计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20180675
57(1):102-123,2020

云计算系统可靠性研究综述

段文雪[1] 胡 铭[1] 周 琼[2] 吴庭明[1] 周俊龙[3] 刘 晓[4] 魏同权[1] 陈铭松[1]
[1](华东师范大学上海市高可信计算重点实验室 上海 200062)
[2](上海外国语大学国际金融贸易学院 上海 200083)
[3](南京理工大学计算机科学与技术学院 南京 210094)
[4](迪肯大学信息技术学院 澳大利亚墨尔本 VIC 3125)
(wenxueduan@gmail.com)

Reliability in Cloud Computing System: A Review

Duan Wenxue[1], Hu Ming[1], Zhou Qiong[2], Wu Tingming[1], Zhou Junlong[3], Liu Xiao[4], Wei Tongquan[1], and Chen Mingsong[1]
[1](*Shanghai Key Laboratory of Trustworthy Computing*, *East China Normal University*, *Shanghai* 200062)
[2](*School of Economics and Finance*, *Shanghai International Studies University*, *Shanghai* 200083)
[3](*School of Computer Science and Technology*, *Nanjing University of Science and Technology*, *Nanjing* 210094)
[4](*School of Information Technology*, *Deakin University*, *Melbourne*, *Australia VIC* 3125)

Abstract As a new computing paradigm, cloud computing has attracts extensive concerns from both academic and industrial fields. Based on resource virtualization technology, cloud computing provides users with services in the forms of infrastructure, platform and software in a "pay-as-you-go" manner. In the meanwhile, since cloud computing provides highly scalable computing resources, more and more enterprises and organizations choose cloud computing platforms to deploy their scientific or commercial applications. However, with the increasing number of cloud users, cloud data centers continuously expand and the architecture becomes increasingly complex, leading to growing runtime failures in cloud computing systems. Therefore, how to ensure the system reliability in cloud computing systems with large scale and complex architecture has become a huge challenge. This paper first summarizes various failures in cloud systems, introduces several methods to evaluate the reliability of cloud computing, and describes some key fault management mechanisms. Since fault management techniques inevitably increase energy consumption of cloud systems, this paper reviews current researches on the trade-off between reliability and energy efficiency in cloud computing. In the end, we propose some major challenges in current research of cloud computing reliability and concludes our paper.

Key words cloud computing; virtualization; reliability; fault management; energy consumption

摘 要 云计算作为一种新型计算模式,已经受到了学术界和工业界的广泛关注.基于资源虚拟化技术,云计算能够以按需使用、按使用量付费的方式为用户提供基础设施、平台、软件等服务.因此,越来越多的企业和组织选择云计算来部署他们的科学或商业应用.然而,随着用户数量的不断增加,数据中心的规模在迅速扩大、架构变得日益复杂,导致云计算系统的运行故障频繁发生,造成了巨大的损失.因此在

收稿日期:2018-09-26;**修回日期**:2019-06-13
基金项目:国家重点研发计划项目(2018YFB2101300);国家自然科学基金项目(61872147)
This work was supported by the National Key Research and Development Program of China (2018YFB2101300) and the National Natural Science Foundation of China (61872147).
通信作者:周琼(zhou.qiong@shisu.edu.cn)

规模巨大、架构复杂的云计算系统中,如何保障系统的可靠性已经成为一个极具挑战性的问题.针对云计算可靠性问题,概述了云计算系统中常见的各种故障,并详细描述了目前云计算中提高可靠性关键的故障管理技术;由于故障管理技术的应用会不可避免地增加系统的能耗,因此介绍了云计算中可靠性与能耗权衡问题的研究现状;最后列举了当前云计算可靠性研究中存在的主要挑战.

关键词 云计算;虚拟化;可靠性;故障管理;能耗

中图法分类号 TP391

云计算(cloud computing)是基于分布式计算(distributed computing)、网格计算(grid computing)、并行计算(parallel computing)等技术发展而来的一种新型计算模式[1].它利用虚拟化技术,将各种硬件资源(如计算资源、存储资源和网络资源)虚拟化,以按需使用、按使用量付费的方式向用户提供高度可扩展的弹性计算服务.由于无需自己为了购买IT基础设施、搭建私有计算平台以及管理升级软硬件资源而投入高昂的费用和人力成本,越来越多的企业选择将他们的计算需求外包给云计算服务提供商.因此,云计算已经引起了学术界和工业界的高度重视,例如,国外的Amazon,Google,国内的阿里、百度、腾讯等企业竞相发布各种云计算平台为用户提供服务.云计算作为一种正在发展中的信息技术,已经被视为信息产业的一次重大革命.Gartner发布的研究报告称,全世界公有云服务的市场收益在2016年为2196亿美元,到2017年已经增加到了2602亿美元,增长率大约为18.5%[2].与此同时,各种形式的软硬件资源仍然在不断地被加入到云计算系统中,数据中心的规模和复杂度也在动态增长,这也进一步促进了云计算技术的飞速发展.

尽管不断发展的云计算技术使得数据中心能够提供动态的、易使用的计算资源,但是由于其具备资源虚拟化、多租户、规模巨大和体系结构复杂等特性,云计算仍然面临着诸多挑战.随着云计算系统中硬件资源的不断增加、规模不断扩大,各种资源的不确定性也在迅速增长,导致系统在运行时频繁发生各种故障[3].2017年8月30日,受系统更新程序故障的影响,Google云的负载均衡系统瘫痪18h,导致美国、欧洲和亚洲等地区的虚拟机都无法连接到Google云后端.云计算系统中频繁发生的故障,无论对用户还是对服务提供商都会带来巨大的损失.根据Google发布的报告称[4],每次修复故障需要的开销包括100美元的技术人员费用和10%的服务器价格(大约为200美元),即单次故障维修大约需要300美元.因此,单台服务器在经历7次故障后,修复硬件需要的开销已经超过购买1台全新服务器所需要的价钱.2016年的一份调查报告显示[5],单次数据中心宕机造成的平均损失在2010年为505502美元,在2013年为690204美元,到2016年已经增加到了740357美元.根据信息周刊发布的报告称[6],每年IT服务中断导致的损失已经超过265亿美元.因此,作为制约云计算技术发展的重要因素,云计算系统的可靠性已经成为亟待研究的问题.

对用户而言,云计算系统的可靠性反映了系统能够提供无故障服务的稳定程度,而云计算系统中的故障则经常体现为某种形式的系统崩溃、服务失效或结果错误.更具体地说,故障是系统不能正常运行的真实原因,而错误是系统故障的一种外在体现,即系统中的失效或错误本质上来源于系统中的某种故障.多个错误可能来自于同一个故障,多个故障也有可能导致同一个错误[7].在云计算系统中,故障的发生可能来自于多方面的原因,如硬件故障、软件故障等.相比于传统的并行计算环境,云计算平台具有高度异构性,计算资源的失效概率相对更高[8].在云系统发生故障时采取合理的措施处理异常情况对保证系统服务的可靠性至关重要.因此,为了保证云计算系统的正确运行,故障管理已经成为云计算系统的重要特性[9].常见的故障管理技术有故障消除(fault removal)、故障预测与避免(fault forecast and avoidance)以及故障容忍(fault tolerance).故障消除是通过传统的软件测试与验证方法移除云计算系统中存在的软件缺陷来保证服务可靠性的一种方法,这种方法适用于系统发布的早期.故障预测与避免是在作业调度的过程中预测故障的发生并采取合理的措施预防故障,它要求系统设计人员具备处理各种类型故障的专业知识[10].故障容忍是系统在发生故障的情况下依然能够正确执行功能的一种能力.云计算环境中的故障容忍技术主要分为2类:主动式容错和被动式容错[10-13].主动式容错是在系统发生故障后采取措施恢复系统的正常功能,常见的主动式容错技术有检查点技术、主动复制技术等.

被动式容错是在不改变系统结构的条件下,从鲁棒控制思想出发设计控制系统,使其对故障不敏感,典型的被动容错是用多副本表决来避免由故障引起错误.

虽然故障管理技术对于提高云计算系统的可靠性、保障服务的性能具有至关重要的作用,但是在系统运行用户的应用程序时,使用故障管理技术必然会增加数据中心消耗的电能,给云计算服务提供商带来额外的运营成本,降低投资回报率.例如 Google 数据中心 1 年的耗电量为 1.12×10^9 kW·h,随之而来的是 6 700 万美元的电费;微软数据中心每年的耗电量为 6×10^8 kW·h,需要支付 3 600 万美元的电费[14].按照目前 35%的增长率发展[15],到 2020 年美国的数据中心消耗的电能将会达到 1.4×10^{11} kW·h,届时美国将可能需要新建 17 个发电站才能满足数据中心的电力需求.由此可见,降低数据中心能耗已经成为云计算服务提供商面临的另一个挑战.因此,在保证云计算系统可靠性的同时,如何降低数据中心的能耗也是在设计故障管理技术时必须要考虑的问题.

1 云计算系统可靠性概述

保障云计算系统的可靠性对于系统向用户提供可靠的云计算服务具有至关重要的意义.本节首先给出云计算系统可靠性的定义,然后介绍可靠云计算系统的设计原则,最后详细描述影响云计算系统可靠性的主要因素.

1.1 云计算系统可靠性定义

云计算系统的可靠性衡量了系统能够提供不中断的无故障服务的稳定程度.一般来说,可靠性可以定义为:在某种规定的条件下,系统能够在约定的时间范围内稳定提供无故障服务的能力[16].如图 1 所示[17],云计算是一种面向服务的架构,客户端可以随时随地通过网络访问云计算系统提供各种服务,如基础设施即服务(infrastructure as a service, IaaS)、平台即服务(platform as a service, PaaS)和软件即服务(software as a service, SaaS).因此,云计算系统的可靠性与具体的服务模型密切相关.为了使云计算系统提供的服务更加可靠,每种服务模型的服务提供商都有责任保障服务的可靠性,并且各自的责任随着服务模型的不同而有所差异.例如,基础设施即服务层的服务提供商应当保证硬件设施一直处于稳定运行的状态,不会因为硬件故障而影响到运行在基础设施上的服务质量;而软件即服务层的服务提供商则需要保证软件系统中不会存在严重的软件缺陷,避免软件故障影响到用户的服务体验.

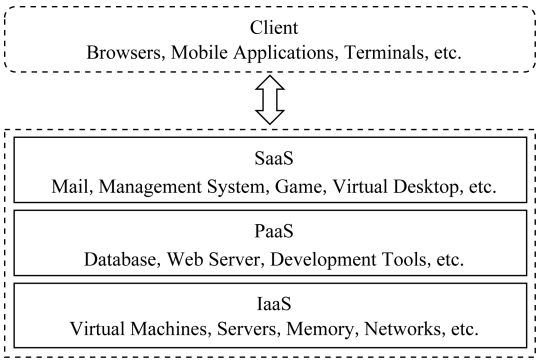

Fig. 1 Service model of cloud computing system

图 1 云计算系统的服务模型[17]

1.2 可靠云计算系统的设计原则

为了避免云计算系统在运行过程中产生故障或者使云计算服务在系统发生故障时具备可恢复的能力,微软公司的 Adams 等人[18]提出了 3 个设计原则来保障云计算系统的可靠性:

1) 弹性设计原则(design for resilience).在不需要人为干预的条件下,云计算服务必须能够容忍系统组件的失效,它应当能够检测到系统中故障的发生并在故障发生时自动采取矫正措施,使用户不会察觉到服务的中断.当服务失效时,系统也应当能够提供部分功能,而不是彻底崩溃.

2) 数据完整性设计原则(design for data integrity).在系统发生故障的情况下,服务必须能够以和正常操作一致的方式操纵、存储和丢弃数据,保持用户托管数据的完整性.

3) 可恢复设计原则(design for recoverability).系统在发生异常情况时,应该能够保证服务可以尽可能快地自动恢复过来;而当服务中断事件发生时,系统维护人员应该能够尽可能快地并且尽可能完整地恢复服务.

遵循这些设计原则有助于提高云计算系统的可靠性并降低故障带来的影响.除了这些原则外,如果某个应用服务实例发生了失效事件,那么未完成的部分服务和被延迟的服务最终也应该能够按规定顺利完成.一旦系统发生了故障,服务提供商或用户都应该采取合适的措施恢复服务,避免因故障导致的服务性能的下降,同时在服务恢复过程中也应尽可能减少人工干预.

1.3 影响云计算系统可靠性的因素

根据 Javadi 等人[19]的总结,系统失效可以定义为系统不能按照约定的方式继续正常运行的事件.当系统偏离了正常功能,就可以认为系统发生了故障.因此,系统中发生的故障是影响系统可靠性的主要因素.在云计算系统中,系统资源规模巨大、体系结构异常复杂、运行的服务数量众多,同时由于各层服务模型之间相互依赖,故障的发生相比于普通系统更加频繁.因而,故障是影响云计算系统可靠性的主要威胁.

在软件测试领域,失效是指软件在运行过程中出现的一种不希望或不可接受的可观察到的外部行为.故障是指软件在运行过程中出现的不希望或不可接受的内部状态,例如软件在执行过程中进入了错误的条件分支,软件便出现了故障[20].如果没有恰当的措施处理故障,此时软件将会出现失效.缺陷是指存在于软件的程序、数据或文档中的不希望或不可接受的差异,如代码错误.当软件在某个条件下出现软件故障,便可以认为软件中存在缺陷.本文引申软件测试中的故障定义,将使硬件、软件等组件或系统的行为发生失效的不正确状态称为故障,例如内存因为读写老化使计算机系统不能正常工作,此时可以称系统中发生了内存故障.

