International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:4, 2008

Memory Leak Detection in Distributed System

Roohi Shabrin S., Devi Prasad B., Prabu D., Pallavi R. S., and Revathi P.

Abstract**—**Due to memory leaks, often-valuable system memory gets wasted and denied for other processes thereby affecting the computational performance. If an application's memory usage exceeds virtual memory size, it can leads to system crash. Current memory leak detection techniques for clusters are reactive and display the memory leak information after the execution of the process (they detect memory leak only after it occur).

This paper presents a Dynamic Memory Monitoring Agent (DMMA) technique. DMMA framework is a dynamic memory leak detection, that detects the memory leak while application is in execution phase, when memory leak in any process in the cluster is identified by DMMA it gives information to the end users to enable them to take corrective actions and also DMMA submit the affected process to healthy node in the system. Thus provides reliable service to the user. DMMA maintains information about memory consumption of executing processes and based on this information and critical states, DMMA can improve reliability and efficaciousness of cluster computing.

Keywords**—**Dynamic Memory Monitoring Agent (DMMA), Cluster Computing, Memory Leak, Fault Tolerant Framework, Dynamic Memory Leak Detection (DMLD).

I. Introduction

DESPITE of great care being taken by developers to develop a fault free application, due to developers' mistakes, applications is bound to error. There are different types of faults that can occur in cluster computing like Application and OS faults, etc Memory leak is one of the major application fault in the cluster computing. Memory leaks are caused when some part of allocated memory is never accessed again. This can degrade program performance and it may lead the program to exhaust systems resource eventually leading to program crash [1]. As per the CERT [2] memory leaks and memory consumption are two common forms of software bugs that severely imperil system availability and security. As per the CERT vulnerability database, 68% of all reported vulnerabilities in the past 2003 were caused by memory leaks or memory corruption. Often High Performance Computing (HPC) systems are subjected to memory leak problem and our objective of DMLD technique to provide efficient resource utilization in the HPC framework. Memory leaks can create bugs in software that are hard to detect. Proper care must be taken in allocation and de-allocation of memory so as to avoid memory leaks. If memory leak can be detected dynamically, it helps for other executing processes preventing from starving for memory and systems crashing.

Roohi Shabrin S., Pallavi R. S. and Revathi P. are with BIT Institute of Technology, Hindupur 515212, India.

Devi Prasad B. and Prabu D. are with Centre for Development of Advanced Computing, SSDG, Knowledge Park, Bangalore 560038, India.

We studied different tools currently available to detect memory leak faults. From our study we found that existing methods gives the memory leak information after the execution of the application. Our team in this paper proposing DMLD technique through DMMA, that is an agent based to deal with memory leak fault dynamically. DMMA designed is lightweight, complete, intelligent dynamic agents that monitor the memory leak fault at runtime and dynamically act to avert and resolve the fault.

This paper is organized as follows. In section 2 we present different types of faults that exist in cluster computing. In section 3 we discussed briefly the existing memory leak detection tools. Section 4 presents the user application with and without DMMA. Section 5 discusses the Implementation of DMMA. Section 6 discusses the experimental evaluation of DMMA; Section 7 presents the conclusions and the future direction.

II. Types of Faults

This section gives different types of faults that may take place in cluster based on fault tolerance studies; cluster faults have been classified into six groups [3]. These six groups are further divided into different sub classes are shown below.

A. Application and OS faults
   1. Memory leak
   2. Resource unavailability

B. Hardware fault likes faults in Memory chips, CPU and Storage Disks

C. Network fault: Node Failure, packet loss, corrupted packets

D.Response fault: Value Fault, Byzantine Error

E. Software fault: Un-handled exception, Unexpected Input,

F. Timeout faults

III. Existing Memory Leak Tools

There are many tools available today to detect memory leaks. Like Valgrind and dmalloc, etc. These tools provide detailed log information about memory leaks after the execution of the application.

