云计算环境下基于失效规则的资源动态提供策略

田冠华[1),2)] 孟 丹[1)] 詹剑锋[1)]
1) (中国科学院计算技术研究所 北京 100190)
2) (中国科学院研究生院 北京 100049)

摘 要 云计算是一个热点研究领域.研究人员提出多种资源共享和资源动态配置策略.然而,很少有工作关注动态提供的资源的可靠性问题.该文提出云计算平台异构服务整合环境下基于失效规律的节点资源动态提供策略.该文的策略通过综合考虑资源需求和资源失效在时间和空间上的规律,保证动态提供的节点资源的可靠性.该文设计实现了一个整合异构负载的云计算模拟器平台和系统资源的多维度失效模型框架,来验证文中提出的策略.该文的云计算模拟器通过模拟异构负载对资源的使用和失效规律,来验证资源动态提供策略的性能.该文基于模拟器平台,使用真实的异构负载评价所提出的策略.结果表明,与 baseline 策略相比,该文提出的策略可以有效提高动态提供的节点资源的可靠性,屏蔽掉大量节点资源的失效,同时对资源使用效率和服务性能不引入负面影响.该文提出的策略对资源失效非一致性分布的情况也有较好的屏蔽能力.针对资源失效在时间空间特性上的评价,表明该文策略适用于云计算环境.此外,该文策略不涉及对系统平台的任何修改或侵入式监测,该文提出的策略有很好的应用前景.

关键词 资源动态提供;可靠性;失效规律;云计算;异构负载
中图法分类号 TP301 **DOI** 号: 10.3724/SP.J.1016.2010.01859

Reliable Resource Provision Policy for Cloud Computing

TIAN Guan-Hua[1),2)] MENG Dan[1)] ZHAN Jian-Feng[1)]
1) (*Institute of Computing Technology, Chinese Academy of Science, Beijing* 100190)
2) (*Graduate University of Chinese Academy of Sciences, Beijing* 100049)

Abstract Cloud computing has become a hot topic, researchers proposed various resource sharing technique and resource provision technique. However, very limited literatures pay attentions to the reliability of dynamically provided resources. This paper proposes failure rules aware node resource provision policies for heterogeneous services consolidated in cloud computing infrastructure, and evaluates the proposed policy with simulation approach, i.e., implements a simulator of heterogeneous service consolidation platform, which take characteristics of heterogeneous services (both characteristics of resource utility and failure rules), into consideration, and uses two production traces to synthesize inputs. In order to evaluate wide ranges of failure rules, this paper proposes a multi-dimension failure modeling framework, i.e., adapt various factors about failure distribution involving temporal and spatial factors to study the proposed policy's capability. The results of evaluation indicate that the proposed resource provision policy is effective for providing robust nodes for heterogeneous services, i.e., the policy can mask more potential node

收稿日期:2010-08-22. 本课题得到国家自然科学基金(60703020,60933003)、国家"八六三"高技术研究发展计划项目基金(2006AA01A102,2009AA01A129,2009AA01Z139)资助.田冠华,男,1980 年生,博士研究生,主要研究方向为集群系统服务质量. E-mail: tianguanhua@ncic.ac.cn. 孟 丹,男,1965 年生,研究员,博士生导师,主要研究领域为高性能计算机体系结构和系统软件、高效通信协议、分布式文件系统与存储服务器、机群操作系统. 詹剑锋,男,1976 年生,博士,副研究员,主要研究方向为集群操作系统、分布式系统、高可靠性软件.

reboot failures from services and leave less chances of unplanned failures, e. g., service failure or node reboot, compared with baseline fault re-provided policy. In addition, the policy is able to mask non-uniformly distribution among resource's reliability system wide. Meanwhile, the policy involves no negative impact on service performance and on node's resource utility, compared with baseline policy. Evaluation with failure rules about temporal and spatial factors indicates that the policy is useful for could computing environment.

Keywords resource provisioning; reliability; failure rules; cloud computing; heterogeneous workload

1 引 言

云计算作为当前的一个研究热点, 主流的信息技术公司, 如 Amazon, IBM, Google, 都对此先后提出各自的云计算基础架构. 研究机构也提出各种原型系统, 如文献[1-2], 并提出了超过 20 种以上的"云"的定义[3]. Vaquero[4] 等人将之总结表述如下: A large pool of easily usable and accessible virtualized resources, which can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. 从中可以看出, 云计算领域研究的一个核心问题是如何实现计算资源的有效的动态配置和共享使用. 研究人员提出多种资源动态提供和管理方案, 如文献[5-6]. 然而, 很少有研究考虑云计算环境下动态提供的资源的可靠性问题.

实际上, 动态提供的资源的可靠性无论对云计算基础运营商, 还是对服务运营商, 都是很关键的. 据报道, Amazon Elastic Compute Cloud (EC2) 和 Amazon Simple Storage Service (S3) 的故障导致服务提供商和资源提供商严重的经济损失. 然而, 文献里很少有涉及保证云计算环境下动态提供的资源的可靠性的研究.

本文将提出基于失效规律的节点资源动态提供策略, 保证动态提供的资源的可靠性. 这种资源可靠性的提高, 不以牺牲系统整体资源使用效率和服务性能为代价, 同时不涉及对云计算平台的硬性修改或侵入式的监测.

云计算平台各种服务有着不同的资源需求模式[3-4,6], 同时动态申请的资源需求往往表现出很强的波动性[3,5]. 此外, 各种服务系统有着各自不同的失效特性[7-10]. 同时保证动态提供的资源的可靠性, 又对资源的使用效率不造成负面影响, 是很有挑战的. 为此, 本文把节点资源或虚拟节点资源看作基本单位, 考虑动态提供的节点的使用效率和可靠性问题. 本文用节点崩溃和无计划的重起失效, 衡量节点资源的可靠性状况. 有研究[8] 指出节点的无计划重起失效可以覆盖大部分失效事件. 同时把节点或虚拟节点作为基本单位[3-5], 可以屏蔽掉性能相互干扰和部件之间故障扩散等问题所引入的复杂性. 事实上, 把虚拟机节点作为基本的资源单位实现系统管理, 正是以 EC2 为代表的云计算环境下的通用作法.

云计算平台下各种服务可以大致归为两大类[2-4,6]: 数据计算密集型服务和交互密集型网络处理服务. 这样的分类可以有效地区分云计算环境下大部分的服务. 同时, 两大类服务在节点粒度上的失效规律[7-8,10-12], 对于研究保证资源的可靠性提供了契机. 本文正是通过考察异构服务的节点失效规律, 提高动态提供的资源的可靠性.

为了评价本文提出的资源提供策略, 我们实现了异构服务整合平台的模拟器和通用的失效模型框架. 我们的模拟器, 考虑高性能计算服务和网络处理服务的资源使用特点和失效规律, 模拟异构服务整合环境下资源动态提供策略. 我们的模拟器使用真实负载, 保证模拟贴近真实场景. 同时, 我们提出的失效模型框架, 可以覆盖广泛的失效模型, 方便对大规模开放的云计算环境下的可靠性的研究和评价.

实验评价结果表明: 本文提出的资源提供策略能够有效地提供可靠的节点资源, 屏蔽掉大量的资源失效. 和不考虑失效规律的 baseline 策略相比, 本文的策略能够提供更可靠的资源, 且对系统整体的资源使用和服务性能没有引入负面影响. 同时, 本文的策略, 对整个系统表现出来的非一致性的失效特性, 有一定的屏蔽能力. 此外, 本文的策略不涉及对平台的任何修改.

本文提出的策略集成在 PhoenixCloud 系统里. PhoenixCloud 是一个由 Phoenix[13] 系统演进而来的云计算基础架构的管理平台. PhoenixCloud 将被

部署在曙光 6000 平台上.