对云计算系统中故障的类型和起因进行概括和总结可以帮助计算机科学家和工程人员设计可扩展的算法、以可容错的方式部署基础设施和软件服务,这也有助于降低故障的修复代价,并使云计算系统提供更加可靠的服务.本节总结了云计算系统中的常见故障以及故障发生的主要原因.

基于故障的特征,云计算系统中的故障主要可以分为 3 种类型:资源故障、服务故障和其他故障.资源故障是指物理资源进入了不正确状态,如硬件故障、软件错误、电源中断、网络中断等.资源故障既可以发生在客户端,也可以发生在服务提供商端.目前大部分容错工作主要集中在资源故障上[21-24].云计算系统中的服务故障是指服务提供商不能提供、或用户不能获取满足服务等级协议(service level agreement, SLA)中规定服务质量的服务.在没有采取容错措施时,资源故障通常会导致服务故障,但在物理资源正常运行时,服务故障依然可能会发生.云计算系统中的其他故障一般是指一些无法预测的由自然或人为原因引起的故障,例如高能粒子、网络攻击和人员操作失误等.为了保证云计算服务的可靠性与可用性,理解故障发生的原因是非常重要的.图 2 总结了云计算系统中常见的故障起因,主要包括资源故障(如硬件故障、软件故障、电源中断和网络故障)、服务故障和其他因素(如软错误、网络攻击和人为因素).

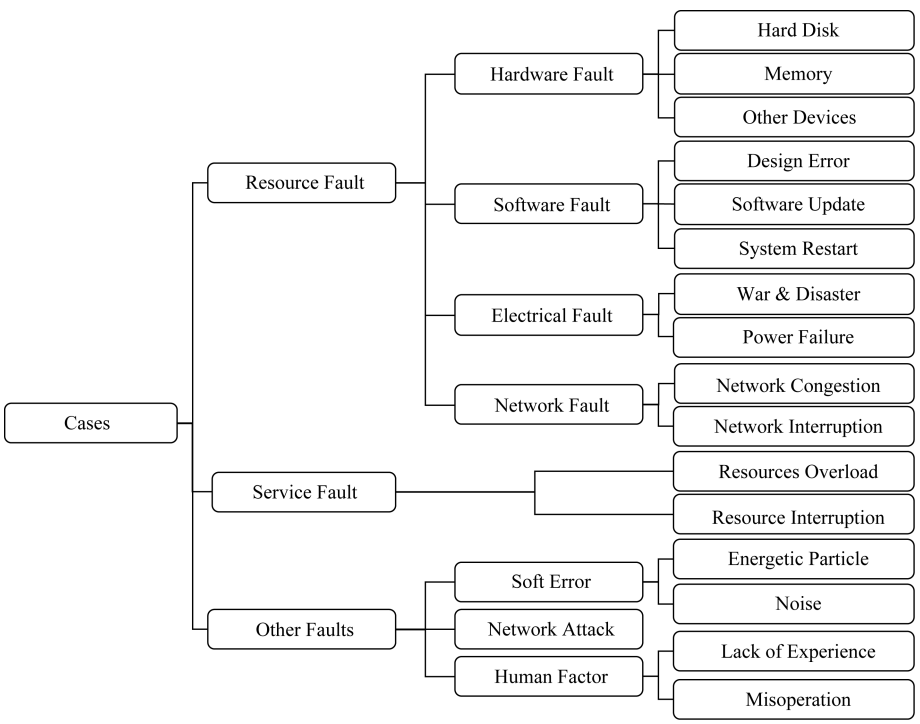

Fig. 2 Common causes of faults in cloud computing system

图 2 云计算系统中常见的故障起因

1.3.1 硬件故障

硬件故障是指由于硬件设施如硬盘、内存等设备不能正常工作导致的系统失效现象.在数据中心发生的所有故障中,大约50%的故障是由硬件引起的[23].随着数据中心的规模和使用时间在增加,硬盘故障的发生频率也在不断增长.据Google发布的报告称[4],硬盘驱动器和内存模块是2007年修复最普遍的2个模块.Vishwanath等人[24]的研究表明,在硬件故障中,有78%的故障是由磁盘驱动器产生的,并且随着使用时间的增加,硬盘驱动器故障的发生次数呈指数规律增长.因此,定期更换磁盘或者使用冗余磁盘阵列可以显著降低磁盘故障的发生概率,增加系统的可靠性.

1.3.2 软件故障

随着云计算系统中的系统和软件变得日益复杂,软件故障已经成为了系统崩溃的一个重要原因.软件故障主要来源于软件的设计错误、更新失败以及系统重启可能带来的功能失效等.在云计算系统中,大约存在40%的服务会由于软件故障而被中断[25].2013年10月份,交易算法中的一个错误导致Knight Captical公司基于云的股票交易软件停止运行45 min,给公司带来了4 400万美元的经济损失[26].有时候,软件更新升级过程中的一个不可预期的错误,也有可能引起整个系统崩溃.2013年,微软的云服务因此瘫痪16 h,据说是因为他们在数据中心的某个区域执行防火墙的常规更新时,某个错误导致了整个系统的崩溃[27].另一个服务中断事例发生在2015年1月份,微软的搜索引擎Bing在代码更新过程中宕机20 min[28].发生故障后,微软的回滚机制没有发生作用,这使得原来在相关联的服务器上的其他服务也被迫关闭.根据Proffitt[29]的调查称,大约20%的重启会由于数据不一致产生失败事件.当然,软件中存在的内存泄漏、不确定的线程、数据误操作、存储空间碎片化等原因也可能造成其他的一些系统故障或系统性能下降.

1.3.3 电力故障

在云计算数据中心中,大约会有33%的服务会由于电力故障而发生服务被迫中断的现象,这在自然灾害或战争时期很容易发生.2012年,在云计算数据中心中发生的所有27次主要的电源中断问题中,有6次是由飓风沙暴造成的[30].2011年,大规模海啸使得日本整个国家在很长一段时期内都处于电力危机中,并且所有客户服务都被中断[31].电力故障的另一个主要原因是不间断电源系统发生故障,它造成了大约25%的电力中断,单次故障会造成大约1 000美元的损失[5].

1.3.4 网络故障

在分布式计算架构中,尤其在云计算系统中,所有的服务都由通信网络所支撑,服务器之间所有的信息都经过网络进行交换和存储.底层网络的中断也可能导致云计算服务的中断.对一些基于云的实时应用来说,网络的性能通常起到了关键的作用.一个很小的网络拥塞可能引起网络传输延迟,使得系统提供的服务违反服务等级协议(SLA).在所有的服务故障中,一部分故障是由网络连接中断引起的,并且网络服务的中断有可能是由物理原因造成的,也有可能是逻辑原因造成的,如加密、带宽等因素.

1.3.5 服务故障

在云计算系统中,无论是否发生资源故障,服务故障都有可能会发生.Dai等人[32]的研究表明,服务故障的发生与提交作业所处的阶段(如作业的请求阶段和执行阶段)是密切相关的.一方面,在作业的请求阶段,用户提交的含有特定服务需求的作业都被保存在就绪队列中.在这个阶段中,资源过载(如服务请求的高峰时间)可能导致服务请求超时,此时用户会访问不到服务.在这种情况下,虽然系统底层资源运行良好,但是它们不能完全容纳全部的请求,从而导致服务故障的发生.另一方面,在作业的执行阶段,作业会被提交到底层的物理资源上,因而资源中断也有会导致服务被中断.

1.3.6 软错误

随着CMOS技术的持续发展和处理器电压的不断调节,软错误已经成为现代计算机系统的一个重要的关注点[33].由于高能粒子、噪声和硬件老化等原因,硬件电路中可能会发生瞬时故障和间歇式故障,这些故障会导致软错误的发生[34].随着软错误在整个系统中传播,它们可能表现为不同形式的系统失效现象,如错误的输出或系统崩溃.在云计算系统级别,这个问题会变得更加严重[4,22,35-37],尤其在由普通商业计算机构成的云计算系统中[38].研究人员已经见证了很多在云中或网格系统中运行科学计算任务产生很高故障率的情况[39-40].尽管不能完全避免系统中产生的软错误,但是系统设计人员可以通过故障预测以及容错措施来消除软错误给服务带来的影响.

1.3.7 网络攻击

近年来,网络攻击成为数据中心故障发生概率

迅速增长的主要原因之一.根据 Ponemon 研究中心发布的报告称,2010 年数据中心大约有 2%的故障是由网络攻击造成的,这个比例在 2013 年增加到 18%,2016 年的最新数据比例为 22%[5].由网络攻击造成的宕机平均损失为 822 000 美元[41].IBM 针对网络安全智能的报告称,55%的网络威胁都来自于可以访问系统的人,如企业的内部员工[42].

1.3.8 人为因素

和网络攻击一样,人为因素在云计算系统的故障起因中也占据了非常大的比重 (22%),由人为因素引起的故障需要的平均代价为 489 美元[5].例如,2015 年 5 月由于员工错误删除了生产服务器上的执行代码,携程旅行网的官网和客户端应用同时崩溃数小时.Schroeder 和 Gibson[43]的研究表明,缺乏经验和操作失误是人为错误发生的主要原因,并且在云计算基础设施部署的早期,人为错误所占的比重非常大.因此,云计算系统的管理人员获取更多经验可以有助于降低人为错误发生的概率.

2 云计算系统的可靠性评估

为了保障云计算系统的可靠性,研究人员需要设计适当的方法和策略来消除故障对系统行为的影响.但是在设计具体的方法之前,研究人员首先需要根据实际需求确定系统可靠性的评估标准.因此,很多学者提出了不同的指标与建模方法来评估云计算系统的可靠性.本节将从衡量系统可靠性的时间指标、云资源系统的可靠性以及云服务系统的可靠性 3 个方面来介绍与可靠性评估相关的工作.

2.1 衡量系统可靠性的时间指标

通常情况下,对提供云计算服务的服务提供商而言,可靠性高的云计算系统一般都具有的特点有:故障的发生次数少、正常工作时间长、在故障发生后需要的修复时间短.在长期的工程实践中,工程人员总结了 3 个时间指标来衡量云计算系统可靠性:平均无故障时间、平均故障修复时间和平均故障间隔时间[44],这些指标体现了系统在规定时间内保持正常运行状态的能力.

1) 平均无故障时间

平均无故障时间(mean time to failure, MTTF)是指系统从开始正常工作到故障发生时所经历的时间间隔的平均值,即系统无故障运行的平均时间.系统的平均无故障时间越长,说明系统的可靠性越高.

2) 平均故障修复时间

平均故障修复时间(mean time to repair, MTTR)是指从系统发生故障到故障维修结束并且可以重新正常工作所经历的时间间隔的平均值.系统的平均无故障时间越长,意味着系统的恢复性能越好.

3) 平均故障间隔时间

平均故障间隔时间(mean time between failure, MTBF)是指系统发生相邻 2 次故障所经历的时间间隔的平均值.系统的平均故障间隔时间越长表示系统具有较高的可靠性并且具有较强的正确工作性能.

2.2 云资源系统的可靠性

在云计算环境中,服务提供商需要管理各式各样的资源组件,如处理器、内存模块、存储单元、网络交换机等.组件越多,云计算系统失效的可能性越大.如果服务提供商了解各种资源组件的失效特征,那么这将可以帮助他们更好地管理计算资源来使系统具备容错机制并提供高性能服务.文献[45]分别从软件组件(包括进程、虚拟机和管理程序)失效和硬件组件(即服务器节点)失效以及组件失效之间的关联关系研究了云计算系统的可靠性.

如图 3 所示,假设一个云计算资源系统由一个含有 ℓ 个进程的应用程序和部署在 k 个服务器节点上的 s 个虚拟机构成,每个节点上运行着一个管理程序和不同数量的虚拟机.当任意 1 个节点或软件组件失效时,系统也将失效.对于硬件节点失效的情况,失效节点将被新节点替换并且系统将被重启;而对于软件组件失效的情况,虚拟机将被重启,然后系统继续运行.

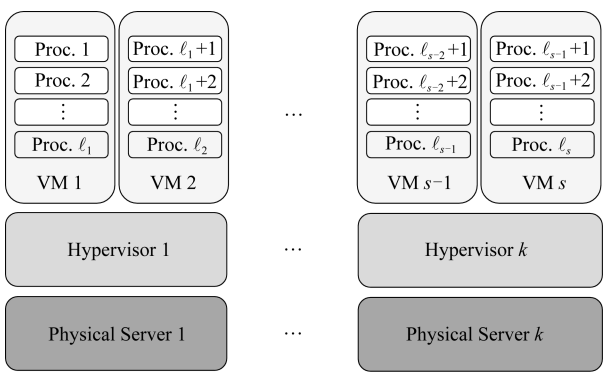

Fig. 3 Architecture of cloud resource system

图 3 云资源系统的架构

文献[45]中将节点从第 j 个节点被替换或某个虚拟机由于失效而被重启后到节点首次发生故障所经历的时间称为节点的无故障时间(time to failure, TTF).文献[45]中对系统中每个节点的失效特征的

假设为:1)每个节点的 TTF 遵循韦伯分布;2)运行在某个虚拟机上进程的 TTF 遵循指数分布;3)节点首次失效会中断整个应用程序;4)节点失效后,失效节点将被新节点替换,然后系统会被重启运行.

