人机物融合操作系统的研究进展与趋势

CCF 系统软件专业委员会

郭　耀[1]　陈海波[2]　胡春明[3]　卜　磊[4]

[1]北京大学，北京

[2]上海交通大学，上海

[3]北京航空航天大学，北京

[4]南京大学，南京

摘　要

操作系统是计算机系统中最为关键的一层系统软件。近年来，由于互联网、云计算、大数据、物联网等新型应用模式的迅速普及，各种面向新型应用模式的操作系统得到了广泛的关注。到了人机物融合的泛在计算时代，为了更好地管理人机物融合环境下的分布异构海量资源，同时为人机物融合的新型应用和服务提供支持，操作系统及相关技术正在产生许多重要的变革。本文首先简要回顾操作系统的发展历史，然后主要针对人机物融合时代操作系统在国内外的最新研究进展进行了总结和对比，并分析了在人机物融合时代操作系统面临的主要挑战和发展趋势。

关键词：操作系统，人机物融合，软件定义，网构软件，泛在操作系统

Abstract

Operating system is one of the key system software layers in a computer system. In recent years, with the prevalence of new application scenarios such as the Internet, cloud computing, big data and Internet of Things, researchers have focused on many new types of operating systems for these new application scenarios. As we are approaching the ubiquitous computing age with human-cyber-physical (HCP) systems, in order to better manage the distributed heterogeneous massive resources in the HCP environment, while providing support for the new types of HCP applications and services, operating systemsand related technology have been going through new evolvements. This report will first briefly review the history of operating systems, then mainly focus on the recent research progress of operating systems in the HCP era, as well as analyze their main challenges and development trends.

Keywords: Operating systems, Human-cyber-physical, Software-defined, Internetware, Ubiquitous operating systems

1　引言

操作系统是计算机系统中最为关键的一层系统软件。按照中国计算机科学技术百科

全书的定义，“操作系统是管理硬件资源、控制程序运行、改善人机界面和为应用软件提供支持的一种系统软件”[1]。简言之，操作系统的主要功能是：向下管理资源（包括存储、外设和计算等资源），向上为用户和应用程序提供公共服务。

结构上，操作系统大致可划分为三个层次，分别是人机接口、系统调用和资源管理[2]。人机接口负责提供操作系统对外服务、与人进行交互的功能，从最简单的命令行操作，到早期 Unix 系统上采用的传统 shell，进而到 Windows 等现代操作系统中采用的 GUI（图形用户界面）窗口系统，人机接口不断向易用性和用户友好发展。资源管理指的是对各种底层资源进行管理，存储、外设和计算单元等都是操作系统管理的对象，随着计算机系统的发展，新的软硬件资源不断出现，操作系统的资源管理功能也越来越庞大和复杂。系统调用是位于人机接口和资源管理之间的一个层次，提供从人机接口到资源管理功能的系统调用功能。

1.1 操作系统的简要历史回顾

1956 年出现了历史上第一个实际可用的操作系统 GM-NAA I/O ㊀，主要是为了弥补处理器速度和 I/O 之间的差异，提供批处理的功能来提高系统效率。公认的第一个现代操作系统是从 20 世纪 70 年代开始得到广泛应用的 Unix 系统[4]。Unix 是第一个采用与机器无关语言（C 语言）来编写的操作系统，从而可以支持更好的可移植性。采用高级语言来编写操作系统具有革命性意义，不仅极大地提高了操作系统的可移植性，还促进了 Unix 和类 Unix 系统的广泛使用。

从 20 世纪 80 年代开始，以 IBM PC 为代表的个人计算机（PC）开始流行，开启了个人计算时代。PC 上的典型操作系统包括 Apple 公司的 Mac OS 系列，Microsoft 公司的 DOS/Windows 系列，以及从 Unix 系统中衍生出来的 Linux 操作系统。PC 时代的操作系统主要面向个人用户的易用性和通用性需求，一方面提供现代的图形用户界面（GUI），可以很好地支持鼠标等新的人机交互设备，另一方面提供丰富的硬件驱动程序，从而使用户可以在不同计算机上都使用相同的操作系统。

进入 21 世纪之后，在个人计算机普及的同时，出现了以智能手机为代表的新一代的移动计算设备，从黑莓（BlackBerry）到 iPhone，再到 Google Android 手机的广泛流行，智能手机已经成为最广泛使用的计算设备。在智能手机上运行的这一类操作系统从核心技术上并无实质性变化，主要是着眼于易用性和低功耗等移动设备的特点，对传统操作系统（例如 Linux）进行了相应的裁剪，并开发了新的人机交互方式与图形用户界面。

近年来，绝大多数计算机采用的处理器已经从单核处理器发展为双核、四核甚至更多核的处理器，然而，目前的多核处理器上采用的操作系统依然是基于多线程的传统操作系统架构，很难充分利用多核处理器的处理能力。为了更好地提高多核处理器的执行

㊀ GM-NAA I/O 输入输出管理系统是通用汽车公司和（General Motors）和北美航空（North American Aviation）联合研制的在 IBM 704 计算机上运行的管理程序，采用该系统的 IBM 704 计算机总共有大约 40 台。

效率，研究人员已经在尝试专门针对多核处理器开发多核操作系统的原型[5-6]，但是，目前均还未得到广泛的推广和应用。

总的来看，单机操作系统发展的主要目的是更好地发挥计算机硬件的效率，以及满足不同的应用环境与用户需求。在 Unix 系统出现之后，单机操作系统的结构和核心功能都基本上定型，在此之后的发展主要是为了适应不同的应用环境与用户需求而推出的新型用户界面与应用模式，以及在不同应用领域的操作系统功能的裁剪。

进入网络时代之后，在单机操作系统的发展主线之外，操作系统的发展的另一个方向主要是扩展操作系统的能力为网络提供支持。操作系统上的网络支持能力大致可以分为两个层次：一个层次是随着局域网、广域网以及 Internet 的逐步普及，通过扩展操作系统的功能来支持网络化的环境，主要提供网络访问和网络化资源管理的能力；另一个层次是在操作系统和应用程序之间出现了新的一层系统软件——**中间件**（middleware），用以提供通用的网络相关功能，支撑以网络为平台的网络应用软件的运行和开发[7]。

在 20 世纪 90 年代还出现了“网络操作系统（Networking Operating System）”的概念，例如 Novell Netware、Artisoft LANtastic 等系统。严格来讲，这一类网络操作系统只是在原来单机操作系统之上添加了对网络协议的支持，从而使得原本独立的计算机可以通过网络协议来访问局域网（或者广域网）之上的资源，这样的操作系统并不是现代意义上的网络化操作系统。

随着过去 20 年互联网的快速发展，操作系统面向的计算平台正在从单机平台和局域网平台向互联网平台转移。操作系统不仅需要提供网络支持能力，更重要的是，需要解决如何管理互联网平台上的庞大的计算资源和数据资源，如何更好地利用分布式的计算能力等诸多问题。在互联网时代，随着单机操作系统的核心功能基本定型，网络化逐渐成为主流。

近年来，面向不同的互联网计算与应用模式，国内外都提出了许多面向云计算和数据中心的云操作系统[9-10]。根据 Techopedia 的定义，“云操作系统（Cloud Operating System）是设计来管理云计算和虚拟化环境的操作系统”[8]。一般来讲，云操作系统是指构架于服务器、存储、网络等基础硬件资源和单机操作系统、中间件、数据库等基础软件之上的、管理海量的基础硬件、软件资源的云平台综合管理系统。云操作系统管理的对象包括虚拟机的创建、执行和维护，虚拟服务器和虚拟基础设施，以及后台的软硬件资源。

除此之外，随着移动互联网和物联网的发展，近年来还出现了面向不同领域的操作系统的概念和实现，包括物联网操作系统、机器人操作系统、企业操作系统、城市操作系统、家庭操作系统等等。究其本质，这些操作系统都是面向新型互联网应用而构建的支持这些应用的开发和运行的网络化操作系统。

1.2 操作系统发展的主要驱动力

分析操作系统发展的历史，可以总结出在操作系统发展过程中的几个主要驱动力[2]：

1）**追求更高效地发挥硬件资源所提供的计算能力。**随着计算机硬件的发展，操作系统必须能够提供更加高效的利用新硬件的能力，利用软件技术统筹管理好硬件以形成灵活、高效、可信、统一的虚拟资源。例如：在 Unix/Linux 中，可以把许多外设都当作文件来进行管理，从而可以提供统一的管理和操作方式，提高应用程序的开发效率；为了解决处理器、内存和 I/O 设备之间的速度差异，OS/360 操作系统通过对内存的划分支持多个程序同时驻留在内存中，从而可以实现多道程序执行的功能；在现代操作系统中，还可以通过虚拟内存的概念扩展计算机的物理内存，使应用程序在运行时几乎可以拥有无穷大的内存空间；在局域网上，大多数操作系统都支持通过网络文件系统（例如 NFS）扩展计算机的本地磁盘，使应用程序可以以透明的方式访问其他计算机上的磁盘空间。在多核异构和分布式的计算机系统中，如何充分利用多核处理器、GPU 和分布式计算系统的能力提高系统的整体效率。

