase; where the agent has little a-prior knowledge about the DRE at the start, 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 through all 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 summarised the acquired information to formulate a service diagnosis matrix as shown in Eq. (1) [18].

$$DRE's\ diagnosed\ service\ matrix = \begin{bmatrix} sdbs_1 & srpl_1 & snet_1 & sose_1 \\ sdbs_2 & srpl_2 & snet_2 & sose_2 \\ \cdots & \cdots & \cdots & \cdots \\ sdbs_n & srpl_n & snet_n & sose_n \end{bmatrix} \quad (1)$$

2) Middle learning phase; after it has gathered enough knowledge about the DRE, 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 knowledgebase gathered from the earlier learning phase as its minibatch. There is a high chance that the NN will predict incorrectly, so in such an event, the RL's Critic will run the validation process through the SD module which corrects and assign the Q-value to the state-action pairs and then store in the Q-table, as well as updating the knowledgebase. The gradual built-up of the knowledgebase will improve DRL's NN prediction accuracy [18].
3) High learning phase. By this stage, the DRL agent would have learned all the states - symptoms that may associate 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 in identifying the faults' matrix without performing excessive checks or validation through the SD module. But during this period, the agent also performs a probability-based decision between exploitation versus exploration; Exploitation where the DRL decided to refer to its knowledgebase to respond the best action for the DRE's state, Exploration where DRL's decide to run all the detailed checks through SD module and get the diagnosis instead of relying on the NN's prediction. At the start of the learning cycle, the probability for exploration will be high at the low learning phase but this diminishes over time when it reaches the high learning phase, where

the exploration rate has decayed over iterations and the preference is shifting more toward knowledge exploitation [18].

3.3 System Diagnostic (SD) Module

The DRE comprise of different software and technology working together to provide the service [19]. Each software and technology have a unique list of configuration, checks, operations, and attributes. So, 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, and others. Referring to Fig. 2, the DRL agent gives instructions to the SD module to do the checks against DRE's environment, ranging from comprehensive top-down checks or a few selective ones. This is on par with the analogy of junior workers that need to perform every check just to make sure, or a senior worker who can deduce roughly which exact area that has to be verified before deducing the root cause. 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 on 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 initialize 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 as shown in Table 1 and sent back to the RL agent. It is a matrix that presents the multiple sub-area 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 amount of details will be too overwhelming for its administrators to go through. The following is a tabulation of the output 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 permission, 3) responsiveness to administrative interaction, 4) communication functionality, 5) data transfer and input-output capability plus 6) software's function and operation status. The vast specific software details can be made available and they will be connected to the future Fault Resolution agent. The four diagnosed service groups are as followed and in the matrix in Eq. (1); Database service, $sdbs_n$, Shareplex replication service, $srpl_n$, Network and communication services, $snet_n$,, supporting the OS environment, $sose_n$.

4 Data Replication Environment (DRE)'s State Representation

For the DRE [18], it is hard to define its state due to its complex multi-tier software setup and the characteristics of the IT applications under its service. A direct method

is needed to identify a state in a database without time properties. Each software's operation information is mined continuously for anomalies and errors. We propose the use of a matrix to capture a list of the events and processes' status of the DRE's software across multiple sources and target instances, *n*. Therefore, the two sections in the state's matrix contain both information from both their logs and process status. 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 representations are 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 to represent the DRE's state in Eq. (2).

$$DRE's\ state = \begin{bmatrix} oralog_1 \\ oralog_2 \\ splxlog_1 \\ splxlog_2 \\ nwlog_1 \\ nwlog_2 \\ oslog_1 \\ oslog_2 \end{bmatrix} \begin{bmatrix} orastat_1 \\ orastat_2 \\ splxstat_1 \\ splxstat_2 \\ nwstat_1 \\ nwstat_2 \\ osstat_1 \\ osstat_2 \end{bmatrix} \begin{bmatrix} orasvc_1 \\ orasvc_2 \\ splxsvc_1 \\ splxsvc_2 \\ nwsvc_1 \\ nwsvc_2 \\ ossvc_1 \\ ossvc_2 \end{bmatrix} \tag{2}$$

Table 1 describes the specific software validation and checks that need to be performed to acquire the DRE' 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 all the DRE's software processes checks, check_all_processes.sh will handle

Table 1. Subsets of memory and logs checks

DRE's software	Service Group checks	Software attribute	Detail checks/description
OracleDB	Process check	Process' stats	DBs' memory process in the OS
	Process check	Operation's stats	DB's mode of operation
	Service check	Tablespace's stats	Tablespaces have enough space on DBs
	……		
Shareplex	Service check	Parameter setting	parameters are valid in Shareplex instances
	Process check Service check	Process's Status and operation	Shareplex's memory process in the OS
	……		
Network	Process check	Listener Process stats	Oracle's listener process on both nodes
	Process check, Service check	Listener.ora availability, and stats	Listeners' availability for service on both nodes
	Process check, Service check	Listener's stats – error or available	Listeners' operations are valid and not in error
	……		
OS	Service check	Disk space	Free space availability on OS for both nodes
	Service check	Primary conf files	Validate /etc/passwd, /etc/shadow, /etc/hosts, /etc/group files
	Process check	Network card operation	Network card status and availability
	….		

this. The last group check is done by check_all_services.sh which validates their specific services.

