

A New Approach of Prediction of Memory Leak in the Cluster Computing Applications

R.Kavitha

C.Kotteeswari

G.Sumathi

Department of Information Technology, Sri Venkateswara College of Engineering
Pennalur, Sriperumbudur.

ABSTRACT

This paper presents a memory leak prediction algorithm for the cluster computing applications. This proposed algorithm uses process characteristics to calculate the exact memory requirement and uniquely identifies maximum memory utilization of an application before the application starts its execution. During the application execution phase, memory leaks in the application processes in the cluster is identified by existing Dynamic Memory Monitoring Agent (DMMA) gives information to the end users to make corrective actions and removes memory leak processes from the affected nodes. This unified approach increases the reliability and fault tolerant in the cluster computing.

Keywords

Cluster Computing, Memory Leak, Fault Tolerance, Maximum memory utilization, Process characteristics.

1. INTRODUCTION

Despite of many advantages provided by the cluster computing it also have several faults[4].They can be classified as Network faults, Software faults, Application and OS faults, Response faults and Timeout faults. The ability of parallel computers to execute multiple instruction streams [14] simultaneously gives rise to the problem of partitioning a program into a set of tasks that can be assigned to different processors. Therefore algorithms that compute optimal partitions of the program must be used for execution for increasing the efficiency.

Memory leak is one of the major application faults in the cluster computing. This can degrade application and overall system performance and it may eventually lead to program crash. This memory leak problem often occurs in High Performance Computing (HPC) systems and this can be solved with the help of existing Dynamic Memory Leak Detection (DMLD)[1] technique. HPC Cluster revolution [13] has set the part of the software stack on a path to become an essential, global binding agent for today's sprawling HPC hardware and software infrastructure.

The algorithm to predict maximum memory utilization limit of an application makes use of the process characteristics to calculate the exact memory value. This helps to obtain improved performance as it provides the maximum memory limit before the execution starts.

This paper is organized as follows. In section II we discussed briefly the existing memory leak detection tools. Section 3 presents the proposed algorithm details and user application with the help of existing DMMA. Section 4 discusses the implementation of the algorithm to predict maximum

memory utilization limit of an application and the experimental evaluation; Finally Section 5 presents the conclusions.

2. EXISTING WORKS

Ccmalloc

CCMALLOC [8] is a memory profiling and malloc debugging library for C and C++ programs that requires the GNU debugger, gdb. All tests were run using the GNU C and C++ compilers. We found CCMALLOC to be easy to install. We consider the documentation for CCMALLOC to be poorly written and often difficult to understand (the file ccmalloc.cfg is the only available user manual). Poor documentation made the product difficult to use. CCMALLOC was not able to detect "writing beyond the bounds of the allocated memory block" errors.

Mss

MSS is free (GPL) software designed to find errors in C and C++ programs caused by the misuse of dynamically allocated memory. We found MSS to be easy to install. MSS requires either the statement `#include "mss.h"` be added to each source file before compiling or one can simply use the `-include` option on the GNU compiler to have the `#include "mss.h"` automatically added to each source file. MSS also requires the insertion of `#define MSS` into each source file or one can use the `-D` compiler option with the gcc compiler and not have to modify source code.

Mpatrol

MPATROL[8] is a debugging tool for detecting run-time errors that are caused by the misuse of dynamically allocated memory for C and C++ programs. MPATROL requires either the statement `#include "mpatrol.h"` be added to each source file before compiling or one can simply use the `-include` option on the GNU compiler to have the `#include "mpatrol.h"` automatically added to each source file. MPATROL works well, but not for all our C and C++ tests. We found that sometimes it was difficult to decide what options to set when running our various tests.