A. Valgrind

Valgrind is an open source tool [4] that can be used to detect memory leaks in the applications. When the tool is used with the application executables as a command line argument, it reports errors in the application [5]. The Valgrind output can also be directed to a log file to view memory leak details. Valgrind tool when made to run along with our application. It consumes more memory. This memory consumption is increased when running the application through Valgrind. It monitors allocated memory object and reports leaked memory by mark-sweeping the virtual memory for unredeemed objects. Mark sweeping the entire virtual address space can add significant overhead, especially for server programs that

usually have large address spaces for buffering and caching. Due to this operation program needs to pause to avoid inconsistency, which makes the service unavailable during entire mark sweeping operation [6].

B. Dmalloc

The dmalloc library replaces the heap library calls normally found in our system libraries with its own versions [7]. When we allocate memory with these functions, the dmalloc library keeps track of a number of pieces of debugging information about our pointer. This information can then be verified when the pointer is freed or reallocated and the details can be logged on any errors. The administration of the library is reasonably complex. If any of the heap maintenance information is corrupted, the program will either crash or give unpredictable results. It even performs fence post checking but library cannot notice when the program reads from these areas, only when it writes values. Fencepost checking also increase the amount of memory the program allocates.

C. Detection of Heap Management

To be able to detect and localize memory leaks from the traces, all the events where components either allocated or deallocate memory are logged. By comparing the allocations and deallocations the memory blocks, which were not properly deallocated and therefore leaked, can be deduced. The traces contain the heap addresses of memory blocks in addition to identification and size information [8]. This approach is more suitable only for component-based embedded systems and also other tools available like, mtrace and purity etc.

IV. Working of DMMA in a Cluster

When a process tries to consume more memory than the virtual memory size, the system may crash. The proposed DMLD prevents above situation. In DMMA, maximum memory consumption limit can be set virtually for each process. When the process tries to consume more memory than its maximum memory limit then process seems to have memory leak, and the execution of that process is stopped in that node and its process execution is submitted to the healthy node in the cluster.

In the absence of proposed DMMA in the cluster, when a process tries to consume large amount of memory for long time (i.e. memory more than its maximum memory consumption limit), then it may cause other processes to starve for memory or may lead to system crash as shown in Fig 1. In presence of proposed DMMA in the cluster, it helps to detect memory leak and gives the memory leak information during the execution of the process and it also prevents the system crash.

