◎研发、设计、测试◎

基于SANs模型的一种并行I/O系统的可用性评估

郑 霄[1,2],李宏亮[2],郑 方[2],郑 翔[2],陈左宁[2]
ZHENG Xiao[1,2],LI Hong-liang[2],ZHENG Fang[2],ZHENG Xiang[2],CHEN Zuo-ning[2]

1.解放军信息工程大学,郑州 450002
2.江南计算技术研究所,江苏 无锡 214083
1.PLA Information and Engineering University,Zhengzhou 450002,China
2.Jiangnan Institute of Computing Technology,Wuxi,Jiangsu 214083,China
E-mail:uu88zheng@126.com

ZHENG Xiao,LI Hong-liang,ZHENG Fang,et al.Availability evaluation based on SANs model for a parallel I/O system. Computer Engineering and Applications,2008,44(19):67-71.

Abstract: The parallel I/O system is one of important components of a HPC system,and it can discount the performance of the whole system if its availability are awful.However,high-throughput and large capacity are still the main target in parallel I/O system's design by far,and few research on its availability evaluation has been published.In this paper,a method managing to evaluate a parallel I/O system's availability based on SANs (Stochastic Activity Networks) model with Mobius modeling environment is expatiated,and the simulation results reveal how the availability of the parallel I/O system is impacted on by the number of global file system,the number of minimum number of OST in a single file system,and the repair period of the whole system.

Key words: Stochastic Activity Networks(SANs);Mobius;parallel I/O system;availability evaluation

摘 要:并行I/O系统是高性能计算机系统的一个重要组成部分,其可用性水平对整机系统性能的发挥具有重要作用。采用SANs(Stochastic Activity Networks,随机行为网)模型及其支持工具Mobius,对一种大规模并行I/O系统建立可用性评估模型,并采用模拟方法进行解析。模拟结果反映了全局文件系统数量、单一文件系统内最小可用OST(Object Storage Target,对象存储目标)数量和系统维修时间等参数的变化对全系统可用度的影响,对于大规模并行I/O系统的设计与维护具有积极的参考价值。

关键词:随机行为网;Mobius;并行I/O系统;可用性评估
DOI:10.3778/j.issn.1002-8331.2008.19.020 **文章编号**:1002-8331(2008)19-0067-05 **文献标识码**:A **中图分类号**:TP302

1 引言

随着高性能计算机系统的迅猛发展,在峰值速度即将突破P级(千万亿次)的同时,系统的可靠性与可用性已经成为制约其发展的重要的瓶颈。并行I/O系统作为高性能计算机系统的一个组成部分,其可用性水平对整机系统的性能发挥有着重要影响。

目前并行I/O系统的设计重点仍是高吞吐率和大容量,而对它的可用性评估研究则较为少见。王红艳等曾在文献[1]中介绍了一种并行I/O系统,并采用基本的串并联逻辑结构建立了它的可用性评估模型。本文尝试使用Mobius工具为该系统建立基于SANs的可用性评估模型,并采用模拟方法进行解析,以研究相关参数在一定范围内的调整对该系统的可用性变化的影响。模拟的结果反映了全局文件系统的数量变化、单一文件系统内可用的最小OST(Object Storage Target,对象存储目标)数量的变化和系统维修时间的变化对全系统可用度的影响。该研究对并行I/O系统的设计与维护起到一定的指导作用。

2 SANs模型与Mobius工具

2.1 SANs模型

系统的可靠性与可用性评估方法有多种,传统的有基于串并联逻辑结构的组合模型,包括RBD(可靠性框图)、FTA(故障树分析)等,其特点是要求事件具有独立性,从而易于简化模型

基金项目:国家高技术研究发展计划(863)(the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z117);博士后基金(20060390963)。

作者简介:郑霄(1976-),男,博士研究生,工程师,主研方向为并行计算机系统高可用;李宏亮(1975-),男,博士,副研究员,主研方向为体系结构、高性能计算、容错计算等;郑方(1984-),男,研究实习员,主研方向为并行计算机系统高可用;郑翔(1976-),男,工程师,主研方向为海量数据存储和管理的研究;陈左宁(1957-),女,院士,博士生导师,主研方向为并行计算机体系结构和分布式操作系统。