5 DRE's Action of Diagnostic Prediction

The DRE's state information from the previous section are the summarised raw input which 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, much like an experienced mechanic that can pinpoint the fault with a car based on the symptoms described by the owner [18]. 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, in respect to 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, as a mean 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.

6 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 situation [18]. 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 on 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 on catastrophic failure, which is irrecoverable and can only be solved by an entire system rebuild.

6.1 The Experimental Set-Up

The experiments are run on two Virtual Machines running on Linux OS and both have Oracle DB and Shareplex installed on them. Each VM has 4 GB of RAM with 100 GB of hard disk storage. The version of the Oracle software is 12 Enterprise edition and the 9.1 for the Shareplex. The network protocol that both VMs use is TCPIP. For the DBs, the simulated faults will impact Oracle's primary memory process such as SMON and PMON. Any failure of either one of these processes will cause the DB service to stop. The script will do a root level kill to simulate the DB outage and a start-up command is required via DB's admin level is required in restoring it. 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 privilege to function, so some scripts simulate the absence and presence of these privileges from the accounts. Likewise, for the schema objects that the user account owns and access; the Shareplex created a list of DB objects under the user account during installation and it continues to use them for its operation. Should there be any changes to their accessibility to the user account of the validity of the object, it will cause Shareplex to malfunction. Scripts are written to simulate this error too. Another factor to note is the availability of free space within DB for the Shareplex to operate on. If there is insufficient space, then Shareplex will not be able to write data into the DB and that results in the suspension of its service. Some scripts constrict and free up the storage space. For the Shareplex's fault simulation, it follows a similar pattern as 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.

As 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 faults 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.

For the OS, the emphasis here 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 3) status of the network card. For each of the software's core functionalities, two of its attributes will be assessed and a metric is associated with it which measures its service's normality. A value of 0 indicates a normal state whereas >0 indicates an abnormality. Table 2 lists all the software components and the respective commands that can simulate and restore their faults.

However, the test does not include malicious or terminal faults to the software if they are either irreversible or require a substantial amount of effort to restore them. 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.

6.2 True Negative Test Results

Besides the data are obtained from the faults inducing scripts in the previous section, another group of scripts has been created to induce software faults that have no impact on their DRE's software functionalities and services. This is to form the 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 and differentiate the environment's state data that can cause service disruption or not.

For the script to induce this group of faults, research has been made across the DRE's software to identify those faults that have a high chance of occurring but they don't have a direct consequential effect that can either disrupt the entire software's stability or create outage on the DRE's functionalities. This is verified by the SD module which confirms the presence of any 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.

6.3 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 are represented by a vector with each element representing the service. And within each element is a scalar value from 0 to 1, values that are >0 indicate the faults' severity whereas 0 is when every component is

Table 2. 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

operating normally. This forms 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 of whether they are improving or degrading. 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.

6.4 Test Results

This section described the results obtained from the FD module after it completed the training and subjected to the evaluation test processes. By this stage, the FD module has been trained thoroughly and it is regarded to be equivalent to achieving the expert level of fault diagnostic capability. The minimum expectation of its prediction accuracy internally is expected to reach 85% accuracy and more. A sample of the DRE's states, including both the predicted and actual service outage results, are shown in Table 3; 1) The DRE state data are derived from the information gathered against the DRE's software components from their logs, internal system statistics, and monitoring after simulating fault are induced. 2) The FD module predicted the service outage results after it received the DRE input based on its learned NN. 3) The SD module produced the real 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 3. 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] [1,0,1,0,0,0,0,0]	[[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
[46968001,0,0,0,0,0,0,64351381] [0,1,0,0,0,0,0,0]	[[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]]	[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.3
[46968001,0,0,0,0,0,0,64351381] [0,1,0,0,0,0,0,0]	[[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]]	[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.2
[46968001,0,0,0,0,0,0,0] [0,0,1,0,0,0,0,0] [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]]	[[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]]	TP	0.1
[46968001,0,0,0,0,0,0,64351381] [0,1,0,0,0,0,0,0] [0,1,0,0,0,0,0,0]	[[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]]	[[0,0,0,1,0,2],[2,2,2,1,0,0],[1,0,0,0,0,0],[0,0,0,0,0,0],[0,0,0,0,0,0]]	TP	0.6
----	----	----	---	----