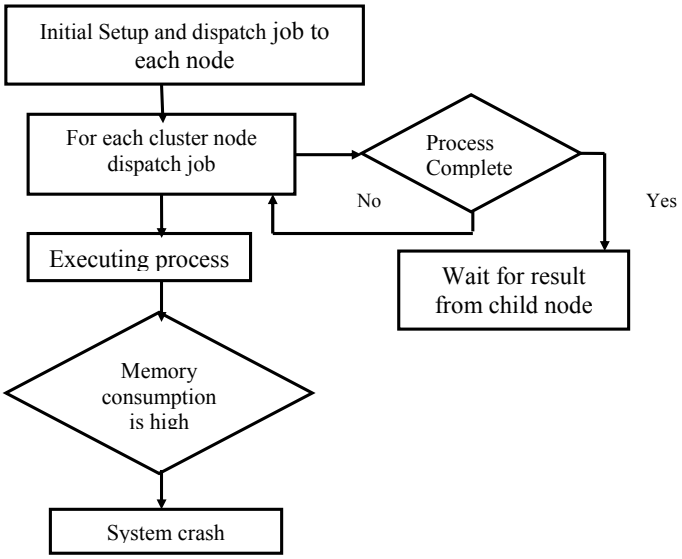

Fig. 1 Normal working of clusters without DMMA

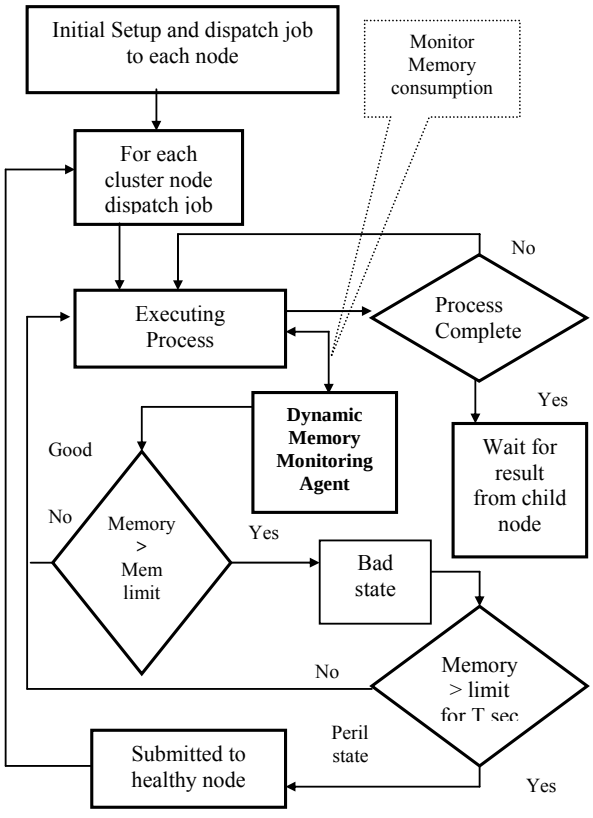

Fig. 2 Working of Clusters with DMMA

Memory leak is an application specific problem where application consumes more memory and never releases it. Memory leaks lead to shortage of memory space for other executing jobs in the cluster that potentially result in the

executing application slowing down which leads to more swapping.

V. Implementation of a Dynamic Memory Monitoring Agent

The proposed DMMA shown in Fig. 2 identifies the memory leak of a particular application. DMMA monitors total memory utilization of executing tasks in a specific time interval. Initially the maximum memory consumption of a currently executing program is set, depending on consumption of memory by the process, limits are set, and we label processes as good, bad or peril state. If memory utilization is higher than the maximum memory consumption limit DMMA consider the process in a bad state and identifies that process running on the node has memory leak. Otherwise DMMA considers the process in good state. If the memory consumption is very high for longer period then that proposed DMMA enables process to go into peril state.i.e, the process execution is stopped and it submit to another healthy node.

The Pseudocode of DMMA presented here is self-explanatory. In this DMMA we are providing maximum memory consumption limit with respect to MPI applications. The different modules of DMMA are shown in the boxes in listing 1 to 7. The DMMA is listed in subsection A and Client server paradigm is shown in subsection B.

A. Listing 1: Dynamic Memory Monitoring Agent Pseudo Code

```
Dynamic Memory Monitoring Agent ( )
 Begin:
  Split the output of ps –ef command.
  Extract the required PID value.
  While 1

   begin:
        Call 'test' subroutine
        Split the output of memdata (program)
        Extract "process total memory size"
            if  (total memory  > maximum memory)
                        Consumption limit then
               begin
                        Call 'bad' sub routine

            end

            else
                begin
                        print "application is in good state"
                        print total mem consumption value
                end
           count=0
           sleep for 12 sec

 end while
```

1. Listing 2: Subroutine of bad state

```
 Bad state ( )
    Begin
       For k=0 to 10 repeat
     Begin
         Call 'test' subroutine
         Split the output of memdata (program)
         Extract "process total memory size"
             if total memory is greater than maximum memory
                        Consumption limit then
           Begin
               Print "risky"
               Print count value
              Count++
          End
           If count is greater than 8
           Begin
               Call 'peril' subroutine
           End
       Sleep for 1 sec
       End for
   End (bad state subroutine)
```

2. Listing 3: Subroutine peril state

```
Peril state ( )
 Begin:
    Split the output of memdata (program)
     Extract "process total memory size"
     Print "application execution stopped"
    Open file descriptor for file.txt
     Write PID value of application,
              Executable name,
              Total memory size, to file.text
    Close file descriptor for file.txt

Send file.txt
from this client to server host
      (from client socket to server socket)
     die "exe terminated"
 end (peril subroutine)
```

3. Lisiting 4: Subroutine test

```
test  ( )
 begin
    split the output of ps-ef command
    check whether the executable to which agent is
  Monitoring
             exists or not
     if executable exists
     begin
        return
    end
    else
         print "executable doesn't exist"
         die "terminated"
 end (test subroutine)
end (memory monitoring agent)
```