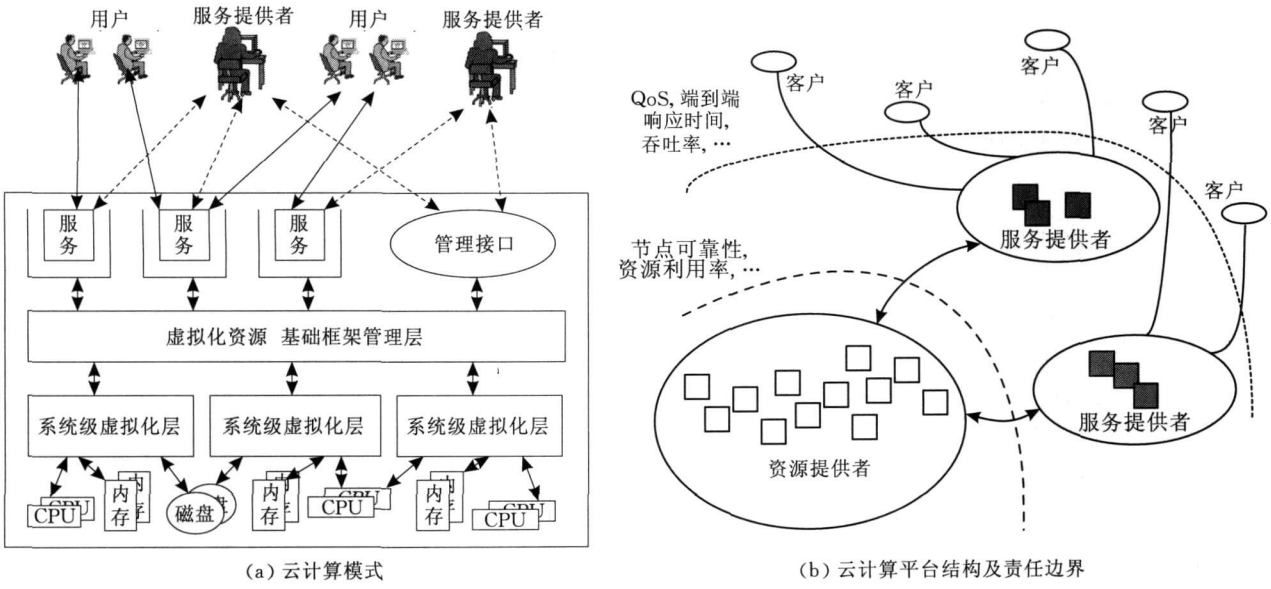
本文工作有以下几个贡献: 提出基于失效规律的节点资源动态提供策略, 通过考虑云计算环境下, 资源的使用和失效规律, 在不损失资源使用效率的前提下, 保证动态提供的资源的可靠性; 针对大规模开放云计算环境失效规律非一致的情况, 提出并讨论单队列和多队列资源提供策略的设计原则和效果评价; 提出和实现一个模拟器及通用的失效模型框架, 来评价策略的有效性, 为研究云计算环境异构负载的资源动态提供策略提供一个有效的研究平台; 对提出的资源提供策略作评价. 结果表明, 和不考虑资源失效规律的 BaseLine 策略相比, 本文的策略提高了动态提供资源的可靠性, 同时对系统整体资源使用率和服务性能没有引入负面影响. 对于失效在时间空间特性上的评价表明本文策略适用于云计算环境.

2　背　景

2.1　云计算模式

传统的服务系统里, 服务提供者往往也是资源提供者, 他们通常是把服务系统部署在独占的机群系统上, 把服务和计算资源放在一起考虑, 以保证服务质量为目标, 从资源使用率或公平性角度设计合适的资源提供策略.

在云计算模式(如图 1(a)[4]) 下, 服务的提供和资源的提供两个功能实现解耦, 如图 1(b) 所示, 服务提供者不再是资源提供者, 他们变成资源提供者的直接客户. 资源提供者需要以 On-Demand 的方式提供计算资源. 同时, 需要确保, 提供的资源是可靠的, 至少不是即将发生失效的.



(a) 云计算模式　　(b) 云计算平台结构及责任边界

图　1

动态提供的资源的可靠性, 对于资源提供者和服务提供者都很关键. 无论是服务提供者要保证提供的服务的 Quality of Services(QoS) 和 Service Level Objects(SLO), 还是资源提供者要保证动态提供的资源的有效性和使用效率, 都依赖于动态提供的资源的可靠性.

为此, 本文提出, 通过考察服务的失效规律, 保证提供的资源的可靠性. 由于云计算平台上的服务可以大致分为两大类[2-4]: 数据计算密集型的服务和交互密集型的网络处理服务, 当前我们考虑这两类服务的节点资源失效率的规律.

失效率可以如下定义[14]: 假定时刻 t 系统没有失效, 失效率是时刻 t 到时刻 $t+\Delta t$ 中的系统失效的一个条件概率. 失效率是时间 t 的一个函数, 我们把失效率函数定义为 $\lambda(t)$. 这个函数定义了一个节点从启动到时间 t 的可靠性. 失效率函数定义式如下:

$$\lambda(t)=\lim_{\Delta t\to 0}\left(\frac{P(t<T\le t+\Delta t\mid T>t)}{\Delta t}\right)=\frac{pdf(t)}{(1-cdf(t))} \tag{1}$$

其中, $pdf(t)$ 是概率密度函数, $cdf(t)$ 是概率函数.

2.2　失效规律的特点

研究人员对节点资源的系统失效规律作过大量研究[7-8, 10-12, 15]. 文献[7-8, 12] 通过对系统失效日志的研究, 发现失效表现出很强的时间空间局部性, 同时把节点无计划重起失效的间隔时间看作是一个随

机过程, 这个随机过程符合参数 $shape$ 小于 1 的 $weibull(scale, shape)$ 分布. 日前我们对大规模云计算实验床的系统失效日志作统计分析, 发现数据密集型服务系统 Hadoop(www.hadoop.com) 的失效规律也服从类似的参数 $shape$ 小于 1 的 $weibull$ 分布.

对于 $weibull(scale, shape)$ 分布, $pdf(t)$ 和 $cdf(t)$ 是

$$pdf(t)=\left(\frac{shape}{scale}\right)\times\left(\frac{t}{scale}\right)^{shape-1}\times e^{-\left(\frac{t}{scale}\right)^{shape}} \quad (2)$$

$$cdf(t)=1-\left(e^{-\left(\frac{t}{scale}\right)^{shape}}\right) \quad (3)$$

这样, 服从 $weibull(scale, shape)$ 分布的系统的失效率函数可以如下计算

$$\lambda(t)=\frac{pdf(t)}{(1-cdf(t))}=\left(\frac{shape}{scale}\right)\times\left(\frac{t}{scale}\right)^{shape-1} \quad (4)$$

假设两个节点: $nodeM$ 和 $nodeN$, 各自的恢复时间是 $uptimeM$ 和 $uptimeN$, 且 $uptimeM > uptimeN$. 失效率是: $\lambda M = \lambda(uptimeM)$ 和 $\lambda N = \lambda(uptimeN)$. 当 $shape < 1$, $\lambda M < \lambda N$ 时, $nodeM$ 比 $nodeN$ 更可靠. 这个规律说明, 刚刚失效的节点更有可能出现失效, 而运行起来之后, 节点会变得越来越稳定. 节点失效的这个特点表明, 把刚刚失效的节点作一段时间的可靠性检测是有必要的[8].

同时, 研究人员[12]指出对于一个典型的高性能计算服务和网络处理服务的混合机群系统, 节点无计划重起失效的概率, 有随运行时间先下降后上升的趋势. 这说明, 节点在刚刚启动时可靠性较低, 容易出故障, 之后随运行时间增加, 逐渐变得越来越稳定可靠, 之后在运行时间超过一定值之后, 故障率上升, 节点变得越来越不可靠. 这种现象对于混合机群里数据库节点尤其明显.

实际上, 研究人员[10-11, 15]发现, 服务系统(如数据中心服务)往往表现出软件老化问题. 软件老化表现为, 软件系统随着运行时间的增加, 性能逐渐下降, 而失效率逐渐上升. 软件老化的原因是系统运行过程中积累了越来越多的内部错误. 研究人员提出软件再生技术[11]来解决软件老化问题. 当软件系统主动再生, 系统内部累积的错误会被清除, 系统重新回到正常状态. 同时, 有效的再生策略可以减少软件再生技术引起的全系统的性能损失.

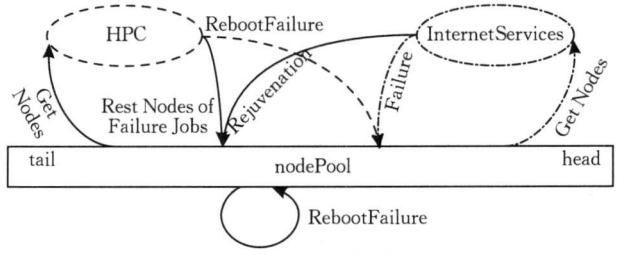
本文把以上失效规律一起加以考虑, 提出基于失效规律的节点资源动态提供策略. 我们的策略的目标是为云计算平台下整合的异构服务, 提供可靠的节点资源.

3 基于失效规律的资源提供策略

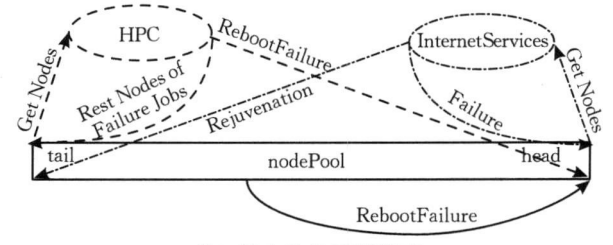
本节, 我们首先给出不考虑资源失效规律的资源提供策略, 作为 BaseLine 策略. 之后给出本文提出的基于失效规律的资源动态提供策略, 其中包括基本的单队列策略和扩展的多队列策略.