Memdebug

MEMDEBUG[8] is a debugging tool for the run-time detection of the following errors in C (and not C++) programs memory leaks, duplicate de-allocations, de-allocating an illegal pointer, out-of- bounds pointer references, off-by-one errors in memory blocks, and unallocated pointer. We found MEMDEBUG to be easy to install and to use. MEMDEBUG requires the addition of `#include "memdebug.h"` to each source file before compiling or one can simply use the `-include memdebug.h` option on the GNU

compiler to have the #include “memdebug.h” automatically added to each source file. MEMDEBUG also requires the insertion of #define MEMDEBUG into each source file or one can use the -D compiler option with the gcc compiler and not have to modify source code. MEMDEBUG was not able to detect general out-of-bounds errors but only off-by-one.

Valgrind

Memory leaks can be detected with the help of an open source tool named Valgrind[5]. It can point the errors when used with the application executables as a command-line argument. The Valgrind output contains memory leak details that can be viewed with the help of log file. The major disadvantage of this tool is when it is made to run along with the application it consumes more memory.

Detection of heap management

In order to identify memory leaks, all the events where the components and their memory is either allocated or de-allocated they are logged. The traces include the heap addresses of the memory blocks with their identification and size information [9]. This technique is mostly used for component-based embedded systems.

3. PROPOSED WORK

We propose an algorithm that can predict maximum memory to be consumed by an application. It also includes effective scheduling algorithm. Effective scheduling [10] is critical for the performance of an application launched onto the grid environment. Once the application is submitted to the cluster, the maximum memory is calculated using the algorithm which considers the following details.

The algorithm takes into account the **file size**. There are various file systems in UNIX. Each file type consumes certain amount of memory. On finding the different file types used by the application their memory is calculated.

The data consumed by the application is predicted on comparing the input values that the application gets during its execution. Some application may use the library files and in this case it uses more data for its execution.

The maximum number of process can be determined on analyzing the type of application submitted to the cluster. The application is split into various numbers of processes, each performing a different task and finally they are combined to get the end result. The **CPU time** is predicted based on the computational complexity of the application.

Stack size specifies the size (in Bytes) of the execution stacks used by each user process on server. An execution stack is an area of server memory where user processes keep track of their process context and store local data.

Working

As shown in Fig 1.1, in the first stage, the jobs are submitted to head node of the cluster. DMMA compares the process memory with the available free memory in each node. The jobs are allocated to the nodes according to the memory available in each node and it starts its execution.

According to the DMMA [1] technique, if the memory consumption of any process exceeds the maximum memory limit which is predicted by the algorithm, the process is put into a bad state. If the memory consumption stays high for a certain period of time then the process goes to peril state. i.e., the process execution is stopped and it is removed from the affected node.