4. Listing 5: Memdata pseudo code

```
memdata ( )
Begin
    Split the output of  ps-ef command
     Extract PID value of required executable from this output
     Extract memory consumption details using the command
       Proc/<PID>/status
   Grep the required information
   Print the total memory consumption value
   Print locked memory usage value
   Print resident set memory usage value
   Print heap memory usage value
   Print stack memory usage value
   Print executable memory usage value
   Print shared memory usage value
End
```

B. Listing 6: Sending Leak Information to Cluster Head Node

Client socket pseudo code: To client socket module we are providing the server hostname and port number as command line arguments.

```
      Client main ( )
 begin
            if command line arguments less than 2
            begin
               print "error, server name, port number not
               specified.
            End
        Create a socket 'sockfd' to send leak information
          to   Server.
         If  sockfd less than 0
           Begin
              Print "error opening socket"
           End
       Connect 'sockfd' to server socket
          If connect fails
           begin
            Print "no connection established"
            Return
           end
         Open the file 'file.txt' in read mode
         If open fails
           begin
                    print "can't open file"
                  end
              copy the data from  'file.txt' into a buffer
              write the data in buffer to 'sockfd'
              if write fails
                begin
                   print "error writing to socket"
                end
end(main)
```

1. Listing 7: Server sockets pseudo code

To server socket we are providing port number as command line argument.

```
Server  main ( )
  Begin
    If command line argument less than 1
    Begin
        Print "error , port no not specified"
        Return
   End
  Create a socket 'sersockfd' to receive data from client
  Bind 'sersockfd'
  Listen 'sersockfd',5
  Set SIGCHLD to ignore
  While 1
   Begin
  Newsockfd= Accept 'sersockfd'
      If accept fails
       begin
      Print "error on accept"
       End
  If fork is equal to 0
    Begin
      Read from 'newsockfd' into buffer
      Print the data in buffer
    Open a file in append mode
   If open fails
    Begin
       Print "can't open file"
    End
    Write data in buffer to file
    Close file
     Close newsockfd
      exit
  End
Else
      Begin
      Close newsockfd
end
end  (while)
end   (main)
```

VI. Experimental Evaluation

The effectiveness of DMMA agent is experimented using MPI (Message Passing Interface) application on PARAM Padma Linux Supercomputer nodes. The MPI job is submitted along with DMMA in the head node (xn02) that acts as server is shown in Fig. 3. The application process running on this node is the parent process. The processes running on other computer nodes (xn01, xn03 and xn04) are known child processes. We have tested our agent on Linux cluster and the output is shown in Fig. 4, having 4 Intel xeon nodes at 3 GHz processor speed and each node has two CPUs, 4GB RAM. The MPI used in our testing is C-DAC MPI [9] and MPICH [10] over PARAMNet-II [11] and Gigabit Ethernet as cluster interconnects.

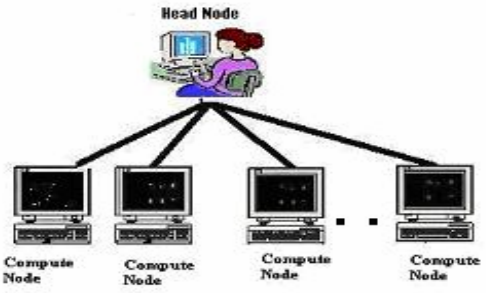

Fig. 3 Computational Cluster showing Head node (Parent process) and Compute nodes (Child process)

Our DMMA runs as a background processes on all these nodes. If the process is subject to memory leak in the cluster, the DMMA immediately send the details of memory leak information to the head node where the application is submitted. The details of memory leak of the child processes and information of the stopped child processes due to memory leak are informed to the users in the head node. This is accomplished by establishing sockets with head nodes and compute nodes as shown in Fig. 8. The Fig. 5 and Fig. 6 shows the list of application process enters into bad state and waits for specific interval of time for its corrective action. Fig. 7 shows the application memory consumption without any memory leak fault.