3.1 BaseLine 策略

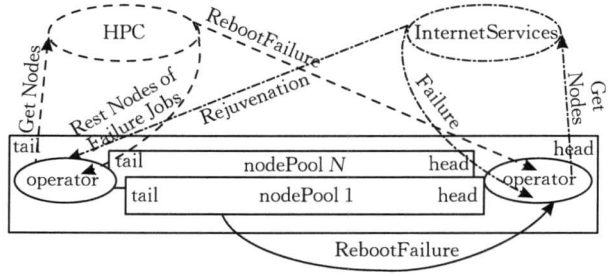
节点资源的动态提供策略是一个 On-Demand 策略. BaseLine 策略维护一个节点资源池, 当一个请求到达时, 资源提供策略从节点资源池里随机的选取一个空闲节点处理到达的请求, 当工作处理完成, 把节点随机地放回节点资源池. 在有节点失效时, BaseLine 策略把失效节点放回节点资源池, 把失效负载放入等待队列. 对于并行任务, 把相关的任务也放入等待队列, 释放节点, 如图 2(a) 所示.



(a) BaseLine策略



(b) 基本的单队列策略



(c) 扩展的多队列策略

图 2

3.2 基于失效规则的资源提供策略

本文提出的基于失效规则的资源动态提供策略

包括单队列策略和多队列策略两大类策略. 其中单队列策略是多队列策略的基础. 我们首先给出单队列策略(如图 2(b) 所示), 和 BaseLine 策略的关键不同在于, BaseLine 策略是随机的选择节点资源, 而我们的策略是维护一个按上次失效恢复时间(uptime) 有序的节点资源池.

根据上面的研究结果, 假定刚刚失效重起的节点的失效规律服从参数 $shape$ 小于 1 的 $weibull$ 分布, 即刚失效的节点比较脆弱, 容易发生故障, 随着运行时间增加, 节点失效机会变少, 会越来越可靠. 因此, 我们把刚失效的节点放在队列头部, 当有节点请求时, 从队列尾部取空闲节点, 保证该节点是空闲资源池中最可靠的节点.

我们的策略是以尽力而为的方式工作, 提供最可靠的节点, 但并没有设置任何 uptime 的门限. 因为, 我们的目标是, 在不影响节点资源使用率的情况下, 提供可靠的节点资源.

对于服务器整合平台, 我们也考虑服务系统的软件老化. 假定网络处理服务服从一个故障率逐渐上升的分布. 同时, 我们对运行时间超过一个门限值的服务节点, 作主动的再生, 把主动再生的节点放入队列的尾部. 根据文献[10-11], 假定主动再生使得服务和节点都是可靠的.

这里, 我们假定用于高性能计算服务的节点服从 $weibull(shape, scale)$, 其中 $shape < 1$, 而当节点用于网络处理服务时, 服从 $shape > 1$ 的 $weibull$ 分布. 当 $shape > 1$ 时, 根据式(4), 两个节点: $nodeM$ 和 $nodeN$, 设它们的服务时间分别是 $ServTimeM$ 和 $ServTimeN$, 并且 $ServTimeM > ServTimeN$. 设失效率: $\lambda M = \lambda(ServTimeM)$ 和 $\lambda N = \lambda(ServTimeN)$. 当 $shape > 1$ 时, $\lambda M > \lambda N$, 节点服务的时间越长, 越容易发生故障, 即节点表现出软件老化现象.

3.3　非一致失效规律下的资源提供策略

对于大规模开放式的云计算环境, 整个系统平台里, 空间上各部分的资源往往会服从各自不同的失效规律和失效到达分布. 基于本文上面提出的单队列的资源提供策略, 可以按照广域空间上资源各自不同的失效规律, 维护各自的有序的空闲资源列表. 当有资源请求到达时, 需要从多个资源列表里选择可靠性最好的资源, 这时可以有多种选择, 比如: (1) 按照各个列表的可靠性程度, 对列表区分优先级; (2) 不对列表设置可靠性的优先级, 而是比较当前各列表的备选资源的可靠性, 选择最合适的资源.

对于回收释放的资源或失效的资源, 也同样需要按照一定的方式, 选择放入哪个列表最适合, 比如: (1) 按所属关系选择放入对应的列表; (2) 按历史统计信息选择列表等. 图 2(c) 表示这种多队列策略, 其中空闲资源池各队列的首尾设置 Operator 操作算子实现这种选择关系.

当前, 我们实现了两种多队列资源提供策略. 对于资源的回收, 两种策略都是按所属关系选择对应的列表; 对于资源提供, 第 1 种策略, 按列表的可靠性等级, 选择可靠的列表提供资源, 当高可靠的列表为空时, 选择次可靠的列表, 依次进行; 第 2 种策略, 按各自失效规律, 计算当前各列表的备选的资源的可靠性程度, 选择最合适的资源.

3.4　策略算法描述

我们提出的基于失效规律的节点动态提供策略, 以事件驱动方式工作. 当 $Node\_Failure\_Event$ 或 $Time\_Schedule\_Event$ 发生时, 策略被调用. $Node\_Failure\_Events$ 是节点失效事件, 到达时间上服从 $Weibull$ 分布, 空间分布上服务 $zipf$ 分布. $Time\_Schedule\_Event$ 是时钟事件, 根据各个时段内 Web 和 HPC 负载的资源需求量, 调整 Web_Node_List 和 HPC_Node_List. 同时对 Web 服务节点作周期性重起, 把运行时间超过门限值 $Rejuvenation\text{-}Threshold$ 的节点放入 $Node\_Pool\_List$.

对于基本的单队列策略, 只维护一个列表; 对扩展的多队列策略, 维护多个列表. 在空闲资源池资源队列的头部和尾部各设置一个 $Operator.Head()$ 和 $Operator.Tail()$ 操作, 用来作合适队列的选择, 实现资源的提供和回收操作. 单队列和多队列策略, 统一的表述如算法 1 所示.

算法 **1**.　资源提供策略的算法描述.

Data_Structure:

Web_Node_List, 是 Web services 使用的节点的列表;

HPC_Node_List, 是 HPC 负载使用的节点的列表;

Node_Pool_Lists, 是空闲资源池里的维护的节点列表;

HPC_Job_Running_List, 是正在运行的 HPC 作业负载的列表;

HPC_Job_Waiting_List, 是等待运行的 HPC 作业负载的列表;

Input:

Web_Node_Failure_Event;

HPC_Node_Failure_Event;

Pool_Node_Failure_Event;

Time_Schedule_Event;

```
Begin:
  get event;
  switch(event.type){
    case Pool_Node_Failure_Event:
      put FailureNode.uptime= CurrentTime;
      Node_Pool_List.head= Operator.Head();
      put FailureNode to Node_Pool_List.head;
      break;
    case Web_Node_Failure_Event:
      put FailureNode.uptime= CurrentTime;
      Node_Pool_List.head= Operator.Head();
      put FailureNode to Node_Pool_List.head;
      break;
    case HPC_Node_Failure_Event:
      put FailureNode.uptime= CurrentTime;
      Node_Pool_List.head= Operator.Head();
      put FailureNode to Node_Pool_List.head;
      put the Job to HPC_Job_Waiting_List.tail;
      put rest nodes of Job to Node_Pool_List.tail;
      break;
    case Time_Schedule_Event:
      For(Web_Node_List){
        ++Node.ServiceAging;
        if(node.ServiceAging> RejuvenationThreshold){
          put Node.ServiceAging= 0;
          Node_Pool_List.tail= Operator.Tail();
          put Node to Node_Pool_List.tail;
        }
      }
      get Web_Request_Node_Resource;
      if(RequestedNodes.size< Web_Node_List.size){
          toReleaseNode.size= WebNodeList.size-
                              RequestedNodes.size;
          take toReleaseNode from Web_Node_List;
          Node_Pool_List.tail= Operator.Tail();
          release Nodes to Node_Pool_List.tail;
      }else{
          toRequestNode.size= RequestedNodes.size-
                              WebNodeList.size;
          Node_Pool_List.head= Operator.Head();
          take toRequestNode from Node_Pool_List.head;
          put these Node to Web_Node_List;
      }
      For(HPC_Job_Running_List){
        if(Job.ExecuteTime+ Job.StartTime>
          CurrentTime){
          put Job.status= finished;
          Node_Pool_List.tail= Operator.Tail();
          release Job's Nodes to Node_Pool_List.tail;
          put Job to HPC_Job_Finished_List;
        }
      }
      For(HPC_Job_Waiting_List){
        if(Job.Nodesize< Node_Pool_List.size){
          Node_Pool_List.tail= Operator.Tail();
          get Node from Node_Pool_List.tail;
          put Job.status= running;
          put Job to HPC_Job_Running_List;
        }
      }
      break;
  }
  End
```

4 评价方法

4.1 云计算模拟器

我们实现了一个服务器整合平台的模拟器,模拟云计算平台下,高性能计算和网络处理两大类服务整合在一起的情况下,节点资源的动态提供策略,见图 3. 在我们的模拟器里,高性能计算和网络处理服务都是以 On-Demand 的方式使用资源.