对于含有 $n(n=\ell+s+k)$个软件组件(进程、虚拟机和管理程序)的 k 节点系统,文献[45]将系统的可靠性定义为系统在第 j 次重启后存活到时刻 x 的概率并将其表示为 $R_j(x)$.$R_j(x)$的值越大,系统的可靠性越高.由于系统中可能存在多种组件之间的失效组合,文献[45]首先假设硬件组件失效和软件组件失效之间相互独立,然后实际考虑了 4 种主要情况:

1) 如果软件组件失效相互独立,硬件失效也相互独立,那么云系统的可靠性可以表示为

$$R_j(x)=\bar{F}(x)=P(X_1>x,X_2>x,\cdots,X_{k+n}>x)=\exp\Big\{-\sum_{i=1}^{k}\lambda_i x_i'\Big\},$$

其中,X_i 表示组件 i 的存活时间;如果第 i 个节点在第 j 次重启时被替换,则 $x_i'=x^c$ $(c>0)$,否则 $x_i'=-t_{ij}^c+(t_{ij}+x)^c$;$\lambda_i$ 是组件 i 的失效率,γ_v 是与组件 v 相关的参数.需要注意的是,由于硬件失效独立性和软件失效独立性的存在,λ_i 和 γ_i 都不为 0.

2) 如果软件失效不相互独立,而硬件失效相互独立,那么云系统的可靠性可以计算为

$$R_j(x)=\exp\Big(-\sum_{i=1}^{k}\lambda_i x_i'-\sum_{v=1}^{n}\gamma_v x-\sum_{\substack{v,w=1\\v<w}}^{n}\gamma_{v,w}x-\sum_{\substack{v,w,z=1\\v<w<z}}^{n}\gamma_{v,w,z}x-\cdots-\lambda_{1,2,\cdots,n}x\Big),$$

其中 $\lambda_{i,j,\cdots}$ 是组件 $i,j,\cdots$的联合失效率;$\gamma_{i,j,\cdots}$ 是与组件 $i,j,\cdots$相关的联合参数.需要注意的是,由于硬件失效独立性的存在,λ_i 不为 0.

3) 如果软件失效相互独立,而硬件失效不相互独立,那么云系统的可靠性可以计算为

$$R_j(x)=\exp\Big(-\sum_{i=1}^{k}\lambda_i x_i'-\sum_{\substack{i,s=1\\i<s}}^{k}\lambda_{i,s}\max\{x_i',x_s'\}-\sum_{\substack{i,s,l=1\\i<s<l}}^{k}\lambda_{i,s,l}\max\{x_i',x_s',x_l'\}-\cdots-\lambda_{1,2,\cdots,k}\max\{x_1',x_2',\cdots,x_k'\}-\sum_{v=1}^{n}\gamma_v x\Big).$$

4) 如果软件失效和硬件失效都不相互独立,那么云系统的可靠性为

$$R_j(x)=\exp\Big(-\sum_{i=1}^{k}\lambda_i x_i'-\sum_{\substack{i,s=1\\i<s}}^{k}\lambda_{i,s}\max\{x_i',x_s'\}-\sum_{\substack{i,s,l=1\\i<s<l}}^{k}\lambda_{i,s,l}\max\{x_i',x_s',x_l'\}-\cdots-\lambda_{1,2,\cdots,k}\max\{x_1',x_2',\cdots,x_k'\}-\sum_{v=1}^{n}\gamma_v x-\sum_{\substack{v,w=1\\v<w}}^{n}\gamma_{v,w}x-\sum_{\substack{v,w,z=1\\v<w<z}}^{n}\gamma_{v,w,z}x-\cdots-\gamma_{1,2,\cdots,n}x\Big).$$

由于文献[45]中的工作假设运行在 k 个物理服务器上的所有 n 个软件组件的失效模式都遵循相同的概率分布,但是这在现实应用场景中是不实际的.云计算系统中存在着新的软件组件和重用的软件组件,并且不同的软件组件是在不同的环境中被开发出来的,因此所有 n 个软件组件并不会遵循相同的概率分布.因此,文献[46]在文献[45]工作的基础之上对其进行拓展.文献[46]中假设云计算环境中的新组件会遵循指数分布,这些新组件会由于更容易产生 bug 而更容易失效;而重用的或旧的软件组件依据它们的到达时间遵循延迟的指数分布.

2.3 云服务系统的可靠性

文献[32]开发了一个云服务系统,其中包含一个云管理系统(cloud management system, CMS).云管理系统能够实现 4 种不同的功能:1)管理由来自不同用户的作业请求组成的请求队列;2)管理互联网上的计算资源(如个人计算机、集群、超级计算机等);3)管理网络上的数据资源(如数据库、网页等);4)划分请求为子任务,并将子任务分配到多个可以访问数据资源的不同计算资源上.当某个用户请求云服务时,CMS 系统首先使用工作流来描述云服务包含的子任务、子任务需要的数据资源以及资源之间的依赖关系,然后将各个子任务分布到各个可以访问数据资源的计算资源节点上.

在云服务运行的过程中,存在各种各样因素会影响云服务的可靠性,具体包括:请求队列溢出、请求超时、数据资源缺失、计算资源缺失、软件失效、数据库失效、硬件失效和网络失效.在文献[32]中,云服务的可靠性被定义为:在特定条件下云服务在用户指定的时间范围内顺利完成的概率.因此,云服务能够顺利完成执行的条件为:用户的作业请求能及时被调度器处理、云服务的子任务能顺利完成、子任务请求的计算/数据资源可用、服务运行期间网络是通畅的.文献[32]中将服务的失效过程分为:请求阶段失效和执行阶段失效.由于第 1 种失效发生在作业请求被成功分配到计算/数据资源之前,而第 2 种失效发生在作业请求被成功分配之后、子任务执行

过程中,所以这 2 个阶段的失效可以是相互独立的.然而,每个阶段中的不同失效是相互关联的.因此,云服务可靠性建模可以被分为请求阶段可靠性建模与执行阶段可靠性建模.

假设 CMS 系统中有 S 个同构服务器服务用户的请求,每个服务器处理单个请求的服务时间满足参数为 μ_r 的指数分布.在请求阶段,用户会为请求的服务设置一个约束时间,即从提交作业请求到作业完成时允许花在该服务上的时间.如果一个作业请求在约束时间之前没有被调度器服务,那么该请求将被丢弃,丢弃率表示为 μ_d;假设请求队列的容量为 N,作业请求的到达行为遵循到达率为 λ_a 的泊松分布.因此,请求队列可以使用如图 4 所示的 Markov 过程建模,其中状态 $n(n=0,1,\cdots,N)$ 表示请求队列中请求的数量.在图 4 中,从状态 n 到状态 $n+1$ 的转移概率为 λ_a.当队列处于状态 N 时,新到达的请求会使请求队列溢出,因此该请求将被丢弃并且队列依然处于状态 N.

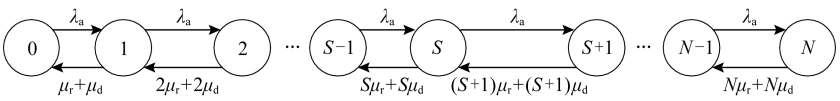

Fig. 4 Markov model for the request queue

图 4 请求队列的 Markov 模型

请求阶段失效包括请求队列溢出和请求超时.请求队列溢出是指用户发出的请求数量大于请求队列的最大容量.请求队列不会溢出的概率为 $R_{\text{overflow}}=\sum_{n=0}^{N-1}q_n$,其中 $q_n(n=0,2,\cdots,N)$ 表示系统处于状态 n 的稳定概率,它可以通过 Chapman-Kolmogorov 方程计算出来.请求超时是指已被接受的请求在请求队列中等待服务的时间超过用户设置的约束时间 T_d.请求超时事件不会发生的概率可以计算为

$$Pr(t<T_d)=\int_0^{T_d}f_n(t)\mathrm{d}t,$$

其中 $f_n(t)$ 表示处理完 n 个请求需要的等待时间的概率密度函数.当 $n<S$ 时,新到达的请求不需要等待便可以立刻被 CMS 系统服务,所以此时请求阶段的可靠性为 $\sum_{n=0}^{S-1}q_n$;当 $n\geqslant S$ 时,只有当请求队列不溢出且队列中的请求不超时,请求阶段才是可靠的,其可靠性为 $\sum_{n=S}^{N-1}q_n\int_0^{T_d}f_n(t)\mathrm{d}t$.因此,请求阶段的可靠性计算为

$$R_{\text{request}}=\sum_{n=0}^{S-1}q_n+\sum_{n=S}^{N-1}q_n\int_0^{T_d}f_n(t)\mathrm{d}t.$$

执行阶段失效包括数据资源缺失、计算资源缺失、软件失效、数据库失效、硬件失效和网络失效.由于服务执行阶段涉及到 3 种类型的节点:硬件节点、软件节点和网络连接节点.由于与某个子任务相关联的所有元素(节点和链路)可以构成一个子任务生成树(subtask spanning tree, SST),并且一个 SST 可以被视为由多个子任务最小生成树(minimal subtask spanning tree, MSST)组合而成,其中每个 MSST 表示顺利完成这个子任务的最小元素集合(任何一个元素失败都可能导致子任务失败).假设第 i 个元素的失效率为 $\lambda(element_i)$,那么其可靠性为

$$R(element_i)=\exp(-\lambda(element_i)\times T_w(element_i)),$$

其中,符号 $T_w(element_i)$ 表示云服务中第 i 个元素的工作时间.软件节点的工作时间为软件程序在某台机器上的运行时间,其与软件程序的工作量和处理器的处理能力相关;网络连接节点的工作时间为在某个网络连接上传输数据需要的时间,其与传输的数据量和网络带宽相关;硬件节点的工作时间为运行在某台机器上所有软件程序的工作时间与连接该机器的所有网络传输时间之和.类似于 MSST,成功完成某个服务需要的所有元素集合可以表示为最小执行生成树(minimal execution spanning tree, $MEST$).如果 $MEST$ 中所有元素都执行成功,那么 $MEST$ 便可以被视为可靠的.$MEST$ 可靠性为

$$R_{MEST}=\prod_{i\in MEST}\exp(-\lambda(element_i)\times T_w(element_i)).$$

如果某个云服务有 N 个 $MEST$,那么任意一个 $MEST$ 的成功执行便意味着该云服务在执行阶段顺利完成.因此,执行阶段的可靠性为

$$R_{\text{execute}}=Pr\left(\bigcup_{i=1}^{N}MEST_i\right).$$

最终,如果云服务的请求阶段和执行阶段都是可靠的,那么该云服务便可以被认为成功执行结束.因此,云服务的可靠性 R_{service} 可以被计算为请求阶段和执行阶段可靠性的乘积:

$$R_{\text{service}}=R_{\text{request}}\times R_{\text{execute}}.$$

类似于文献[32]中的工作,文献[47]在研究云计算服务系统运行过程的基础上,将云计算服务系统可靠性定义为云计算系统在给定的条件和规定时间内完成用户请求的能力.文献[47]中首先对云计算服务系统分阶段建模,包括用户服务请求达到CMS系统的任务请求阶段、调度系统对服务请求的子任务进行调度的调度阶段、子任务开始被执行到全部完成的执行阶段,然后建立了云计算服务系统的可靠性模型.

如果使用$R(t)$表示云服务系统的可靠性,即用户提交的服务请求能够被云计算系统在指定的时间t内成功完成的概率;m表示用户提交的服务请求在云计算系统中执行时被划分成子任务的个数;$P_{\mathrm{B}}^{(m)}$表示服务请求队列已满时用户的服务请求被阻塞的概率;$T_{\mathrm{SRT}}^{(m)}$表示用户请求从提交到最终执行完成的时间,即服务响应时间,那么云计算服务系统的可靠性可以被计算为

$$R(t)=(1-P_{\mathrm{B}}^{(m)})\times Pr\{T_{\mathrm{SRT}}^{(m)}<t\}.$$

3 云计算系统故障管理技术研究

为了使云计算系统能够提供可靠的服务,研究人员在设计云计算系统的架构时应该考虑到故障的管理,所有针对运行良好的云计算环境设计的架构和技术都应当能够适用于容易发生故障的云环境.为了管理云环境中容易发生的资源故障以及保证服务的可用性,研究人员已经设计了各种技术与方法,使云计算系统在容易发生故障的环境中能够提供可靠的服务.由于云计算使用了面向服务的架构,所有的技术与方法都应该从服务可靠性的角度开始着手设计.如图5所示,云计算系统中主要的故障管理技术分为3类:故障消除、故障容忍、故障预测与避免.

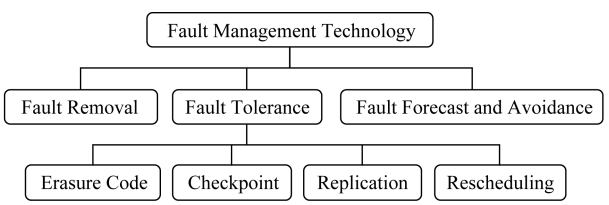

Fig. 5 Main fault management techniques in cloud computing

图5 云计算中主要的故障管理技术

3.1 故障消除

故障消除技术是运用传统的软件测试与验证方法发现并移除云计算系统中潜在的故障以提高系统服务可靠性的一种方法,主要针对于云计算中的软件故障.软件测试对于移除软件中的故障、提高系统的容错能力具有至关重要的作用,通过测试的系统组件通常可以被认为已经具备了较高程度的可靠性.文献[48-49]表明,在分布式环境中部署软件之前,如果软件没有经过充分的测试,也有可能会产生故障,带来巨大的经济损失.

为了保证软件系统的可靠性,研究人员已经设计了各种各样的方法和工具来检测云计算系统中潜在的故障,其中比较典型的一种方法是故障注入.它是一种通过向代码中引入故障来提高软件系统测试覆盖率的软件测试技术,经常和压力测试一起被用来测试软件系统的鲁棒性.故障注入常被应用在分布式系统部署的早期[50-52],也可以被用于普通的软件系统[53].Hoara等人[54]提出了故障注入语言FAIL及其集群版本的故障注入工具FCI,可以用来在分布式应用中进行故障注入.FAIL语言不仅可以让用户不再编写低级代码,以简单的方式设计复杂的故障场景;它还可以被用于构造一些概率性场景、确定性场景和可再生场景,让用户能够研究系统在某些特定情景下的行为模式.他们的故障注入工具FCI包含一个编译器、运行时库和在分布式应用中进行故障注入的中间件平台,支持与其他编程语言直接交互,不需要用户修改应用软件的源代码,对应用运行时间的影响特别小.Monnet等人[55]开发了一个故障注入工具,该工具可以让开发者在不改变应用程序代码的条件下向应用中注入故障.实验结果表明,他们的工具能够以可再生的方式提供精确的控制粒度,发现大规模网格环境中的独立故障和有依赖关系的故障.