2）**通过凝练应用软件中的共性并进行复用，尽可能提高软件开发和运行的效率。**从前面对操作系统发展的历史分析可以看到，操作系统的很多功能是通过凝练应用软件中的共性而得到的，并将应用程序的共性沉淀为操作系统（中间件）的一部分，通过复用这些共性功能，可以为软件的开发和运行提供更为丰富的应用程序接口和运行时函数库，提高软件开发和运行的效率。例如，在 Unix 操作系统中，集成了完整的开发环境，包括 cc（C 语言编译器）、make（build 管理工具）、头文件、系统库（主要是 libc）等开发工具；在图形用户界面的发展过程中，从 1991 年开始发布的 Linux 操作系统中并没有包括图形化桌面，KDE 和 Gnome 图形桌面环境是分别在 1996 年和 1997 才发布的，而直到更晚的时候，它们才逐渐成为 GNU/Linux 操作系统标准发布版中的一部分。为了支撑移动应用的开发和运行，Android 操作系统又在 Linux 内核上添加了用来应用框架层（application framework）和运行时（Android runtime）。

3）**更好地满足用户对易用性的需求。**随着新的应用模式和人机交互方式的出现，操作系统本身还引领了新型人机交互界面及相关技术的发展。早期的 Unix shell 就是一种非常经典的用户界面，直到现在还是许多系统程序员的首选。随着个人计算机的流行，基于图形用户界面（GUI）的窗口系统成为使用最为广泛的交互方式。随着互联网的迅速发展和应用的服务化，在浏览器中可以执行原来本地应用程序的大部分功能，因此 Web 浏览器已经不再是提供简单的网络浏览功能，而正在成为一种新的用户界面模式，出现了像 Chromium OS、Facebook OS 这样的基于 Web 的操作系统。

1.3 人机物融合时代的操作系统

随着移动互联网和物联网的迅速发展，计算模式和软件应用都在逐步演化为更加复杂和动态的形式。在新的计算模式和应用场景中，除了传统的计算设备（“机”）和新兴的物联网设备（“物”），还逐渐融入了一种新的重要元素，即“人”的参与，从而形成人机物三元融合（human-cyber-physical）的计算环境。举例来说，在新的人工智能应用场景中，例如自动驾驶，不仅要考虑到计算设备和汽车中各式各样的传感器，还一定要

考虑到人的因素，其中包括驾驶员以及路上的行人等等。

20 世纪 90 年代，Mark Weiser 提出了泛在计算（Ubiquitous Computing）[11] 的概念，认为计算是广泛存在的，计算机可以是任意尺寸大小、任意形状的，可以嵌入到每个物体中为人们提供服务。21 世纪以来，物联网（Internet of Things）[12] 提倡将物品通过信息传感设备，按照约定的协议与互联网连接起来，以实现智能化识别和管理。例如，Android Things[13] 是 Google 在 2015 年宣布的基于 Android 的物联网操作系统平台，专为在小内存、低功耗设备上运行而设计，支持蓝牙和 Wi-Fi，开发者可以通过网络统一收集设备数据进行设备管理。

随着泛在计算概念的深化，计算机可管理资源的范围将进一步扩大，机器人和智能家居开始成为操作系统的可管理资源，出现了机器人操作系统（ROS）[14] 和家庭操作系统（HomeOS）[20] 等新的操作系统概念。随着人机物融合的应用模式的进一步演化，操作系统的概念还会不断延伸和泛化，梅宏院士等人在 2018 年《IEEE Computer》杂志发表了题为“走向泛在操作系统：一种软件定义的视角”的文章，首次提出“泛在操作系统（Ubiquitous Operating System）”的概念[21]，来表示不限于 Windows/Linux 这样的传统操作系统的未来新型操作系统。泛在操作系统的概念指的就是在人机物融合的泛在计算模式下支持泛在应用开发和运行的操作系统平台。

本文主要关注最近几年国内外在新型操作系统领域的学术和产业进展，特别是在面向新的人机物融合领域的新型操作系统以及关键技术。在总结和对比国内外操作系统研究的基础上，分析面向人机物融合的操作系统的技术挑战和发展趋势，为操作系统相关领域的研究提供建议。

2 国际研究现状

在现实世界中，操作系统的研究和应用所涉及的不仅仅是操作系统本身，同时也与一个操作系统所支持的计算机硬件系统以及运行在其上的软件应用一起构成操作系统的生态。目前主流的操作系统生态包括 Microsoft 公司主导的 Windows 生态、Apple 公司主导的 Mac OS/iOS 生态，以及开源的 Linux 生态（也包含 Android 等从 Linux 上派生出来的操作系统）。与此相对应，学术界很多研究也会关注这些主流操作系统的优化，而由于 Linux 操作系统的开源特性，大多数这类研究都集中在 Linux 系统的优化和新特性上。另外，学术界和产业界也针对人机物融合时代下的不同计算机硬件系统和新的应用场景提出了各种不同的新型操作系统及相关技术。

2.1 学术界研究进展

在学术界，从近年来操作系统重要会议的论文发表可以看出，由于计算机系统和操作系统（及中间件）领域的研究越来越趋于应用模式导向，因此，目前操作系统领域的

研究呈现出了一个非常鲜明的特点：以新型应用模式作为主要驱动力，拥有大量数据和基础设施的大型互联网公司在很大程度上引领了研究方向和技术潮流，而学术界和科研机构则关注于追踪解决具体技术问题和进行优化，也在新型操作系统的相关研究方向上取得很多有价值的突破。

由于操作系统的相关研究方向众多，本文主要关注与人机物融合操作系统的发展较为相关的几个重要研究方向，包括新型操作系架构的设计，与人工智能相关的操作系统技术，以及面向新型应用场景的操作系统等。

2.1.1 新型操作系统架构设计

除了像 Windows 和 Linux 这样的传统操作系统架构之外，学术界和产业界还提出了很多新的操作系统结构，其中基于能力的操作系统和基于构件的操作系统是早期重要的操作系统架构设计方案。随着网络的发展，网络开始成为操作系统架构设计的重要方面，一方面，云环境成为新的应用场景，新的网络应用模式也对操作系统设计提出了新的要求，要求操作系统开始针对网络进行专门的优化，做到高吞吐、低延迟，于是人们开始使用基于库的操作系统（library OS）架构设计来提高操作系统对网络的使用效率；另一方面，网络成为操作系统设计的重要组成部分，操作系统开始使用网络作为通信介质，并通过网络来连接传统的计算机系统硬件，提出了像分离内核操作系统[37]这样的全新架构。

（1）基于能力的操作系统

基于能力的系统（Capability- based System）是指使用基于能力的安全机制（Capability-based Security）的操作系统。**能力**（Capability）通常被定义为授予拥有者访问计算机系统中的实体或对象权限的令牌或密钥。基于能力的系统使用能力来管理访问权限，保护操作系统的资源。操作系统的应用程序会根据能力列表来访问对象，并通过传递能力来传递访问对象的权限。与访问控制列表（Access Control List）功能相同，都提供了访问控制权限管理的功能，从不同角度记录了权限矩阵的信息。但是基于能力的系统更加直观，程序直接持有访问对象权限，权限控制粒度更细，权限变化更加灵活和方便。同时基于能力的系统还可以提供最小特权操作并且避免混淆代理问题（Confused Deputy Problem）。

KeyKOS[26]是一个早期的基于能力的面向对象操作系统，最初实现是出于为 Tymnet 主机上的应用程序提供安全、可靠、24 小时可用的服务，同时保证可以在单个硬件系统上运行多个操作系统实例。KeyKOS 内核包括约 20 000 行 C 代码，可以在不到 100KB 的内存中运行，其设计主要基于三个体系结构概念：无状态的内核、单级存储和能力系统。KeyKOS 系统的能力被称为键（Key），只有拥有键才能够发送消息调用服务。系统设计遵循最小权限原则，对象依赖键来持有权限。KeyKOS 系统包含多个模块，每个模块以最小权限运行。因为 KeyKOS 系统采用基于能力的系统设计，所以内核无须关注用户是否持有操作权限、文件名解析等问题，简化了内核设计，因此 KeyKOS 内核比一般微内核更小，被称为纳米内核（NanoKernel）。

2019 年发表的 SemperOS[27] 是在 M3[28] 操作系统上设计的基于能力的多核操作系统，M3 系统中将内核和存储器集成到了分组交换片上网络（Network-on-Chip，NoC）中，并为每个内核配备了数据传输单元（Data Transfer Unit，DTU）作为通用硬件组件，并将该包含核心、本地内存和 DTU 的组合称为处理元素（Processing Element，PE）。SemperOS 保留了 PE 的概念，并使用分组管理的思想将 PE 分组，每个组都由一个独立的内核管理，并使用能力系统来进行资源的分配和权限管理。SemperOS 重点关注 PE 组之间能力的分配和回收设计。每个 PE 组通过能力来访问资源，通过请求能力和分配能力实现能力的交换，在组之间传递和共享资源，并提出了能力回收机制来回收权限和资源。内核通过系统调用从另一个 PE 组请求能力，受请求方会复制能力并交给请求方，请求方通过能力来访问资源。当不再需要共享时，则通过递归撤销能力来收回对资源的访问权限。

（2）基于构件的操作系统

基于构件的操作系统吸收了微内核操作系统提出的构件化思想，以此来保证的操作系统灵活性、安全性和高性能，并在此基础上提供了一种构造新操作系统的方法。基于构件的操作系统常运用于构建定制化的操作系统，例如嵌入式操作系统。