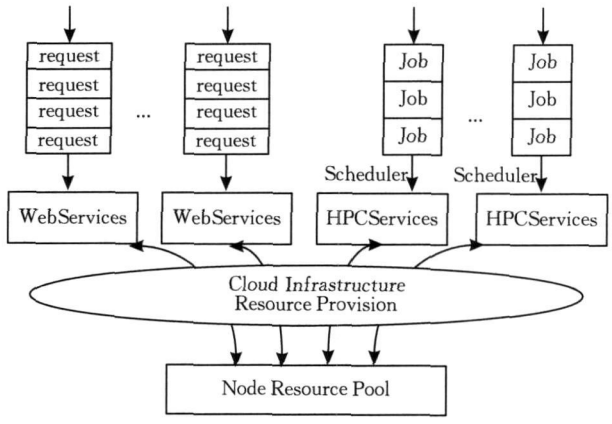


图 3 云计算模拟器

两大类服务在资源使用、管理时间粒度、负载类型和性能评价指标等方面有很大差异性. 具体来说:(1) 负载特征不同. 对于 Internet 数据中心交互式服务,负载由一系列的单链接或多链接的请求序列组成,对于 HPC 计算服务,负载是并行批处理作业;(2) 资源使用特征不同[3]. 对于 HPC 计算服务,往往需要独占的资源处理作业,对于 Internet 交互式服务,请求可以在共享资源上并发的执行;(3) 服务性能指标不同. 对于 HPC 计算服务,用户容忍作业的排队等待,直到获得运行的资源,对于 Internet 交互式服务,用户请求需要在线立即地响应,请求要在 QoS 容忍的门限时间内作出响应;(4) 管理时间粒度不同[4]. 对于 Internet 交互式服务,请求处理过程相对短,响应时间要求高,管理时间粒度小,对于 HPC 计算服务,资源管理的时间粒度相对大.

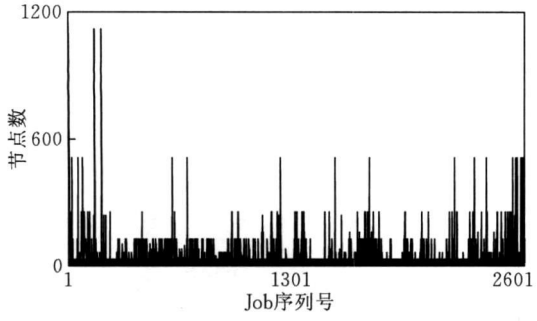
因此,在我们的模拟器里,两类服务被相互独立

的模拟. 这种设计架构, 使得我们的模拟器可以在节点资源池、网络服务、高性能计算之间动态地调整节点资源的配置和使用.

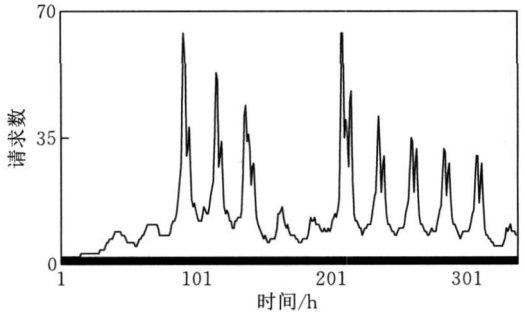
设 $Hres$ 是分配给高性能计算的节点数, $Ires$ 是网络处理服务的节点资源数, $Pres$ 是节点资源池里的空闲节点数. 我们改变模拟器的整个节点资源量(即 $Hres$, $Ires$, $Pres$ 的总和) 评价我们提出的节点资源动态提供策略的性能.

4.2　异构负载

我们使用两个真实系统负载 trace, 模拟两类服务整合下的负载. 其中高性能负载使用 SDSC BLUE trace(www. cs. huji. ac. il/ labs/ parallel/ workload/) , 见图 4(a) . 网络处理服务负载使用 World Cup 98 trace(ita. ee. lbl. gov/ html/ contrib/ WorldCup. html) . World Cup 98 trace 记录每个时间段内处理的请求数. 而我们主要是模拟资源的使用情况, 为此, 我们首先把吞吐率转换成资源的使用量. 这个转换是按照 CPU 占用不高于 80%, 且 SLO 不被破坏的情况下, 正常处理所需的节点资源量取整数得到的, 见图 4(b) .



(a) SDSC BLUE trace 中作业的节点资源需求序列



(b) World Cup98 trace 中的节点资源需求序列

图　4

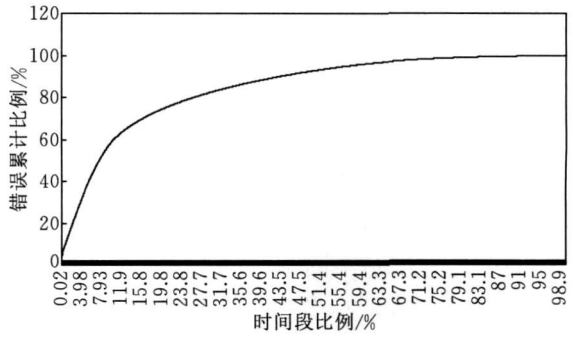
同时, 为评价两种负载不同混合比例下, 对资源提供策略的影响, 我们调整 World Cup 98 资源需求量, 以 1 倍、2 倍、3 倍等, 与单倍的 SDSC BLUE trace 负载相混合. 我们把这种不同比例的混合负载标记为 wc1、wc2、wc3 等.

4.3　失效模型框架

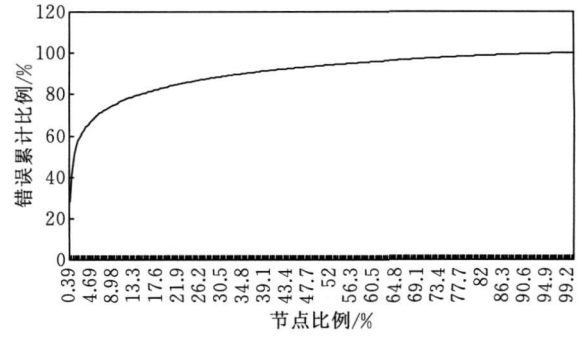
之前的系统失效的研究, 表明不同的负载和软硬件配置, 会对系统的失效模式带来不同的影响, 因此我们认为, 在大规模开放式的云计算环境下, 整个系统的失效规律也不会是一致的, 而是会在时间和空间因素上, 表现出较大的一个覆盖区间. 为了方便研究资源动态提供策略对于保证资源可靠性的能力, 我们构建一个通用的失效模型框架. 我们提出的失效模型框架, 包含失效在 3 个纬度上的信息, 即失效发生的时间分布信息、空间分布信息和失效引入的恢复开销.

其中, 失效的时间分布信息, 刻画失效发生的时间间隔的分布以及失效发生时间的关联性; 失效的空间信息, 刻画失效的局部性特征, 即整个系统资源的可靠性不一致, 失效集中在一部分节点; 失效引入的恢复开销, 刻画不同部件的各种失效类型恢复的开销.

日前, 我们对一个大规模云计算实验床的系统失效日志作统计分析(结果如图 5) , 发现 Hadoop 类数据密集型服务系统, 表现出很强的时间局部性(74. 5% 的失效事件发生在 20% 时间段内) 和空间局部性(14. 45% 的节点上发生了 80% 的错误事件) , 同时失效到达时间的分布服从参数 $shape$ 小于 1 的 $weibull(scale, shape)$ 分布(其中 $shape$ 在 0. 025~ 0. 05 之间) , 这与文献中对于传统机群的研



(a) 数据密集型应用失效的时间局部性



(b) 数据密集型应用失效的空间局部性

图　5

究结果有一定的区别, 如文献[7-8] 发现机群的节点失效的 *weibull* 分布的 *shape* 落在 0.7~0.8 区间内. 此结果也能说明我们提出的通用失效模型框架对于研究资源失效和可靠性的意义.

根据 Hadoop 失效日志的分析结果和文献中的结果, 我们假定资源失效的时间间隔服从 *weibull* 分布, 通过调整 *weibull* 分布的参数, 评价我们提出的策略的效果; 同时我们考查不同的失效空间局部性下, 我们策略的效果. 为此, 我们使用 *zipf* 分布来模拟空间上非一致的失效分布. 研究[16] 发现, 机群系统中的失效局部性符合 *zipf* 规律. *zipf* 作为通用的数学模型工具, 可以通过调整 *zipf* 的参数, 评价本文策略在不同的失效局部性情况下的效果. 另外, 由于本文只涉及对节点无计划重起失效的处理, 所以当前把失效开销设为一个常数值.

4.4 评价参数

为了比较 BaseLine 策略和我们的策略的性能差异, 我们从几个方面作评价: (1) 动态提供的节点资源的可靠性; (2) 不同策略对服务性能的影响; (3) 对节点资源使用率的影响; (4) 管理和恢复开销的影响.

对于动态提供的节点资源的可靠性方面, 我们提出几个评价参数: 节点池内空闲节点的失效数; 网络服务主动再生的次数; 网络服务无计划失效的次数; 高性能计算无计划失效的次数.