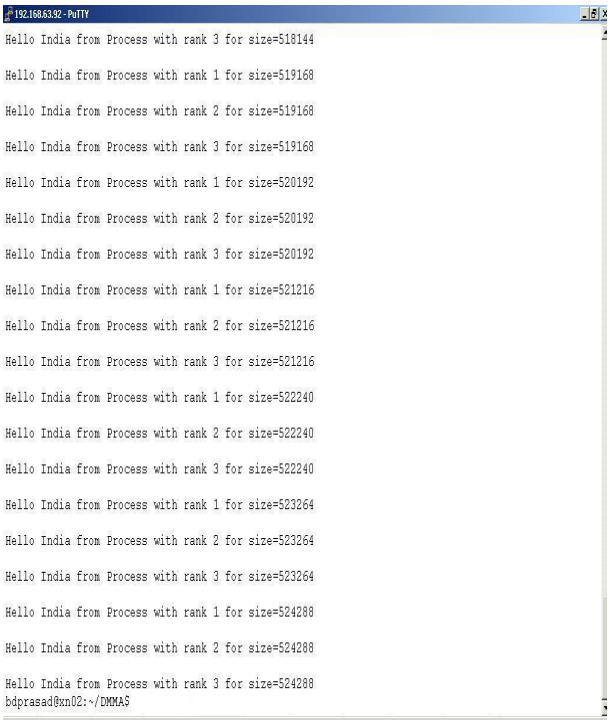

Fig. 4 Successful execution of MPI application

Fig. 5 Terminated application execution in xn01 due to memory leak

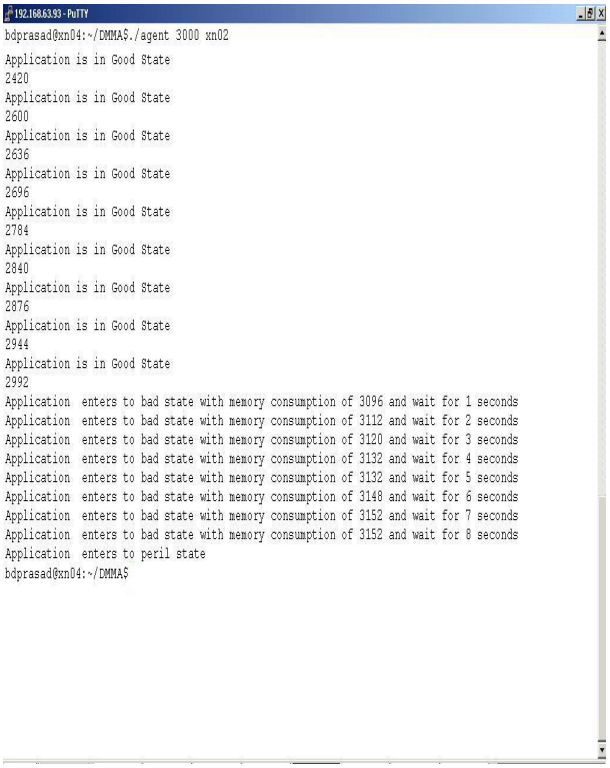

Fig. 6 Terminated application execution in xn04 due to memory leak

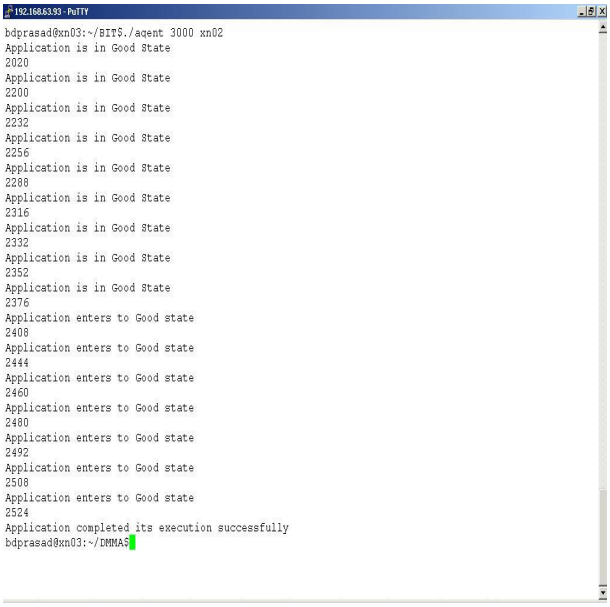

Fig. 7 Application completed successfully in xn03 without memory leak

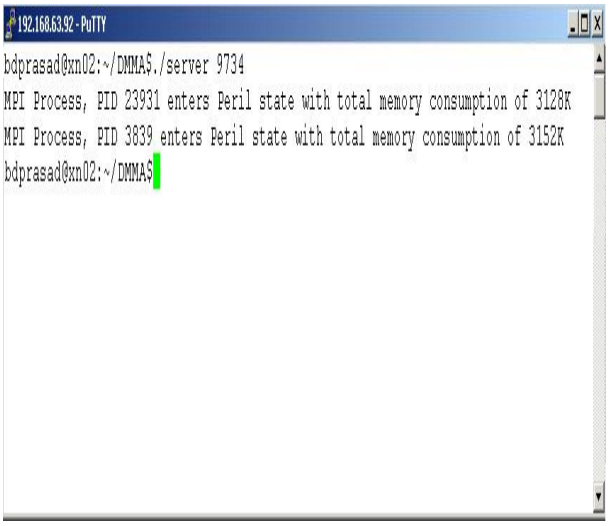

Fig. 8 Head node-xn02 receives leak information from affected child nodes

VII. Conclusion and Future Work

The proposed DMMA is intelligent and lightweight in nature. It provides memory leak detection dynamically in a cluster-computing environment. This approach helps the user to take preventive measures in their application, because it detects the memory leak information before it occurs. Hence it improves the performance and provides reliable service to its end users. The possible future research works extensions related to DMMA framework are: Incorporating an algorithm that can be predict desirable maximum memory utilization limit of application. DMLD technique can also be extended to support memory leak management for grid computing.

Acknowledgment

The authors would like to thank Dr. Prahlada Rao BB, Head SSDG CDAC Bangalore-38 India, for his invaluable feedback and review comments. The authors convey immense reverence and thankfulness to Shri Mohan Ram N Centre Head, CDAC Bangalore-38 for providing the suggestion and guidance to this project. We also thank Shri Ramakrishnan, Director General, CDAC India, for his encouragement and kind support for this project.

