iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 8 **Issue:** VII **Month of publication:** July 2020

DOI: https://doi.org/10.22214/ijraset.2020.29240

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue VII July 2020- Available at www.ijraset.com

Preemptive Fault Tolerance in DDS based Distributed System using Application Migration

Rakesh[1], Manjeet Gupta[2], Aditya Baunthiyal[3], Amit Kr. Jain[4]
[1, 2, 3, 4]*Central Research Laboratory, Bharat Electronics Limited, Ghaziabad (India)*

Abstract: ***Software availability is the main concern of distributed systems regardless of design and framework. Faults are inevitable to any system and system should continue to function even in presence of these faults. Fault tolerance (FT) techniques should make the system fault tolerant by detecting and handling these faults. This paper introduces the technique of proactively anticipating the possible fault and then avoiding that fault by application migration. The possible fault is predicted through sudden or unusual utilization of CPU, RAM, fan speed or CPU temperature. Whenever CPU, RAM, fan speed or temperature breaches the permissible limits of any machine or an alert is generated based on historical data analysis; system takes the necessary steps to redistribute the application load based on software or hardware analytics. The proposed solution has proved to be more efficient than recovery from failure via traditional reactive FT mechanisms such as checkpoint/restart. The use of data distribution service (DDS) as communication middleware has helped in convenient state transfer and state maintenance of migrating application.***

Key Terms**: Fault Tolerance, Migration, Distributed systems, Load balancing, DDS**

I. INTRODUCTION

Software fault tolerance means that system should able to with-stand failures and operation shall continue without compromise. There are many reasons of system failure in a distributed system due to complexity and heterogeneity. Despite these failures system should be able to recover from fault and perform the intended function. In legacy distributed system, fault tolerance solutions can be broadly classified as reactive and proactive. [1]

Reactive fault tolerance solution is used to reduce the impact of failures on a system when the failures have already occurred. Solution based on this policy are retry, replication, check-pointing and message logging. Retry is the most common failure recovery mechanism and it assumes that no matter whatever the cause of the failures it will not encountered in the later attempt [2]. In replication multiple instance of an application are created and then they are deployed on different hosts so that all replicated application instance do not hang (due to a host crash, host network disconnection, resource crunch etc.) and the execution of the application would proceed successfully [2]. In check pointing, the basic method is that the system regularly saves its current state on a reliable and stable storage. So that on a crash, when the system is restarted, its start from the last preserved state instead of from the beginning [3].As check-pointing is an expensive operation, many new method have been brought to minimize the number of checkpoints. Message logging is one of those methods. The basic criteria of this method is that, if the transmission of message scan be replayed in a well-defined order, we can always achieve a consistent state of the system, without making restore from the coherent state saved in stable storage. While, a check-pointed state is consider as a starting point, and all messages that have been logged after that, are simply retransmitted and handled accordingly [3].

Proactive fault tolerance predicts the faults in advance and places working components in place of unhealthy components, to prevent failure from faults and errors. Many method follow this policy such as software rejuvenation, self healing, preemptive migration and load balancing. In software rejuvenation on each reboot of the system a planned change on the state is taken into consideration. In self healing the basic criteria is to automate the control of failure of an application instance running on multiple virtual machines. Preemptive Migration is another method which continuously monitors the application critical resource like RAM and CPU usage and if during operation any application required higher resource than same application shall migrated to least loaded hardware in the system before the system dried out with resource on current hardware. In another case if hardware temperature is getting increase and it is about to fail than all application shall uniformly migrated to least loaded hardware in the system. The faulty hardware gets reported to operator for maintenance. Figure1 shows the Fault tolerance taxonomy.