收稿日期:2008-03-06 **修回日期**:2008-04-16

的描述和求解，适于精度要求不高的系统可靠性与可用性评估；随着系统复杂度的增长和用户对评估精度要求的提高，能够反映出系统内部复杂关联的状态空间模型被越来越多地用于可用性评估，包括马尔可夫模型和 Petri 网系列模型等。其中，Petri 网系列模型具有比马尔可夫模型更高抽象级别的描述能力，不仅大大简化了建模过程，而且部分解决了马尔可夫模型建立与求解过程中的状态空间爆炸问题。在 Petri 网系列模型中，随机行为网（Stochastic Activity Networks，SANs）继承了随机 Petri 网（Stochastic Petri Nets，SPNs）对复杂系统行为的并发、异步、分布和时延等特性的描述能力，同时通过基本元素的扩展，增强了对行为发生的条件、功能与分支情况等方面的描述能力。在相关工具的支持下，SANs 常被用于复杂计算机系统与网络的性能与可信性评估。SANs 一般采用图形方式描述，模型中各个基本元素的符号与含义如表 1 所示。除此之外，作为系统可用性建模的一种标准方法，SANs 还有严格的形式化定义，详情参见文献[2]。

表 1 SAN 模型的基本元素的符号与含义[3]

名称			符号	功能含义
中文		英文		
库所		Place		可能的系统局部状态（条件或状况）。与传统 Petri 网相同，其值或状态由令牌（token）数量表示。所有库所某同一时刻的令牌数分布情况决定此刻系统的状态
行为	瞬时行为	Immediate Activity		立即引起系统状态发生改变的事件，其完成时间很短、可以忽略
	时延行为	Timed Activity		其发生或者引发系统状态改变存在一定时延的事件，其完成时间可以为常数，也可以为某种概率分布，如指数分布等
弧		Arc	→	由库所指行为向时为输入弧，表示该库所状态是使该行为进入使能状态的一个先决条件 由行为指向库所时为输出弧，表示该行为的发生会对该库所的状态产生影响
分支选择		Case		依附在行为之上，表示该行为完成后会有两种以上的可能情况发生。这些情况互斥，每次只会有其中的一种会发生
输入门		Input Gate	▷	放在输入弧上，用来控制行为执行的使能谓词（enabling predicate，即使能条件也可以包括执行时要完成的功能）
输出门		Output Gate	◁	放在输出弧上，用来控制行为执行完成时要完成的功能

2.2 Mobius 工具

Mobius 是美国伊利诺斯大学开发的一种集成化建模环境[4]，它采取了框架式设计结构，能够兼容多种模型描述方法和多种模型求解方法，为建立复杂系统的 SANs 评估模型提供了良好的支撑。

运用 Mobius 进行系统评估时，一般需要建立四个不同层次的模型：原子模型、组合模型、回报模型和求解模型，如图 1 所示[5,6]。

（1）需要为系统的部件或子系统建立原子模型。原子模型是 Mobius 的基本模型，由状态变量和行为组成。状态变量是用来保存模型状态信息的基本实体，而行为则是引起模型状态发生改变的机制。目前 Mobius 的原子模型所支持的形式化描述语言包括随机行为网、排队网络、故障树分析和随机过程代数等。

（2）若建模对象是一个复杂系统，或待评估指标无法根据单一模型求出，那么就需要构造组合模型。Mobius 提供了两种组合机制：复制（Replicate）和联合（Join），前者将一个模型复制成若干份，所有模型备份间通过一个或多个状态变量来共享，从而形成模型间的关联组合后者通过一个或多个状态变量的共享来连接若干个不同类型的模型。

（3）在原子或组合模型的基础上增加一些回报变量，从而使组合模型变成一种新的、指明了用户所关心的评估指标模型，即回报模型。回报变量有两种，一种是“速率”回报，用于累积模型处于某种状态的概率，另一种是“脉冲”回报，用于累计某个行为的完成次数。回报变量本身也有状态，它的值随着模型中它所定义的行为的发生而改变。