Pebble[29] 是一个基于构件的操作系统，其设计遵循以下原则：

- 最小特权模式：尽量在用户级运行程序，特权模式内核仅负责在保护域之间进行切换。最小化特权模式可以减少特权代码数量，减少执行基本特权模式服务所需的时间。
- 操作系统由细粒度的可替换构件组成，使用内存保护加以隔离。
- 保证将线程从一个保护域转移到另一个保护域对性能的影响较小。
- 在保护域之间转移线程是通过硬件中断处理（称为端口遍历）来完成的。端口代码是动态生成的，并执行端口特定的操作。硬件中断、IPC 和 Pebble 等价的系统调用都由 Pebble 提供的端口机制处理。

Pebble 与传统微内核操作系统架构相似，系统内核运行在特权级，系统服务运行在用户模式下。Pebble 的内核仅包含将线程从一个保护域转移到另一个保护域的代码，以及少量需要内核模式的支持功能。而微内核在特权模式下实现了 IPC 和虚拟内存管理。同时 Pebble 提供了端口机制来管理保护域之间的通信等，仅当两个保护域之间存在端口时，才能调用另一个保护域提供的服务，以此保证系统的灵活性与安全性。并且通过端口解决硬件中断、异常等，简化了操作系统的设计。

基于构件的操作系统中关键技术之一是如何提供灵活构建操作系统内核的工具，例如 FLUX OSkit[30]、THINK[31] 等都提供了组装操作系统内核的工具，可以灵活地绑定和组装操作系统构件。与 Pebble 不同，OSkit 和 THINK 框架中的不同构件之间通过接口绑定来对不同构件进行组装。OSkit 使用动态绑定机制来组装构件，使用类似于 C ++ 的虚拟函数表的动态分配表和功能指针来允许构件在运行时由客户端 OS 动态绑定在一起。THINK 则根据不同通信形式（例如远程通信和本地通信）生成不同的绑定方式所对应的代码。同时基于 THINK 框架设计了 KORTEX 内核组装库，提供了线程、内存管理和进程管理等多种模型，并实现了不同形式的绑定方式，例如系统调用、信号、进程间通信和

远程进程通信等。在 KORTEX 的基础上还实现了一个类似于 L4 的内核服务，与传统 L4 的性能相比并没有下降。

（3）基于库的操作系统

最早的库操作系统（Library Operating System）是基于外核（Exokernel）操作系统[32]的概念来开发的，但是随着云计算技术的发展和虚拟化技术的成熟，库操作系统开始运行在虚拟机监视器（Hypervisor）上。与传统操作系统运行环境不同，云环境中硬件资源以虚拟化的形式提供给使用者，系统服务则一般运行在虚拟机上。安全性、专业化、定制化、高性能成为人们新的要求，人们开始绕过操作系统内核来直接管理虚拟硬件资源，因此，基于库的操作系统提供了良好的解决方案。

例如，Unikernel[33]是一种面向云计算环境、运行在虚拟环境中的库操作系统，具有以下特点：

- 接口统一：Unikernel 在设计时考虑到了当前应用程序和操作系统使用的语言不同，很难通过统一的源代码分析来对整个系统进行静态推断，因而使用高级语言框架来统一为内核和用户空间的不同接口。高级语言框架提供了静态类型检查、动态内存管理、模块化和元编程功能。统一的高级语言框架简化了 Unikernel 系统的设计和构建。
- 文件配置：Unikernel 通过将库和配置代码链接到一起生成应用程序。配置文件可以用来简化部署过程，静态配置信息在应用程序开发过程中设置，结构类似于树形结构文件系统，在编译时所有相关库会遍历子目录提取配置信息并设置初始值。Unikernel 仅支持通过简单函数调用来动态修改配置文件信息，而专门目标的配置信息更新需要通过重新编译。大部分的静态配置信息可以借助于静态分析工具和编译器的类型检查器，大大减少配置复杂的应用程序虚拟环境的工作量。
- 链接优化：库的链接是根据代码配置文件按需进行的，可以通过死代码消除等技术来减小生成代码的体积。编译后生成的文件运行在统一的虚拟空间中，且因为配置信息通常是静态的，所以运行时不需要动态链接支持，优化了运行时的性能。

MirageOS 是在 Unikernel 基础上设计的库操作系统，基于 OCaml 语言设计，在 Xen 虚拟机上运行。相比传统操作系统，它简化了应用程序的运行架构，用户应用程序代码直接运行在 Mirage runtime 上面。在传统操作系统上，应用程序操作底层硬件需要通过系统调用，由系统内核来实际操作底层硬件，程序需要等待内核响应；调用共享库提供的函数常常需要操作系统通过动态链接来加载库。同时系统内核提供了许多应用程序使用不到的功能，如果运行在虚拟机上，虚拟镜像体积较大。MirageOS 将应用程序编译后和运行时使用的库链接在一起生成虚拟镜像，缩小了虚拟镜像的体积，且应用程序和共享库运行在统一的虚拟地址空间，而无须进行特权转换和上下文切换，可以直接通过共享库操作底层硬件资源，大大提高了运行时的效率。

同时 MirageOS 在其上开发了 Mirage Web 服务器来提供网络服务，实验证明 Mirage Web 服务器的启动时间与 Linux 内核的启动时间相近，少于启动 Linux 内核后再启动 Apache 服务器；且 Mirage Web 服务器的可伸缩性要优于 Apache 和 Linux 的组合，更适用

于高并发的环境。

利用基于库的操作系统的概念，还可以为每个应用程序构建专门的操作系统，从而可以提高单个应用的运行效率。例如，EbbRT[34]就是这样一个可以为单个应用构建专门的库操作系统的框架，由于它既允许应用直接控制硬件，又能获得通用操作系统所提供的服务，因此可以降低开发成本。实验表明，EbbRT 可以获得比 Linux 高 2.08 倍的吞吐量，并且在将 node.js 移植到 EbbRT 时展示了其广泛适用性和易用性。Microsoft 公司也曾经尝试将其 Windows 7 操作系统改造成基于库的操作系统[35]，从而可以为像 Excel 或 PowerPoint 这样的应用提供专用的库操作系统，可以显著提高单个应用的运行效率。

总的来说，基于库的操作系统因为在运行应用程序时可以直接访问硬件资源，从而比传统操作系统的性能更好，更适用于对性能要求较高的运行环境，因而常用于数据中心领域，提高其上应用的运行性能。但是基于库的操作系统在处理多任务并行时较为困难，常常需要重写硬件驱动，由于当今硬件变化速度较快，从而会产生巨大的工作量。虚拟机可以提供高度隔离的虚拟硬件设备，所以基于库的操作系统更多地运行在虚拟环境中，为虚拟硬件设备提供驱动程序，由虚拟机管理硬件的多路并发和重复使用，同时虚拟机也为库操作系统上运行的应用提供了良好的隔离。面向云计算等虚拟化环境的库操作系统研究正在成为一个重要的主流研究方向。

（4）分离内核操作系统

LegoOS[37]是一种专门为分解化（Disaggregated）硬件系统架构设计的操作系统。将传统服务器中的硬件资源分解到通过网络连接的硬件构件中：每个构件是独立的隔离的实体，都有一个控制器和一个网络接口，且可以独立运行。硬件的发展使得硬件资源的分解成为可能：首先是网络速度的迅速增长使得可以通过网络快速访问通过网络分解的硬件构件；其次网络接口越来越接近硬件，未来硬件可以通过网络接口直接访问网络；最后，硬件设备的处理能力不断增强，应用程序和操作系统逻辑可以在单个硬件上运行。

LegoOS 采用了分离内核（splitkernel）的思想来设计操作系统。分离内核的设计关注以下要点：1）将传统操作系统的功能分解到监视器（monitor）中，每个监视器管理一个硬件构件，对其进行虚拟化和保护；2）在硬件构件上运行监视器，为每一个硬件构件配置低功耗通用内核作为控制器来支持监视器的运行，例如 ASIC，FPGA 等；3）使用网络进行通信，各个构件使用消息传递来保持一致性；4）包含全局资源管理和故障处理功能。

LegoOS 设计了三种构件，管理 CPU 的进程构件 pComponent，管理 DRAM 的内存构件 mComponent 和管理 SSD、HDD 的存储构件 sComponent。当需要访问缓存时，pComponent 根据虚拟地址向 mComponent 端请求内存数据，每个 mComponent 可以选择不同内存分配技术以及虚拟到物理内存地址的映射方法。为了保证计算效率，pComponent 端保留少量高速缓存，mComponent 则负责管理大部分的内存资源，出现脏数据时，写回 mComponent。当生成新线程时，LegoOS 使用全局资源管理来选择当前线程数最少的内核来生成新线程，保证每个内核负载均衡，尽可能减少线程调度和上下文切换的性能损耗。并使用两级方法管理分布式内存空间，进程相关的 mComponent 使用粗粒度的虚拟内存分

配，其他 mComponent 使用细粒度的虚拟内存分配，以此减少网络通信，保持良好的负载平衡和内存利用率。sComponent 端则在支持分层文件接口的基础上使用哈希表加快文件查找。

与原始 Linux 相比，LegoOS 的网络利用效率要优于 Linux，但是网络速度仍然限制了缓存访问和数据存储的性能。并对比了实际应用 Tensorflow 和 Phoenix 的运行效果，结果表明 LegoOS 的性能明显优于具有本地 SSD 的 Linux 服务器和具有远程存储器的 Linux 服务器。