其中, 节点池内空闲节点的失效数, 体现了资源提供策略通过避免将不可靠的节点用于服务系统处理负载, 从而屏蔽掉这部分资源失效的能力. 因此, 在相同的节点失效分布情况下, 这个值越大说明资源提供策略对于屏蔽失效的能力越好.

网络服务主动再生的次数, 体现了网络处理节点的可靠性程度, 只有当节点连续无故障的工作超过一定的时间门限, 才会被主动再生. 因此, 这个值越大说明, 资源提供策略提供给网络处理服务的节点的可靠性越高, 说明资源提供策略的性能越好.

网络服务无计划失效的次数和高性能计算无计划失效次数, 体现了两类服务里节点的故障情况. 这个值越小, 说明资源提供策略提供的节点资源可靠性越高.

对于服务性能影响方面, 由于我们当前对网络处理服务只是模拟其资源使用情况, 未模拟其性能, 所以我们主要是从高性能服务的性能角度, 评价不同的资源提供策略对性能的影响. 评价参数包括: 平均完成的作业数, 作业的平均 turn around 时间, 作业的平均执行时间. 完成作业数越多, 平均执行时间越短, turn around 时间越短, 说明资源提供策略的性能影响越小.

对于节点资源使用率影响方面, 完成相同负载, 使用的资源量越少, 说明资源提供策略提供的资源越有效, 资源提供策略的性能越好.

对于管理和恢复开销方面, 考虑到在正常负载处理过程中, 系统故障会引入一系列的资源重新配置和应用的重起及状态恢复等开销, 由于当前我们并不区分负载失效的类型, 因此我们把负载执行过程中无计划的失效数看作对于恢复开销的衡量指标, 即无计划的失效数越少, 资源提供策略的性能越好.

5 实验评价

5.1 失效屏蔽能力的评价

我们首先评价我们提出的基本的资源提供策略对于屏蔽不可靠节点的能力. 我们把 Web 负载和 HPC 负载的混合比例按照: 1∶1, 2∶1, …, 14∶1 的方式, 作多组实验. 同时我们评价整体资源总数的变化对策略的影响, 把节点资源总数从 150 个按 30 个节点的步长逐渐增加到 1350 个.

实验结果如图 6 所示. 其中 x 轴是节点资源总数, y 轴是两类负载的 14 种混合比例, z 轴是高性能计算中节点失效数和高性能节点失效数与被屏蔽的节点失效数的比例.

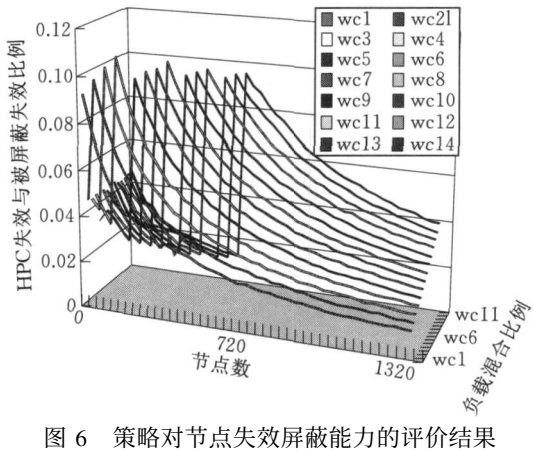


图 6 策略对节点失效屏蔽能力的评价结果

图 6 可以看出, 大多数的失效(不少于 90%) 发生在空闲节点池里, 说明我们的策略可以有效屏蔽掉大部分节点失效. 图中可以看出, 资源整体规模对于动态提供的节点资源的可靠性, 有一个正面的影响, 即给定负载情况下, 整体资源越多, 刚失效的不稳定节点有机会在空闲资源池里停留越长的时间,

这样可以尽量屏蔽掉刚失效节点不稳定又继续失效给服务带来影响的可能性, 达到提高动态提供的资源的可靠性的目的. 同时, 负载量规模对于动态提供的节点资源的可靠性的影响是负面的, 即给定资源总量, 负载量越大, 需要越多的资源处理, 节点在空闲资源池里停留的时间越短, 刚失效的节点被使用的可能性越大, 因此动态提供的节点资源的可靠性就会受影响.

同时, 图中可以看到一个巨大的突起. 这是因为, 节点资源总量沿 x 轴方向增加, 在突起的左侧, 系统处于资源饥饿态. 在突起右侧, 系统资源处于正常状态. 同时, 整体负载量沿 y 轴方面增加, 资源饥饿态是随着负载的增加在 x 轴方向逐渐右移. 这是因为, 负载的增加使得整体资源需求量增加, 使得沿 x 轴方向, 资源饥饿持续更久, 表现为资源饥饿态是随着负载的增加在 x 轴方向逐渐右移.

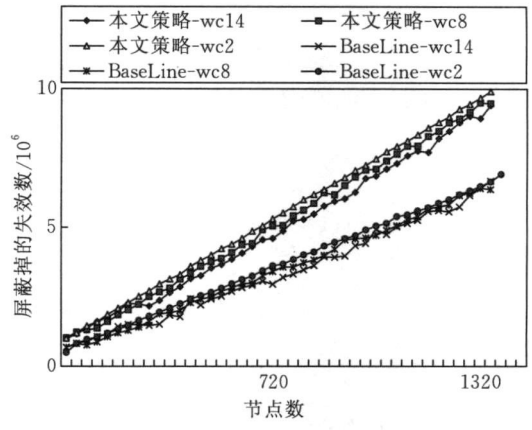
5.2　节点失效的屏蔽能力的比较

本节比较 BasLine 策略和我们提出的基本策略, 对节点失效的屏蔽能力. 负载混合比例是 Web: HPC, 2: 1, 8: 1, 14: 1. 节点资源总量是从 150 增加到 1350, 步长是 30.

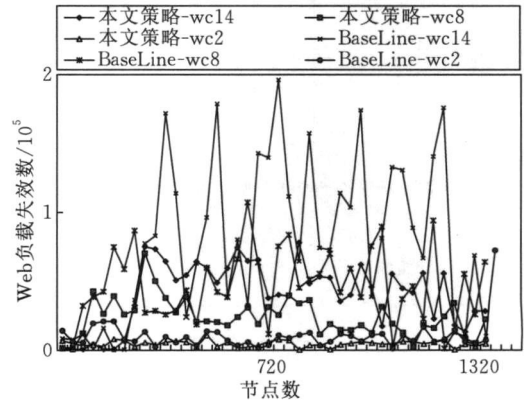
从图 7(a) 中可以看出, 与 BaseLine 策略相比, 我们的策略可以屏蔽掉更多的空闲节点失效. 这是因为, 按节点失效时间的排序, 有助于把刚刚失效的不稳定节点放在空闲池中保持尽量长的时间, 使节点度过不稳定时期, 避免节点的失效发生在负载处理过程中. 同时, 我们可以看出, 资源整体规模对屏蔽掉空闲节点失效有正面效果. 这是因为, 空闲资源增多, 可以使刚刚失效的节点有更多机会处于空闲态. 这就使我们的策略, 对资源按 uptime 排序可以比 BaseLine 的随机策略, 更好地选择可靠的空闲节点, 给刚刚失效不稳定的节点提供更多屏蔽的机会.

从图 7(b) 中可以看出, 与 BaseLine 策略相比, 我们的策略使得 Web 服务节点更可靠, 失效的次数更少. 同时从图 7(c) 中可以看出, 我们策略提供的节点, 有更多的机会无故障地连续处理 Web 服务, 当服务时间超过一定门限, 而被主动再生. 图 7(b) 和图 7(c) 可以看出, 我们的策略与 BaseLine 策略相比, 提供的节点资源的可靠性有一个明显的提高.

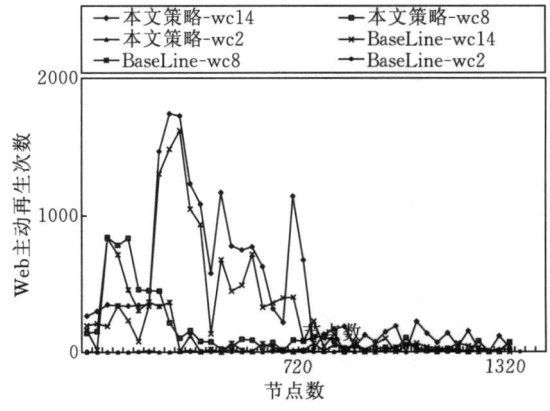
图 7(d), 我们把 HPC 节点失效和 Web 节点失效看作无计划失效, 把空闲节点失效和 Web 再生看作主动屏蔽和恢复, 那么比较我们的策略和 BaseLine 策略中, 无计划失效占整个重起处理的比例. 可以看出, 我们的策略使得无计划失效明显减少, 说明我们的策略提高平台整体资源的可靠性.