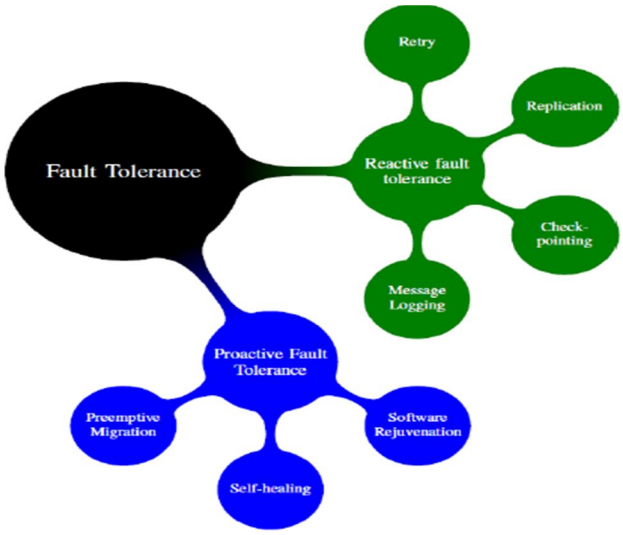


Figure 1. Fault tolerance taxonomy

Many attempts have been made to improve fault tolerance of distributed systems using aforementioned techniques. Some of the implementations of above techniques are discussed here. Los Alamos proposes solution based on message passing interface (LA-MPI) [8], which is used for in tera scale cluster. Inspire by the need of additional software to ensure end-to-end reliability in a high performance message-passing system. LA-MPI uses the retry method is the fault tolerant technique. In [9], the authors suggested new technique in proactive policy for fault tolerance in MPI applications based on task migration and load balancing capabilities of Charm++ and AMPI, the main idea in this method is, when a fault is near to happen just before the crash system proactively try to migrate execution off that application using processor virtualization.

Automatic application self healing using rescue points (ASSURE) introduce rescue points, a new application self-healing method for detecting, tolerating and recovering from application failure in server applications. In [10], they use the existing quality assurance testing method to generate known invalid inputs to an application, in order to identify candidate rescue points.

The proposed solution in this paper takes the pro-active approach where application is actively monitored for any abnormal resource usage and is then migrated to other machine to avoid any malfunctioning. It prevents node failures by preemptively migrating application from a node which are about to fail. Pre fault indicators such as CPU fan speed, CPU temperature and CPU/RAM utilization are used to avoid imminent failure through anticipation and reconfiguration. As application is shifted from one hardware to another hardware, result in application failure avoidance leading to extension of application mean time to failure (AMTTF) beyond system mean time to failure (SMTTF). Application failure reducing through preemptive migration is significantly and more efficient than recovery from failure. [4]Proactive Fault tolerance solution using preemptive application migration is based on feedback-loop control mechanism (figure 2) where health of each application running on node is monitored and preemptive action is taken to avoid immediate application failure by migrating high resource consuming application from unhealthy to healthy compute nodes [4].

II. PROPOSED SOLUTION: MONITORING AND MIGRATION

The proposed solution is based on preemptive application migration by continuously monitoring of system resources viz. CPU, R

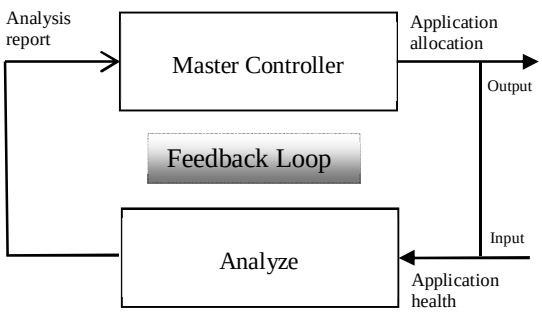


Figure 2. Feedback loop control for preemptive migration

AM, fan speed and CPU temperature of machine with the help of two distributed applications known as agent and controller. Agent is the local level software responsible for monitoring and performing local level functions. Agent monitors only a subset of applications which are in its scope and the possible candidate of migration is considered only amongst them. Controller is the system level software responsible for proactive application migration.