除了故障注入外,研究人员还设计了一些其他的测试技术.Song等人[56-57]利用大量的系统消息流创建系统组件依赖图,通过分析网格系统来发现关键的hub组件.由于包含故障的hub组件容易影响到整个系统的操作,因而它们具备较高的测试优先级,对hub组件进行检测有助于更快地发现系统中存在故障的区域.文献[58]中利用认证编译器产生能够在网格资源中执行的机器码,生成的机器码中包含了可验证的子句,它可以在软件系统的运行过程中自动验证代码属性.该方法不仅能验证代码的安全性,也可以检测系统中的故障和可疑代码.

文献[48-58]所述的方法都是利用测试技术对系统进行故障检测的,它们可以被用于确定哪些系统

组件应该被优先测试与验证,以及哪些测试(如组件测试、集成测试、交互测试等)具备更低的测试开销.因此,以前适用于软件组件的故障预测研究也可以为识别大规模云计算系统中的故障可能区域提供很好的基础.然而,由于在特别复杂的云计算系统中很难有效地实施故障消除技术,并且虚拟机的失效现象是不可避免的[59],所以故障消除技术通常也很难完全发现系统中潜在的故障.因此,和后文中将要提到的故障容忍方法相比,软件测试与验证在故障管理中获得的关注度并不是特别高.

3.2 故障容忍

对云计算系统而言,容错已经成为了数据存储、传输和计算必不可少的需求之一.提供数据存储级的容错相对直接[60],RAID[61]和 Amazon 的 EBS (elastic block storage)[62]都通过局部冗余存储来提高磁盘故障的容忍程度.目前复制和检查点是 2 种广泛使用的故障容忍机制[63].纠删码[64]作为存储系统容错的主要方法受到了越来越多的重视.在数据传输方面,目前很多知名的网络协议都可以通过被重新设计来恢复云计算中的数据传输错误,例如,微软的云计算服务平台 Azure 就使用了基于队列的消息检索/重传机制来保证数据的传输可靠性[65].在云计算系统中,处理计算过程中发生的错误与资源故障和服务故障都有关系.

尽管用户可以继续使用传统的算法级别容错方法来处理错误[66],但由于公有云对用户呈现出不透明性[67],由云计算服务提供商来提供容错机制则更加有效.在故障发生时,云计算系统应当能采取合理的措施让服务从故障中恢复.目前比较流行的高级容错措施包括空间冗余和时间冗余.空间冗余的概念可以追溯到 3 模块冗余系统(triple modular redundancy, TMR)[68].当空间冗余概念首次出现在云计算环境中时,模块便被替换成了虚拟机副本或处于锁步状态的任务副本[69-70].目前,这种实现已经被具有高可用性的云计算解决方案所采用,如 VMware[71]和 VGrADS[72].时间冗余通常会引起任务的重新执行.任务在发生错误时可以被重新提交[39,73],或者回滚到之前的一个正确状态并重新执行[74].类似地,检查点方法周期性地将主虚拟机的状态保存为 1 个检查点,并存储在另一个虚拟机上,例如,Remus[75]中的虚拟机对就是以 leader-follower 的方式执行的.Kemari[76]和 HydraVM[77]是使用虚拟机检查点的另一个例子.通过故障容忍技术,可以有效地降低硬件故障、网络故障以及软错误等故障所带来的影响.表 1 展示了本节将要描述的 4 种容错技术.

Table 1 Comparison of Four Fault Tolerance Technologies

表 1 4 种容错技术的对比

Tolerance Technology	Working Principle	Scope of Application	Advantages	Disadvantages
Erasure Code	Spatial Redundancy	Storage,Communication	Good Data Protection	Large Space Overhead
Checkpoint	Time Redundancy	Process	High Reliability	Large Time and Space Overhead
Replication	Spatial Redundancy	Storage,Process	High Fault Tolerance Rate	High Computing and Storage Requirements and Low Storage Efficiency
Rescheduling	Time Redundancy	Process	Simple to Implement	Extra Delay

3.2.1 纠删码

纠删码[78]的设计思想是将数据对象划分成若干个数据块,然后在每个数据块中添加额外的信息并对数据块进行编码.当存储数据块的某个节点失效时,利用编码后的数据块的一个子集便可以恢复出完整的原始数据集.通常情况下,纠删码可以被描述为三元组(n,k,r),其中 k 为编码前数据对象被划分的块数,n 为编码后数据块的个数,r 为 1 个不小于 k 的整数.纠删码将 k 个原始数据块编码生成 n 个数据块,其中包含 $n-k$ 个冗余数据块.因此,编码后的数据具有 $n-k$ 个数据块的容错能力,当发生不多于 $n-k$ 个数据块损坏或丢失时,我们便可以使用存活的 $r(r\geqslant k)$ 个数据块修复数据.下面介绍基于纠删码的备份与恢复方案[79].

假设需要存储的数据对象 $\boldsymbol{F}=(F_0,F_1,\cdots,F_{k-1})$可以被划分成长度固定的 k 个数据块 $F_i(0\leqslant i\leqslant k-1)$,纠删码$(n,k,r)$将 k 个原始数据块编码成$n(n>k)$个数据块.如果编码时选择 n 个长度为k 的向量 $\boldsymbol{g}_i=(g_{i0},g_{i1},\cdots,g_{i(k-1)})(0\leqslant i\leqslant n-1)$,并且其中任意 k 个向量都线性无关,那么编码后的数据块 $\boldsymbol{D}_i(0\leqslant i\leqslant n-1)$为

$$\boldsymbol{D}_i=\boldsymbol{g}_i\cdot\boldsymbol{F}=g_{i0}\cdot F_0+g_{i1}\cdot F_1+\cdots+g_{i(k-1)}\cdot F_{k-1}.$$

并且这 n 个编码后的数据块中的任意 k 个都

可以被用来重构原始数据.如果:

$$\boldsymbol{G}=(g_{ij})(0\leqslant i\leqslant n-1,0\leqslant j\leqslant k-1),$$

那么可以得到:

$$\boldsymbol{G}\cdot\begin{pmatrix}F_0\\F_1\\\vdots\\F_{k-1}\end{pmatrix}=\begin{pmatrix}\boldsymbol{D}_0\\\boldsymbol{D}_1\\\vdots\\\boldsymbol{D}_{n-1}\end{pmatrix},$$

所以 1 个纠删码(n,k,r)可以被表示为 $\boldsymbol{D}=\boldsymbol{G}\cdot\boldsymbol{F}$.其中 $\boldsymbol{F}=(F_0,F_1,\cdots,F_{k-1})$是原始数据对象;$\boldsymbol{D}=(\boldsymbol{D}_0,\boldsymbol{D}_1,\cdots,\boldsymbol{D}_{n-1})$是编码后的数据;$\boldsymbol{G}$ 是 $n\times k$ 的矩阵,称为纠删码(n,k,r)的生成矩阵.

需要注意的是:k 与 n 的比值 k/n 称为该纠删码的编码率;$(n-k)/k$ 称为该编码的冗余度.

纠删码冗余机制要求 $\boldsymbol{G}$ 的任意 k 行组成的子矩阵 $\boldsymbol{G}'$均可逆,所以使用任意 k 个数据块都可以重构出原始的 k 个数据块.除此之外,系统码的使用使得任何生成矩阵可以通过运算转化为

$$\boldsymbol{G}=\begin{pmatrix}\boldsymbol{I}_k\\\boldsymbol{P}\end{pmatrix}=\begin{pmatrix}1&0&\cdots&0\\0&1&\cdots&0\\\vdots&\vdots&&\vdots\\0&0&\cdots&1\\P_{0,0}&P_{0,1}&\cdots&P_{0,k-1}\\P_{1,0}&P_{1,1}&\cdots&P_{1,k-1}\\\vdots&\vdots&&\vdots\\P_{n-k-1,0}&P_{n-k-1,1}&\cdots&P_{n-k-1,k-1}\end{pmatrix},$$

其中,$\boldsymbol{I}_k$ 为 $k\times k$ 的单位矩阵,$\boldsymbol{P}$ 为$(n-k)\times k$ 矩阵.

在所得的编码数据中,其前 k 位由单位矩阵 $\boldsymbol{I}_k$ 决定,因而和原始数据 $\boldsymbol{F}$ 中的各位数据相同,而其余的 $n-k$ 位被称为校验数据,是 k 个原始数据块的线性组合.

采用这种系统码编码数据块时,会生成较多的分块,这些分块中不仅包含了数据分块还包含了冗余分块,这样数据分块和冗余分块的数量便大于原始的数据分块.此时,系统便可以将数据分块和冗余分块发送给存储节点进行存储.当用户读取数据时,只要获取其中一定数量的分块,无论这些分块是数据分块还是冗余分块,都可以通过这些分块重构出原始数据对象.由于该算法在生成冗余分块时引入了纠错思想,因而其可以极大地避免了恢复数据与原始数据存在差异或无法重构的现象.

当需要恢复数据时,在所有的 $r(r\geqslant k)$个存活数据块中选择任意 k 个数据块便可以重构原始数据.由于生成矩阵中每个行向量都与 1 个数据块一一对应,当从 r 个数据块中选取 k 个数据块进行解码运算时,这 k 个数据块对应的行向量可以组成 k 阶方阵 $\boldsymbol{Q}$.如果将这 k 个数据块记作$(\boldsymbol{R}_0,\boldsymbol{R}_1,\cdots,\boldsymbol{R}_{k-1})$,$\boldsymbol{Q}$ 记作$(\boldsymbol{q}_0,\boldsymbol{q}_1,\cdots,\boldsymbol{q}_{k-1})$,那么 $\boldsymbol{Q}$ 与原始 k 个数据块相乘便可以得到这些存活下来的 k 个数据块,即:

$$\boldsymbol{Q}\cdot\begin{pmatrix}F_0\\F_1\\\vdots\\F_{k-1}\end{pmatrix}=\begin{pmatrix}\boldsymbol{R}_0\\\boldsymbol{R}_1\\\vdots\\\boldsymbol{R}_{k-1}\end{pmatrix},$$

由于生成矩阵中任意 k 个向量线性无关,即 $\boldsymbol{Q}$ 可逆,因而使用 $\boldsymbol{Q}$ 的逆矩阵 $\boldsymbol{Q}^{-1}$便可以计算出原始数据 $\boldsymbol{F}$,其计算过程为

$$\boldsymbol{F}=\boldsymbol{Q}^{-1}\boldsymbol{Q}\cdot\begin{pmatrix}F_0\\F_1\\\vdots\\F_{k-1}\end{pmatrix}=\boldsymbol{Q}^{-1}\begin{pmatrix}\boldsymbol{R}_0\\\boldsymbol{R}_1\\\vdots\\\boldsymbol{R}_{k-1}\end{pmatrix}.$$

在实际应用中,节点的失效可能引起原始数据块的损坏,也可能引起冗余数据块的损坏.因此,数据块的恢复过程分为 2 个部分.

1) 原始数据块.在恢复原始数据时,使用 $\boldsymbol{Q}^{-1}$中的第 i 行与 $\boldsymbol{R}$ 进行数量积运算,所得的积再做异或运算,即可得到损坏的原始数据块 $\boldsymbol{D}_i(0\leqslant i\leqslant k)$:

$$\boldsymbol{D}_i=\boldsymbol{Q}_i\cdot\boldsymbol{R}=Q_{i,0}^{-1}\cdot\boldsymbol{R}_0\oplus Q_{i,1}^{-1}\cdot\boldsymbol{R}_1\oplus\cdots\oplus Q_{i,k-1}^{-1}\cdot\boldsymbol{R}_{k-1}.$$

2) 冗余数据块.恢复由编码过程生成的冗余数据 $\boldsymbol{D}_k,\boldsymbol{D}_{k+1},\cdots,\boldsymbol{D}_{n-1}$时,将编码矩阵 $\boldsymbol{P}$ 中对应行 $\boldsymbol{P}_i$ 与 k 个数据块进行数量积运算,所得积再做异或运算,便得到损坏的冗余数据块 $\boldsymbol{D}_i(0\leqslant i\leqslant n-k-1)$:

$$\boldsymbol{D}_{k+i}=\boldsymbol{P}_i\cdot\boldsymbol{R}=P_{i,0}\cdot\boldsymbol{R}_0\oplus P_{i,1}\cdot\boldsymbol{R}_1\oplus\cdots\oplus P_{i,k-1}\cdot\boldsymbol{R}_{k-1}.$$

使用纠删码能够提供优化的数据冗余度,防止数据丢失,恰当地使用纠删码可以提高空间的利用效率并获得较好的数据保护效果.目前该方法在通讯方面已经得到了广泛应用.此外,将纠删码引入云存储系统中代替副本备份策略,可以有效提高云存储系统的可靠性.但是由于纠删码需要存储额外的冗余数据,因而该方法存在一定的存储空间开销.