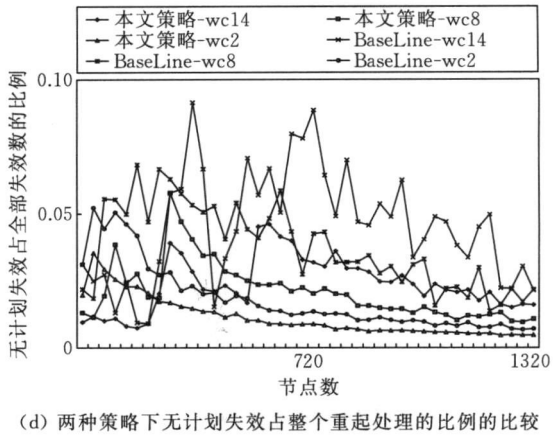
(a) 两种策略对节点失效能力的屏蔽结果的比较



(b) 两种策略提供的 Web 服务节点的失效数比较



(c) 两种策略下 Web 服务主动再生次数的比较



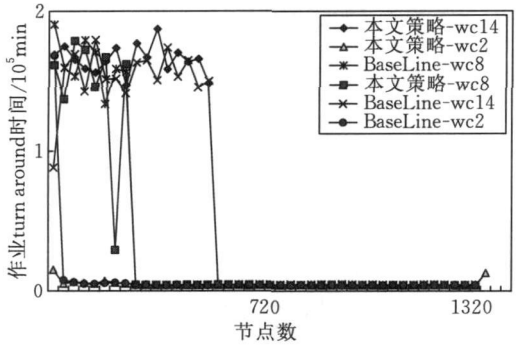
(d) 两种策略下无计划失效占整个重起处理的比例的比较

图 7

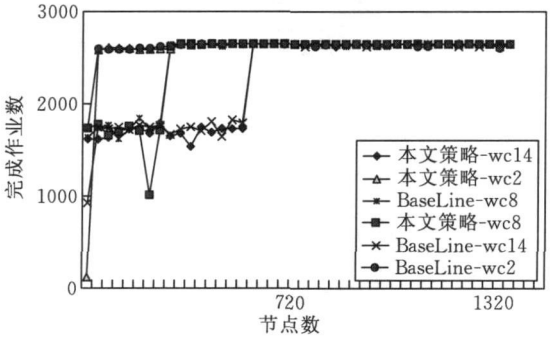
5.3 服务性能比较

本节比较 BasLine 策略和我们提出的基本策略对服务性能的影响. 负载混合比例仍然是 2∶ 1, 8∶ 1, 14∶ 1. 同时, 节点资源总量从 150 增加到 1350, 步长是 30.

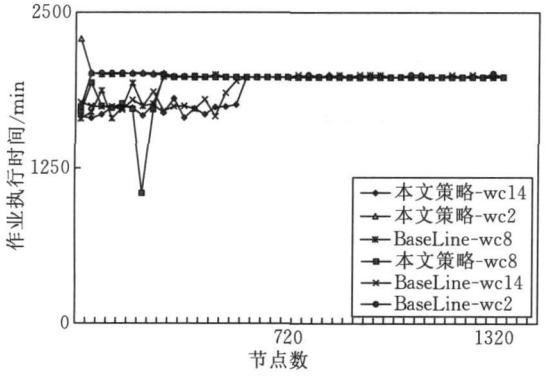
从图 8 中可以看出, 两种策略下负载的性能在资源饥饿和不饥饿情况下都是几乎一致的, 说明我们的策略, 与 BaseLine 策略相比, 提高动态提供的节点资源的可靠性, 而对于负载性能没有引入负面影响.



(a) 两种策略下作业平均 turn around 时间的比较



(b) 两种策略下作业平均完成数的比较



(c) 两种策略下作业平均执行时间的比较

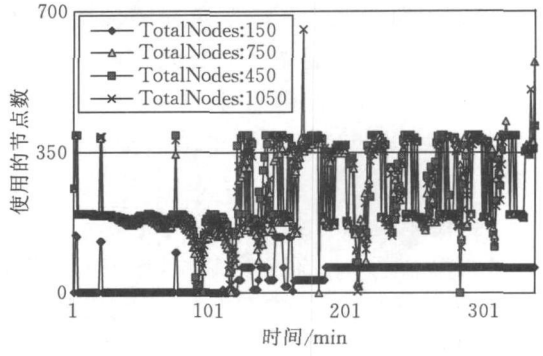
图 8

当前我们的模拟方法对 Web 服务只模拟了资源需求, 没有模拟 Web 的处理细节. 因此, 只比较了 HPC 负载下的性能, 我们计划在未来加入对 Web 处理细节的模拟和性能的比较.

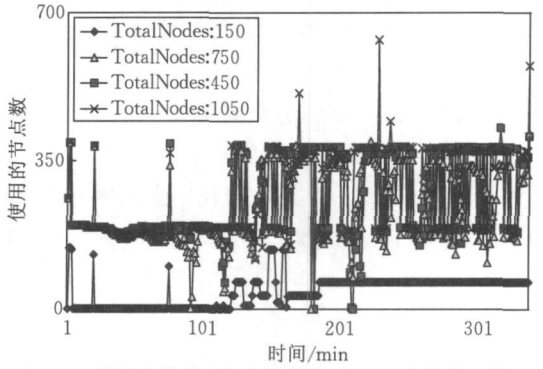
5.4 动态的资源使用率的比较

我们比较 BasLine 策略和我们提出的基本策略下, 节点资源的动态使用率. 负载混合比例是 8∶ 1, 14∶ 1. 节点资源总量从 150 增加到 1350, 步长 300.

图 9 是 8 倍 Web 负载和 1 倍 HPC 负载下的结果,

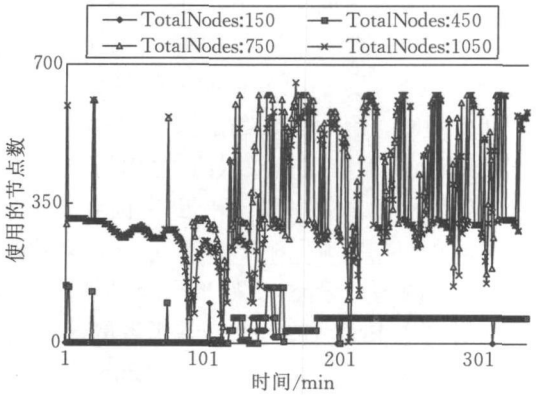


(a) 8∶1 混合负载 BaseLine 策略下 HPC 节点使用情况

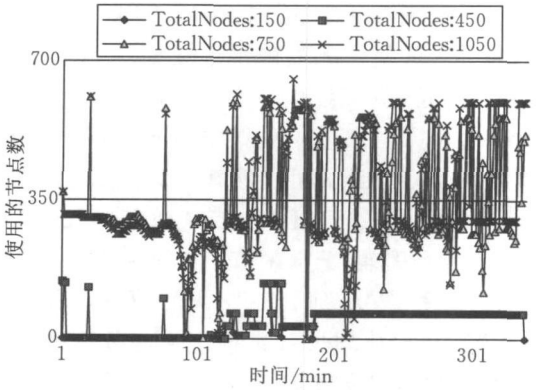


(b) 8∶1 混合负载本文提出的策略下 HPC 节点使用情况

图 9



(a) 14∶1 混合负载 BaseLine 策略下 HPC 节点使用情况



(b) 14∶1 混合负载本文提出的策略下 HPC 节点使用情况

图 10

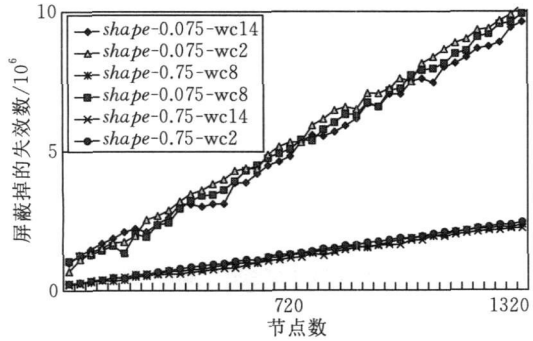
图 10 是 14 倍 Web 负载和 1 倍 HPC 负载下的结果. 图 9 和图 10 比较, 可以看出负载越大, 资源饥饿越严重, 图 9 中 450 个节点使得系统正常执行, 图 10 中系统仍是资源饥饿态.

尽管负载量的不同, 会给资源使用造成不同影响, 但比较图 9(a) 和图 9(b) 及图 10(a) 和图 10(b), 可以看出, 相同情况下, 我们的策略与 BaseLine 策略的资源使用率是几乎相同或略有改进的. 说明我们的策略可以提高动态提供的节点资源的可靠性, 同时对资源使用率不造成负面影响.

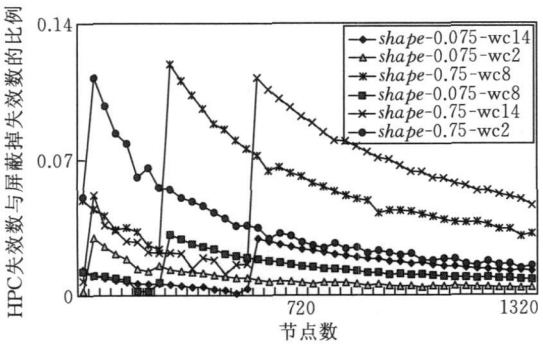
5.5　资源失效规律的时间特性的影响

我们评价失效达到的时间特性对于本文提出的基本策略的影响. 负载混合比例是 8∶ 1, 14∶ 1. 节点资源总量从 150 增加到 1350, 步长 300.