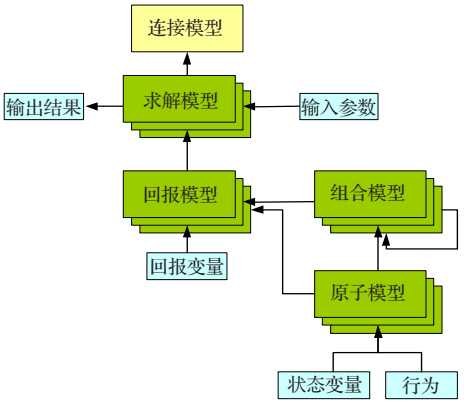


图 1 Mobius 的模型层次

（4）建立求解模型。Mobius 支持数值解析和模拟两种求解方法，用户可根据需要选择适当的方法，然后输入必要的模型参数，形成可实际执行的求解模型。根据所选的方法、输入的参数以及模型的精度要求，求解模型执行后输出的结果可以是精确的、近似的或者是统计的。结果可以是任何回报变量的平均值、变量值、或者概率函数，甚至可以是通过求解某个子模型所构建的更加抽象的模型。当然，这一结果可能就是用户最终所关心的评估指标，也可能只是一个用于下一步计算的中间结果。若是后者，那么还需要将多个求解模型连接一个更大的模型，即连接模型（connected model）。连接模型间通过共享由子模型的输出结果进行交互。与组合模型一样，连接模型支持构建层次化模型，但方式更为松散。

3 并行 I/O 系统的可用性评估

3.1 系统组成

本文要评估的并行 I/O 系统[1]是一种高性能计算机系统的子系统之一，其可用性水平对整机系统的性能发挥有着重要的影响。该系统主要由 n 个 Lustre 全局文件系统[8]组成；每个全局文件系统内含有 $2(m+1)$ 个 I/O 节点，分成 $(m+1)$ 个 I/O 节点对；其中一个节点对由两个元数据服务器（MDS）节点组成，用于存放文件的 inode 信息，其余节点对的节点用作文件的对象存储服务器（OSS）。另外，每个 I/O 节点对内的两个 IO 节点之间通过心跳线互相监测运行状态，且所有 I/O 节点同时与两套网络互连。图 2 给出了系统的结构框架示意图。

每个 Lustre 文件系统内，MDS 节点对采用 active-standby failover 模式与一个共享盘阵互连，组成 MDS 单元，如图 3（a）所示。正常情况下，两个MDS 都启动，其中一个工作，另一个等

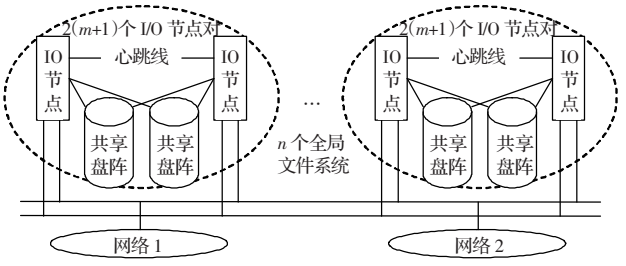
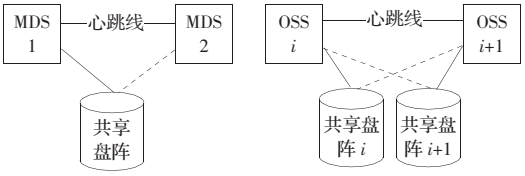


图 2　并行 I/O 系统的结构框架

待;当工作 MDS 失效时,另一个自动接管失效 MDS 的工作。因此,MDS 单元可用的条件是它的盘阵和不少于一个 MDS 正常工作。



(a)MDS 单元　(b)一个 OST 单元

图 3　MDS 单元及一个 OST 单元

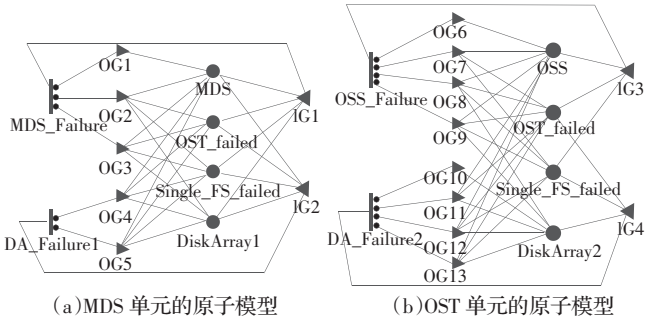
OSS 节点对采用 active-active failover 模式与两个共享盘阵互连,组成一个对象存储目标(OST)单元,如图 3(b)所示。正常情况下,两个 OSS 同时工作,但每个 OSS 仅与其中的一个盘阵交互,如图 3(b)的实线所示;当其中的一个 OSS 失效时,另一个才会接管失效 OSS 的盘阵,如图 3(b)的虚线所示。