6.5 Service Outage Classification Results

The test is conducted with a list fault inducing scripts with 80 entries. 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 4 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 outage to the information received from the DRE's environment. While it has high capability in recognizing most of the induced faults that can affect the DRE's software functionalities, it fair less when it comes to the detection of those in the other groups. Based on the result, the FD's sensitivity is 0.355, specificity is 0.645, precision is 0.871. The SD module has shown to be accurate

Table 4. 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)

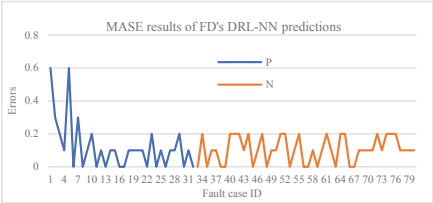

Fig. 3. MASE score of True and positive predicted results

enough that its prediction can produce the correct category of service outage for the given environment state's data input. It has the competency to differentiate if the inputs are related to DRE's service functionalities.

6.6 Service Outage's Prediction Accuracy

For this test, The SD module forms the baseline in 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 Mean Square Error 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 Fig. 3. Based on the results, the accuracy is below the mark of 0.3 and below except for one entry that scored 0.6. This can be since this is a DRL based Fault diagnosis that learns adaptively with the environment. While it is experienced to recognize the fault scenario that it was had trained for. However, for new and unfamiliar ones, it has some deviations. One possible 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. However, the list of potential faults that can affect DRE is controlled and limited, the next 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.

7 Conclusion

The FD module has been proven that it be able to 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 predict the outage gradually as it interacts with it and learns with the help of the SD module that corrects its prediction. It is adaptive to the DRE and available to configure to support heterogeneous platforms and software without restriction to the data replication architecture.

References

1. Milani, B.A., Navimipour, N.J.: A systematic literature review of the data replication techniques in the cloud environments. Big Data Res. **10**, 1–7 (2017)

2. Tabet, K., et al.: Data replication in cloud systems: a survey. Int. J. Inf. Syst. Soc. Chang. (IJISSC) **8**(3), 17–33 (2017)
3. Iacob, N.: Data replication in distributed environments. Ann.-Econ. Ser. **4**, 193–202 (2010)
4. Chen, X., Feng, Z.J.: Time-frequency space vector modulus analysis of motor current for planetary gearbox fault diagnosis under variable speed conditions. Mech. Syst. Signal Process. **121**, 636–654 (2019)
5. Chen, L., Zhang, Z., Cao, J.: A novel method of combining generalized frequency response function and convolutional neural network for complex system fault diagnosis. PLoS ONE **15**(2), e0228324 (2020)
6. Jia, F., et al.: A neural network constructed by deep learning technique and its application to intelligent fault diagnosis of machines. Neurocomputing **272**, 619–628 (2018)
7. Ding, Y., et al.: Intelligent fault diagnosis for rotating machinery using deep Q-network based health state classification: a deep reinforcement learning approach. Adv. Eng. Inform. **42**, 100977 (2019)
8. Dai, W., et al.: Fault diagnosis of rotating machinery based on deep reinforcement learning and reciprocal of smoothness index. IEEE Sens. J. **20**, 8307–8315 (2020)
9. 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). IEEE (2019)
10. Venkatasubramanian, V., et al.: A review of process fault detection and diagnosis: part I: quantitative model-based methods. Comput. Chem. Eng. **27**(3), 293–311 (2003)
11. Venkatasubramanian, V., Chan, K.: A neural network methodology for process fault diagnosis. AIChE J. **35**(12), 1993–2002 (1989)
12. Zhang, D., Lin, Z., Gao, Z.: A novel fault detection with minimizing the noise-signal ratio using reinforcement learning. Sensors **18**(9), 3087 (2018)
13. Zhang, D., Gao, Z.: Reinforcement learning–based fault-tolerant control with application to flux cored wire system. Meas. Control **51**(7–8), 349–359 (2018)
14. Kyte, T., Kuhn, D.: Expert Oracle Database Architecture. Apress, New York (2014)
15. Quest Software: Shareplex for Oracle v9.1.4 (2018)
16. Shotts, W.: The Linux Command Line: A Complete Introduction. No Starch Press, San Francisco (2019)
17. Fujimoto, S., Van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477 (2018)
18. Wee, C.K., Nayak, R.: Adaptive database's performance tuning based on reinforcement learning. In: Ohara, K., Bai, Q. (eds.) Knowledge Management and Acquisition for Intelligent Systems. PKAW 2019. Lecture Notes in Computer Science, vol. 11669. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30639-7_9
19. Wee, C.K., Nayak, R.: Data replication optimization using simulated annealing. In: Le, T., et al. (eds.) Data Mining. AusDM 2019. Communications in Computer and Information Science, vol. 1127. Springer, Singapore (2019). https://doi.org/10.1007/978-981-15-1699-3_18