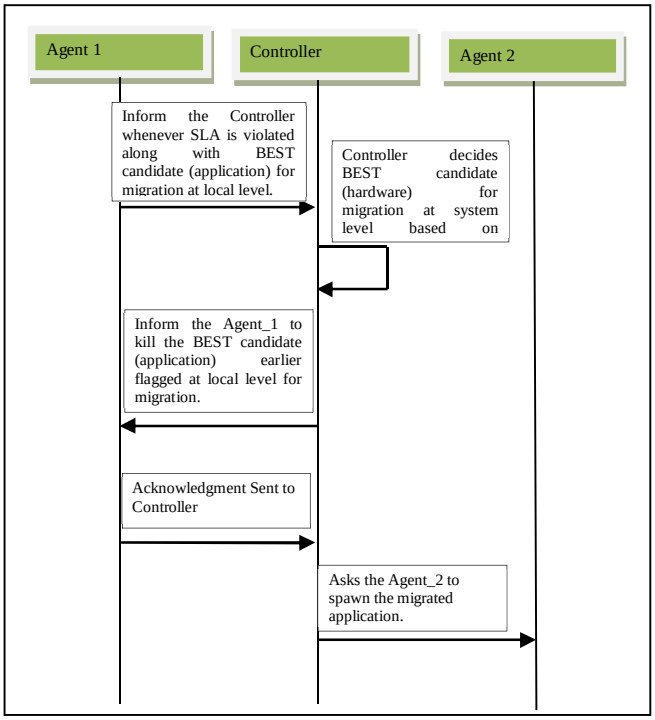


Figure 3. Interaction during SLA breach

The permissible limits of resource utilization hence forth referred as service level agreement (SLA) are pre-decided for each machine and configured through a XML based configuration file. If either of the resources breaches the service level agreement, then local agent informs the master controller about this violation. Master controller in turn decides the migration strategy to the least loaded node. The interaction between local agent and controller can be visualized through the sequence diagram (figure 3).
SLA rules are configured for a node on startup as per figure 8, violating which triggers the application migration. Local agent keeps track of CPU, RAM, fan speed and CPU temperature avoiding occasional spikes in resource usages.
Whenever the SLA breach is reported by the local agent, high resource (CPU/RAM) intensive application is migrated on minimum loaded node by the controller as described in figure 5.This type of migration also helps to achieve hardware redundancy where application can migrate in case of complete failure of one of the hardware. In this case all the application is migrated to other machines regardless of SLA violations. In this approach we are using data distribution service (DDS) [5][6][7] middleware. DDS uses data publish/subscribe distribution paradigm and the data-centric approach. The underlying data distribution model uses a Global Data Space (GDS) to identify data circulating in the system. DDS publishers (P in Fig.4) send their data into this GDS, and the DDS middleware (not shown in figure) propagates the information to all interested DDS Subscribers (S) that make data locally available.

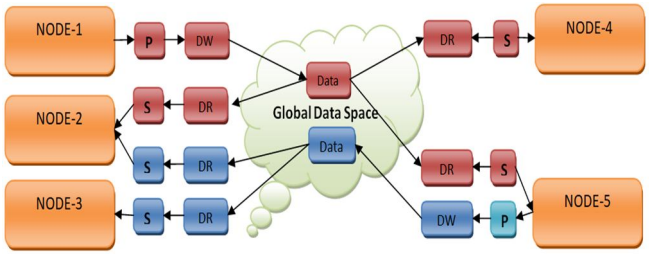


Figure 4. DDS Global Data Space

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue VII July 2020- Available at www.ijraset.com

In this approach application are using DDS transient quality of service (QoS) to checkpoint or store their current state on DDS cloud. DDS cloud helps them to store and retrieve their state on newly assigned hardware without any data loss. DDS provide this in-built feature of data storage in real time scenario. DDS data storage service runs on all nodes and data is always in synchronization. Whenever any application joins as a late joiner in system, application first retrieve its last stored state from DDS data storage service and after that start working on live data.