3.2.2 检查点

检查点技术(checkpointing)是目前被广泛使用的一种故障容忍技术.检查点技术是将正在运行的进程的当前状态周期性地保存在某个稳定存储资源上,每次保存的状态则称之为一个“检查点”.当

检测到任务发生失效事件时,进程的状态会被回滚到最近保存的检查点状态并从该状态开始重新执行.如图 6 所示,在任务的执行过程中,每隔一段时间系统会将任务的状态保存为一个检查点,在相邻检查点之间的时间间隔叫做检查点区间(checkpoint interval).当一个检查点区间结束时,系统会使用某种错误检测技术检查任务的执行是否正确.当检测到任务的状态发生错误时,任务会被回滚到临近的一个检查点状态并重新执行从该检查点开始的检查点区间内的工作.由于检查点技术能够将任务从错误状态中恢复过来,目前已经有很多云计算管理套件整合了检查点机制来提供不中断的云计算服务,如 Oracle 的 UniCloud,Intel 的数据中心管理器(data center manager, DCM)等.基于检查点的工作原理,检查点技术可以被分为 3 种类型:非一致性检查点、一致性检查点和通信引导的检查点[80].

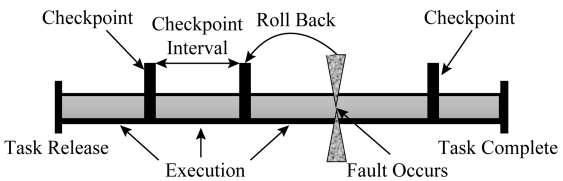

Fig. 6 Fault management using checkpoints

图 6 使用检查点进行故障管理

非一致性检查点允许属于同一应用程序的每个进程自主决定何时执行检查点操作.由于每个进程可以相互独立地在不同时刻执行检查点操作,因而这种方式的优点是进程执行检查点操作具有较大的自由度,不需要受到其他进程状态的限制.但是这种方式也存在诸多缺点,其中最大的一个缺点便是多米诺效应.当一个进程执行回滚操作时,之前接受过这个进程发送消息的进程也需要回滚.如果这些被迫回滚的进程之前也发送过消息,那么它们的回滚必然又会引起其他进程进行回滚.此时,系统就发生了多米诺效应,使得很多进程都做了无用功,甚至可能回归到计算的起点.此外,由于非一致性检查点迫使每个进程维护多个检查点,因而这种方式需要进程周期性地进行垃圾收集回收无用的检查点.

在一致性检查点中,同一应用程序的进程之间需要同步检查点来确保每个进程保存的检查点相互一致,并且所有的检查点合并起来也需要形成一致的全局状态.一致性检查点的恢复过程相对容易,每个进程总是从最近的检查点重新执行,因此该方法不易受到多米诺效应的影响.此外,一致性检查点要求每个进程在稳定存储上只需要存储一个持久检查点,因而它可以显著降低存储开销,也不需要进行垃圾收集.然而,一致性检查点的主要缺点是系统提交输出时会产生很大的延迟.

在通信引导的检查点中,它不需要同步所有的检查点,因而可以避免多米诺效应.在这个协议中,进程会执行 2 种检查点:局部检查点和强制检查点.进程可以独立地、有选择地执行局部检查点,但是必须执行强制检查点来保证恢复过程的一致性.

文献[81]表明,在容易发生故障的环境中,不采用检查点技术时成功完成某个任务所需的时间远大于采用检查点技术时完成该任务所需要的时间.虽然检查点技术能够保证云计算服务的可靠性,但值得注意的是,由于在稳定存储介质上保存检查点需要一定的时间,因此应用检查点技术必然会带来一些时间开销[82].Fu[22] 的实验表明,在像云计算这样的大规模系统中,频繁地应用检查点机制会带来巨大的额外开销.据估计,在千万亿次操作的系统中,1 个 100 h 的作业会花费约 150 h 的检查点创建开销[21];相反,如果降低使用检查点技术的频率,那么在程序发生故障后重新执行又会增加程序的总执行时间.因此,如何确定最优检查点区间并减少检查点开销已经成为众多研究人员的研究目标[83-84].例如,为了优化云计算环境中的检查点/重启机制,Di 等人[85]首先设计了一个计算公式为具备不同失效事件分布的云作业计算最优检查点数量;然后设计了一个自适应算法优化检查点带来的各种影响,如检查点开销、重启开销等;最后他们在 Google 系统的运行数据上评估了他们的方法,实验结果表明他们的方法可以降低任务的执行时间平均达 50~100 s.由于云计算底层计算基础设施具备动态性,科学工作流系统中经常会发生任务执行延迟的现象,导致大量的时序违反事件发生.对时序违反事件进行处理会消耗大量的时间,因此必须采取合理的措施,在满足时序一致性要求的同时,尽可能降低不必要的时序违反事件处理开销.Liu 等人[86] 发现,通常情况下,在某个检查点处检测到的时序违反事件带来的时间赤字是非常小的,而工作流后续活动的盈余时间(时序约束和执行时间之间的时间差)基本可以弥补该时间赤字,他们将这种现象命名为“自动恢复”.例如工作流中的某个任务由于时序违反产生 10 s 的时间赤字,但是很有可能该任务后面的任务会产生 10 min 的平均冗余时间,那么这些冗余时间便可以用来弥补该任务的时间赤字.基于这个发现,他们提出了一个自适应的时序处理点选择策略,它只选择

关键的而非所有的检查点作为处理点来解决时序违反问题.仿真实验表明,他们的方法可以在满足时序一致性和成本高效的条件下,显著避免不必要的时序违反事件的处理开销.

3.2.3 复制

复制(replication)是提供系统容错能力的另一种故障管理技术,这种方法使用多个计算资源同时运行同一任务的多个进程副本并维持相同的状态.根据副本的更新方式,复制策略可以分成2种类型:主从复制和主动复制.如图7(a)所示,在主从复制中,客户端仅仅将请求发送给主副本,只有主副本会处理用户请求.主副本在处理完成用户的请求后,会将其状态转发给所有从副本并通知它们更新状态,在所有从副本完成状态更新并向主副本发送确认消息后,主副本向客户端发送反馈.如图7(b)所示,在主动复制中,客户端会将请求发送给所有副本,所有副本在接收到请求后都会执行请求.在客户端接收到所有副本的第1个响应后,可以继续处理其他的事务.

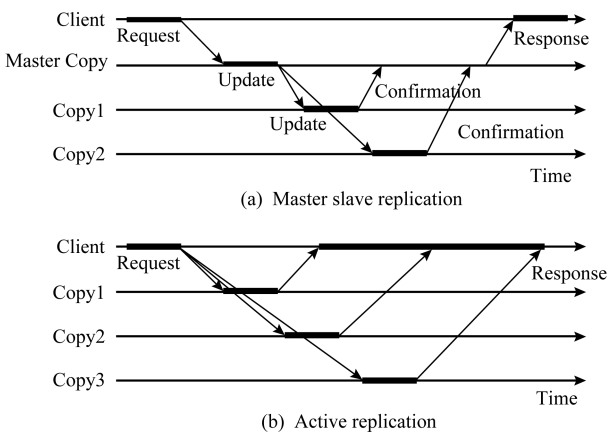

Fig. 7 Replication technology

图7 复制技术

目前已经有许多云计算服务提供商使用复制机制来提供不同级别的容错保障.微软的Azure云计算平台使用虚拟机副本来提供虚拟机级别的容错.当虚拟机发生故障时,Azure总是使用副本虚拟机来接管发生故障的虚拟机,继续执行故障虚拟机中正在运行的任务.在基础设施即服务级别,OpenStack云计算平台将文件和对象分布在不同数据中心的多个服务器的磁盘上,使用数据副本来保证数据的可靠性.在Apache Hadoop,Amazon的EBS等DFS系统中也应用了资源复制来保证系统的可靠性.此外,还有很多应用复制机制的其他例子,例如Benoit等人[87]提出了一个容错调度算法,该算法基于资源复制策略,致力于在不增加应用延迟时间的条件下,将有依赖关系的多个任务映射到异构系统中,并使其能够容忍多次处理器失效.为了考虑任务之间的通信竞争并保证具有依赖关系的一组任务的执行可靠性,Benoit等人[88]又提出了一个竞争感知的容错调度算法,该算法使用了任务复制策略,在不牺牲任务总执行时间的情况下能够容忍多次处理器失效.Wang等人[89]提出了RMSR(replication-based scheduling for maximizing system reliability)算法,它将任务通信整合到系统可靠性中,使用任务复制技术在异构系统中进行可靠性感知的任务调度.为了最大化通信可靠性,他们提出了一个算法为当前任务搜索具有最优可靠性的所有通信路径,在任务复制阶段,由用户决定任务的可靠性阈值并确定每个任务的动态副本.实验结果表明,他们的方法能够显著提高系统的可靠性.

虽然应用复制技术可以保障云计算服务可靠性,但是它对计算资源、存储资源的需求较高,同时存在存储效率低下的缺点.目前,应用该技术存在诸多挑战,其中最大的3个挑战为:1)设计算法确定副本的最优放置策略以达到最佳的容错效果;2)设计方法同步副本状态,保持副本之间的状态一致性;3)降低副本数量,减少维护可靠性需要的成本.这3个问题都是开放性研究问题,很多学者就此提出了各种各样的解决方案.

1) 副本选择与放置.在应用复制技术时,由于每个进程或数据选择副本数量的不同以及每个副本在不同计算资源上的放置方式,对复制技术能够实现的容错效果和该技术能获得的存储空间利用率具有很大的影响.因此,该问题要求设计合适的副本选择算法与最优放置策略以达到最佳的容错效果、实现最高的资源利用率.文献[90]尝试对在动态分布式系统中进行副本放置的自适应算法的容错能力及可扩展性等属性进行评估.Weissman等人[91]设计了一个副本管理系统,它可以基于用户的需求动态地分配副本资源.当某个副本失效时,用户的资源请求可以被系统透明地重新路由,测试实验验证了他们方法的有效性.文献[92]提出了一个SRIRAM研究系统,它可以在网格系统和分布式系统中自动对计算资源进行备份,提高计算资源的可用性和容错能力.Valcarenghi等人[93]提出了一个服务复制方法,他们首先让副本之间相互临近形成网络中的服务岛,然后使用混合整数规划评估不同的复制配置来决定

哪种服务岛配置具备较高的容错能力.仿真实验表明,这种方法能够发现长距离的服务间连接,降低了副本的需求量,同时也简化了副本的管理.文献[94]提出了一个用于通信公司内的计算网格资源分配系统,该系统使用动态进程复制来提供容错能力,满足服务等级协议.在 e-Demand 项目中,Townend 等人[95]提出了一个基于复制的方法来检测由多个任务组成的工作流中的错误计算,一个工作流会在多个服务副本上同时执行,同时,该方法还使用一个投票过程来选择哪个副本集合应该将结果返回给用户.实验表明,该方法可以显著提供工作流的容错能力.Genaud 等人[96]为基于 MPI 的网格环境设计了一个容错机制,它使用资源复制来提高并行环境的容错能力,实验表明这种方法具备非常好的可扩展性.

2) 副本同步.在大规模计算环境中,应用复制技术需要保持不同副本状态的一致性.然而,副本同步会产生巨大的额外开销.因此,研究人员需要设计合适的方法同步副本状态,在保持副本之间的状态一致性的同时,降低副本同步带来的开销.当前有很多研究尝试解决在云计算中应用复制机制带来的副本一致性问题[83-84],其中比较典型的算法有 Paxos 算法[97]和 Raft 算法[98].Raft 算法比 Paxos 算法更容易理解和实现.这 2 种算法都可以被用来在分布式环境中同步副本间的状态,它们无论在局部环境中还是在广域网环境中都展示出了较好的性能.Zhang 等人[99]基于主从复制方法在多个主机上复制服务,构建可靠的、高可用的网格服务.他们的方法能够在局部环境中保证较高的服务可用性,但是为了同步服务副本间的状态进行开放网格服务基础设施(open grid service infrastructure, OGSI)通知所产生的额外开销也是非常可观的.Dasgupta 等人[100]提出了一个框架将冗余副本资源的预留整合到服务等级协议中,当主副本失效时允许程序切换到从副本继续运行.仿真实验表明这个方法可以提高系统资源分配的效率.

3) 副本数量的确定.在云计算系统中,虽然采用复制策略能够提高云服务的容错能力,但过多的副本会增加云服务的成本.因此副本数量与成本之间的折中也是采用复制技术时应该考虑的问题.研究人员需要设计副本管理策略,降低副本数量,减少维护可靠性需要的成本.为了在减少云存储消耗的同时满足数据中心数据可靠性要求,Li 等人[101]提出一个成本高效的数据可靠性管理机制 PRCR,该方法通过主动检测副本的状态来减少副本的数量,可以在保证数据可靠性的同时,降低存储成本.Amazon 云也提供了低冗余存储选项[102],让客户以较低的冗余级别来存储非关键性的可再生数据.

3.2.4 重新调度

除了应用检查点和复制策略,一个失败的任务可以使用不同的资源进行重新调度.如图 8 所示,当检测到任务产生错误时,调度程序使用额外的计算资源重新提交并执行任务,因此重新调度可以有效地避免检查点和副本同步带来的开销.然而,重新调度技术也必然会引起调度结果的延迟.文献[103]提出了一个框架,通过资源的有效监控和作业执行状态的跟踪,在任务失败时做出高效的调度策略,该框架能够在具备高度动态性的环境中以容错的方式完成任务的执行.重新调度在某些情况下被认为是一种可行的容错机制,但是,由于重新调度会产生额外的延迟,它可能会影响工作流中与之并发执行的任务或工作流中依赖于它的其他任务的执行过程.或许正是由于这个原因,这个技术一直没有成为被广泛研究的焦点.