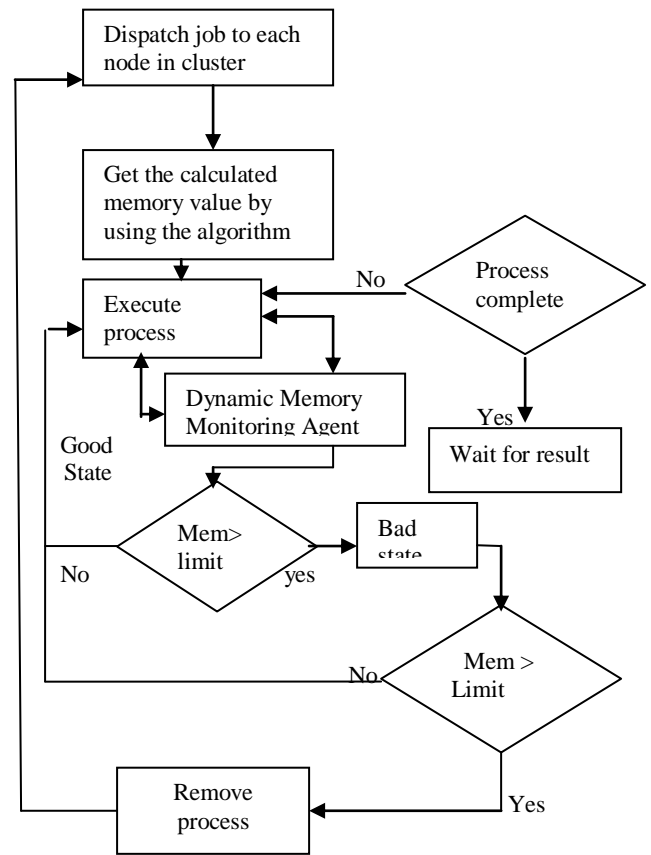


Fig 1.1 Working of Dynamic Memory Monitoring Agent in clusters

In this work proposed algorithm is used along with the existing DMMA [1] technique to avoid memory leak problems very effectively. In this proposed algorithm the process characteristics of an application like file size, data size, CPU time, stack size and numbers of processes are taken into account for calculating maximum memory. Once this value is calculated it is given as one of the input to the DMMA. The proposed algorithm doesn't assume virtually the maximum memory as in the case of existing DMMA. It takes the calculated memory value to execute DMMA, that inturn improves the efficiency of the DMMA technique to achieve maximum fault tolerance in clusters.

Fig 1.1 shows the working of Dynamic Memory Monitoring Agent in clusters. The implementation of DMMA in clusters consist 3 sub modules

- Job Submission
- Memory monitoring
- Re-scheduling

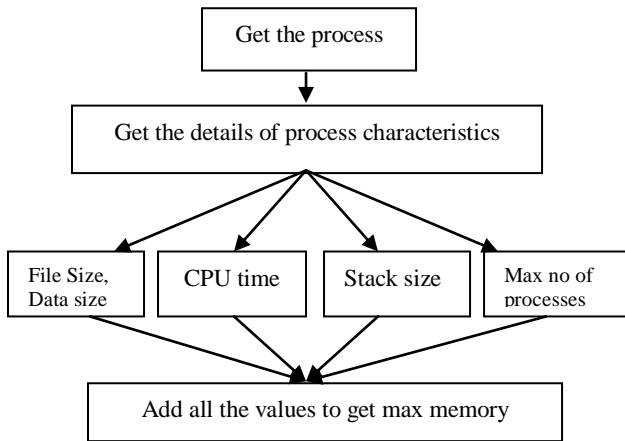


Fig 1.2 Calculation of maximum memory utilization

An algorithm which is used to predict the maximum memory utilization of an application is incorporated along with the existing DMMA [1]. This helps to achieve greater performance in the cluster environment. Fig 1.2 explains the algorithm

As given in Fig 1.2, the detail of the process which is submitted to the head node is taken. This includes the data size which can be in any range starting from KB to MB. Then the other parameters taken into account include file size, CPU time, stack size and the maximum number of process available in that node.

This algorithm doesn't assume virtually the maximum memory. It takes the calculated memory value to execute DMMA. So with the help of this algorithm we can achieve greater tolerance in the cluster environment.

4. PERFORMANCE STUDY

The algorithm to predict maximum memory utilization limit of an application is implemented in a cluster. The cluster configuration steps are done by configuring Domain name System (DNS) and Network Information Service (NIS) of the head node and the compute nodes. Then remote shell (rsh) is used to communicate between the nodes. The first step is to submit the job to cluster. The number of jobs to be submitted to the head node of the cluster is given by the user.

We tested our work using Linux cluster having five nodes as shown in Fig 1.3. Each node in the cluster has two CPUs, 2GB RAM and 4GHz processor speed.

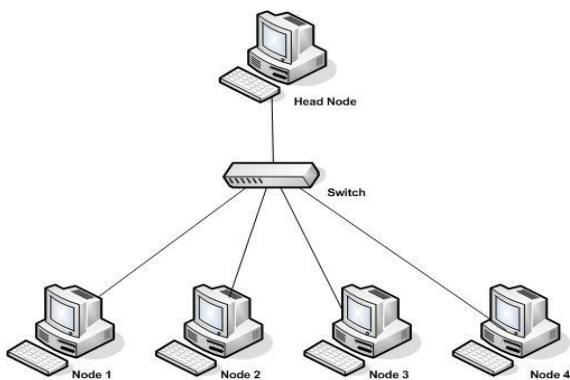


Fig 1. Cluster with five nodes

```

    root@server1:/home
    [root@server1 ~]# cd /home/
    [root@server1 home]# ls
    a.out client.c cluster new node testdma.c
    [root@server1 home]# clear

    [root@server1 home]# cc testdma.c
    [root@server1 home]# ./a.out
    STARTING DMMA APPLICATION
    Enter the no of application to be submitted to the cluster:3

    Enter the maximum memory limit of head node:1000

    enter the maximum memory limit of compute node:2600

    enter the id and size for application 1:1    600
    enter the id and size for application 2:2    2300
    enter the id and size for application 3:3    4000

    Application id  Applicationsize
    1                600
    2                2300
    3                4000
  
```

Fig 1.4 Head Node Inputs

As shown in Fig 1.4, the processes to be executed along with the memory requirements and maximum memory consumption limit calculated by using the algorithm are given to the head node of the cluster as the inputs. The memory requirement of each process is compared with maximum memory available in each node of the cluster. The processes are allocated to the nodes according to their memory requirements. At some point of time during execution when memory is not available for the execution of the program, then it is given to the node that has enough memory for its execution. All the processes are scheduled and executed.

If the application gets the sufficient memory to complete its job, it finishes its job successfully and this can be visually seen as shown in Fig 1.5.

If the process is able to finish its execution successfully, then it is considered to be in good state. The memory consumption value is monitored at some specific time interval. This is clearly shown in Fig 1.5 where the memory value gets increased as the application starts its execution.

```

    root@server1:/home
    Enter the no of application to be submitted to the cluster:3

    Enter the maximum memory limit of head node:1000

    enter the maximum memory limit of compute node:2600

    enter the id and size for application 1:1    600
    enter the id and size for application 2:2    2300
    enter the id and size for application 3:3    4000

    Application id  Applicationsize
    1                600
    2                2300
    3                4000
    Application 1 submitted to node 1
    Application 1 is in good state with memory consumption value 600

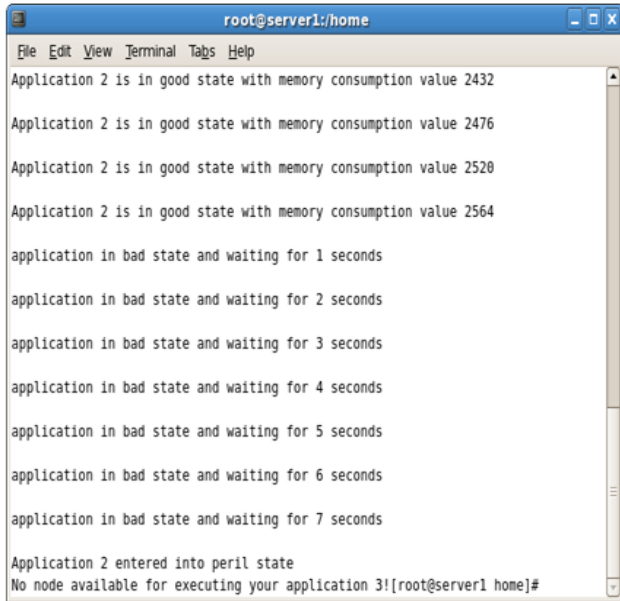
    Application 1 is in good state with memory consumption value 668

    Application 1 is in good state with memory consumption value 736

    Application 1 has finished execution successfully
  
```

Fig 1.5 Successful execution of an application

When the process starts to consume more memory than the memory available in that node, it enters into a bad state where it waits for certain amount of time and if it still continues to consume more memory it enters into peril state. This is shown in Fig 1.6.



```
root@server1:/home
File Edit View Terminal Tabs Help
Application 2 is in good state with memory consumption value 2432
Application 2 is in good state with memory consumption value 2476
Application 2 is in good state with memory consumption value 2520
Application 2 is in good state with memory consumption value 2564
application in bad state and waiting for 1 seconds
application in bad state and waiting for 2 seconds
application in bad state and waiting for 3 seconds
application in bad state and waiting for 4 seconds
application in bad state and waiting for 5 seconds
application in bad state and waiting for 6 seconds
application in bad state and waiting for 7 seconds
Application 2 entered into peril state
No node available for executing your application 3!root@server1 home#
```

Fig 1.6 Termination of an application

If the process enters into peril state, the execution is stopped and it is removed from that node. Once it is removed, re-scheduling has to be done for that process. A low-complexity dominant sequence clustering (DSC) algorithm [15] is available for scheduling parallel tasks on an unbounded number of completely connected processors. On rescheduling, there may be many nodes available for executing the application. So, the rescheduling algorithm is designed based upon certain factors. The important factors include,

Based upon the type of application, decide the capacity of hard disk (512MB, 2GB, 4GB) that the application needs and the job are allocated to the appropriate node.

Check the processor speed of each node. If the application needs faster performance allocate it to the suitable node.

Since each node can have a different database server, select the node which has the database which it needs. For example, if more security is needed in the case of military application choose the more secure database like Oracle.

5.CONCLUSION

The proposed algorithm detects very efficiently the maximum memory utilization limit of an application when used along with the existing DMMA technique helps to achieve greater fault tolerance, reliability and efficiency of cluster computing.

REFERENCES

[1]. Roohi Shabrin S., Devi Prasad B., Prabu D., Pallavi R. S., and Revathi P., “Memory Leak Detection in Distributed System”, World Academy of Science, Engineering and Technology 16 2006.

[2] R.Hastings and B.Joyce, “Purify Fast detection of memory leaks and access errors”, Proceedings of USENIX winter 1992 Technical conference, pages 125-136, Dec 1992.

[3]US-CERT vulnerability notes database
<http://www.kb.cert.org/vuls>

[4] 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.

[5] “Valgrind A Program Supervision Framework”, Nicholas Nethercote and Julian Seward.Electronic Notes in Theoretical Computer Science 89 No.2, 2003.

[6] “Ramandeep singh, “Get the better of memory leaks with Valgrind”, Linux J., February2006 (106), 2006.

[7] J.Seward, N.Nethercote, and Fitzhardinge, “valgrind, an open-source memory debugger for x86- gnu/Linux” <http://valgrind. Kde.org/>.

[8] “Gray Watson, Debug Malloc Library”, Published by Gray Watson, Version 5.4.2; October 2004.

[9] Heike Verta, T.S. “Detection of heap management flaws in Component-based software”, In EUROMICRO, 2004, Rennes, France IEEE.

[10]C. D. Polychronopoulos , D. J. Kuck, Guided self-scheduling “A practical scheduling scheme for parallel supercomputers, IEEE Transactions on Computers, v.36 n.12, p.1425-1439”, Dec. 1987

[11]Frank D. Anger , Jing-Jang Hwang , Yuan-Chieh Chow, “Scheduling with sufficient loosely coupled processors, Journal of Parallel and Distributed Computing, v.9 n.1, p.87-92”, May 1990.

[12]Glenn R. Luecke, James Coyle, Jim Hoekstra, Marina Kraeva, Ying Li, Olga Taborskaia, and Yanmei Wang, “A Survey of Systems for Detecting Serial Run-Time Errors”,2006.

[13]. HPC Management Software “Reducing the Complexity of HPC Cluster and Grid Resources”.

[14]. M. Girkar C. Polychronopoulos “Partitioning programs for parallel execution”, Proceedings of the 2nd international conference on Supercomputing.

[15] T. Yang A. Gerasoulis “ Scheduling Parallel Tasks on an Unbounded Number of Processors” IEEE Transactions on Parallel and Distributed Systems 1994.