3.2　可用性建模

并行 I/O 系统可分为两个层次:对全系统而言,只要多个全局文件系统中至少有一个工作正常,则该并行 I/O 系统可用;对单一的全局文件系统而言,由于采用了某种数据高可用技术,其 MDS 单元可用,并且不少于一定数量的 OST 单元可用,该文件系统才可用。在可用性建模中,首先建立两个基本单元的 SANs 原子模型,再利用 Mobius 工具的复制与联合机制,由原子模型构造出全系统的可用性模型。在此基础上,根据需要指定相应的回报变量,然后给出必要的初始参数,采用模拟方法进行解析。

3.2.1　原子模型

图 4 给出了 MDS 单元和 OST 单元的 SAN 原子模型。



(a)MDS 单元的原子模型 MDS_module　(b)OST 单元的原子模型 OST_module

图 4　MDS 单元和 OST 单元的 SAN 原子模型

在 MDS_module 模型中,库所 MDS 和 DiskArray1 是局部变量,分别代表一个 MDS 单元中可用的 MDS 和盘阵数量,初始令牌数分别为 2 和 1;库所 OST_failed 和 Single_FS_failed 是共享变量,分别代表本 MDS 单元所在文件系统内失效的 OST 单元数和全系统中失效的全局文件系统数量,初始令牌数均为 0。行为 DA_Failure1 表示 MDS 单元内的盘阵失效,由于该盘阵唯一,所以必然会导致 MDS 单元失效;而 MDS 单元在全局文件系统中必不可少,所以其后果只会是下列 2 种情况之一:(1)MDS 所在的全局文件系统失效,但因为还有其它文件系统可用,所以该并行 I/O 系统仍可用;(2)MDS 所在的全局文件系统失效,同时由于此前其它文件系统均已失效,该并行 I/O 系统失效。

行为 MDS_Failure 表示发生 MDS 节点失效事件,其处理可分为:(1)失效前若两个 MDS 都正常,则不影响本文件系统的使用,否则转步骤(2)或步骤(3);(2)MDS 单元失效,导致所在的文件系统失效,但因为还有其它文件系统可用,所以该并行 I/O 系统仍可用;(3)MDS 单元失效,导致所在的文件系统失效,同时由于此前其它文件系统均已失效,因此该并行 I/O 系统失效。

在 OST_module 模型中,库所 OSS 和 DiskArray2 是局部变量,分别代表一个 OST 单元内可用的 OSS 节点和盘阵数量,初始令牌数均为 2;库所 OST_failed 和 Single_FS_failed 是共享变量,分别代表本 OST 单元所在文件系统内失效的 OST 单元数和全系统中失效的全局文件系统数量,初始令牌数均为 0;行为 OSS_Failure 和 DA_Failure2 分别表示发生 OSS 节点和盘阵失效事件,它们的后果均有 4 种可能:(1)失效前如果两个 OSS(或盘阵)都正常,那么本 OST 单元仍可工作,否则(2)、(3)或(4);(2)本 OST 单元失效,但它所在的文件系统内总的失效 OST 单元数量不超过系统允许的阈值,所以该文件系统仍可用;(3)本 OST 单元失效,导致它所在的文件系统内总的失效 OST 单元数量超过系统允许的阈值,引起该文件系统失效,但因为还有其它文件系统可用,所以该并行 I/O 系统仍可用;(4)本 OST 单元失效,导致它所在的文件系统内总的失效 OST 单元数量超过系统允许的阈值,引起该文件系统失效,同时由于此前其它文件系统均已失效,因此该并行 I/O 系统失效。

MDS_module 和 OST_module 原子模型中各元素的属性设定如表 2 所示。

表 2　原子模型中的元素属性

库所名	初始令牌数	所属原子模型
MDS	2	MDS_module
DiskArray1	1	
OSS	2	OST_module
DiskArray2	2	
OST_failed	0	共享
Single_FS_failed	0	

与图 4 中各行为相关的设置与代码如下:

行为 1　MDS_Failure

失效率:Node_failure_rate*MDS->Mark()

输入门:

IG1:(MDS->Mark()>0)&&(DiskArray1->Mark()>0)&&(Single_FS_failed->Mark()<Num_FS)&&(OST_failed->Mark()<Num_OST-Min_Num_OST+1)

分支情况:

case1:if (MDS->Mark()==2) return(1.0); else return(0.0);

case2:if (MDS->Mark()==1 && Single_FS_failed->Mark()<Num_FS-1)　return(1.0); else return(0.0);

case3:if (MDS->Mark()==1 && Single_FS_failed->Mark()==Num_FS-1) return(1.0); else return(0.0);

输出门:

OG1:MDS->Mark()--;

OG2:MDS->Mark()=0;DiskArray1->Mark()=0;OST_failed->Mark()=Num_OST;Single_FS_failed->Mark()++;

OG3:MDS->Mark()=0;DiskArray1->Mark()=0;OST_failed->Mark()=Num_OST;Single_FS_failed->Mark()=Num_FS;

行为 2 DA_Failure1

失效率:DiskArray_failure_rate*DiskArray1->Mark()

输入门:

IG2:(MDS->Mark()>0) && (DiskArray1->Mark()>0) && (OST_failed->Mark()<Num_OST-Min_Num_OST+1) && (Single_FS_failed->Mark()<Num_FS);

分支情况:

case 1:if(Single_FS_failed->Mark()<Num_FS-1)return(1.0); else return(0.0);

case 2:if(Single_FS_failed->Mark()==Num_FS-1)return(0.0); else return(1.0);

输出门:

OG4:MDS->Mark()=0;DiskArray1->Mark()=0;OST_failed->Mark()=Num_OST;Single_FS_failed->Mark()++;

OG5:MDS->Mark()=0;DiskArray1->Mark()=0;OST_failed->Mark()=Num_OST;Single_FS_failed->Mark()=Num_FS;

行为 3 OSS_failure

失效率:Node_failure_rate*OSS->Mark()

输入门:

IG3:(OSS->Mark()>0) && (DiskArray2->Mark()>0)&& (OST_failed->Mark()<Num_OST-Min_Num_OST+1)&& (Single_FS_failed->Mark()<Num_FS);

分支情况:

case1:if (OSS->Mark()==2) return(1.0); else return(0.0);

case2:if((OSS->Mark()<2)&& (OST_failed->Mark()<Num_OST - Min_Num_OST)) return(1.0); else return(0.0);

case3:if (OSS->Mark ()<2 && OST_failed->Mark ()==Num_OST -Min_Num_OST && Single_FS_failed ->Mark ()<Num_FS-1) return(1.0);else return(0.0);

case4:if((OSS->Mark()<2) && (OST_failed->Mark()==Num_OST-Min_Num_OST)&&

(Single_FS_failed->Mark()==Num_FS-1)) return(1.0); else return(0.0);

输出门:

OG6:OSS->Mark()--;

OG7:OSS->Mark()=0;DiskArray2->Mark()=0;OST_failed->Mark()++;

OG8:OSS->Mark()=0;DiskArray2->Mark()=0;OST_failed->Mark()=Num_OST; Single_FS_failed->Mark()++;

OG9:OSS->Mark()=0;DiskArray2->Mark()=0;OST_failed->Mark()=Num_OST;Single_FS_failed->Mark()=Num_FS;

行为 4 DA_Failure2

失效率:DiskArray_failure_rate*DiskArray2->Mark()

输入门:

IG4:(OSS->Mark()>0)&& (DiskArray2->Mark()>0)&& (OST_failed->Mark()<Num_OST-Min_Num_OST+1)&& (Single_FS_failed->Mark()<Num_FS)

分支情况:

case1:if (DiskArray2->Mark()==2) return(1.0); else return(0.0);

case2:if(DiskArray2->Mark()<2 && OST_failed->Mark()<Num_OST-Min_Num_OST) return(1.0); else return(0.0);

case3:if(DiskArray2->Mark()<2 && OST_failed->Mark()==Num_OST -Min_Num_OST && Single_FS_failed ->Mark ()<Num_FS-1) return(1.0); else return(0.0);

case4:if (DiskArray2->Mark()<2 && OST_failed->Mark()==Num_OST-Min_Num_OST && Single_FS_failed->Mark()==Num_FS-1) return(1.0); else return(0.0);

输出门:

OG10:DiskArray2->Mark()--;