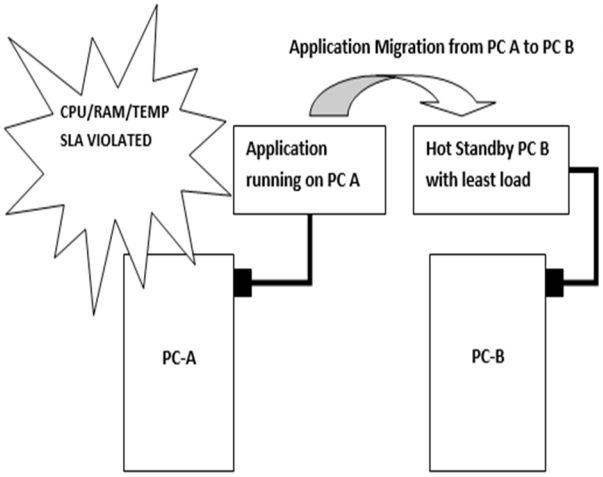


Figure 5. Pre-emptive application migration

III. ARCHITECTURE

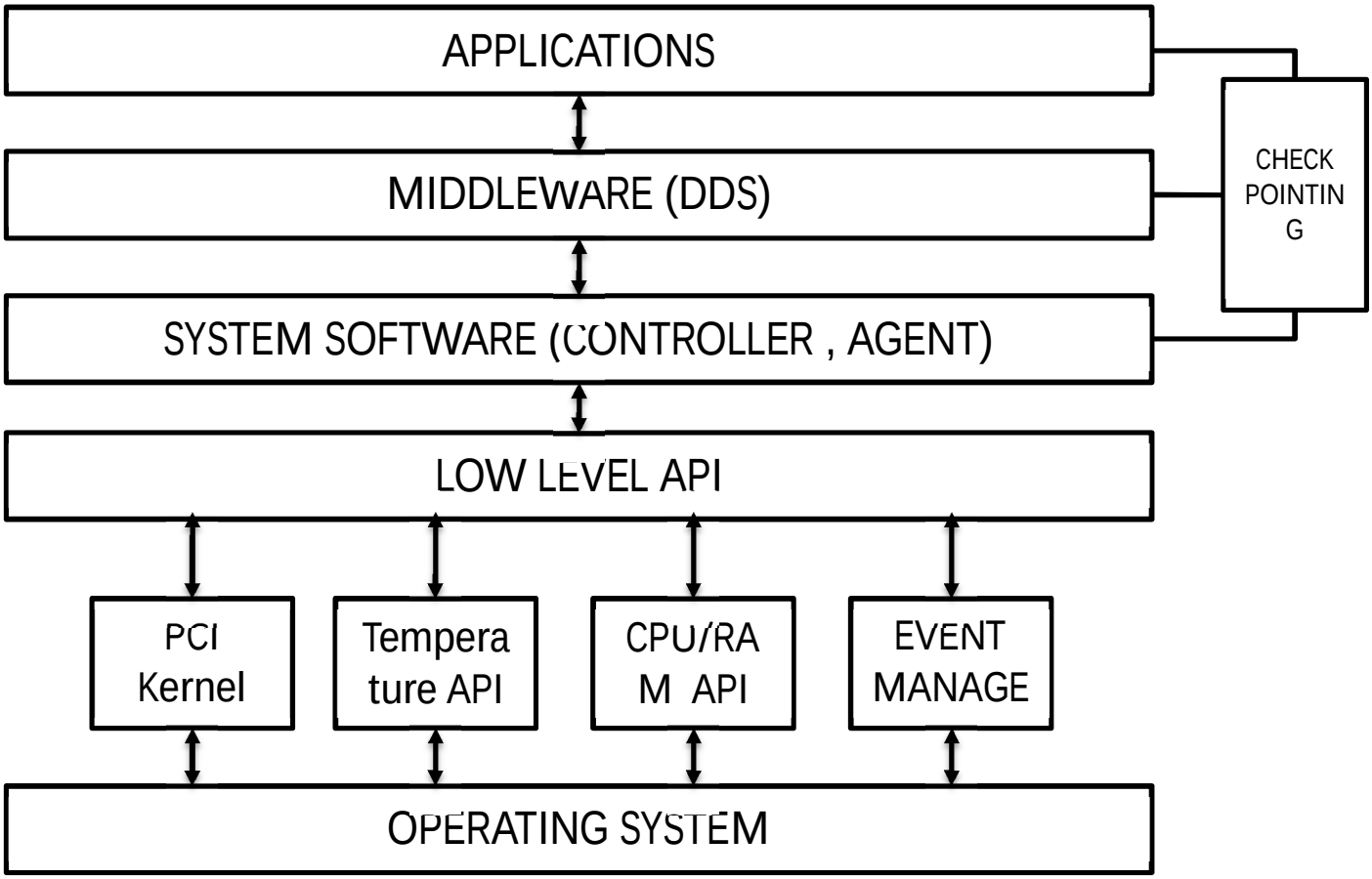


Figure 6. Software architecture

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue VII July 2020- Available at www.ijraset.com