图 11 对比 *weibull(shape, scale)* 分布 *shape*= 0.075 和 *shape*= 0.75 情况下本文策略的性能. 其中图 11(a) 是 HPC 负载的失效数与 HPC 负载加空闲资源池中失效总数的比值; 图 11(b) 是空闲资源池中失效数.



(a) 不同 *shape* 参数下, 空闲资源池屏蔽掉的资源失效的次数



(b) 不同 *shape* 参数下, HPC 失效数与空闲资源池失效数比例

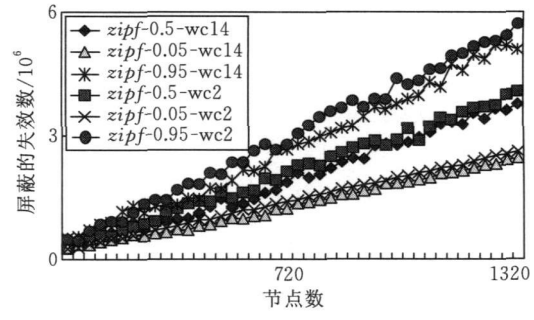
图　11

从图中可以看出, 本文策略, 在 *shape* 参数越小的情况下, 对于即将到来的失效的屏蔽能力越好. 原因是 *weibull(shape, scale)* 分布在 *shape* 越接近零的时候, *pdf(t)* 越向接近零的时间区间集中, 即刚刚失效的资源即将失效的可能性越大, 因此对于 *shape* 较小的情况, 使用空闲资源池的策略可以更有效地屏蔽掉即将到来的失效, 进而提高动态提供的资源的可靠性. 以我们日前对数据密集型应用的云计算实验平台的系统失效日志的分析结果, 发现这类系统的失效间隔时间服从的 *weibull* 分布的 *shape* 值(0.025~ 0.05) 小于传统机群系统的失效分析[7-8] 中得到的 *shape* 值(0.7~ 0.8), 因此对于数据密集型应用类的云计算平台, 我们的策略是更加有效的.

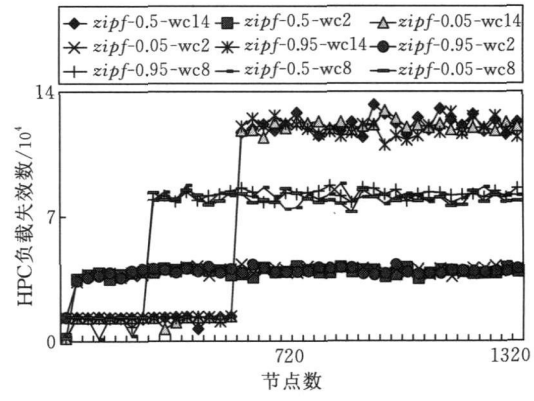
5.6　资源失效的空间特性的影响

我们评价失效的空间特性对于本文提出的基本策略的影响. 负载混合比例是 2∶ 1, 8∶ 1, 14∶ 1. 节点资源总量从 150 增加到 1350, 步长 300.

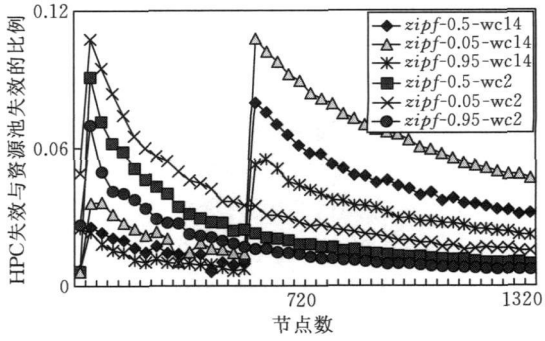
图 12 对比空间非一致失效分布情况下(*zipf* = 0.05, 0.5, 0.95), 本文策略的效果. 其中, *zipf* = *n*%



(a) 不同 *zipf* 参数下, 空闲资源池屏蔽掉的资源失效的次数



(b) 不同 *zipf* 参数下, HPC 作业的失效数



(c) 不同 *zipf* 参数下, HPC 作业失效数与资源池中屏蔽掉的资源失效的次数的比例

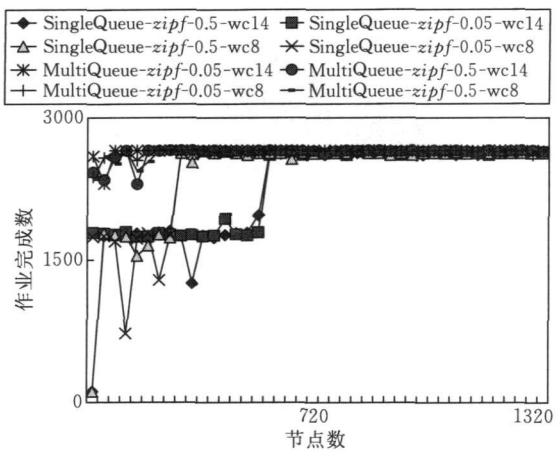
图　12

表示系统中 $n\%$ 的资源易出现失效, 而剩余的 $1-n\%$ 的资源相对较可靠. 图 12(a) 是空闲资源池屏蔽掉的失效数, 图 12(b) 是 HPC 负载的失效数与空闲资源池失效数的比值. 可以看出本文策略对于不可靠的资源配置情况($zipf = 0.95$), 有更显著的失效屏蔽能力. 图 12(c), 是 HPC 资源的失效数. 可以看出, 在不同负载下(WC14, WC8, WC2), 由于系统整体的负载量的变化, 提供给 HPC 的资源的可靠性不同. 从 WC2~ WC14, 随着负载量增加, 系统整体资源量逐渐紧张, 使得可以提供给 HPC 负载的资源的可靠性下降. 但从图中可以看出, 在相同的负载量不同失效分布($zipf = 0.05, 0.5, 0.95$) 情况, 本文策略所动态提供的资源的可靠性, 并没有造成明显的影响, 即本文的策略可以一定程度上屏蔽掉, 系统资源的非一致失效分布带来的影响. 对于大规模开放的云计算平台, 整个系统的资源可能分布在一个广泛的空间区域, 假定资源满足完全一致的失效分布是不太可能的. 而本文的策略, 正可以在一定程度上屏蔽掉资源失效分布的差异性引入的问题.

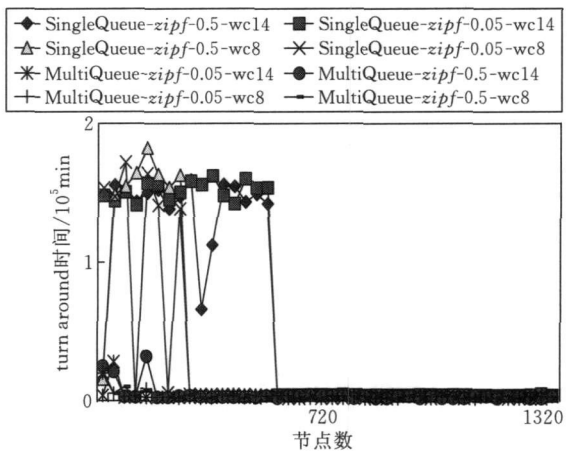
5.7 单队列与多队列策略的比较

我们评价单队列与多队列策略的效果. 负载混合比例是 8∶1, 14∶1. 节点资源总量从 150 增加到 1350, 步长 300.

图 13, 给出相同情况下($zipf = 0.05$ 和 0.5, 负载为 WC14 和 WC8), 单队列和多队列策略下, 系统负载的性能比较. 其中, 图 13(a) 是平均完成数, 图 13(b) 是平均 turn around 时间. 从图中可以看出, 与单队列的策略相比, 多队列策略可以有效的保证系统在高负载的情况下的服务性能. 原因是, 对于失效规律非一致分布的情况下, 多队列策略, 可以更好的选择可靠的资源执行系统负载, 进而很大程度上缓解系统资源的瓶颈, 保证服务性能.



(a) 单队列和多队列情况下, 作业平均完成数的比较



(b) 单队列和多队列情况下, 作业平均 turn around 时间的比较

图 13

6 相关工作

6.1 资源管理和动态提供

研究人员提出了多种方案解决异构负载下资源管理和动态提供问题. Nurmi[2] 和 Rochwerger[1] 分别提出面向云计算平台用以整合异构负载的大规模虚拟机群资源管理平台 Eucalyptus 和 RESERVOIR. Irwin[5] 等人提出面向异构负载的虚拟机资源租用框架 Shirako 及支持异构负载的资源共享加权公平的调度模型 Winks. Padala[6] 等人研究异构负载下虚拟机群系统的资源使用特征, 提出适用于计算密集型和交互密集型整合负载的资源动态提供策略. 上述工作集中于处理异构负载下的资源管理和资源动态提供问题, 但没有考虑动态提供的资源的可靠性问题.