References

[1] R.Hastings and B.Joyce Purify: Fast detection of memory leaks and access errors. In proceedings of USENIX winter 1992 Technical conference, pages 125-136, Dec 1992.
[2] US-CERT vulnerability notes database http://www.kb.cert.org/vuls
[3] Mohammad Tanvir Huda, Heinz W.Schimdt, Ian D.Peake, An agent oriented dynamic fault tolerant framework for Grid computing 2005, Monash University: Melbourne.p.84.
[4] Valgrind: A Program Supervision Framework Nicholas Nethercote and Julian Seward.Electronic Notes in Theoretical Computer Science 89 No. 2, 2003.
[5] Ramandeeep singh, Get the better of memory leaks with Valgrind Linux J., February2006 (106), 2006.
[6] J.Seward, N.Nethercote, and Fitzhardinge.valgrind, an open -source memory debugger for x86– gnu/Linux http://valgrind. Kde.org/.
[7] Gray Watson, Debug Malloc Library, Published by Gray Watson, Version 5.4.2; October 2004.
[8] Heike Verta, T.S. Detection of heap management flaws in Component-based software. In EUROMICRO, 2004, Rennes, France: IEEE.
[9] CDAC-MPI, http://www.cdac.in/html/ssdgblr/cmpi.asp
[10] William Groups, Ewing Lusk, Nathan Doss and Anthony Skjellum. "A High-Performance, Portable Implementation of MPI Message Passing Interface Standard". Available at http://www.mcs.anl.gov/mpi/.
[11] PARAMNet, CDAC www.cdac.in/HTmL/pdf/PARAMNet.pdf