In the proposed architecture as described in figure 6, application is using DDS middleware for communication and check pointing. System software (controller and agent) also uses DDS as communication middleware for communication and check pointing. Low levels API are used agent to get RAM, CPU, CPU temperature and fan speed. Event manager provides integrated configuration, event, alarming, and availability management capabilities. The event manager is composed of Linux kernel and application components. Kernel services supports full dynamic insertion and deletion of hardware devices into the PCI configuration. Agent receives system alert and hardware details from event manager and kernel service.

IV. EXPERIMENT AND RESULT

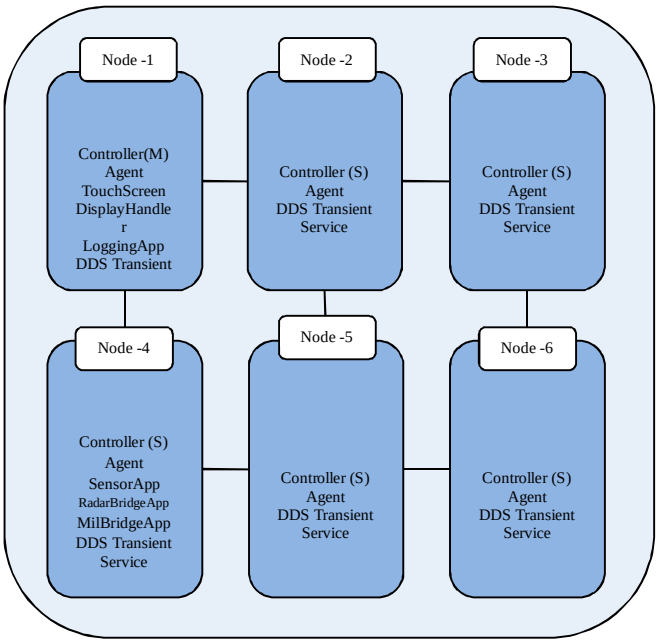


Figure 7. Experimental setup

As per figure 7, a system is a group of three or more nodes configured as identical servers. Each node typically runs on an independent physical machine and has its own controller and agent. The controller and agent work together to get nodes resource utilization.

At any given time, one controller in the system is designated as the master controller. The master controller is responsible for coordinating resource updates across the entire system. The master controller also coordinates the addition or removal of any node to or from the system due to configuration changes, failure, or administrator actions. The master controller is chosen each time the system is started and can change overtime. Normally the first controller to start will become the master and on the failure of master controller, another controller to be chosen as master thorough a consensus algorithm.

All controllers except the master are slave controllers. Slave controllers have direct connection to all agents and they always remain in synchronization with master controller. Agent monitors local resources of node as per configuration like CPU, RAM, fan speed and CPU temperature. Updated resource consumptions are published on DDS cloud by agent. The master controller serializes these multiple updates, performs constraint checks and finally creates a list of nodes in order of resource consumption. The master as well as slave controller update the list and publish it on DDS cloud for check pointing. The agent monitor raises the alarm on SLA violation and updates the same to master controller, controller receives the SLA violation alarm and based on type of SLA takes the decision. In experiment, CPU usage of nodes is considered as base for application migration and SLA are configured as described in figure 8.

*SLAConstraintsConfiguration.xml

```
<?xml version='1.0' encoding='UTF-8' standalone='no'?>
<root>
  <SLAConfiguration>
    <PartID>0</PartID>
    <ConstraintSequences>
      <Constraint>
        <Metric>CPU</Metric>
        <UpperBound>90</UpperBound>
        <LowerBound>0</LowerBound>
        <ActionOnRuleVoilation>ALERT</ActionOnRuleVoilation:
      </Constraint>
      <Constraint>
        <Metric>RAM</Metric>
        <UpperBound>80</UpperBound>
        <LowerBound>0</LowerBound>
        <ActionOnRuleVoilation>ALERT</ActionOnRuleVoilation:
      </Constraint>
      <Constraint>
        <Metric>TEMPERATURE</Metric>
        <UpperBound>60</UpperBound>
        <LowerBound>0</LowerBound>
        <ActionOnRuleVoilation>ALERT</ActionOnRuleVoilation:
      </Constraint>
      <Constraint>
        <Metric>HDD</Metric>
        <UpperBound>90</UpperBound>
        <LowerBound>0</LowerBound>
        <ActionOnRuleVoilation>ALERT</ActionOnRuleVoilation:
      </Constraint>
    </ConstraintSequences>
```