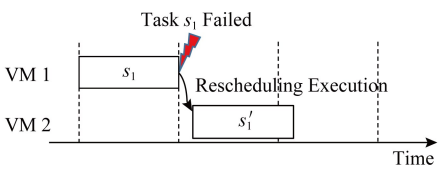

Fig. 8 Fault management using rescheduling mechanism

图 8 使用重新调度机制进行故障管理

3.3 故障预测与避免

由于故障容忍技术存在巨大的存储和计算开销并且实现方式非常复杂,云服务提供商已经开始采用故障预测与避免机制来保障云服务的可靠性.故障预测与避免机制会在故障发生之前采取预防措施避免故障的发生.

故障预测与避免机制的性能主要依赖于对故障发生的正确预测[104-105],基于故障预测的结果,故障避免机制会采取合理的措施预防故障.例如,系统可以将正在运行的进程从可疑的资源上迁移到正常的资源上以保证服务可以不被中断地运行.因此,对故障进行精确的预测可以使故障管理更加高效且可靠.迁移是配合故障预测的被广泛使用的一种容错方式.随着高速网络和分布式架构的发展,迁移正在运行的任务已经变得可行.随着云计算的出现,迁移主要可以分为进程迁移[106]和虚拟机迁移.考虑到云计算基础设施的动态性,在云计算中主要采用虚拟机迁移来保障服务的可靠性.

4 云计算可靠性与能耗的权衡问题

在第 3 节中,我们已经讨论了很多可以用来保证云计算系统可靠性的技术,这些技术已经被证明是高度优化且非常高效的[82].它们通过使用存储资源来存储日志或检查点,或者使用额外的计算资源重新执行任务片段,让系统在发生故障或服务被中断时恢复到最近的正确状态.因此,这些技术都或多或少地增加了计算资源或存储资源的冗余程度.通过使用这些技术,云计算服务提供商声称他们可以在大于 99.99%的时间内提供可靠的服务[107],即每年只允许大约 52.6 min 的服务中断时间.然而,增加额外资源必然会增加系统的能耗、降低服务提供商的收益,并会给环境带来负面的影响.随着资源冗余程度的增加,云计算系统的可靠性得到提高,而系统的能效则迅速下降.因此,研究人员在为云计算系统设计可靠性保障措施时也应考虑可靠性与能耗的权衡问题.

当前,云计算数据中心的资源利用率仅为 6%~12%,较低的资源使用率导致系统消耗了较高的电能[108].然而,在应用或任务被提交之前,精确地预测它们的资源需求是非常困难的,有可能会存在过度供给或过少供给的现象.因此,根据应用的需求,精确地供给资源可以使云计算服务更加可靠与节能.由于能耗管理机制能够通过对系统可靠性和硬件资源的管理来降低系统的能耗,所以很多学者采用能耗管理机制对云计算系统的可靠性与能耗进行折中.

目前被用来降低系统能耗的方法主要有能耗感知的资源调度或任务调度、硬件节能技术等.在资源过度供给的情况下,通过资源调度或任务调度可以降低系统能耗和其他操作开销.

为了在多用户公有云系统中进行能耗感知的容错调度,Gao 等人[109]提出了一个统一的资源分配与容错调度框架.该框架包含 2 种调度:静态调度和动态调度.静态调度首先为每个用户分配足够的虚拟机资源,保证用户的任务能够在规定的截止时间内完成;接着为每个用户的应用选择合适的错误检测机制与容错方法(即确定副本因子),保证应用具备较高的容错确信度;最后将用户的任务副本映射到具体的物理服务器上,获得一个临时调度.为了应对云计算系统的运行时变化,动态调度通过任务迁移保证用户应用能够在规定时间内完成.他们的框架能够使云服务提供商在可靠性、性能和能耗 3 个因素之间进行权衡,在保证所有用户的应用程序满足截止时间约束的前提下,最小化系统全局能耗,并获得较高的故障覆盖率和容错确信度.

Faragardi 等人[110]从服务的截止时间角度考虑服务质量,提出了一个基于整数线性规划的数学模型来调整云计算系统的可靠性和能耗.利用这个模型,他们提出了一个基于帝国竞争算法[111]的群体智能资源调度算法以故障感知和节能的方式分配资源.和混合遗传算法相比,他们提出的方法可以降低能耗达 17%,提供系统可靠性达 9%.

Diouri 等人[112]从能耗的角度评估了检查点协议和一致性与非一致性容错协议.对非一致性协议而言,日志可以保存在 HDD(hard disk drive)中,也可以保存在 RAM(random access memory)中.文献[112]经过实验表明,使用 RAM 保存日志消耗的能耗要低于使用 HDD 保存日志消耗的能耗,因而在大规模分布式系统中使用 RAM 存储日志要优于基于 HDD 的消息日志协议.对于一致性协议而言,它的能耗模式和非一致性协议的能耗模式类似,只是一致性协议的能耗会依赖于进程的协调过程.较差的同步过程则意味着更长的协调过程以及更多的能耗消耗.因此,减缓最快速进程的执行过程可以显著降低系统的能耗.

虚拟机迁移和任务合并是降低云计算系统能耗的关键技术:系统可以将使用率较低的服务器上运行的虚拟机迁移到其他主机上或者通过任务合并来使利用率较低的资源进入休眠模式或关闭状态.然而,随着虚拟机合并和任务迁移频率的增加,虽然系统的能耗会被降低,但是频繁地进行虚拟机迁移或任务合并也会影响到系统的可靠性.Choi[113]从虚拟机布局的角度提出了一种考虑节能和服务器可靠性的虚拟机放置算法.该算法在最小化服务器消耗功率的同时避免虚拟机的密集放置而导致的热岛,从而提升服务器可靠性.Zhou 等人[114]提出了一种虚拟机放置策略——最优冗余虚拟机布局(optimal redundant virtual machine placement, OPVMP).该方法在满足 k-容错(k-fault-tolerance)约束下,当主虚拟机故障需要迁移至备份虚拟机时,最小化网络资源的开销.

此外,动态电压和频率调节(dynamic voltage and frequency scaling, DVFS)作为一种非常高效的功耗管理技术,已经被广泛用在数据中心中来降低服务器的能耗.然而,使用 DVFS 降低处理器的执

行频率,会不可避免地增加处理器发生瞬时故障(即软错误)的概率,从而降低服务器系统的可靠性.因此,对大规模异构计算系统如云计算数据中心而言,高可靠性和低能耗是服务提供商应当考虑的2个目标.为了在异构集群中降低有依赖关系的任务集的能耗并最大化其可靠性,Zhang等人[115]提出了3个算法:RHEFT(reliability-aware heterogeneous earliest finish time),RCPOP(reliability-aware critical-path-on-a-processor),RMEC(reliability maximization with energy constraint).这3个算法都具有3个阶段:确定任务优先级、处理器频率选择、任务到处理器的映射.在确定任务优先级阶段,计算有向无环图中所有任务的拓扑顺序和优先级并将任务放入一个优先级队列中.在处理器频率选择阶段,在考虑任务集可靠性的前提下,为优先级队列中的每个任务确定最优的执行频率和电压.在任务到处理器的映射阶段,算法利用基于HEFT算法[116]的改进算法将每个任务分配到不同的处理器上.通过仿真实验结果表明,RMEC算法在可靠性和能耗指标上明显要优于其他2种算法.尽管文献[115]对有向无环图的任务集使用DVFS技术时可以保证应用具备较高的可靠性,但是RMEC算法并没有考虑故障发生情况下的容错问题.与之不同,Li等人[117]也使用DVFS技术降低实时应用执行的频率、最小化应用消耗的电能.他们利用检查点技术在任务之间的松弛时间内重新执行任务,保证应用的可靠性.他们提出的方法能够在保证实时应用的可靠性与截止时间要求的情况下,最小化系统的能耗.与Li等人的工作类似,Wu等人[118]通过任务调度将DVFS技术应用于工作流应用的执行过程.当处理器发生瞬时故障时,他们的方法可以使用检查点与回滚-恢复技术重新执行任务片段.该方法可以在同时满足截止时间与可靠性约束的条件下,最小化工作流应用消耗的电能.

5 云计算可靠性面临的挑战

云计算由于能够以“按需使用,按使用量付费”的方式向用户提供动态可扩展的服务,其已经成为了备受关注的一种计算模式.当前,存在着非常多的新应用已经基于云计算被开发和部署,并且有越来越多的应用程序正在被从传统的计算平台上迁移到基于云计算的平台上.未来,随着科学技术的进一步发展,云计算技术的应用也必将更加广泛.然而,由于云计算基于虚拟化技术实现,并且其具有多租户、规模巨大和体系结构复杂等特性,精确地管理云计算系统中的软硬件资源存在非常大的难度.同时,信息安全、边缘计算等技术的发展以及它们与云计算的深度融合,也给云计算的可靠性带来了诸多挑战.综合来看,未来云计算可靠性的研究工作可以分别从5个方面继续展开:

1)虚拟化技术在硬件资源失效场景下的服务可靠性问题.虚拟化技术是云计算得以实现的关键.通过虚拟化技术,云计算平台可以向用户提供隔离的虚拟机,虚拟机之间相互独立,一个虚拟机的失效并不会影响到其他虚拟机的正常运行.因此,当某个虚拟机失效时,服务提供商可以很容易地使用另一台虚拟机来替换失效的虚拟机,使失效虚拟机上运行的应用服务可以不被中断地运行.在硬件资源失效时,为了保持用户虚拟机的正确运行状态,服务提供商也需要将虚拟机从异常主机迁移到正常主机上.然而,虚拟机的替换和迁移过程必将会影响虚拟机中服务的运行可靠性.如何在硬件资源失效时不降低虚拟机中应用服务的可靠性是目前存在的一个挑战.

2)系统可靠性引发的信息安全问题.随着云计算市场的持续快速增长,用户对云计算的依赖程度也在逐渐增加,越来越多的用户将个人数据放到云中.但是由于用户对云计算系统的管控能力较弱,用户对云中的个人数据保护能力不足.同时,云计算系统中存在的漏洞也给系统攻击者提供了访问他人隐私数据的机会,网络入侵很可能会导致大范围的安全问题和服务隐患.因此,如何通过提高云计算系统可靠性来保障用户的信息安全,已经成为云计算服务提供商面临的一大挑战.在未来的研究工作中,研究人员应该投入更多的精力保证云计算系统的可靠性,避免可靠性问题带来的信息安全问题.

3)可靠性导致服务成本增长的问题.为了提高云计算系统提供的服务的可靠性,很多服务提供商采用冗余资源来提高服务对故障的容错能力.虽然冗余资源的增加可以显著提高服务的可靠性,但是它也相应地增加了系统提供服务需要付出的成本.这不仅会增加用户的使用云计算服务需要付出的费用,还可能会降低服务提供商的收益率.因此,在研究如何提高系统可靠性的同时,设计合理的资源分配策略来降低云计算服务的使用成本也是未来研究人员需要考虑的问题.

4)可靠性导致服务性能下降的问题.为了提高云计算系统对系统中故障的容忍程度,目前几乎所有

的数据中心都具备容错能力.在系统或服务发生故障时,系统管理程序利用容错措施将系统或服务从故障中恢复过来.但是,在处理故障保证服务可靠性的过程中,用户服务的性能必然会受到影响,例如,在虚拟机执行用户任务的过程中使用检查点机制虽然可以增加任务的容错能力,但是它也会增加任务的完成时间,影响用户的满意程度.因此,如何在保证用户服务可靠性的同时降低容错机制对服务性能的影响也是云计算研究面临的一大挑战.未来,研究人员在设计容错措施时,应该仔细地考虑到恢复过程对服务性能的影响,在提高服务的可靠性的同时,保障服务的高效性.

5) 云计算、雾计算和物联网深度融合场景中的可靠性问题.随着物联网的持续深度发展,智慧城市、智能家庭等物联网应用在可预见的未来将会极大地丰富人们的生活.但是目前市场上的各种智能终端设备的计算资源、存储资源有限,智能程度普遍不能令用户满意.云计算由于具有无限的资源、方便的使用模式,很自然地成为了给物联网中大量终端设备提供智能的首要选择.然而,云计算由于以集中式的方式提供大量资源,具有网络延迟高、网络拥塞和可靠性较低等不足,并不适用于对实时性、可靠性要求较高的物联网应用场景.为了弥补云的不足、满足未来应用的需求,人们在这种背景下提出了雾计算[119].如图 9 所示,作为云计算的一种延伸,雾计算使用边缘网络中数量庞大的雾节点(即专门部署的本地服务器或已经部署在网络中的路由器、交换机、网关等),弥补单一物联网设备的不足.然而,雾中设备相对分散、中心控制困难,而且雾节点的资源相对受限、节点间协调困难,这些限制都会对雾计算提供的服务的性能产生很大影响.如何在云计算、雾计算和物联网深度融合的场景中保证应用服务的可靠性是未来的一项重要研究工作.

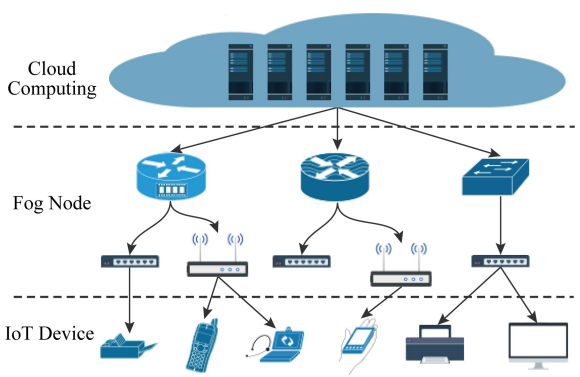

Fig. 9 Integration of cloud computing, fog computing and Internet of things

图 9 云计算、雾计算和物联网深度融合

6 总 结

云计算作为一种新型计算模式,能够为用户提供高度可扩展的计算服务,已经吸引了学术界和工业界的广泛关注.然而,随着云计算数据中心的规模、复杂度不断增加,云计算系统的运行故障也日益频繁,可靠性问题已经成为制约云计算技术发展的一大挑战.本文以云计算的故障为主线,介绍了云计算中常见的故障,并详细介绍了云计算中关键的提高可靠性的 3 类故障管理技术:故障消除、故障容忍和故障预测与避免.其中故障消除技术主要采用软件测试和验证技术,针对于云计算中的软件故障;故障容忍技术又包括纠删码、检查点、复制和重新调度等,通过增加时间或空间的冗余提高系统的容错能力,可以有效地降低硬件故障、网络故障和软错误等故障带来的影响;故障预测与避免机制基于故障预测的结果,采取合理的措施预防故障.