OG11:OSS->Mark()=0;DiskArray2->Mark()=0;OST_failed->Mark()++;

OG12:OSS->Mark()=0;DiskArray2->Mark()=0;OST_failed->Mark()=Num_OST;Single_FS_failed->Mark()++;

OG13:OSS->Mark()=0;DiskArray2->Mark()=0;OST_failed->Mark()=Num_OST;Single_FS_failed->Mark()=Num_FS;

3.2.2 组合模型

在上述两个原子模型的基础上，建立全系统的组合模型，如图 5 所示。其中,Rep1 表示将原子模型 OST_module 复制 Num_OST 份，这些模型复制品共享库所 OST_failed 与 Single_FS_failed;Rep1 与 MDS_module 模块联合在一起,共同构成一个全局文件系统，并保持对库所 OST_failed 与 Single_FS_failed 的共享;最后,Rep2 将 Join 复制 Num_FS 份,构成全系统的完整模型;在 Rep2 一级,库所 Single_FS_failed 保持共享。

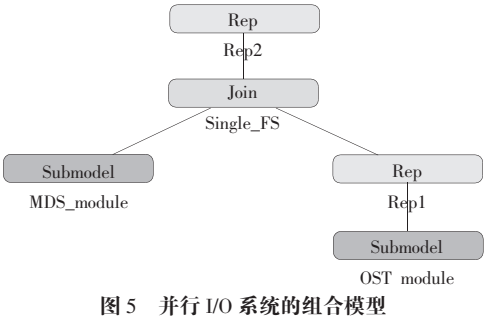


图 5 并行 I/O 系统的组合模型

3.2.3 回报变量与模型解析

该模型的回报变量设为不可用性(unavailability),并与模型中失效的文件系统数量相关联,回报函数设定如下:

if(MDS_module->Single_FS_failed->Mark()==Num_FS)

return(1.0/Num_FS);

在模型解析过程中,当失效文件系统数量等于全局文件系统总数时,模型返回回报变量值 $r_i(i=1,2,\cdots)$;r_i 的累积值即为系统的总失效时间长度。对于模拟解析方法而言,假设模拟系统运行的时间长度为 T(本实验将它设为系统的一个维修周期),则系统的可用度可通过以下公式来反映:

$$A_{sys}=1-\sum_{i=1}^{k} r_i/T, k \in N$$

本模型的模拟解析所采用的相关参数如表 3 所示。其中，单一全局文件系统内的 OST 实际数量为 100；通过采取适当的 OST 冗余策略，全局文件系统允许有少量 OST 失效，因此最小 OST 单元数量可变，从 95 到 100，即允许 OST 失效的数量在 0 到 5 之间；另外，为了计算通过一定数量的全局文件系统备份对并行 I/O 系统的可用性提高效果，全局文件系统的数量在 2 到 6 之间进行变化。

表 3　模型的相关参数列表

参数符号	参数值	说明
DiskArray_failure_rate	1.0E-6	盘阵失效率，按 MTTF 为 100 万小时估算
Node_failure_rate	1.0E-5	节点(服务器)失效率，按 MTTF 为 10 万小时估算
Num_FS	2~6	全局文件系统数量
Num_OST	100	单个文件系统内的 OST 单元数量
Min_Num_OST	95~100	单个文件系统可用所需的 OST 单元最小数量

根据上述模型，在处理器为 Pentium 3.0 GHz 的计算机上分别模拟了系统的维修周期为 1 年(8 760 小时)和半年(4 380 小时)情况下系统的失效时间与可用度，模拟结果如表 4 和图 6 所示。

表 4　系统在 1 年内和半年内的不可用时间随文件系统数量与最小 OST 数量变化的情况

1 年内不可用时间/h		单一文件系统可用的最小 OST 数量					
		100	99	98	97	96	95
全局文件系统数量	2	6 044.53	2 824.69	1 056.33	319.35	77.49	34.16
	3	5 149.67	1 826.35	430.47	82.38	14.01	1.79
	4	4 364.35	1 224.65	225.05	23.70	2.84	0.00
	5	3 747.15	837.33	106.15	8.83	0.00	0.00
	6	3 307.75	595.93	60.40	2.13	0.00	0.00
半年内不可用时间/h		单一文件系统可用的最小 OST 数量					
		100	99	98	97	96	95
全局文件系统数量	2	2 414.58	753.29	164.82	28.46	7.80	0.00
	3	1 823.65	334.14	33.26	2.72	0.00	0.00
	4	1 379.52	152.25	7.33	0.00	0.00	0.00
	5	1 036.52	74.87	1.80	0.00	0.00	0.00
	6	803.35	34.30	0.09	0.00	0.00	0.00