Figure 8. Node SLA configuration

As shown in figure 9 SensorApp, RadarBridgeApp and MilBridgeApp applications are deployed and started by agent on node-4. The resource consumed y each application on node-4 is shown in figure 10.

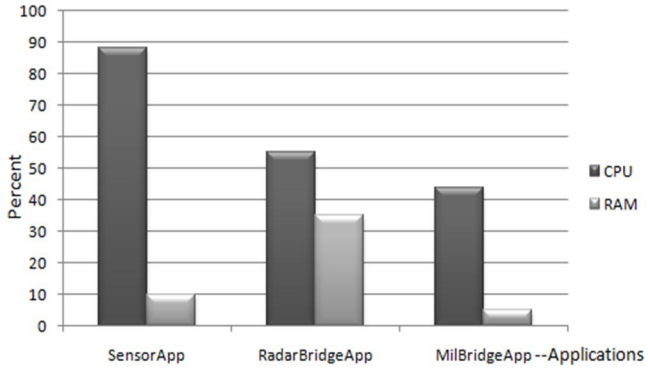


Figure 9. Node-4 Applications statistics

TouchScreen, DisplayHandler and LoggingApp applications are deployed on node-1 and resource consumption is as per figure 10. In system, SLA for CPU violation across the nodes is set as 90 percent. No applications are deployed on node-2, node-3, node-5 and node-6. But agent shall consider these nodes for migration purpose and their CPU, RAM, fan speed and temperature related stats are reported by agent to controller. The figure 11 represent the graph of CPU and RAM collected by master controller before SLA violation.

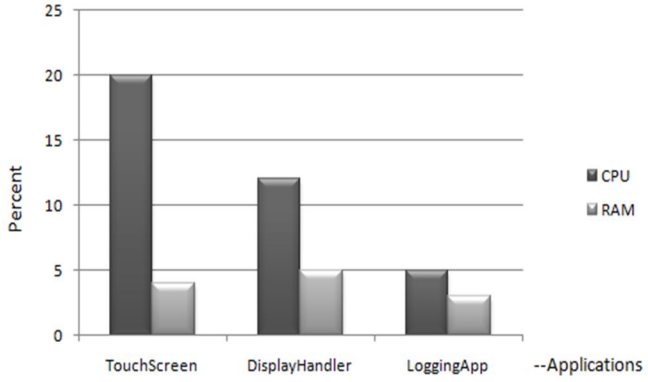


Figure 10. Node-1applications statistics

Master controller collect the data from each agent every 1 sec and averages the data of last five cycle and update internal map to avoid jitter in RAM and CPU usage. Master controller prepares a list of nodes as per load (TOTAL_CPU_USAGE+ TOTAL_RAM_USAGE).

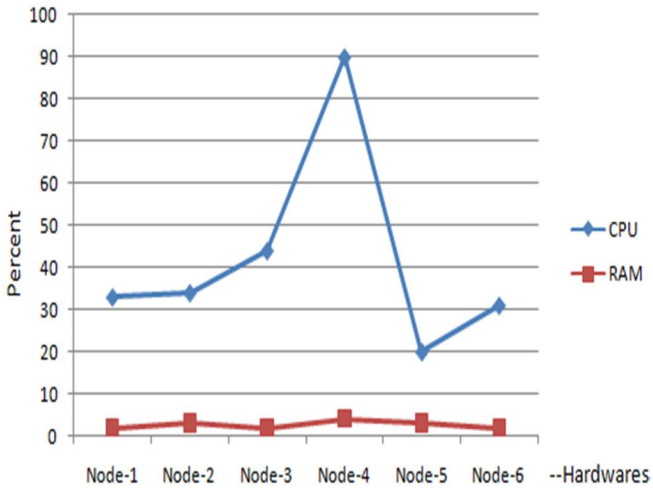


Figure 11. Nodes statistics before migration

Agent running on node-4 raises the alarm for SLA violation of CPU as soon as its overall nodes CPU usage touches 90 percent up to maximum 5 seconds. Controller receives this alarm and pro-actively migrate the highest CPU usage application to least loaded node which is node-4 as per figure 11.