由于引入故障管理技术,不可避免地提高了云计算系统的能耗,因此云计算系统可靠性和能耗的权衡问题成为当下的研究热点之一.目前被用来降低系统能耗的方法主要有能耗感知的资源调度或任务调度、硬件节能技术等.

除此之外,如何解决提高系统可靠性带来的成本增长和服务性能下降问题、如何发现和预防提高系统可靠性而带来的潜在的信息安全问题都将会是很大的挑战.同时,如何在硬件资源失效时不降低虚拟机中应用服务的可靠性也是目前存在的一个挑战.在未来,随着云计算与边缘计算、物联网等新兴技术与云计算的深度融合,云计算可靠性将会迎来更多、更大的挑战.

参 考 文 献

[1] Foster I, Zhao Yong, Raicu I, et al. Cloud computing and grid computing 360-degree compared [C] //Proc of Grid Computing Environments Workshop. Piscataway, NJ: IEEE, 2008: 1-10

[2] Gartner, Inc. Gartner forecasts worldwide public cloud services revenue to reach $260 billion in 2017 [EB/OL]. (2017-10-12) [2019-04-01]. https://www.gartner.com/en/newsroom/press-releases/2017-10-12-gartner-forecasts-worldwide-public-cloud-services-revenue-to-reach-260-billionin-2017

[3] Engelmann C, Geist A. Super-scalable algorithms for computing on 100,000 processors [C] //Proc of the 5th Int Conf on Computational Science. Berlin: Springer, 2005: 313-321

[4] Barroso L, Hoelzle U. The datacenter as a computer: An introduction to the design of warehouse-scale machines [J]. Synthesis Lectures on Computer Architecture, 2009, 8(3): 1-107

[5] Ponemon Institute. Cost of data center outages [R]. Traverse City, Michigan: Ponemon Institute, 2016: 1-21 [2019-04-01]. https://planetaklimata.com.ua/instr/Liebert_Hiross/Cost_of_Data_Center_Outages_2016_Eng.pdf

[6] Cohen G. Downtime, outages and failures-understanding their true costs [EB/OL]. [2019-04-01]. http://www.evolven.com/blog/downtime-outages-and-failures-understanding-their-true-costs.html

[7] Selic B. Fault tolerance techniques for distributed systems [R]. Amund City, NY: IBM, 2004

[8] Nazir B, Qureshi K, Manuel P. Replication based fault tolerant job scheduling strategy for economy driven grid [J]. Journal of Supercomputing, 2012, 62(2): 855-873

[9] Sun Dawei, Chang Guiran, Miao Changsheng, et al. Analyzing, modelling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments [J]. Journal of Security and Networks, 2013, 66(1): 193-228

[10] Haider S, Ansari N. Temperature based fault forecasting in computer clusters [C] //Proc of the 15th Int Multi Topic Conf. Piscataway, NJ: IEEE, 2012: 69-77

[11] Garg R, Singh A. Fault tolerance in grid computing: State of the art and open issues [J]. Journal of Computer Science and Engineering Survey, 2011, 2(1): 88-97

[12] Ganga K, Karthik S, Paul A C. A survey on fault tolerance in workflow management and scheduling [J]. International Journal of Advanced Research in Computer Engineering & Technology, 2012, 1(8): 88-97

[13] Zhang Yang, Mandal A, Koelbel C, et al. Combined fault tolerance and scheduling techniques for workflow applications on computational grids [C] //Proc of the Int Symp on Cluster Computing and the Grid. Los Alamitos, CA: IEEE Computer Society, 2009: 244-251

[14] Asfandyar Q. Power-demand routing in massive geo-distributed systems [D]. Cambridge, MA: MIT Press, 2010

[15] Andrea P. The sony pictures hack, explained [EB/OL]. (2014-12-18) [2019-04-01]. https://www.washingtonpost.com/news/the-switch/wp/2014/12/18/the-sony-pictures-hack-explained/?utm_term=.3d569458d3d5

[16] QuEST Forum. Quality excellence for suppliers of telecommunications forum [EB/OL]. (2013-01-28)[2019-04-03]. https://www.questforum.org/wp-content/uploads/2015/01/Antitrust_Guidelines.pdf

[17] Wikipedia. Cloud computing [EB/OL]. (2019-04-03) [2019-04-04]. https://en.wikipedia.org/wiki/Cloud_computing

[18] Adams M, Bearly S, Bills D, et al. An introduction to designing reliable cloud services [EB/OL]. 2014 [2019-04-04]. https://www.microsoft.com/en-au/download/details.aspx?id=34683

[19] Javadi B, Thulasiraman P, Buyya R. Enhancing performance of failure-prone clusters by adaptive provisioning of cloud resources [J]. Journal of Supercomputing, 2013, 63(2): 467-489

[20] Stephen B, Joe T, Tom L, et al. Software Testing: Principles and Practice [M]. 2nd ed. New York: Pearson Education, 2017

[21] Javadi B, Abawajy J, Buyya R. Failure-aware resource provisioning for hybrid cloud infrastructure [J]. Journal of Parallel and Distributed Computing, 2012, 72(10): 1318-1331

[22] Fu Song. Failure-aware resource management for high-availability computing clusters with distributed virtual machines [J]. Journal of Parallel and Distributed Computing, 2010, 70(4): 384-393

[23] Egwutuoha I, Chen Shiping, Levy D, et al. A proactive fault tolerance approach to high performance computing (HPC) in the cloud [C] //Proc of the Int Conf on Cloud and Green Computing. Los Alamitos, CA: IEEE Computer Society, 2012: 268-273

[24] Vishwanath K, Nagappan N. Characterizing cloud computing hardware reliability [C] //Proc of the ACM Symp on Cloud Computing. New York: ACM, 2010: 192-204

[25] Philp I. Software failures and the road to apetaflop machine [C] //Proc of Int Symp on High Performance Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2005: 128-128

[26] Bresiger G. Knight capital computer meltdown just waiting to happen [EB/OL]. (2013-10-26) [2019-04-01]. http://nypost.com/2013/10/26/knight-capital-computer-meltdown-just-waiting-to-happen/

[27] Jones P. Overheating brings down Microsoft data center [EB/OL]. (2013-03-14) [2019-04-01]. http://www.datacenterdynamics.com/content-tracks/power-cooling/over-heating-brings-down-microsoft-data-center/74543.fullarticle

[28] Perez S. Microsoft and Yahoo confirm search outages [EB/OL]. (2015-01-02) [2019-04-01]. https://techcrunch.com/2015/01/02/following-bing-coms-brief-outage-search-yahoo-com-goes-down-too/

[29] Proffitt B. Software-as-a-Service: The dirty little secrets of SaaS [EB/OL]. (2013-03-05) [2019-04-01]. http://readwrite.com/2013/03/05/software-as-a-service-the-dirty-little-secrets-of-saas

[30] Budnik U. Lessons learned from recent cloud outages [EB/OL]. (2013-02-27) [2019-04-01]. http://www.rightscale.com/blog/enterprise-cloud-strategies/lessons-learned-recent-cloud-outages

[31] Banham R. Lessons from Japan: Why cloud is the future of disaster preparedness [EB/OL]. (2016-06-14) [2019-04-01]. https://www.forbes.com/sites/delltechnologies/2016/06/14/lessons-from-japan-why-cloud-is-the-future-of-disaster-preparedness/#4baf777818bf

[32] Dai Yuanshun, Yang Bo, Dongarra J, et al. Cloud service reliability: Modeling and analysis [C] //Proc of the 15th IEEE Pacific Rim Int Symp on Dependable Computing. Los Alamitos, CA: IEEE Computer Society, 2009: 1-17

[33] Weaver C, Emer J, Mukherjee S, et al. Techniques to reduce the soft error rate of a high-performance microprocessor [C] //Proc of the 31st Int Symp on Computer Architecture. Los Alamitos, CA: IEEE Computer Society, 2004, 32(2): 264-275

[34] Shubu M. Architecture Design for Soft Errors [M]. Amsterdam: Elsevier, 2008

[35] Haider S, Nazir B. Fault tolerance in computational grids: Perspectives, challenges, and issues [J]. SpringerPlus, 2016, 5: 1991

[36] Liang Yinglung, Zhang Yanyong, Sivasubramaniam A, et al. BlueGene/L failure analysis and prediction models [C] //Proc of the Int Conf on Dependable Systems and Networks. Los Alamitos, CA: IEEE Computer Society, 2006: 425-434

[37] Nguyên T, Désidéri J. Resilience issues for application workflows on clouds [C] //Proc of the Int Conf on Computational Science and Its Applications. Berlin: Springer, 2012: 418-433

[38] Hamilton J. Architecture for modular data centers [C] //Proc of the 3rd Biennial Conf on Innovative Data Systems. Asilomar, CA: Online Proceedings, 2007: 306-313

[39] Li Jie, Humphrey M, Cheah Y, et al. Fault tolerance and scaling in e-science cloud applications: Observations from the continuing development of MODIS Azure [C] //Proc of the Int Conf on e-Science, Piscataway. NJ: IEEE, 2010: 246-253

[40] Kandaswamy G, Mandal A, Reed D. Fault tolerance and recovery of scientific workflows on computational grids [C] //Proc of the Int Conf on Cluster Computing and the Grid. Los Alamitos, CA: IEEE Computer Society, 2008: 777-728

[41] Sherly A, InduShobha C. An overview of social engineering malware: Trends, tactics, and implications [J]. Technology in Society, 2010, 32(3): 183-196

[42] Ponemon Institute. The third annual study on the cyber resilient organization [EB/OL]. (2018-03) [2019-04-01]. https://www.ibm.com/downloads/cas/6RAEXRY5

[43] Schroeder B, Gibson G. A large-scale study of failures in high-performance computing systems [C] //Proc of the Int Conf on Dependable Systems and Networks. Los Alamitos, CA: IEEE Computer Society, 2006, 294-258

[44] Elefans. MTTR/MTTF/MTBF diagram [EB/OL]. (2010-01-09) [2019-04-01]. http://www.elecfans.com/baike/cunchushebei/cidaiku/20100109156891.html (in Chinese)
(Elefans. MTTR/MTTF/MTBF 图解 [EB/OL]. (2010-01-09) [2019-04-01]. http://www.elecfans.com/baike/cunchushebei/cidaiku/20100109156891.html)

[45] Thanadech T, Raja F, Chokchai L, et al. A reliability model for cloud computing for high performance computing applications [C] //Proc of the Int Conf on Parallel processing workshops. New York: ACM, 2012: 474-483

[46] Yadav N, Singh V B, Kumari M, et al. Generalized reliability model for cloud computing [J]. Journal of Computer Applications, 2014, 84(14): 13-16

[47] Chen Bingquan. Cloud service system reliability modeling [J]. Electronic Product Reliability and Environmental Testing, 2014, 32(2): 22-28 (in Chinese)
(陈冰泉. 云计算服务系统的可靠性建模研究[J]. 电子产品可靠性与环境实验, 2014, 32(2): 22-28)

[48] Research Triangle Institute. The economic impacts of inadequate infrastructure for software testing [EB/OL]. 2002 [2019-04-01]. https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf

[49] Demmy W, Petrini A. Statistical process control in software quality assurance [C] //Proc of the IEEE National Aerospace and Electronics Conf. Piscataway, NJ: IEEE, 2002: 1585-1590

[50] Barton J, Czeck E, Segall Z, et al. Fault injection experiments using FIAT [J]. IEEE Transactions on Computers, 1990, 38(4): 575-582

[51] Carreira J, Costa D, Silva J. Fault injection spot-checks computer system dependability [J]. IEEE Spectrum, 1999, 36(8): 50-55

[52] Dawson S, Jahanian F, Mitton T. ORCHESTRA: A probing and fault injection environment for testing protocol implementations [C] //Proc of Int Computer Performance and Dependability Symp. Piscataway, NJ: IEEE, 1996: 56-56

[53] Kanawati G, Kanawati N, Abraham J. FERRARI: A flexible software-based fault and error injection system [J]. IEEE Transactions on Computers, 1995, 44(2): 248-260

[54] Hoara W, Tixeuil S. A language-driven tool for fault injection in distributed systems [C] //Proc of the Int Workshop on Grid Computing. Piscataway, NJ: IEEE, 2005: 194-201

[55] Monnet S, Bertier M. Using failure injection mechanisms to experiment and evaluate a grid failure detector [C] //Proc of the Int Conf on High Performance Computing for Computational Science. Berlin: Springer, 2006: 610-621

[56] Song C. Assessing reliability of grid software systems using emergent features [C] //Proc of Workshop on Reliability and Robustness in Grid Computing Systems. Muncie, IN: OGF, 2007

[57] Topkara U, Song C, Woo J. Connected in a small world: Rapid integration of heterogenous biology resources [C] // Proc of the 2nd Int Workshop on Grid Computing Environments. New York: ACM, 2006: 109-118

[58] Chang B, Crary K, DeLap M, et al. Trustless grid computing in concert [C] //Proc of the 3rd Int Workshop on Grid Computing. Berlin: Springer, 2002: 112-125