6.2 系统的失效规律

研究人员对系统失效规律作了大量研究. Iyer 等人研究负载类型负载强度和失效率之间的关系. Heath[8] 等人研究了机群系统的失效日志, 发现节点的无计划重起失效的时间间隔服从 $weibull(scale, shape)$ 分布, 其中 $shape < 1$, 即刚刚失效的节点很不稳定. Schroeder[7] 等人对洛杉矶国家实验室的机群系统长达 9 年的失效日志作研究, 证实 HPC 计算机群节点的婴儿期脆弱现象. Sahoo[12] 等人对 IBM 的混合服务的机群系统作研究, 发现与文献[7-8] 相似的节点失效规律, 同时发现当连续处理时间超过一定值后, 系统故障率上升, 这种老化现象在数据库节点尤为显著.

6.3 软件老化和再生

Huang 和 Trivedi[10-11] 等人发现电信和网络服

务等系统在连续运营过程中存在软件老化现象, 并提出软件再生技术, 通过主动重起老化系统, 清理内部积累的错误状态, 使系统恢复正常. 软件再生技术作为一种简洁有效的主动恢复技术, 得到广泛研究, 主要工作分两大类: 基于系统状态模型的再生策略(如 Markov model[11]、Stochastic Reward Net[10]、Stochastic Activity Net[9]) 基于系统参数监测和预测的主动恢复技术(如时间序列分析[17]、多维分型分析方法[15]).

6.4　系统可靠性保证

研究人员提出了多种改进资源可靠性的方法. Heath[8] 针对集群节点失效规律, 提出资源可靠性验证方法; Zhang[16] 针对 HPC 服务, 提出和分析多种检查点策略对于减少由系统故障引入的性能损失的能力; Hacker[18] 分析 HPC 的作业队列结构, 对于改进作业运行的可靠性的能力. 相比之下, 我们是针对云计算环境下, 异构服务在时间空间上的失效规律, 提出保证动态提供的资源可靠性的策略.

6.5　云计算环境模拟器

Buyya[19] 设计实现 CloudSim 模拟器, 用以研究云计算环境下资源提供和使用过程中涉及经济关系的租用模型和处理行为. 与之不同, 我们的模拟器是从考察异构负载环境下资源动态使用行为和可靠性角度设计的. 我们的模拟器适用于为研究云计算异构负载整合环境下资源动态提供策略.

7　结　论

本文提出面向云计算环境的基于失效规则的节点资源动态提供策略. 我们的策略综合考虑了资源需求和失效规律, 保证动态提供的节点资源的可靠性.

我们提出和实现了一个整合异构负载的云计算模拟器平台和系统资源多维度的失效模型框架, 来验证本文提出的策略的效果. 我们的云计算模拟器通过模拟异构负载下资源使用和失效时间空间上的规律, 验证资源动态提供策略的性能.

我们基于模拟器平台利用真实的异构负载, 评价本文提出的策略. 结果表明, 与 BaseLine 策略相比, 本文提出的策略可以有效提高动态提供的节点资源的可靠性, 屏蔽掉大量节点资源的失效, 同时对资源使用效率和服务性能不引入负面影响. 对于失效在时间空间特性上的评价表明, 本文策略适用于云计算环境. 本文提出的策略不涉及对系统平台的任何修改或侵入式监测, 具有很好的应用前景.

参　考　文　献

[1] Rochwerger B, Breitgand D, Levy E et al. The Reservoir model and architecture for open federated cloud computing. IBM Journal of Research and Development, 2009, 53(4): 1-17

[2] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk et al. The eucalyptus open-source cloud-computing system//Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. 2009: 124-131

[3] Armbrust M, Fox A, Griffith R et al. Above the clouds: A berkeley view of cloud computing. UC Berkley, USA: Technical Report No. UCB/EECS-2009-28, 2009: 1-25

[4] Vaquero L M, Rodero-Merino L, Caceres J et al. A break in the clouds: Towards a cloud definition. ACM SIGCOMM Computer Communication Review, 2009, 39(1): 50-55

[5] Irwin D, Chase J S, Grit L et al. Sharing networked resources with brokered leases//Proceedings of the USENIX Technical Conference. Boston, MA, USA, 2006: 199-212

[6] Padala P, Shin K G, Zhu Xiao-Yun et al. Adaptive control of virtualized resources in utility computing environments//Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007. Lisbon, Portugal, 2007: 289-302

[7] Schroeder B, Gibson G A. A large-scale study of failures in high-performance computing systems//Proceedings of DSN2006. Philadelphia, Pennsylvania, USA, 2006: 249-258

[8] Heath T, Martin R P, Nguyen T D. Improving cluster availability using workstation validation//Proceedings of the ACM SIGMETRICS. Marina Del Rey, California, USA, 2002: 217-227

[9] Tai A T, Tso K S, Sanders W H et al. Chau: A performability-oriented software rejuvenation framework for distributed applications//Proceedings of the 35th DSN 2005. Yokohama, Japan, 2005: 570-579

[10] Vaidyanathan K, Harper R E, Hunter S W et al. Analysis and implementation of software rejuvenation in cluster systems//Proceedings of the ACM SIGMETRICS 2001. Cambridge, MA, USA, 2001: 62-71

[11] Huang Y, Kintala C, Kolettis N et al. Software rejuvenation: Analysis, module and applications//Proceedings of the 25th Symposium on Fault Tolerant Computer Systems. Pasadena, California, 1995: 381-390

[12] Sahoo R K, Sivasubramaniam A, Squillante M S et al. Failure data analysis of a large-scale heterogeneous server environment//Proceedings of the DSN 2004. Florence, Italy, 2004: 772-784

[13] Zhan J, Sun N. Fire Phoenix cluster operating system kernel and its evaluation//Proceedings of the Cluster2005. Boston, MA, USA, 2005: 1-9

[14] Ross S. A First Course in Probability. New Jersey: Prentice Hall, 2002

[15] Shereshevsky M, Crowell J, Cukic B et al. Software aging and multifractality of memory resources//Proceedings of the 33rd DSN 2003. San Francisco, CA, USA, 2003: 721-730

[16] Zhang Y, Squillante M S, Sivasubramaniam A et al. Performance implications of failures in large-scale cluster scheduling//Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing JSSPP 2004. New York, NY, USA, 2004: 233-252

[17] Vaidyanathan K, Trivedi K S. A comprehensive model for software rejuvenation. IEEE Transactions on Dependable and Secure Computing, 2005, 2(2): 124-137

[18] Hacker T J, Meglicki Z. Using queue structures to improve job reliability//Proceedings of the ACM HPDC 2007. Monterey, California, USA, 2007: 43-54

[19] Buyya R, Ranjan R, Calheiros R N. Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities//Proceedings of the 7th High Performance Computing and Simulation. Leipzig, Germany, 2009: 1-11

TIAN Guan-Hua, born in 1980, Ph. D. candidate. His research interests include quality of service in cluster system and cloud computing.

MENG Dan, born in 1965, Ph. D., professor, Ph. D. supervisor. His research interests include architecture of high performance computing system, system software, high performance communication protocol, distributed file system, operating system for cluster system.

ZHAN Jian-Feng, born in 1976, Ph. D., associate professor. His research interests include cluster operating system, distributed system, high availability software.

Background

Cloud computing, as a novel computing paradigm, become popular. almost all leading IT enterprises propose their Cloud architecture and infrastructure to facilitate managing and sharing massive scale computing resources. In fact, researchers proposed various resource sharing techniques and resource provision techniques. However, very limited literatures pay attentions to the reliability of dynamically provided resources.

This paper proposes a failure rules aware node resource provision policy for heterogeneous services consolidated in cloud computing infrastructure.

This work is supported by National Science Foundation of China under grant of Nos. 60703020, 6093300 and the National High Technology Research and Development Program (863 Program) under grant of Nos. 2006AA01A102, 2009AA01A129, and 2009AA01Z139. The project focus on improving the availability of large scale systems. This work improves reliability of dynamically provisioned resources in cloud computing.

The group is always doing researches in the field of dependability and availability for large scale systems, and has developed Fire Phoenix system, a scalable and fault-tolerant cluster management system for Dawning4000 series, and Dawning5000 series. The work is published in Proceedings of the 7th IEEE Cluster Computing 2005. Meanwhile, the group also got some interesting achievements in the relative fields, e.g., QoS guarantee, fault diagnosis, performance debugging. Jiang Y et al. introduced a promising admission control policy to guarantee QoS of cluster systems, and her work is published in Proceedings of the 8th IEEE Cluster Computing 2006. Wu L et al. proposed a fault diagnose algorithm for cluster systems, and his work is published in Proceedings of the 20th IEEE IPDPS 2006. Zhang Z H et al. proposed a precise request causal path reconstruction algorithm, the work published in Proceedings of the 39th IEEE DSN 2009.