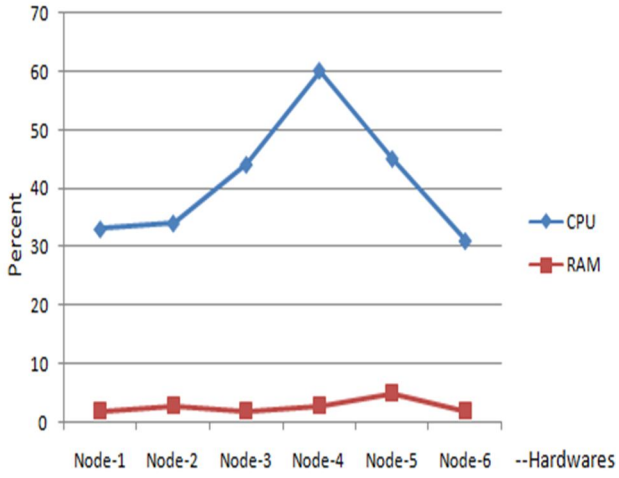


Figure 12. Nodes statistics after migration

After SensorApp application migration, overall CPU usage of node-4 is now 60 percent which is below SLA violation. The overall CPU usage of node-5 after migration increase from 20 to 45 percent which is under SLA violation limit (figure 13). In this approach when application is migrated from one computing hardware to another computing hardware, previous running application will not be stopped until migrated application gets functional on newly assign hardware to ensure no data loss during migration.

V. CONCLUSION

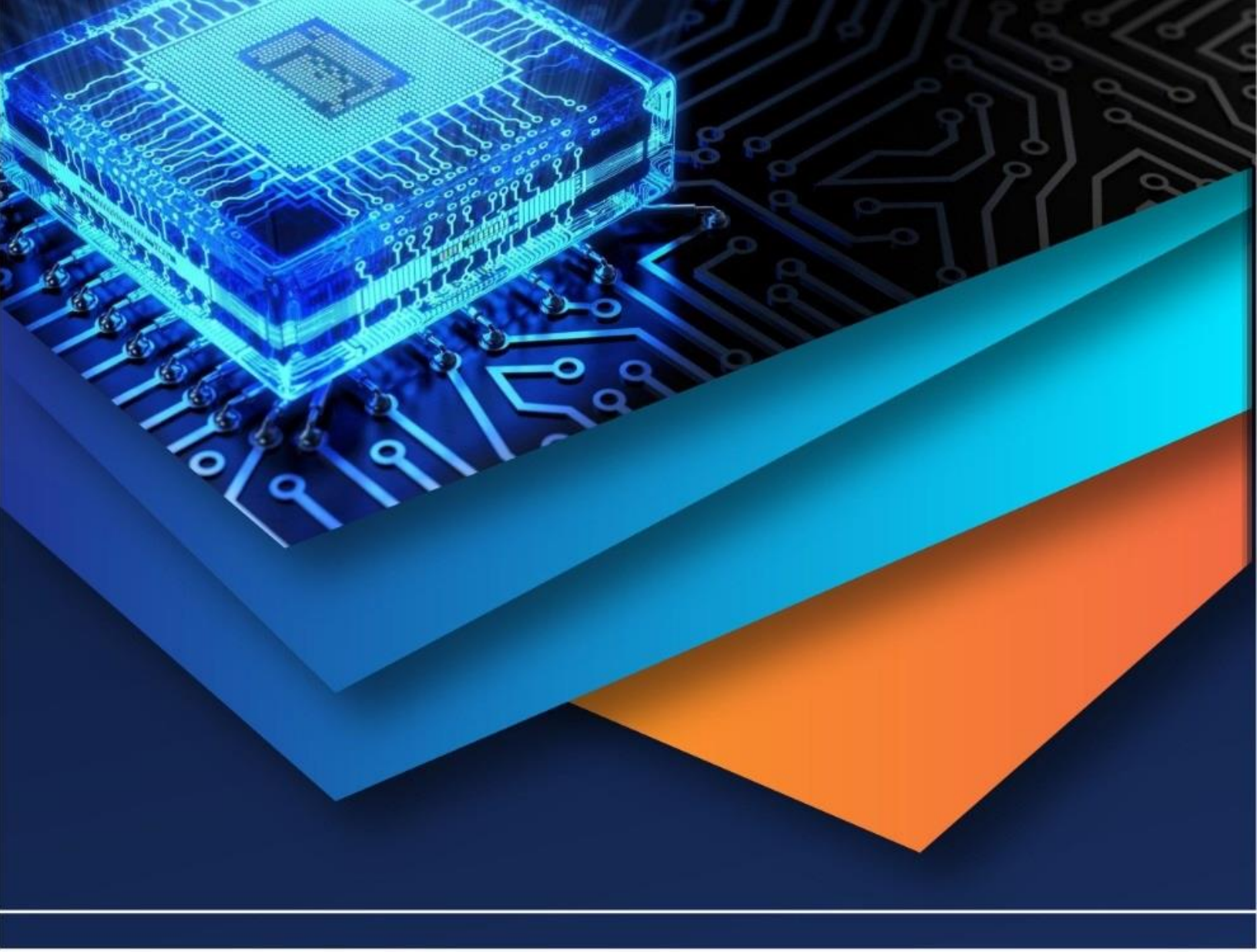
The proposed solution employs pro-active approach for fault detection and uses that knowledge for application migration. The application which might create failure in future is migrated to least loaded node to distribute the load uniformly across computing resources. This approach has been successfully used in distributed system. The experimental results show that the method can quickly and efficiently re-locate software on least loaded hardware to reduce the workload. It can also efficiently migrate applications on complete hardware failure/breakdown.