由表 4 和图 6 可以看出：

(1)在全局文件系统数量固定的情况下，系统的失效时间基本上随着单一系统内允许的失效 OST 数量增加而加速下降。例如，当全局文件系统数为 3 时(评估时间为 1 年)，若不允许 OST 失效(即要求最小 OST 数量为 100)，则系统失效时间为 5 149.67 小时；若允许 1 个 OST 失效，则系统失效时间降到 1 826.35，降幅比约为 64.5%；随着最小 OST 数量从 99 逐次降到 95，系统失效时间的降幅比依次约为 76.4%、80.9%、83% 和 87.2%。

(2)适当增加全局文件系统的数量可以降低系统的失效时间。当系统内允许的失效 OST 数量越小，全局文件系统的数量对系统的失效时间的影响越明显。如评估时间为 1 年时，当最小 OST 数量依次等于 100、99、98、97、96 和 95 时，随着全局文件系统从 2 逐次增加到 6，系统失效时间的降幅比依次约为 15%、30%、50%、72%、80%和 95%。

(3)缩短系统的维护周期(即评估时间)可以明显地改善系统的可用度。从本实验来看，在全局文件系统数量和单一文件系统内的最小可用 OST 数量不变的情况下，当评估时间由 1 年缩短到半年时，系统的可用度都会有不同程度的提高。例如，当单一文件系统内可用的最小 OST 数量为 100、全局文件系统数量为 2 时，若评估时间从 1 年缩短到半年，则系统的可用度会从原来的 30.968%提升到 44.873%，增幅为 13.9%，增幅比接近于 45%。

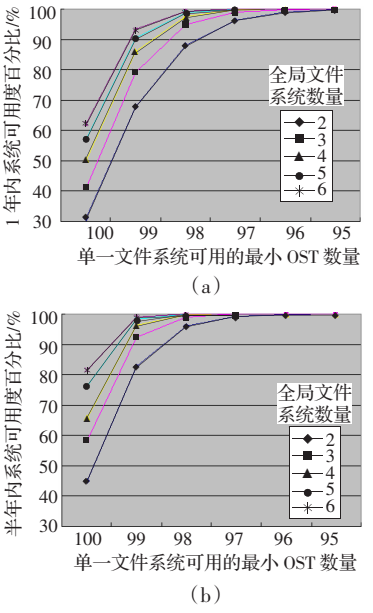


图 6　1 年内与半年内系统的可用度随文件系统数量与最小 OST 数量变化的情况

4　结束语

本文对一种大规模并行 I/O 系统建立了基于 SANs 模型的可用性评估模型，并且通过 Mobius 进行求解。结果显示：(1)适当增加全局文件系统的数量或单一文件系统内可用的最小 OST(Object Storage Target，对象存储目标)数量均可显著提高全系统的可用度；(2)当全局文件系统不小于 2、单一文件系统内允许失效的 OST 数量不少于 3 时，可保证全系统在半年时间内的可用度在 99.9%以上；(3)考虑到增加全局文件系统的成本和减少单一文件系统可用所需的最小 OST 数量的技术难度，适当的减少维修周期(如半年甚至更短)，也可保证系统的可用度。该项研究将为并行 I/O 系统的设计与维护提供较好的参考价值。

致谢　本文所使用的建模工具 Möbius，是向美国伊利诺斯大学 William H. Sander 教授领导的 PERFORM 研究小组申请的、用于学术研究目的的免费试用版本(2.1.3 版)，在此对他们表示感谢。

参考文献：

[1] 王红艳，朱建涛，郑翔.一个并行 I/O 系统的可用性评估模型[J].高性能计算技术，2004，6：30-34.

[2] Sanders W H，Meyer J F.Stochastic activity networks：formal definitions and concepts[C]//LNCS 2090.Berlin：Springer，2001：315-343.

[3] Sanders W H.Module 4：stochastic acitivity networks，ECE/CS 541 computer system analysis[R].2005.

(下转 83 页)