[59] Zhang Qi, Cheng Lu, Boutaba R. Cloud computing: State of the art research issues [J]. Journal of Internet Services and Applications, 2010, 1(1): 7-18

[60] Abadi D. Data management in the cloud: Limitations and opportunities [J]. IEEE Computer Society Technical Committee on Data Engineering, 2009, 32(1): 3-12

[61] Patterson D, Gibson G, Katz R. A case for redundant arrays of inexpensive disks (RAID) [C] //Proc of the Int Conf on Management of Data. New York: ACM, 1988: 109-116

[62] Amazon Web Services, Inc. Amazon elastic block store [EB/OL]. 2018 [2019-04-01]. http://aws.amazon.com/cn/ebs/

[63] Zhou Ao, Wang Shangguang, Zheng Zibin, et al. On cloud service reliability enhancement with optimal resource usage [J]. IEEE Transactions on Service Computing, 2016, 4(4): 452-466

[64] Zheng Qingji. Research on erasure code for secure storage system [D]. Shanghai: Shanghai Jiao Tong University, 2009 (in Chinese)
(郑清吉. 安全存储系统中纠删码技术研究[D]. 上海: 上海交通大学, 2009)

[65] Lu Wei, Jackson J, Barga R. AzureBlast: A case study of developing science applications on the cloud [C] //Proc of the 19th ACM Symp on High Performance Distributed Computing. New York: ACM, 2010: 413-420

[66] Deng Jing, Huang S, Han Yunghsiang, et al. Fault-tolerant and reliable computation in cloud computing [C] //Proc of Globecom Workshops. Piscataway, NJ: IEEE, 2010: 1601-1605

[67] Warneke D, Kao O. Nephele: Efficient parallel data processing in the cloud [C] //Proc of the Workshop on Many-Task Computing on Grids and Supercomputers. New York: ACM, 2009: No.8

[68] Lyons R, Vanderkulk W. The use of triple-modular redundancy to improve computer reliability [J]. IBM Journal of Research and Development, 1962, 6(2): 200-209

[69] Jhawar R, Piuri V, Santambrogio M. A comprehensive conceptual system-level approach to fault tolerance in cloud computing [C] //Proc of the Int Systems Conf. Piscataway, NJ: IEEE, 2012: 1-5

[70] Bressoud T, Schneider F. Hypervisor-based fault tolerance [C] //Proc of the 15th ACM Symp on Operating Systems Principles. New York: ACM, 1955: 1-5

[71] VMware, Inc. Fault tolerance [EB/OL]. [2019-04-01]. https://www.vmware.com/cn/products/vsphere/fault-tolerance.html

[72] Ramakrishnan L, Koelbel C, Kee Y, et al. VGrADS: Enabling e-Science workflows on grids and clouds with fault tolerance [C] //Proc of the Conf on High Performance Computing Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2009: No.47

[73] Frey J, Tannenbaum T, Livny M, et al. Condor-G: A computation management agent for multi-institutional grids [J]. Cluster Computing, 2002, 5(3): 237-246

[74] Rejinpaul N, Visuwasam L, Maria M. Checkpoint-based intelligent fault tolerance for cloud service providers [J]. Journal of Computers and Distributed Systems, 2012, 2(1): 59-64

[75] Cully B, Lefebvre G, Meyer D, et al. Remus: High availability via asynchronous virtual machine replication [C] //Proc of the USENIX Symp on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2008: 161-174

[76] Tamura Y, Sato K, Kihara S, et al. Kemari: Virtual machine synchronization for fault tolerance [C/OL] //Proc of USENIX Annual Technical Conf. Berkeley: USENIX Association, 2008 [2019-04-01]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.580.7704&rep=rep1&type=pdf

[77] Hou Kaiyuan, Uysal M, Merchant A, et al. HydraVM: Low-cost, transparent high availability for virtual machines, HPL-2011-24 [R/OL]. Palo Alto, CA: HP Laboratories, 2011 [2019-04-01]. https://www.hpl.hp.com/techreports/2011/HPL-2011-24.html

[78] Luo Xianghong, Shu Jiwu. Summary of research for erasure code in storage system [J]. Journal of Computer Research and Development, 2012, 49(1): 1-11 (in Chinese)
(罗象宏, 舒继武. 存储系统中的纠删码研究综述[J]. 计算机研究与发展, 2012, 49(1): 1-11)

[79] Lu Xin, Liu Yuan. Cloud storage backup and recovery strategy based on erasure codes [J]. Computer Engineering and Applications, 2016, 52(4): 56-60 (in Chinese)
(芦欣, 刘渊. 基于纠删码的云计算存储备份及恢复策略[J]. 计算机工程与应用, 2016, 52(4): 56-60)

[80] Elnozahy E, Alvisi L, Wang Yimin, et al. A survey of rollback-recovery protocols in message-passing systems [J]. ACM Computing Surveys, 2002, 34(3): 375-408

[81] Duda A. The effects of checkpointing on program execution time [J]. Information Processing Letters, 1983, 16(5): 221-229

[82] Pedram M. Energy-efficient datacenters [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012, 31(10): 1465-1484

[83] L'Ecuyer P, Malenfant J, Computing optimal checkpointing strategies for rollback and recovery systems [J]. IEEE Transactions on Computers, 1988, 37(4): 491-496

[84] Daly J. A higher order estimate of the optimum checkpoint interval for restart dumps [J]. Future Generation Computer Systems, 2006, 22(3): 303-312

[85] Di Sheng, Robert Y, Vivien F, et al. Optimization of cloud task processing with checkpoint-restart mechanism [C] // Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2013: 1-12

[86] Liu Xiao, Yang Yun, Yuan Dong, et al. Do we need to handle every temporal violation in scientific workflow systems? [J]. ACM Transactions on Software Engineering and Methodology, 2014, 23(1): 1-34

[87] Benoit A, Hakem M, Robert Y. Fault tolerant scheduling of precedence task graphs on heterogeneous platforms [C] // Proc of Int Symp on Parallel and Distributed Processing. Piscataway, NJ: IEEE, 2008: 1-8

[88] Benoit A, Hakem M, Robert Y. Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems [J]. Parallel Computing, 2009 35 (2): 83-108

[89] Wang Shuli, Li Kenli, Mei Jing, et al. A task scheduling algorithm based on replication for maximizing reliability on heterogeneous computing systems [C] //Proc of IEEE Int Parallel & Distributed Processing Symp Workshops. Los Alamitos, CA: IEEE Computer Society, 2014: 1562-1571

[90] Andrzejak A, Graupner S, Kotov V, et al. Algorithms for self-organization and adaptive service placement in dynamic distributed systems, HPL-2002-259 [R]. Palo Alto, CA: HP Laboratories, 2002

[91] Weissman J, Lee B. The virtual service grid: An architecture for delivering high-end network services [J]. Concurrency and Computation: Practice and Experience, 2002, 14(4): 287-319

[92] Verma D, Sahu S, Calo S, et al. SRIRAM: A scalable resilient autonomic mesh [J]. IBM Systems Journal, 2003, 42 (1): 19-28

[93] Valcarenghi L, Castoldi P. QoS-aware connection resilience for network-aware grid computing fault tolerance [C] //Proc of Int Conf Transparent Optical Networks. Piscataway, NJ: IEEE, 2005: 417-422

[94] Lac C, Ramanathan S. A resilient telco grid middleware [C] //Proc of the 11th IEEE Symp on Computers and Communications. Piscataway, NJ: IEEE, 2006: 306-311

[95] Townend P, Groth P, Looker N, et al. FT-Grid: A fault-tolerance system for e-science [C] //Proc of the 4th UK e-Science All Hands Meeting. Swindon, UK: EPSRC, 2005

[96] Genaud S, Rattanapoka C. P2P-MPI: A peer-to-peer framework for robust execution of message passing parallel programs on grids [J]. Journal of Grid Computing, 2005, 5 (1): 27-42

[97] Lamport L. Paxos made simple [J]. ACM SIGACT News: Distributed Computing Column, 2001, 32(4): 18-25

[98] Diego O, John O. In search of an understandable consensus algorithm [C] //Proc of the USENIX Annual Technical Conf. Berkeley: USENIX Association, 2014: 305-320

[99] Zhang Xianan, Zagorodnov D, Hiltunen M, et al. Fault-tolerant grid services using primary-backup: Feasibility and performance [C] //Proc of the Int Conf on Cluster Computing. Los Alamitos, CA: IEEE Computer Society, 2004: 105-114

[100] Dasgupta G, Dasgupta K, Purohit A, et al. QoS-GRAF: A framework for QoS based grid resource allocation with failure provisioning [C] //Proc of the Int Workshop on Quality of Service. Piscataway, NJ: IEEE, 2006: 281-283

[101] Li Wenhao, Yang Yun, Yuan Dong. Ensuring cloud data reliability with minimum replication by proactive replica checking [J]. IEEE Transactions on Computers, 2006, 65 (5): 1494-1506

[102] Amazon Web Services, Inc. Amazon S3 reduced redundancy storage [EB/OL]. [2019-04-01]. https://aws.amazon.com/cn/s3/reduced-redundancy/

[103] In J, Avery P, Cavanaugh R, et al. SPHINX: A fault-tolerant system for scheduling in dynamic grid environments [C] //Proc of the Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2005: 12-21

[104] Fu Song, Xu Chengzhong. Exploring event correlation for failure prediction in coalitions of clusters [C] //Proc of the ACM Conf on Supercomputing. New York: ACM, 2007: No.41

[105] Islam S, Keung J, Lee K, et al. Empirical prediction models for adaptive resource provisioning in the cloud [J]. Future Generation Computer Systems, 2012, 28(1): 155-162

[106] Milojičić D, Douglis F, Paindaveine Y, et al. Process migration [J]. ACM Compute Survey, 2000, 32(3): 241-299

[107] Shen Xing. Aliyun's four overseas data centers have opened one after another within a month, data center availability over 99.99% [EB/OL]. (2016-11-21) [2019-04-01]. https://yq.aliyun.com/articles/64704?spm=5176.10695662.1996646101.searchclickresult.7d5c693e6WnoIh (in Chinese)
(身行. 阿里云四大海外数据中心月内相继开服，数据中心可用性99.99%以上[EB/OL]. (2016-11-21) [2019-04-01]. https://yq.aliyun.com/articles/64704?spm=5176.10695662.1996646101.searchclickresult.7d5c693e6WnoIh)

[108] Thibodeau P. Data centers are the new polluters [EB/OL]. (2014-08-26) [2019-04-01]. http://www.computerworld.com/article/2598562/data-center/datacenters-are-the-new-polluters.html

[109] Gao Yue, Gupta S, Wang Yanzhi, et al. An energy-aware fault tolerant scheduling framework for soft error resilient cloud computing systems [C] //Proc of Design, Automation and Test in Europe Conf and Exhibition. Piscataway, NJ: IEEE, 2014: No.94

[110] Faragardi H, Rajabi A, Shojaee R, et al. Towards energy-aware resource scheduling to maximize reliability in cloud computing systems [C] //Proc of the Int Conf on High Performance Computing and Communications. Piscataway, NJ: IEEE, 2013: 1469-1479

[111] Atashpaz G, Lucas C. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition [C] //Proc of the Congresson Evolutionary Computation. Piscataway, NJ: IEEE, 2007: 4661-4667

[112] Diouri M, Glück O, Lefevre L, et al. Energy considerations in checkpointing and fault tolerance protocols [C] //Proc of the Int Conf on Dependable Systems and Networks Workshops. Piscataway, NJ: IEEE, 2012: 1-6

[113] Choi J. Virtual machine placement algorithm for energy saving and reliability of servers in cloud data centers [J]. Journal of Network and Systems Management, 2018, 27 (1): 149-165

[114] Zhou Ao, Wang Shangguang, Chen Bo, et al. Cloud service reliability enhancement via virtual machine placement optimization [J]. IEEE Transactions on Service Computing, 2017, 10(6): 902-913

[115] Zhang Longxin, Li Kenli, Xu Yuming, et al. Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster [J]. Information Sciences, 2015, 319(3): 113-131

[116] Topcuouglu H, Hariri S, Wu Minyou. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274

[117] Li Zheng, Ren Shangping, Quan Gang. Energy minimization for reliability-guaranteed real-time applications using DVFS and checkpointing techniques [J]. Journal of Systems Architecture, 2014, 61(2): 71-81

[118] Wu Tingming, Gu Haifeng, Zhou Junlong, et al. Soft error-aware energy-efficient task scheduling for workflow applications in DVFS-enabled cloud [J]. Journal of Systems Architecture, 2018, 84: 12-27

[119] Bonomi F, Milito R, Zhu Jiang, et al. Fog computing and its role in the Internet of things [C] //Proc of the 1st Edition of the MCC Workshop on Mobile Cloud Computing. New York: ACM, 2012: 13-16

Duan Wenxue, born in 1994. Master candidate. Her main research interests include cloud computing, and model checking.

Hu Ming, born in 1995. PhD candidate. His main research interests include cloud computing, and program synthesis.

Zhou Qiong, born in 1981. PhD, lecturer, master supervisor. Her main research interests include statistics and big data, managerial research method, and applied economics.

Wu Tingming, born in 1993. Master. His main research interests include cloud computing.

Zhou Junlong, born in 1988. PhD, assistant professor. Senior member of CCF. His main research interests include real-time embedded systems, cloud computing, and cyber physical systems.

Liu Xiao, born in 1982. PhD, senior lecturer of Deakin University. Senior member of CCF. His main research interests include software engineering, workflow management systems and cloud computing.

Wei Tongquan, born in 1973. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include real-time embedded systems, green and reliable computing, parallel and distributed systems, and cloud computing.

Chen Mingsong, born in 1982. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include design automation and verification of cyber physical systems, cloud/embedded/parallel computing, computer architecture, and Internet of things.