Controller and agent are scalable with any number of nodes because it is based on generic data centric framework. This solution uses XML configuration which allows us to configure hardware and software resources. It also allows setting different SLA rules for CPU, RAM, and temperature and fanning speed. Another major benefit of this approach is that it provides fault tolerance even in case of complete failure of hardware. On complete hardware breakdown, it migrates all applications without any SLA breach to a functional hardware.

In future, machine learning based data analytics shall be implemented on historical data for better fault prediction and application migration. It is also envisaged to monitor application critical component like thread, IPC, exit and entry points and application health in application migration.

REFERENCES

[1] A. Ledmi, H. Bendjenna, and H.S. Mounine, Fault Tolerance in Distributed Systems: A Survey in2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS).
[2] S. Haider, et al. Fault tolerance in distributed paradigms, In 2011 International Conference on Computer Communication and Management, Proc. of CSIT.
[3] A.S. Tanenbaum and M. V. Steen, Distributed systems:principles and paradigms. 2007: Prentice-Hall.
[4] C.Engelmann,G. R Vallee, T. Naughton, S. L. Scott, Proactive fault tolerance using preemptive migration, Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA, [Online]. Available: http://www.fastos.org/ras.
[5] Object management group, [Online]. Available: http://www.omg.org/ [Nov 2010].
[6] Object Management Group (OMG), Data Distribution Service for Real-Time Systems, [Online].Available: http://www.omg.org/spec/DDS/1.2.
[7] Real-Time Innovation (RTI) DDS Project Home Page: [Online]. Available: http://www.rti.com/products/data _distribution/.
[8] R.L.Graham, et al., A network-failure-tolerant message passing system for tera scale cluster, In International Journal of Parallel Programming, 2003. 31(4): p.285-303
[9] S.Chakraborty, C.L.Mendes, and L.V. Kale, Proactive fault tolerance in MPI applications via task migration, In International Conference on High-Performance Computing. 2006. Springer.
[10] S. Sidiroglou, et al., Assure: Automatic software self-healing using rescue points. ACM SIGARCH Computer Architecture News, 2009. 37(1): p. 37-48.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY