doi:10.11887/j.cn.202205005 http://journal.nudt.edu.cn

并行程序运行故障原因识别*

刘 轶,高玉林,张国振
(北京航空航天大学 计算机学院,北京 100191)

摘 要:高性能计算系统的复杂性和规模的不断增长使得系统的平均无故障时间越来越短,因此系统的硬软件故障导致并行程序运行出错的概率随之增加。此外,并行程序本身可能存在的编程错误也会导致运行出错。由于处理上述两类故障原因的措施迥异,所以在程序运行出现故障时,用户需要关注故障原因的类别。针对这一问题,设计和实现了一种基于作业管理系统 Slurm 的并行程序运行故障原因识别系统。通过对 Slurm 进行扩展,监控作业状态,重提交和重运行作业。根据作业运行结果,区分故障原因类别。故障注入方式进行的实验表明,该系统具有较高的识别准确率。

关键词:高性能计算系统;Slurm;运行故障;故障检测
中图分类号:TP302 **文献标志码**:A **文章编号**:1001-2486(2022)05-045-08

Identifying causes of execution failure for parallel programs

LIU Yi, *GAO Yulin*, *ZHANG Guozhen*
(School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

Abstract: With the increasing of scale and complexity of high-performance computing systems, the mean time between failures is getting shorter, which causes an increasing probability of execution-failure caused by the hardware and software failures for parallel programs. In addition, the possible programming errors (i.e. bugs) that exist in parallel programs can also lead to execution failure. Approaches to deal with the above two types of execution failures are totally different, therefore, when an execution-failure occurs, the programmer must figure out if the failure is caused by a system fault or a programming bug. In response to this issue, a system to identifying causes of execution-failures for parallel programs was designed and implemented on the basis of the Slurm. The system has all the supported features of Slurm, as well as the ability to monitor job status, re-submit and re-run jobs. The experimental results of the job execution show that the system can distinguish the type of program execution-failures. Experiments conducted with fault injection also demonstrates the accuracy of the system.

Keywords: high performance computing system; Slurm; execution failure; fault detection

高性能计算(high performance computing, HPC)系统广泛应用于国防建设、科学研究以及国民经济等重要领域,这些重要领域内应用需求的日益增长促进了高性能计算系统的迅速发展和规模增长。日本于2020年6月发布的超级计算机 Fugaku[1]拥有158 976个节点,节点处理器包含48个计算核及4个辅助核,系统峰值浮点性能高达513 PFLOPS。神威·太湖之光[2]是性能优越的国产超级计算机,拥有40 960个处理器,共10 649 600个处理器核,系统峰值性能达125.436 PFLOPS。

随着高性能计算系统规模的日益庞大以及系统内组件数量的迅速增加,硬软件复杂性不断提高、系统日益复杂,这使得高性能计算系统的平均无故障时间(mean time between failures, MTBF)越来越短。HPC 系统上通常运行并行程序,由于参与并行程序运行的节点数巨大、程序运行时间长,系统 MTBF 的降低使得程序在运行过程中发生硬软件故障的可能性也随之增加。突发的系统硬软件故障,如节点死机、操作系统崩溃、互连网络故障等,会影响程序执行,甚至导致程序运行失效。

消息传递接口(message passing interface, MPI)是 HPC 系统中用于进程间通信的并行编程接口。尽管 MPI 应用广泛,但它自身并不具备容错机制。如果其中一个计算节点崩溃,整个 MPI 并行程序将会崩溃。在 HPC 系统中,失效的程序通常比成功的程序消耗更多的资源。

大多数 HPC 系统使用作业管理系统(如 Slurm[3]、PBS[4])来管理资源,并执行不同用户的多个作业。程序执行过程中,用户不能直接与程

* **收稿日期**:2020-11-12
基金项目:总体技术及评测技术与系统研究资助项目(2016YFB0200100)
作者简介:刘轶(1968—),男,河北安新人,教授,博士,博士生导师,E-mail: yi.liu@buaa.edu.cn

序进行交互。同时,作业管理系统只能反映作业的排队、运行状态,并不能发现程序在运行时出现的所有异常情况。当程序自身存在编程 bug,如死锁、死循环等情况时,程序不会被作业管理系统中止,这将占用计算资源,并且影响程序的正常执行。在硬软件故障方面,除了元器件和部件故障导致死机、程序崩溃等常见的故障之外,还有一些间歇性的故障。例如,内存工作不稳定或接触不良可能导致系统故障。这些故障会偶发出现,具有不确定性。

虽然用户可以在小规模下调试程序,但是许多问题只有在进程足够多时才会出现,而且某些调试工具虽可以收集程序状态,但不能提供参与程序执行的计算节点的硬软件状态。因此,如果程序在 HPC 上运行失败,用户应该首先区分运行失败是由于程序自身 bug 还是系统硬软件故障。

Slurm 是一个大规模 Linux 机群的轻量级开源资源与作业管理器,可以为用户提供对机群资源的共享访问。由于轻量且高效的性能,Slurm 受到超算中心的关注,超过 60% 的 500 强超级计算机使用 Slurm[3]。本文基于 Slurm 提出一种并行程序运行故障原因识别系统,将程序运行失效的原因识别为系统硬软件故障或程序自身 bug,减轻了调试过程的平均故障间隔时间缩短对未来百亿亿级超级计算机的影响,并提高了调试大型并行程序的效率。

1 作业管理系统 Slurm 简介

Slurm 是一种开源的 Linux 机群管理和作业调度系统。它具有三个主要功能:首先,它提供了一个框架,用于在高性能计算机群上提交、启动和监视分配所有计算节点上的作业;然后,它使用排队策略管理等待资源分配的作业队列,以提高计算节点资源的利用率;最后,它统筹分配计算节点的资源给用户的作业,使得用户的作业可以运行。

Slurm 的组织架构如图 1 所示。Slurm 主控制节点运行 slurmctld 守护程序,用于管理资源、调度和监视作业。当主控制节点出现故障时,备用控制节点对系统进行管理,提高了 Slurm 的容错性。每个计算节点都运行 slurmd 守护程序,该守护程序负责等待和执行 slurmctld 分配的作业,并监视作业的状态、响应 slurmctld 对机器状态和作业信息的请求、发送机器状态和作业状态的变化到 slurmctld。slurmstepd 由 slurmd 守护进程在作业启动时生成,并在作业完成后终止。slurmstepd 是 Slurm 的作业步管理进程,负责管理作业步的输入、输出(stdin、stdout 和 stderr)及信号处理。

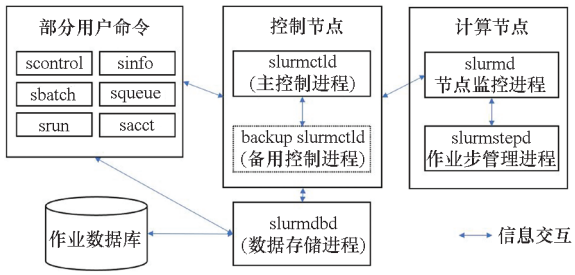

图 1 Slurm 组成结构

Fig. 1 Slurm architecture

Slurm 用户命令包括 sbatch, srun, sinfo, squeue 和 scontrol 等。sbatch 和 srun 用于提交作业;sinfo 和 squeue 可以报告系统所有计算节点状态和作业队列中的作业状态;scontrol 用于监视和修改机群中的配置和状态信息。当普通用户在 Slurm 上调试大型并行程序时,只能通过 Slurm 提交待调试的作业,等待 Slurm 分配计算资源,最后等待执行结果。在程序执行期间,普通用户只能使用 squeue 和 scontrol show job 命令查看作业的状态,不能得到节点硬软件状态的数据。在节点失效、运行超时等情况下,scontrol show job 命令可以输出作业的信息(如 NODE FAILURE、TIMEOUT 等),但不能定位具体的故障位置和原因。

HPC 上的大多数并行应用程序需要很长时间来运行,它们的测试运行通常持续几个小时甚至几天,普通用户不可能时刻查看程序状态和节点状态。如果程序运行失效后,普通用户没有第一时间发现程序运行结束,并且在发现运行结束后很难区分程序运行失效是由于程序自身 bug 还是节点的硬软件故障,这将大大增加程序调试的时间和降低调试的效率。因此,需要一种基于 Slurm 的并行程序运行故障原因识别机制,通过监控程序和节点状态,当程序出错时识别程序的故障原因。

2 并行程序运行故障原因识别系统

2.1 运行故障原因分类

HPC 系统上并行程序运行失败按照故障原因分类有多种维度。

1)按照故障现象分类(外在表现):有节点死机、程序异常退出、程序运行停滞、程序执行时间异常和程序运行结果错误等类别。

2)按照故障原因分类:有程序编程错误和系

统硬件/软件故障两种。

3)按故障的确定性分类:有确定性故障和非确定性故障两种类别。确定性故障为每次运行都会出现的故障,具有确定性和可重复性;非确定性故障即在运行期间可能会出现的故障,具有不确定性。

把上述三种故障分类里 2 和 3 的故障类型进行组合,就可以得到以下四种故障类型:

a)编程错误导致的确定性故障;

b)系统硬/软件失效导致的确定性故障;

c)编程错误导致的非确定性故障;

d)系统硬/软件失效导致的非确定性故障。

本文的目标在于检测出并行程序的运行故障,并将故障原因识别为上述 4 种故障类型,从而帮助程序员提高在 HPC 上运行和调试程序的效率。

确定性故障具有确定性和可重复性,当用户程序在 Slurm 上运行出错时,首先检测错误原因是否为确定性故障。当程序运行出错时,重新提交程序并且组合使用换节点不换程序、换程序不换节点等方法,检测出错原因是否为确定性故障。具体思路为:指定首轮节点,在首轮节点上运行另一种经过验证的程序,如果验证程序出错,可推断为 b;排除首轮节点,在其他节点上执行该程序,在使用验证程序确认新节点中无故障的情况下,如果该程序仍然出错,则可推断为 a。

非确定性故障存在随机性,且很难通过 1 ~ 2 次程序运行来复现和识别故障原因。通过采用在不同节点上多次重复执行程序的方法,在运行次数比较多、得到足够的运行结果后进行判断。如果在多个节点上程序运行仍会出现故障,可推断为 c;如果在不同的节点上没有出现故障,只在特定节点上出现故障,可推测为 d。

2.2　系统架构与实现

为了实现并行程序运行故障原因识别系统,本文在 Slurm 19.05.2 发行版的基础上进行了扩展。扩展后的系统架构如图 2 所示,其中阴影填充的部分为扩展的模块。

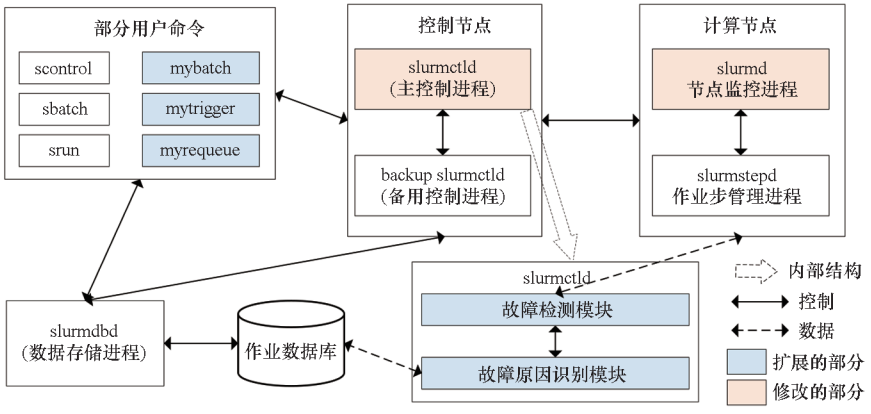

图 2　扩展后的 Slurm 架构

Fig. 2　Extended Slurm architecture

为了进行故障现象的检测和故障原因的识别,首先在 Slurm 的主控制进程 slurmctld 中扩展了故障检测模块和故障原因识别模块,用于程序运行故障的检测和故障原因的识别。

同时在 Slurm 中扩展了三个用户命令,分别是 mybatch、mytrigger 和 myrequeue 命令。三个命令对应功能如表 1 所示。

基于在 Slurm 中实现的命令和模块,并行程序故障原因识别的流程为:

步骤 1:mybatch 命令提交用户程序,提交成功后返回作业编号。

步骤 2:根据作业编号,故障检测模块对程序和程序所在运行节点进行状态监控。

表 1　实现的命令列表

Tab. 1　List of implemented commands

命令	作用
mybatch	提交作业,把用户程序放进作业队列中,进行自动化的检测
mytrigger	定时根据用户程序的编号监控程序的状态和程序所在节点的状态
myrequeue	将上次运行节点加入排除节点中,将作业放入作业队列里重新排队;如果作业上次运行失败,那么指定上次运行的节点,运行验证程序

步骤 3:作业运行结束后,故障检测模块停止

监控,根据程序运行的结果进行下一步:如果程序运行结果为成功完成、取消、超出内存限制、超过截止时间四种情况,根据运行结果可以知道运行结果的原因,结束故障检测,转到步骤5;如果程序是其他运行结果,转到步骤4。

步骤4:如果程序重运行次数少于2次,将上轮节点加入程序的排除节点中,重新运行程序,同时指定上轮节点运行验证程序,转到步骤2;如果程序重运行次数大于或等于2次,转到步骤5。

步骤5:故障原因识别模块从作业数据库中获取用户程序和验证程序的多次运行结果,综合两种程序的运行结果,识别故障的类型。

2.3　故障检测模块

在用户程序运行期间,为了及时检测程序bug或系统硬软件故障,需要对程序状态和节点状态进行监控,因此在slurmctld中扩展了故障检测模块。故障检测模块的工作流程如图3所示。

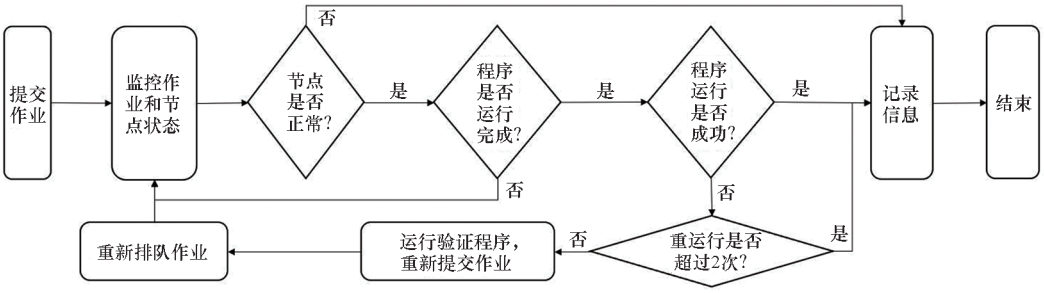

图3　故障检测流程

Fig. 3　Fault detection flow chart

在程序运行期间,每隔一段时间(预设15 s)根据用户的程序编号检查程序的状态和程序所在节点的状态。如果在程序运行期间某个节点通过心跳机制没有响应,那么该节点很可能出现了严重的硬件故障,导致了节点死机/崩溃,记录该节点存在问题,并取消程序的运行。

当程序结束运行后,需要根据用户程序的运行结果对程序进行处理。当程序运行结束后,记录程序运行数据,并将运行数据发送到故障原因识别模块。如果本次程序执行是非正常结束(如失败、超时、节点崩溃),则把程序本次运行的节点集合加入该程序的排除节点列表,以使该程序在其他计算节点上重新运行。然后把该程序放到Slurm作业队列中重新排队,并且重新对该程序的状态进行监控。同时指定程序当次运行的节点集合运行验证程序。验证程序为预先选择的且没有编程故障的程序,验证程序可以在系统中提前运行,那么运行验证程序只需指定节点,然后将验证程序重新排队运行即可。

2.4　故障原因识别模块

程序运行结束后,作业数据库中的数据发送到故障原因识别模块。使用A_n表示程序第n次运行的结果,S_n表示程序第n次运行时所在节点集合,C_n表示在S_n上运行验证程序的结果,该模块的故障原因识别流程为:

1)如果A_1成功,则认为程序正常执行;

2)如果A_1失败、C_1失败,则认为节点集合S_1中发生了系统硬软件故障导致的确定性故障;

3)如果A_1失败、C_1成功,但A_2失败,说明S_1正常工作。A_2失败后,还需对S_2增加验证程序的检验。如果C_2成功,说明S_2没有故障,则认为用户程序有bug,发生了由编程错误导致的确定性故障;如果C_2失败,说明节点集合S_2中发生了系统硬软件的确定性故障。

4)如果A_1失败、C_1成功、A_2成功,则认为存在非确定性的故障。但是非确定性故障的具体类型不确定,需要进一步识别。

进一步识别需要手动执行myrequeue和mytrigger命令,重提交并监控用户程序,通过比较作业的运行结果和输出结果,判断作业是否出错。如果出错,则在节点上运行验证程序,综合在不同节点上程序和验证程序的运行结果。如果在多个节点组合上程序运行出错或者输出结果不同,可推断为用户程序有bug;如果用户程序在不同的节点组合上运行均成功,验证程序也正常运行,可推测为S_1中存在非确定性的硬件故障。

3　实验测试

3.1　实验环境与测试方法

实验部署在4台服务器(b1 ~ b4)搭建的小规模机群中,其中b1是小规模机群的Slurm控

制节点。机群中文件共享的软件是网络文件系统(network file system, NFS),b3 是 NFS 服务器。应用程序采用高性能计算基准测试程序 NPB[5]。由于实验环境的限制,选择 NPB 中的 5 个程序——IS、EP、LU、BT 和 SP,测试规模使用 C 级。表 2 列出了小规模测试集群的环境配置参数。

表 2　测试环境配置

Tab. 2　Configurations of experimental environment

配置	参数
操作系统	CentOS 7
处理器	Intel(R)Core(TM)i7 -7700@3.60 GHz
计算节点内存	3 GB
带宽	100 Mbit/s
作业管理系统	Slurm 19.05.2
共享文件系统	NFSv4
MPI 环境	MPICH 3.3
基准测试程序	NPB3.4

实验主要分为两个部分:准确性测试和运行开销测试。其中准确性测试包括检测准确率、识别准确率和识别延迟的测试;运行开销测试主要针对系统对程序执行的影响。

3.2　准确性测试

在实验中,设置的评估指标为:

1)故障检出次数(fault detection times, FDT):故障注入后,故障检测模块成功检测出来的次数;

2)故障识别次数(fault identification times, FIT):注入的故障检测出来后,故障原因识别模块正确识别出故障原因的次数;

3)故障识别延迟:故障注入后,到故障原因识别模块识别出故障原因所需要的时间。

针对程序自身 bug 导致的故障,在 MPI 并行程序源代码中加入 MPI_Recv 阻塞死锁、while 无限循环、堆栈溢出等问题代码。实验中程序的最大运行时间为 10 min。实验结果如表 3 所示。通过实验发现,死锁和 while 无限循环等 bug 会导致程序运行超时,无限递归导致的堆栈溢出等错误会导致程序运行崩溃。程序自身的 bug 会导致程序在不同节点上的多次运行都失败。从表 3 的实验结果可以看出该系统可以有效地检测和识别程序自身的 bug。

表 3　程序自身 bug 实验结果

Tab. 3　Program's own bug experiment results

故障类型	实验次数	FDT	FIT	程序平均运行时间/s	平均识别延迟/s
程序死锁	10	10	10	616.0	1 362.2
无限循环	10	10	10	612.3	1 355.7
堆栈溢出	10	10	10	1.5	150.0

文献[6]实现了 HPC-SFI[6],可以有效地在一个 HPC 系统中注入三种类型的系统故障,分别是节点内故障、互连网络故障和存储/并行文件系统故障。本文使用 HPC-SFI 来随机注入节点内处理器故障、互连网络故障和整个节点的故障。故障注入的实验结果如表 4 所示。使用的测试程序是 NPB 的 SP. C. 2,验证程序为 BT. C. 2,设置最大程序运行时间均为 10 min。

表 4　HPC-SFI 硬件故障注入实验结果

Tab. 4　HPC-SFI hardware fault injection experiment results

故障部位	实验次数	FDT	FIT	平均识别延迟/s
互连网络	10	10	10	131.4
处理器	10	10	10	653.7
整个节点	10	10	10	168.1

注入的互连网络故障和整个节点的故障,会导致计算节点与控制节点之间的通信中断,可以很快被故障检测模块发现;处理器故障导致作业运行时异常崩溃,作业运行提前结束。通过表 4 的实验结果可以得出,本系统能够 100% 检测和识别 HPC-SFI 注入的硬件故障。

从表 3 和表 4 的平均识别延迟可以看出,故障原因识别延迟与故障类型存在一定的联系。即在相同的测试程序下,平均故障延迟因故障的不同而不同。其中的原因是,故障检测模块需要至少运行 2 次用户程序和一次验证程序,同时在 Slurm 中重排队的作业需要等待 2 min 才能再次被分配资源开始运行。所以当故障导致程序运行超时时,程序运行了最大运行时间后因 Slurm 超时机制而中止,平均识别延迟会较高;当故障导致作业运行崩溃或节点崩溃时,作业运行因故障提前中止,平均识别延迟较低。

潜在的故障是指一个故障在发生之前的潜在行为,潜在的故障是普遍存在的。许多节点故障

不是突然变化的结果,而是长期性能下降(软件老化)的结果。在潜在故障存在期间,节点的性能已经出现了偏离,比如计算速度慢、计算出错率高等问题,但节点尚未失败。超过20%的机器故障都是由这些潜在的故障引起的[7]。

在实验中分别模拟互连网络丢包、CPU运行变慢等潜在硬件故障。在模拟潜在故障的实验中,使用的实验程序是NPB的SP. C. 4,最大运行时间为10 min;验证程序为BT. C. 4,最大运行时间5 min。

通过Linux内核的流量控制器Traffic Control对网卡发送的数据包进行限制,通过设置不同的丢包率模拟HPC系统中互连网络的丢包、网络中断等故障。本文分别在丢包率为1%、5%和10%下运行了10次实验。实验结果如表5所示。

表5 网络丢包故障实验结果

Tab. 5 Network packet loss experiment results

丢包率	程序运行结果	实验次数	FDT	FIT
1%	正常(373.5 s)	10	0	0
5%	超时	10	10	10
10%	超时	10	10	10

本文使用开源工具cpulimit[8]限制用户程序的某个进程的CPU使用率,从而模拟CPU速度异常变慢的故障。在程序运行后随机选择一个节点将SP程序的进程CPU使用率设置在90%、70%、50%,各运行了10次。实验结果如表6所示。

表6 CPU故障实验结果

Tab. 6 CPU slow down experiment results

CPU限制	程序运行结果	实验次数	FDT	FIT
90%	正常(371.0 s)	10	0	0
70%	超时	10	10	10
50%	超时	10	10	10

在表5和表6的实验中,总体的故障识别准确率是66.7%。通过模拟不同程度的潜在硬件故障,发现不同程度的潜在硬件故障对程序的影响结果不同。只有当故障严重程度超过一定程序时,导致作业运行超时,本系统才能探测出硬件故障。如果故障只是轻微或者瞬间发生的故障,系统很有可能探测不出来。

此外,在网络丢包实验中发现,设置的程序最大运行时间会影响故障识别的准确率。如果将实验程序SP. C. 4的最大运行时间设置为20 min,当丢包率为5%时,此时10次实验程序的平均运行时间为15 min 43 s,程序均运行成功,将检测不到故障。用户对其程序的运行时间估计得越准确,并行程序故障原因识别系统的准确率越高。

3.3 运行开销测试

在运行正常的情况下,对实验使用的5个NPB基准程序分别运行20次,其中10次为在Slurm的sbatch命令进行提交,另外10次为在扩展后的Slurm中使用mybatch命令进行提交。当MPI并行进程数为2和4时,各基准程序的平均运行时间如图4和图5所示。

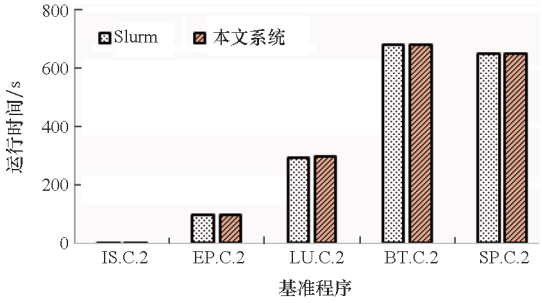

图4 进程数为2时基准程序平均运行时间

Fig. 4 Average running time of benchmark program when the number of processes is 2

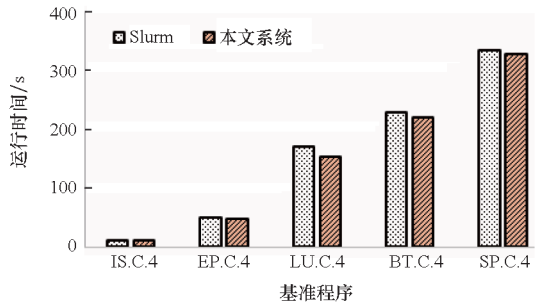

图5 进程数为4时基准程序平均运行时间

Fig. 5 Average running time of benchmark program when the number of processes is 4

从图4和图5的实验结果中可以看出,在程序不出现故障的情况下,系统运行开销保持稳定在-9.1%到3.8%之间变化,并且不会随着基准程序的类型和进程数发生变化。这是因为在程序正常运行期间,故障检测模块主要开销在于定时对作业状态和节点状态进行轮询。轮询主要依靠slurmctld和slurmd之间的通信,不会影响作业在计算节点上的运行。这意味着并行程序故障原因识别系统的开销很小,不被程序类型和程序运行规模所影响,扩展性好。

4　相关工作

虽然已经开发了许多方法和工具,但调试大型并行程序仍然是一个艰巨的问题。交互性调试(例如 TotalView[9]、Arm DDT[10])是一种传统方法,通过允许在一个窗口中的许多进程和线程同时进行调试,可以在一个单独的线程内或任意进程或线程的组合内对代码逐行执行运行、步进和终止,来完成对程序执行的完全控制。但是这种方法不能自动检测程序故障。此外,在运行大规模程序时,人工分析数以万计的进程数据耗时耗力。一些调试工具在执行期间收集各种程序状态,并将其提供给程序员以进行后期分析。例如,堆栈跟踪分析工具(STAT[11])从并行程序的所有进程中收集堆栈跟踪。通过这些跟踪,程序员可以生成调用图前缀树,以前缀树的形式汇编应用程序的行为。但是,STAT 不适合普通用户进行调试,因为 STAT 依赖于许多软件,这些软件需要系统管理员预先安装并且超出普通用户的权限;STAT 无法自动识别程序故障,需要人工干预,调试运行时间较长;STAT 没有考虑程序调试期间硬件故障的影响,因此无法在发生故障时区分硬件/软件故障和程序错误。

为了简化大规模执行下并行程序确定性重放的复杂性,Guo 等[12]设计实现了一种使用两个节点重新运行大规模 MPI 并行程序的系统,大大减少了重放所需的节点数。并行程序的一个进程在计算节点上运行时,需要与其他进程通信,但是它与之通信的进程可能没有在运行。为了建立一个使重新运行的进程正常执行的环境,需要在计算节点上模拟 MPI 通信环境,这需要比较长的时间来完成。Zhai 等[13]根据并行程序中许多进程具有相似的计算模式这一特点,首先使用层次聚类方法对运行程序的进程进行聚类,然后从每个进程聚簇中选择一个具有代表性的进程进行重放。这种“代表性”重放的方法减少了重放的进程数量,从而显著减少重放所需的时间。但是这种方法只能用来预测 HPC 的性能,不能用于故障检测。同时他们的方法都忽略了大规模并行程序调试中硬件故障的影响。本文方法可以自动识别程序故障,并进一步区分硬件故障和程序错误故障,以进行更好的调试。

目前针对高性能计算系统故障的研究热点是基于系统日志和传感器数据的软件/硬件故障检测。基于特殊领域的高性能计算机群运行的作业具有相对稳定的运行行为,Wu 等[14]提出了一种基于异常活动分析的高性能系统运行时失效检测方法,但这种方法只能针对具体领域内的重复性作业,具有很强的局限性;Gabel 等[7]相同的时间下在多台配置相同的机器上执行相同的任务,通过传感器获取机器的运行性能,如果一台机器的性能与其他机器显著不同,就会被标记为可疑,即怀疑该节点存在潜在故障;Ghiasvand 等[15]定时采集系统日志,得到系统日志生成频率,如果某个节点日志生成频率与大多数节点的偏差超过一定阈值,则认为该节点出现了故障。由于传感器数据与系统日志有固定的格式和规律,是可以根据关联规则进行挖掘的文本,因此机器学习也是用于高性能计算机群故障探测主要方法之一[16-17]。但是很多时候作业在 HPC 上运行时,运行速度与用户预期相差较大;同时因为很多作业是首次提交,没有作业运行的数据记录,不能依靠系统日志和传感器数据来检测故障。本文的系统不依赖于系统日志和传感器数据,根据用户程序与验证程序的运行结果可以区分程序运行错误与节点的硬软件故障。

5　结论

并行程序自身存在的编程 bug 会导致程序运行出错,另外,高性能计算系统硬软件故障概率的增加导致并行程序在运行时出错的概率随之增加。本文从普通权限的编程人员/用户角度,提出一种在 HPC 上程序调试失效时识别故障原因的思路。通过对作业管理系统 Slurm 进行扩展,监控作业和节点状态,指定节点和排除节点重提交程序。根据程序多次运行结果,识别故障原因的类别,从而提高程序员在 HPC 上识别编程错误或系统硬软件故障的效率。实验表明,该系统具有较低的开销和较高的准确率。

参考文献(References)

[1] RIKEN Center for Computational Science. Fugaku project [EB/OL]. [2020-10-10]. https://www.r-ccs.riken.jp/en/fugaku/research/covid-19/projects/.

[2] 国家超级计算无锡中心. 硬件资源[EB/OL]. [2020-10-10]. http://www.nsccwx.cn/swsource/5d2fe23624364f0351459262. National Supercomputing Center in Wuxi. Hardware resources [EB/OL]. [2020-10-10]. http://www.nsccwx.cn/swsource/5d2fe23624364f0351459262. (in Chinese)

[3] SchedMD. Slurm workload manager[EB/OL]. [2020-07-20]. https://slurm.schedmd.com/overview.html.

[4] OpenPBS. PBS professional open source project[EB/OL]. [2020-10-10]. https://www.openpbs.org/.

[5] NASA Advanced Supercomputing. NAS parallel benchmarks

[EB/OL]. [2020 - 10 - 10]. https://www.nas.nasa.gov/publications/npb.html.

[6] WANG Y Q, ZHANG Q, LIU Y, et al. HPC-SFI: system-level fault injection for high performance computing systems[C]//Proceedings of IFIP International Conference on Network and Parallel Computing, 2018: 103 - 113.

[7] GABEL M, SCHUSTER A, BACHRACH R G, et al. Latent fault detection in large scale services[C]//Proceedings of 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2012: 1 - 12.

[8] MARLETTA A. CPU usage limiter for Linux[EB/OL]. [2020 - 11 - 10]. https://cpulimit.sourceforge.net/.

[9] TotalView. TotalView HPC debugging software[EB/OL]. [2020 - 10 - 10]. https://totalview.io/products/totalview.

[10] Arm. Arm DDT[EB/OL]. [2020 - 10 - 10]. https://www.arm.com/products/development-tools/server-and-hpc/forge/ddt.

[11] ARNOLD D C, AHN D H, DE SUPINSKI B R, et al. Stack trace analysis for large scale debugging[C]//Proceedings of IEEE International Parallel and Distributed Processing Symposium, 2007: 1 - 10.

[12] GUO Y Y, LIN F, LIU Y, et al. Re-running large-scale parallel programs using two nodes[C]//Proceedings of IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, 2018: 485 - 492.

[13] ZHAI J D, CHEN W G, ZHENG W M. PHANTOM: predicting performance of parallel applications on large-scale parallel machines using a single node[C]//Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2010: 305 - 314.

[14] WU L P, LUO H B, ZHAN J F, et al. A runtime fault detection method for HPC cluster[C]//Proceedings of 12th International Conference on Parallel and Distributed Computing, Applications and Technologies, 2011: 68 - 72.

[15] GHIASVAND S, CIORBA F M. Anomaly detection in high performance computers: a vicinity perspective[C]//Proceedings of 18th International Symposium on Parallel and Distributed Computing, 2019: 112 - 120.

[16] DANI M C, DOREAU H, ALT S. *K*-means application for anomaly detection and log classification in HPC[C]//Advances in Artificial Intelligence: From Theory to Practice, 2017: 201 - 210.

[17] BORGHESI A, LIBRI A, BENINI L, et al. Online anomaly detection in HPC systems[C]//Proceedings of IEEE International Conference on Artificial Intelligence Circuits and Systems, 2019: 229 - 233.