2.1.2 人工智能与操作系统技术的融合

近年来，随着深度学习等人工智能技术的广泛应用，人工智能及相关技术和系统已经步入了一个蓬勃发展的新阶段，同时也对操作系统的研究产生了重要的影响。在 2017 年操作系统顶级会议 SOSP 上，针对深度学习应用的自动化测试工具 DeepXplore[40] 虽然与操作系统并没有特别直接的关系，却获得了本次会议的最佳论文奖，从一个侧面表明了人工智能与操作系统技术的深度融合趋势。

人工智能（AI）与操作系统技术的融合主要体现在两个方面：基于 AI 技术对操作系统进行优化（AI for Systems），以及针对 AI 相关系统的研究（Systems for AI）。

（1）基于人工智能技术对操作系统进行优化

当下，研究人员尝试将人工智能（AI）的研究成果应用到系统软件中，提升系统软件的性能表现、可靠性、可用性等指标。

- **基于日志挖掘的系统故障检测与诊断：**在很多数据中心和云计算平台中，部署着大规模的软件系统。由于大规模软件系统具有规模大、逻辑复杂、并发性高、错误难以定位等特点，如何检测和诊断系统故障对系统开发者和维护者构成了极大的挑战。得益于软件系统开发的规范性要求，软件系统运行过程的很多状态信息都会被输出到日志文件中，日志文件对于理解系统状态和发现潜在的性能问题很有帮助。因此，如何从日志文件中挖掘出有价值的信息，以辅助系统故障的检测和诊断引发了很多工作的探索。日志挖掘的方法有多种。基于规则的方法[88] 首先以人为的方式给定规则集，然后根据规则集对日志进行匹配和规则推理，从而达到检测和诊断异常事件的目的。该类方法的缺点在于，领域知识的缺乏和事件之间复杂的关系导致规则集很难编写。另外，规则推理这一过程非常耗时，使得基于规则的方法不适用于大规模的日志。基于日志统计特征的主成分分析方法结合日志和源代码[89-91]，能够将非结构化数据转化成结构化数据，并构造得到日志的统计特征，然后利用主成分分析方法在线检测与诊断系统故障。基于贝叶斯网络的故障定位[92] 通过构建故障推理贝叶斯网络，来推理得到每个模块包含故障的概率。基于不变性挖掘的共现模式匹配方法[93-94] 根据程序执行流存在的不变性先验，挖掘不同事件共同出现的模式，然后通过检验共现模式是否满足条件来判断异常事件的发生。这种方法只能捕获事件之间的线性关系。基于深度学习的方法[95] 利用日志中事件类型的信息和事件变量的信息，包括时间戳、事件变量值

等，通过长短期记忆网络学习事件的非线性关系，并输出事件变量值，然后比较预测值跟输入值是否匹配来判断是否发生异常事件。该方法能实时接收用户的反馈从而修正模型，并能够在线更新模型。

- **基于机器学习的分布式资源分配**：对于分布式计算任务，人们总是希望能够找到高效的计算资源分配（调度）方式，使得计算任务尽可能快速完成的同时，计算资源能够得到充分利用。近年来，研究人员提出了基于机器学习（尤其是强化学习）的方法来进行计算资源分配，让机器自己通过自我学习获得更好的资源分配策略。Mao 等人提出 DeepRM[96]，实现对集群中多任务和多种资源（例如内存、I/O、CPU 等资源）进行调度。DeepRM 使用基于 Policy Gradient 的强化学习算法[97]，使用神经网络作为决策器，以当前资源分配图和任务等待序列作为神经网络的输入。DeepRM 取得了比非机器学习调度算法更好的性能。在神经网络的训练中，人们也使用深度强化学习方法对神经网络中各个节点进行计算资源（CPU 和多 GPU）的分配，从而最大程度减少神经网络的运行时间。目前已有方法主要还是实验模拟，如何将其真正投入实际应用还需要进一步探索。
- **基于机器学习的索引构建**：传统的索引构建方法，如 B 树、哈希表、布隆过滤器，目标都是将关键字映射到对应的位置，但都没有考虑具体的数据分布。基于机器学习的索引构建方法旨在利用实际的数据分布，将机器学习的方法引入到索引构建中，以提高索引性能。已有基于机器学习的索引构建围绕三类索引任务展开，包括：基于 B 树的索引、基于哈希表的索引和基于布隆过滤器的索引。基于 B 树的索引构建方法特点是记录位置的条目是排好序的。传统的 B 树可以看作是基于决策树的模型，该模型可以很好地利用 cache 来加速。现有研究[98]使用简单的神经网络模型和决策树模型来构造循环索引模型，相较于传统模型，循环索引模型能达到更好的索引效果。现有研究[99]还提出了基于机器学习的哈希表索引结构，使用了循环索引模型学习哈希函数，相较于传统的哈希函数，基于机器学习的哈希表索引结构可以达到更低的冲突率和更低的存储要求。但查询速度不如传统的哈希函数。基于布隆过滤器的索引模型主要目标是判定查询关键字是否存在。现有研究[100]提出了可学习的布隆过滤器模型，可以达到更低的假正例率和更低的存储要求。

（2）针对人工智能系统的研究

另一方面，研究者也在操作系统等系统软件方面为智能融合时代的 AI 系统做出优化与新型设计。AI 系统作为一个系统软件，其性能与存储、网络、调度和模型优化等方面都密切相关。研究者针对 AI 系统中模型训练的特殊性，提出了大量原创的技术，对 AI 系统性能进行不断优化。这些技术与具体的 AI 平台无关，可以结合不同平台的架构进行集成实现。

- **存储层次优化**：深度学习框架仅可与部分现有存储系统集成，可能无法获取某个存储系统上的数据进行训练，导致训练效率和效果降低。分布式存储系统（如 HDFS，Ceph）和云存储（如 AWS S3，Azure Blob Store，Google 云存储）为用户

提供多种存储方式的同时，也增加了机器学习系统正确配置和使用不同存储系统的困难性。除此之外，计算资源与存储资源分离导致需要使用远程数据时，额外的网络传输会提高系统成本并增加处理数据的时间。Alluxio[101]可以解决机器学习系统的数据访问问题。Alluxio 最简单的形式是一个虚拟文件系统，它透明地连接到现有的存储系统，并将它们作为一个单一的系统呈现给用户。使用 Alluxio 的统一命名空间，可以将许多存储系统挂载到 Alluxio 中，包括 S3、Azure 和 GCS 等云存储系统。由于 Alluxio 已经与存储系统集成了，因此深度学习框架只需与 Alluxio 进行交互即可访问所有存储中的数据。这为从任何数据源获得数据并进行训练打开了大门，从而可以提高深度学习学得的模型的性能。Alluxio 还提供常用数据的本地缓存。当数据远离计算时，例如存储环境中的计算分离，该缓存技术非常有用。由于 Alluxio 可以在本地缓存数据，所以不需要通过网络 IO 来访问数据，从而使得深度学习训练的成本会更低，并且花费的时间会更少。

- **网络层次优化**：随着数据量增多，模型增多，大规模分布式处理成为必须途径。服务器间传输数据所造成的资源消耗问题也随之产生。服务器之间如何互联、参数在各个节点之间如何高效传输也是热门研究方向。香港科技大学陈凯教授团队在此方向发表了一系列成果，并发布了分布式机器学习系统星云 Clustar[102]。该系统基于 RDMA 的新型数据中心网络技术，还利用了智能网络计算平台（smart in-network computing）技术，并且在应用感知的路由和调度的算法上做了优化。卡耐基梅隆大学（CMU）和 Petuum 推出了新一代高效的分布式深度学习通信架构 Poseidon[103]。Poseidon 是一个易于使用，并能提升深度学习程序在 GPU 集群性能的通信架构。Poseidon 利用深度程序中的层级模型结构而叠加通信与计算，这样以减少突发性网络通信。此外，Poseidon 使用混合的通信方案，并根据层级属性和机器数量优化每一层同步所要求的字节数。已存的深度学习程序不需要更改代码就能通过 Poseidon 在多个机器上自动最优地实现并行化，加速效果和机器数量呈线性增长关系。
- **模型训练任务调度优化**：整体同步并行计算模型 BSP 是一种异步 MIMD-DM 模型（DM：distributed memory）。BSP 模型能够保证模型的收敛性，但各节点之间等待会造成大量计算资源浪费。为缓解这一问题，Eric Xing 提出延迟同步并行计算模型 SSP 模型[104]，利用机器学习算法的容错性构建的一种并行计算模型。机器学习的容错性是指在误差容许的范围内，算法迭代计算过程中可以存在一定的误差。给定了一个时间窗口，并保证了两个线程维持在这个时间窗口内，这样使得误差保持在一定的范围，并且有理论收敛性的保证。该方法既能减轻慢节点拖慢整个系统运行速度的程度，又能保证模型参数的最终收敛。然而参数可能是相互依赖的，因此随机的并发更新可能会导致收敛速度缓慢甚至导致算法失败的错误。同时模型参数以不同的速率收敛，收敛慢的参数子集会限制 ML 算法的完成。因此 CMU 的 Eric Xing 教授团队提出了调度式的模型并行性（schedule model parallelism，SchMP），该编程方法考虑了参数依赖和不均匀收敛，通过有效地调

度参数更新，提高了 ML 算法的收敛速度。为了在规模上支持 SchMP，他们开发了一个分布式框架 STRADS[105]，它优化了 SchMP 程序的吞吐量，并将四个通用的 ML 应用程序编写为 SchMP 程序。

2.1.3 面向新型应用场景的操作系统

除了面向传统计算设备的新型操作系统之外，随着大数据、物联网、“互联网 +”等新型应用模式的普及，还出现了各种面向具体应用领域的新型网络化操作系统。虽然有些操作系统也是由某些公司主导研发的，但是由于这些操作系统原型尚未进入大规模产业应用阶段，因此我们把这些新型操作系统也放到学术界研究进展进行探讨。

- **家庭操作系统**：Microsoft 公司曾经提出家庭操作系统 HomeOS[20] 的概念，为家庭智能应用的用户和开发者提供了类似于 PC 的技术抽象，包括网络设备抽象接口、跨设备的任务接口，以及专为家庭环境设计的管理界面。HomeOS 提供了与特定设备及其功能无关的家庭操作系统内核，用户和应用程序可以通过集中式操作系统查找、访问和管理智能设备。HomeOS 为应用程序开发者提供独立于设备底层信息的高级 API，将设备抽象化并提供统一的管理原语和接口，方便家庭智能设备应用的开发和用户管理使用。家庭操作系统扩大了传统操作系统设备的范畴，并提供了面向智能设备的抽象和管理方法。HomeOS 还提供了面向家庭的应用程序商店，拥有数十个家庭管理应用，并在多个真实家庭环境中进行了长时间试用。
- **机器人操作系统**：斯坦福大学最早研发的 Robot Operating System（ROS）[14-16] 是一个面向机器人编程的操作系统。虽然 ROS 出现较早，但是近年来逐渐成为面向机器人系统的一个主流操作系统，也出现了针对 ROS 的很多研究工作[17-19]。ROS 实际上是一个元级操作系统，它与传统操作系统提供的服务类似，会提供硬件抽象、低级别的设备控制、执行常用的功能函数、消息传递、包管理等服务。但是，ROS 运行在传统操作系统上，而非传统底层硬件上，同时它所提供的硬件抽象、设备控制和服务的主要对象是机器人而不是传统硬件设备。有别于传统操作系统基于硬件一致性缓存的通信，ROS 的通信方式主要基于网络实现的，例如基于服务的同步远程过程调用（RPC）通信，基于 Topic 的异步数据流通信，还有参数服务器上的数据存储。同时 ROS 提供了跨平台编写代码的工具和库，能够简化不同平台下机器人的开发过程。
- **智慧城市操作系统**：英国公司 Living Plan IT 最早提出一种“智慧城市”操作系统 Urban OS[38]。该操作系统提供把包括水、电、交通等服务连接在一起的统一城市平台。与传统操作系统相比，这种城市操作系统更注重系统的鲁棒性，因为在该系统上会运行许多生死攸关的服务。该公司同时计划通过嵌入成千上万的传感器作为监控设备，为一个新建设的办公街区建立智能照明和供热系统。维也纳理工大学的研究人员也提出智慧城市操作系统（Smart City Operating System）[38-39]，用来构建基于云计算的智慧城市应用生态系统，无缝连接所涉及的不同利益相关者，支持智慧城市应用的高效构建、部署和运行。

- **传感器网络操作系统**：TinyOS[43-44]是加州伯克利大学提出的面向无线传感器网络（Wireless Sensor Networks，WSN）的操作系统，由于它结构简单、所需内存非常小，可以运行在能力受限的无线传感器网络节点之上，在早期得到了较为广泛的应用。随着传感网络的广泛部署，分布式计算平台推动了对于适用于传感网络的编程范型的研究。在加州伯克利大学领导的 TerraSwarm 项目的支持下，研究了拥有海量的智能传感器网络环境下的计算问题。Swarm 是一个基于对象的适用于传感网络的分布式系统[41]，主要创新在于无缝的将现实时空中的物体集成到应用程序的计算环境中。在这个项目中提出了面向大规模传感网络的操作系统 SwarmOS[40]，可以为上千个传感器提供编程抽象以及系统层面的支持。同时还提出了面向 ManyCore 的 Tessellation 操作系统[41]，用来支持实时、响应式和高吞吐量的大规模并行应用。

2.2 产业界现状

在产业界，除了广为应用的 Windows、Mac OS/iOS、Linux 等主流操作系统之外，也有很多新的操作系统得到了推广和应用，其中典型的包括云操作系统（Cloud Operating Systems）[45]、Web 操作系统、物联网操作系统等，除此之外，像微软、Google、Facebook 这样的大公司也在着力打造自己的操作系统生态。这里我们主要针对大型互联网公司在操作系统领域的布局和进展来进行探讨。

2.2.1 Google

Chrome 操作系统[106-107]是 Google 的云连接桌面操作系统。这款专注于网络应用的网络操作系统可为大多数廉价的 Chromebook 提供动力，为中等水平或基本需求的人们提供了低成本的笔记本电脑选项。Chrome OS 与 Android 应用程序的兼容性赋予了 OS 新的生命，并带来了数百万种新软件的选择。Chrome 操作系统是 Google 设计的基于 Linux 内核的操作系统。它源自免费软件 Chromium OS，并使用 Google Chrome 浏览器作为其主要用户界面。Chrome 操作系统是 Google 的云连接桌面操作系统，其中应用程序和用户数据都驻留在云中：因此，Chrome OS 主要运行 Web 应用程序。Chrome 操作系统具有集成的媒体播放器和文件管理器。它支持类似于本机应用程序的 Chrome 应用程序，以及对桌面的远程访问。Android 应用程序于 2014 年开始可用于操作系统，并于 2016 年在受支持的 Chrome 操作系统设备上引入了对 Google Play 整体的 Android 应用程序的访问。这是通过在轻量级 Linux 内核运行虚拟机，并且在其中运行容器所实现的。Chrome 操作系统只能预安装在 Google 制造合作伙伴的硬件上。

Chrome 操作系统最初用于上网本等辅助设备，而不是作为用户的主要 PC。虽然 Chrome 操作系统支持硬盘驱动器，但 Google 出于性能和可靠性方面的考虑以及其访问应用程序具有较低容量要求，要求其硬件合作厂商使用固态驱动器和远程服务器。Chrome 操作系统的工程总监 Matthew Papakipos 称，Chrome 操作系统消耗的驱动器空间是

Windows 7 的六十分之一。Chrome Enterprise 于 2017 年推出，包括 Chrome OS，Chrome 浏览器，Chrome 设备及其用于企业用途的管理功能。教育部门是 Chromebook，Chrome OS 和基于云的计算的早期采用者。Chromebook 在教室中得到广泛使用，基于云的系统的优势也已在其他部门（包括金融服务，医疗保健和零售）中赢得了越来越多的市场份额。Google 已与多家领先的 OEM 厂商合作开发了 Chrome 设备，其中包括 Acer、ASUS、Dell、HP、Lenovo 和 Samsung。

Fuchsia[108-109]是 Google 目前正在开发的开放源代码的新一代操作系统。当该项目于 2016 年 8 月以自托管形式 git 出现而没有任何正式公告时，它才首次对外公开。GitHub 项目称 Fuchsia 可以在许多平台上运行，从嵌入式系统到智能手机，平板电脑和个人计算机。2019 年 7 月 1 日，Google 宣布了该项目的首页 fuchsia. dev，该网站提供了新发布的操作系统的源代码和文档。尽管没有正式宣布，但对代码的检查表明该操作系统可以在通用设备上运行，包括汽车仪表娱乐系统，交通信号灯和数字手表等嵌入式设备，一直到智能手机，平板电脑和个人电脑。该操作系统与 Android、Chrome OS 不同，因为它基于 Zircon 微内核（以前称为 Magenta），而不再是基于 Linux 内核。Chrome 和 Android 的高级副总裁 Hiroshi Lockheimer 认为 Fuchsia 是 Google 围绕新一代操作系统的实验之一。

Fuchsia 操作系统的内核 Zircon 源自 Little Kernel，一种用于嵌入式系统的小型操作系统。福布斯将 Zircon 的演变描述为：Zircon 以前被称为 Magenta，它的设计目的是可以扩展到从嵌入式 RTOS（实时操作系统）到各种移动和台式设备的任何应用程序。因此，人们一直猜测 Fuchsia 将成为 Android 和 Chrome OS 的继承者，会将两者的功能相结合并保持向后兼容，可以运行基于二者之一的旧版应用程序。简而言之，该产品旨在在从 32 位或 64 位 ARM 内核到 64 位 x86 处理器的任何设备上运行。Zircon 微内核的软件文档详细介绍了其提供的功能。Zircon 包括一个微内核和一些运行在用户态的系统服务、驱动程序和库。目前 Zircon 提供了 170 多个系统调用。

有报道称 Fuchsia 的愿景是成为人和人工智能之间的交互接口。多年来，智能手机变得越来越强大，但它们仍然停留在 10 年前由 iPhone 创建的应用程序范式中。用户要订购比萨，订购电影或音乐会门票，预订机票，支付停车费，听音乐，观看 Netflix 等等，对于所有这些事情，都需要使用一个单独的应用程序。但是实际上用户并不在意应用程序本身，而只是需要其提供的功能，那么在智能时代用户还需要把要做的每件事都在手机上排成一行应用程序吗？在新的人机物智能融合时代，聊天机器人可以为用户订购比萨、鞋子、演唱会门票等等，应用程序的必要性不再突出，这也是 Facebook 也在致力于研发下一代操作系统的原因之一。Google 借助 Fuchsia，试图摆脱过去几十年来我们以应用程序为中心的方法，从而创建一个随处可用的智能平台，并且可以随设备在各个设备上进行跟踪。因此，Fuchsia 不仅仅是一个操作系统，它还可能会定义出一种新的软件制作方式，可以在信息不再成为数据孤岛的情况下创造交互式体验，从而成为用户和人工智能之间的接口。

2.2.2 Facebook

Facebook 公司正在自主研发操作系统以摆脱对于谷歌 Android 操作系统的依赖[110]。

目前，Facebook 推出的智能设备包括 Portal 智能显示器和 Oculus VR 产品等使用的都是 Android 操作系统。新的操作系统将在当前和将来的 Facebook 设备上使用，包括新一代的智慧显示屏、虚拟现实设备以及智能眼镜等可穿戴设备。Facebook 更换使用自己的操作系统也将为 Facebook 诸多应用系统，包括一直在开发的语音助手，提供更好的运行环境。Facebook 首席执行官马克·扎克伯格（Mark Zuckerberg）在 2019 年的公司年度股东大会上指出，语音技术对公司的未来非常重要，并且将成为公司未来制造的多种智能设备的关键部分。自 Facebook 终止了 Facebook Messenger M 助手 M 之后，新的语音助手自 2018 年初开始投入使用。

Facebook 在 2013 年曾尝试为智能手机构建自己的操作系统，但带有原型的 HTC 手机以失败而告终。如今，在人机物融合时代到来的契机下，Facebook 再次尝试构建面向智能应用的操作系统。除了操作系统，该公司同时研究操作系统之下的芯片，希望能够取得像苹果公司同时控制 iPhone 和 iOS 一样的潜在优势。Facebook 计划在未来推出智能设备上安装自己的操作系统和应用程序，并且用自己的语音助手取代 Alexa、Siri 和 Google Assistant。Facebook 将以新一代操作系统为切入点打造属于自己的生态。通过迁移到自己的操作系统，Facebook 可以拥有更大的自由度来将社交互动（以及数据隐私）更深地植入其设备。它还可以防止 Google 与 Facebook 之间的分歧使其产品路线图脱轨。Facebook 正在探索智能应用和设备究竟需要什么，包括可能与其他公司合作或构建专门用于增强现实的自定义操作系统。

2.2.3 Apple

除了 Mac OS 和 iOS 之外，Apple 公司也在开发面向新的硬件设备的其他操作系统。Apple tvOS[111-112]是运行在第四代和第五代 Apple TV 上的操作系统，可在 Apple 机顶盒上提供易于导航的电视观看体验。tvOS 支持从应用商店下载可在 Apple TV 上使用的各种不同的应用程序和游戏，并且更加注重人工智能的使用，比如利用 Siri 语音控制，Apple Remote 或 iPhone 和 Apple Watch 上的 Remote 应用程序获得想要观看的内容。Apple 会定期向 tvOS 添加新功能，而 2020 年对 tvOS 的更新为 tvOS14。在 iOS 上一样，Siri 可以打开应用和游戏并响应命令，而不仅仅是简单的内容搜索。例如，Siri 可以显示体育比分，电影时间，天气和库存状态。Siri 还可以通过“打开增强的语音”命令来更改特定设置，该命令可以增强对话并柔化音乐和声音效果，或者为字幕提供“打开隐藏式字幕”功能。tvOS 上 Siri 的理解能力很强，并且可以回答基于主题的搜索，例如“向我展示 80 年代的电影”或“向我展示具有恐龙的电影”或“查找有关建筑的纪录片”。当一个命令中包含多个主题时，例如“向我展示 1960 年代的间谍电影”或“向我展示 90 年代的高中喜剧”，Siri 也可以理解。tvOS 是基于 iOS 设计的，tvOS 继承了 iOS 和 macOS 的许多辅助功能，非常注重人际交互的设计。tvOS 包括 Apple 的 VoiceOver，Zoom 和 Siri 技术，以帮助盲人和视力障碍者。Apple 的屏幕阅读器 VoiceOver 支持 30 多种语言，可让视障用户知道视觉显示内容，并输入对屏幕提示的响应。VoiceOver 使用的手势与其他 Apple 产品（轻拂、敲击和旋转）相似。

2.2.4 Microsoft

在机器人操作系统领域，Microsoft 启动了 ROS for Windows 的项目[113]，标志着把 Windows 10 IoT Enterprise 的可管理性和安全性带入创新的 ROS 生态系统。当下，先进的机器人正在改变我们在工作和家庭中的生活。仓库机器人可以在第二天把货物交付给在线购物者，许多宠物主人依靠机器人吸尘器保持地板清洁。制造业、交通运输、医疗保健和房地产等各行各业都从机器人中受益。随着机器人的发展，开发工具也随之发展。许多开发人员利用机器人操作系统（ROS），这是一组库和工具，可用于构建复杂的机器人。ROS 在全球许多尖端机器人项目中得到了使用。Windows 已经成为机器人和工业系统值得信赖的一部分。借助适用于 Windows 的 ROS，开发人员将能够使用熟悉的 Visual Studio 工具集以及丰富的 AI 和云功能。微软希望通过将先进的功能（如硬件加速的 Windows 机器学习、计算机视觉、Azure 认知服务、Azure IoT 云服务和其他 Microsoft 技术）应用于家用、教育、商业和工业机器人，从而将智能优势带入机器人领域。制造商希望使机器人更加了解周围环境，使其更易于编程并且更安全地进行操作。世界各地的政府，制造商和学者都在投资于下一代制造业，有时也称为“工业 4.0”。

从以上例子可以看到，国际主流的大公司都在关注人机物融合时代的操作系统，面向物联网和泛在计算等各个不同领域的信息化需求正在构建各种不同的新型操作系统。同时，各大公司也非常重视操作系统生态的建设，特别是与现有操作系统之间的兼容性、应用跨平台执行和应用开发环境等。

3 国内研究进展

在操作系统相关领域，国内在学术和产业上均处于明显的落后状态。在产业应用方面，国内计算机厂商大多数长期采用微软的 Windows 操作系统，导致国产操作系统，特别是桌面操作系统，缺乏良性发展的生态环境。在学术研究方面，一方面由于缺乏具体的应用场景，另一方面由于操作系统领域的研究投入大、周期长、见效慢，在国内组织起高水平的操作系统研究队伍相对不是很容易，因此与计算机学科中的其他领域相比，国内在操作系统领域与国际先进水平之间的差距相对较大。

3.1 学术界进展

虽然国内在操作系统的基础研究方面还相对落后，但是经过以上海交通大学为代表的多个操作系统研究团队的努力，在最近几年已经取得了长足进展，特别是在近 5 年来，在 OSDI、SOSP、ATC、FAST 等操作系统顶级学术会议都可以看到来自国内学术界的论文数量在持续增加，研究范围也涉及很多操作系统的关键技术方向，包括操作系统的系统安全和形式化验证、多核操作系统、云计算操作系统、机器人操作系统等。

随着自动驾驶以及 IoT 等人机物融合场景的不断涌现，微内核有着越来越重要的地位。虽然微内核有着宏内核难以获得的许多好处，如代码行数少、易于形式化验证、强隔离性和更好的容错性等。然而，由于进程间通信（IPC）等隔离域间的跨域调用的开销，微内核架构要达到或超越宏内核的性能方面存在一些挑战。针对当前的操作系统中资源隔离带来的性能开销，上海交通大学提出了基于现有硬件特性的直通式跨域调用方式，消除调用过程中操作系统的干预，在带来细粒度隔离的同时提供高性能[64-65]；结合相关设计研制了新型跨域调用方案 SkyBridge[66]，相比 seL4、Google Fuchsia、Fiasco 等主流微内核系统的跨域调用在保持强隔离性的同时性能提升超过 10 倍。上海交通大学进而提出低时延跨隔离域交互方法 UnderBridge[67]，将原本为用户态设计的 Intel PKU 特性用于在内核态构建隔离执行域和高效跨域通信方法，从而能够把微内核中原本运行在用户态独立地址空间的系统服务拉回到内核态隔离执行域中运行，从而在保证微内核隔离性的同时极大降低进程间通信的开销，针对进程间通信频繁的应用提升性能 2 到 13 倍。上海交通大学还通过使用软硬件协同的方法提出微内核场景下的进程间通信机制 XPC[68]，让微内核能够超越宏内核的性能。XPC 提出了两个新的硬件原语，并且基于 RISC-V 和 ARM64 等多个平台提出面向微内核的 IPC 设计方案，通过 FPGA 仿真实现并进行了性能测试。XPC 将 IPC 性能提升几个数量级，并显著提升了 seL4、Fiasco、Google Fuchsia 等操作系统的性能。

虚拟化技术与操作系统的实际使用紧密相关，在人机物融合的操作系统中，也至关重要。随着系统虚拟化功能的不断完善，虚拟机管理软件的代码规模也日趋庞大，导致虚拟机管理软件漏洞的增多。因此，提升虚拟化方法的可信性的一个关键是有效减少虚拟化系统的可信基。上海交通大学提出了资源管理与隔离分离的虚拟化方法，打破资源管理功能与隔离功能的深度耦合。基于此方法，设计了基于嵌套虚拟化的极小化可信基构建方法，研制的 CloudVisor 系统[69]将虚拟化平台可信基从数十万行代码减少至数千行。基于此工作，上海交通大学进而提出解耦式嵌套虚拟化技术，在恶意虚拟机监控器环境下实现虚拟机隐私的高效保护。基于此技术开发了 CloudVisor-D 系统[70]，将传统嵌套虚拟化的架构分解为两个模块，能在不引起虚拟机下陷的情况下将大多数虚拟机操作转交给可信的守卫模块处理，大幅度提升嵌套虚拟化技术的性能。

GPU、TPU 等新的异构计算设备与 NVM 等异构存储设备的发展推动着操作系统的发展，同时为自动驾驶、IoT 等人机物融合系统提供了技术保障。然而这些异构资源缺乏原生虚拟化能力，从而只能选择独占以支持强隔离或弱隔离下的共享，并且缺乏跨主机进行动态迁移的能力。上海交通大学提出隔离与功能分离的软硬件协同虚拟化方法，首次实现了在一个物理 TrustZone 上虚拟出多个 TrustZone[71]，该技术带来的性能损耗低于 5%；提出了基于访问频率的 NVM 虚拟化方法[72]，首次在虚拟化环境中实现了 NVM 虚拟化；提出了基于动态状态跟踪与推断的一致迁移方法[73]，克服了异构硬件（如 SGX 等）的部分状态难以被软件管理、迁移过程难以保持一致等难题，研制了异构虚拟机迁移方法，支持异构虚拟机动态迁移，且性能开销小于 5%；提出了利用多 CPU 与多网卡同时迁移虚拟机状态的方法，解决了异构资源虚拟机迁移状态多、难以收敛等难点问题，

研制的虚拟机并行迁移技术，将虚拟机在线迁移的总延迟降低 10 倍，将服务的离线时延降低最高 280 倍[74]。

Java 等托管语言方便开发人员进行上层应用等开发和维护，因而在人机融合操作系统中同样需要进行关注。Java 等托管语言依赖于 Java 虚拟机等语言层虚拟化技术，其中重要的问题之一是内存的垃圾回收机制的性能。针对之前并行垃圾回收器的不足，上海交通大学提出了一种新型垃圾回收器 Platinum[75]，在垃圾回收期间为应用线程和垃圾回收线程提供了隔离的执行环境，使其互不干扰。Platinum 还基于 Intel MPK 特性消除了传统并行垃圾回收器中的屏障指令，进一步降低了软件开销。Platinum 与之前的并行垃圾回收器相比能降低最多 74% 的 p99 尾时延，并保持较低的 CPU 资源占用率，使应用能够同时达到高吞吐和低时延的目标。

为了在人机物融合环境中更高效地保存数据，除了 NVM 的虚拟化之外，在如何使用 NVM 等新型存储设备的问题，上海交通大学也开展了一系列研究工作。针对 NVM 写时延超过 DRAM 近一个数量级的问题，上海交通大学提出并实现了 SoupFS 文件系统[76]。其将持久化元数据与最新元数据分离，通过异步操作维护二者一致性，从而克服了 NVM 高写时延的问题，并消除文件系统元数据操作的关键路径中内存刷除等高延迟操作。针对高级语言缺乏对 NVM 的有效支持问题，上海交通大学提出了 Espresso 系统[77]，首次在 Java 运行时内部对 NVM 提供了支持，使 Java 用户能以访问普通 Java 对象的方式直接操作 NVM 中的数据。针对不同应用的需求，Espresso 还提供了不同粒度的接口来操作和管理对 NVM 中的数据，并基于 Java 虚拟机的特点，保证了 JVM 元数据的崩溃一致性，使 JVM 在错误退出后依然能够重新启动并正常恢复。在保证用户数据和 JVM 元数据持久性的同时，相比之前基于 JNI 的工作 PCJ 有最多 256 倍的性能提升。为了在 NVM 上提供具有 ACID 属性的事务支持，近年来研究者提出并研究持久化事务内存（PTM）编程模型。上海交通大学提出了第一个基于快照隔离且对读操作友好的 PTM 系统 Pisces[78]。Pisces 通过复用重做日志作为新版本的方法，高效地将多版本控制（MVCC）引入到 PTM 设计之中；进一步提出双版本控制（DVCC）和三阶段提交方法，从而能够实现几乎不阻塞读操作。实验数据表明，Pisces 能够达到现有系统高达 10 倍以上吞吐量，且具有更好的可扩展性。NVM 的可字节寻址性，让用户态可以直接访问存储设备，进而使实现用户态文件系统成为可能。上海交通大学提出用户态文件系统 ZoFS[79]，通过设计新的抽象，将文件系统的保护和管理职责分离，减少了不必要的性能开销。在保证安全和隔离性的前提下，在用户态更加充分地发挥出 NVM 的性能优势。

在进入到多核乃至众核时代后，操作系统的可拓展性成为衡量操作系统性能的重要标准之一。为了在人机物融合操作系统中充分发挥多核处理器的性能，提升操作系统的可扩展性尤为重要。在众核系统中由于 CPU 缓存的串行访问，多个核均需通过全局变量进行通信，导致核间通信时间显著增加，从而影响操作系统性能。为此，上海交通大学提出了去通信化读写同步方法，将读的计数器进行分布式化，并通过一个基于版本的共识协议，将读写数据均需要通信的同步机制优化为只有写数据需要通信，该方法将通信开销由 O(n * c) 降低到 O(c)，其中 n 为核的数量，c 为无竞争时每次访问缓存的时间，

消除了 Linux 内存管理模块 90% 以上的通信开销[80-81]，提高了操作系统与虚拟化的扩展性。

随着使用 ARM 架构的低功耗、高性能服务器的兴起，包括亚马逊、华为在内的国内外多家企业已经开始部署 ARM 服务器，向 ARM 服务器上迁移现有应用迫在眉睫。ARM 与 x86 架构的主要区别之一在于其使用了不同的内存模型。为此，上海交通大学还针对 ARMv8 服务器中多种不同访存保序方案的性能影响进行了全面、详尽的分析与测试[82]。基于这些分析，上海交通大学为开发者移植现有应用到 ARMv8 服务器提供了在不同场景使用保序方案的指导意见。针对内存屏障的特殊使用模式，提出了一种有效的机制，利用硬件访存的原子性消除了使用硬件内存屏障导致的巨大开销，带来 10% 到 380% 的性能提升，最终达到近似没有使用硬件内存屏障的理想性能。

软件可靠性在部分人机物融合场景中至关重要。软件缺陷（Bug），尤其是系统软件的缺陷在自动驾驶等场景中可能会造成使用人员的生命危险。形式化验证利用数学方法，从源头验证避免漏洞，保证系统正确，是避免软件缺陷、提升软件可靠性的重要方法之一。上海交通大学对并发程序精化验证的相关理论成果进行扩展，在定理证明工具 Coq 中设计并实现了支持帮助机制的并发精化验证框架，成功地验证了 AtomFS[83]，保证了 AtomFS 的接口在任意并发环境下的功能正确性和原子性。除了在正确性上，与 Ext4 等成熟文件系统相比，AtomFS 具有良好的多核扩展性，在运行实际的应用程序时能取得合理的性能。

分布式通讯是人机物融合场景中必不可缺的关键部分之一。而分布式事务是一种有保障的分布式通讯和处理方法。传统的分布式事务通过两阶段提交等方式进行多轮大规模消息同步，存在时延高、可扩展性差等问题。随着新型网络设备的传输时延已经接近本地访存速度，上海交通大学提出将传统“移动计算而非移动数据”的模式变为“移动数据而非移动计算”的原位计算模式，提出了并发控制协议和 RDMA 一致性同步方法，实现了本地事务内存到分布式事务内存的转换，研制了分布式内存事务系统 DrTM[84-85]。原位计算模式将分布式事务变为本地事务，避免了两阶段提交的多轮大规模消息同步等问题。DrTM 在 6 个节点的集群中获得了超过每秒 500 万事务的吞吐能力，相比之前最快的 Calvin 系统提升超过 19 倍。上海交通大学进一步提出分布式乐观并发控制协议并实现了 DrTM + R 系统[86]，提供了事务的高可用性。根据事务处理不同阶段设计了混合式并发处理协议，充分结合原位计算与 fork-join 模型的优势[87]，相比单一模式性能进一步提升超过 40%。

在云计算操作系统方面，陈左宁院士等人[47]从虚拟机管理器设计、虚拟资源调度与管理、分布式文件系统与容错、大规模数据处理与服务、应用感知的系统优化等角度，研制云计算共性基础核心软件，并从构建自主可控的云计算生态角度入手，在容器在线迁移、新型硬件的虚拟化技术及跨机器的虚拟化架构、大规模计算集群在离线负载混合调度、基于纠删码的高可靠自维护存储系统等方面取得突破，提出了云 OS 最小内核和应用 API 标准规范，通过提供云 OS 最小内核的开源参考实现，逐步形成具备多层次一致标准的云计算应用生态。国防科技大学等单位提出以网络资源的按需聚合和自主协同为核心，建立虚拟计算环境（iVCE）[48]的思路。以“聚合与协同”为构建虚拟计算环境的新

途径，针对互联网资源的自然特性，研究了开放环境下的按需聚合问题、分布自治资源的自主协同问题以及聚合与协同的计算性质。iVCE 建立在开放的网络基础设施之上，为终端用户或应用系统提供和谐、可信、透明的一体化服务。近年来，在 iVCE 的基础上，进一步发展了云际计算（Joint Cloud computing）[49]，以多个云服务实体之间开放协作为基础，通过多方云资源深度融合，方便开发者通过“软件定义”方式定制云服务，创造云价值的新一代云计算模式。

在透明计算模型的基础之上，清华大学从用户控制的云计算（Customer Controlled Cloud Computing）角度出发，以用户端提供用户服务、网络服务器提供程序和数据的存储、对网络软硬件资源进行管理提出了基于透明计算的云计算操作系统（TransOS）的概念。把传统操作系统，例如 Linux、Windows 等也定义为云操作系统中的资源。把云计算操作系统定义为运行在传统操作系统与计算机主板 BIOS 之间，对包含各种传统操作系统在内的网络资源进行管理的元级操作系统（Meta OS）。近年来，清华大学在分布式图计算系统[52-53]、类脑计算系统[54]等领域也取得了很多较好的研究成果。

在面向机器人的操作系统方面，杨学军院士等人在面向单个机器人的 ROS 操作系统基础上，研制了面向机器人集群应用的 micROS 操作系统[50]。MicROS 设计了以集群机器人系统作为整体进行管理的分布式的架构，并将每个机器人系统本身按照层次化进行管理。MicROS 还提供了支持 OODA（观察 - 调整 - 决策 - 行动）行为模型的自主行为管理机制，并支持机器人集群的群体智能策略。

针对网络环境下的复杂网构软件，北京大学、南京大学等单位研究了网构软件（Internetware）[55-57]的概念和技术体系，研究满足质量需求并保障可信度和服务质量的软件构造方法，以及凝练共性管理功能并保证软件可信、高服务质量运行的软件运行支撑技术。在前期操作系统和中间件方面多年的研究工作基础上，北京大学正以网构软件这一新型软件形态作为切入点，针对新型企业计算（Enterprise Computing）的需求，构造面向网构软件的网络化操作系统，简称网构操作系统（Internetware Operating System）[24,22]。

北京大学研制的燕云管理平台[58]是一个面向云管理的网构操作系统，实现了对服务器、存储、网络、软件平台等基础软硬件资源的集成与配置管理，支持公有、私有与混合 IaaS 云的按需构造与管理。燕云提供了 Power 和 x86 混合 IT 架构的支持，并支持基于 Power 平台的 PowerVM 以及基于 x86 平台的 KVM、VMware、Xen、Hyper-v。燕云不仅管理了北大、清华、南大、中科院等单位的数十台服务器并提供 IaaS（Infrastructure as a Service）和 PaaS（Platform as a Service）服务；而且通过 OEM 的方式陆续转化为联想和方正等多个 IT 企业的云管理产品，广泛应用于政务、交通、电信、医疗等多个行业领域的云计算平台，拥有逾百家大中型客户。另外，北京大学团队还提出了面向校园的网构操作系统 CampusOS[59]。与 HomeOS 和智慧城市操作系统类似，CampusOS 主要用于管理大学校园里的联网资源和校园应用的操作系统，可管理教师、学生、课程、组织机构数据，以及用户设备上生成的数据。CampusOS 为校园应用的开发提供 SDK 支持，并提供了相应校园应用市场。同时，CampusOS 的功能和 SDK 中的 API 也可以由开发者自由地添加，支持灵活的应用构建与良好的可扩展性。

为了解决信息孤岛业务数据和功能与第三方系统之间难以进行高效互操作的问题，北京大学在多年系统软件研究的基础上，发现了信息系统内部基于云－端融合特性的反射回路，提出了颠覆式的互操作技术途径—“黑盒”思路，发明了云－端融合系统的资源反射机制与高效互操作技术[60]。该技术将信息系统视为黑盒，通过对系统客户端的外部监测与控制来实现系统业务数据和功能的高效互操作，可以将信息孤岛开放效率平均提升 2 个数量级。基于该技术构建了面向云端融合系统，特别是遗产系统的数据管理和共享的网构操作系统，基于该系统的产品和解决方案应用于国家政务信息系统整合共享工程、国家互联网＋政务服务试点工程等一系列国家重大任务。

由于操作系统的基础性和重要性，形式化验证技术对于操作系统的正确性和安全性可以起到至关重要的作用。南京大学和中国科技大学的研究团队在利用形式化方法对操作系统内核进行验证方面进行了很多研究工作。其中包括一个用来验证抢占式操作系统内核的实用验证框架[61]，通过描述语言来定义操作系统内核的高层抽象模型，并通过 Coq 证明器证明一个商用的抢占式内核 C/OS-II 的许多机制的正确性，包括调度器、中断处理、消息队列和互斥锁等。另外，为了形式化验证运行在 SPARCv8 处理器上的嵌入式操作系统，还对 SPARCv8 处理器指令集体系结构（ISA）进行了形式化的描述和自动化验证[62-63]。

3.2 产业界进展

在操作系统的产业应用方面，虽然国内计算机系统主体采用的还是基于 Windows、Mac 和 Linux 的系统解决方案，但是很多企业与研究机构合作，经过多年的研发努力，也研制了多个有影响力的操作系统，并在一定范围内取得了规模化应用。

国产服务器和桌面操作系统的主要代表之一是麒麟操作系统（Kylin OS），包括银河麒麟和中标麒麟等不同版本，可以支持服务器和桌面计算机。麒麟操作系统是由国防科技大学、中软公司、联想公司、浪潮集团和民族恒星公司合作研制的商业操作系统，同时其社区版也提供桌面版本和服务器版本供个人用户下载使用。麒麟操作系统最早基于 FreeBSD 进行研发，目前的版本是以 Linux 内核为基础，可广泛支持主流的处理器架构，包括 X86 及龙芯、申威、众志、飞腾等国产 CPU 平台。

2019 年，国内多家厂商联合推出了统信操作系统（Unity Operating System）㊀。统信操作系统基于 Linux 的 Debian 发行版进行开发，也包括桌面操作系统和服务器操作系统的版本。统信操作系统可支持 x86、龙芯、申威、鲲鹏、飞腾、兆芯等国产 CPU 平台，其桌面版本能够满足不同用户的办公、生活、娱乐需求，服务器版本能够满足企业级用户对服务器高性能、高稳定性、高可靠性的要求。

国内在云计算领域有巨大的市场和产业需求，因此国内云计算和数据中心提供商也提出和实现了多个面向云计算的操作系统。例如，华为 FusionSphere ㊁是华为公司面向多

㊀ https://www.uniontech.com/

㊁ https://e.huawei.com/en/cloud-computing/fusionsphere-openstack

行业客户推出的云操作系统产品，基于 OpenStack 架构开发，整个系统专门为云设计和优化，提供强大的虚拟化功能和资源池管理、丰富的云基础服务组件和工具、开放的 API 接口等，可以帮助客户水平整合数据中心物理和虚拟资源，垂直优化业务平台。浪潮云海 OS 操作系统是浪潮云数据中心操作系统，包括 Incloud OpenStack 发行版、面向软件定义计算的 InCloud Sphere、面向软件定义存储的 InCloud Storage、面向软件定义网络的 InCloud Network、数据中心运维管理平台 InCloud Manager 和面向软件定义安全的 InCloud Security 六大模块，用户可以灵活选择任意模块或整体解决方案部署到业务环境当中。

华为于 2019 年 8 月正式对外发布基于微内核的全场景分布式操作系统鸿蒙 OS。鸿蒙 OS 利用确定时延引擎和高性能 IPC，保证高优先级任务对资源的使用，降低应用响应时延 25.7%，并较现有系统提高 IPC 性能 5 倍。同时，鸿蒙操作系统将形式化验证技术应用于可信执行环境（TEE），保障系统正确性和可信执行环境等安全。

在构建新的操作系统的同时，华为也不断加强在 Linux 社区的投入与贡献。据 Linux 社区统计，华为长期以来在 Linux 社区贡献排名前十，也是国内唯一一家进入前十的企业。在 Linux 5.4 进入 Patch 贡献排名前五，这也是我国企业首次进入前五。在最新的 Linux 5.8 中，华为的 Patch 贡献数也首次进入了前三。此外，华为还积极将其 EulerOS 发行版进行了开源，联合国内操作系统发行商构建了 OpenEuler 社区。

华为长期以来也不断推动新兴技术的使用。其率先推动 F2FS 文件系统在智能终端设备上的使用。如今 F2FS 已经广泛应用于不同品牌的智能终端设备上。同时，华为构建了我国在 Linux 内核中的第一个文件系统 EROFS[114]。EROFS 通过改变压缩方式，在节约存储空间的同时提升文件系统读取性能，被广泛应用于华为智能手机中。

在自动驾驶方面，华为自动驾驶操作系统内核已于 2020 年 3 月获得业界 Safety 领域最高等级功能安全认证 ISO 26262 ASIL-D 认证，结合其于 2019 年在 Security 领域获得的高等级信息安全认证（CC EAL 5 +），使其成为业界首个同时拥有 Security 与 Safety 双高认证的商用 OS 内核。

阿里巴巴公司在国内拥有领先的云计算技术和业务，并自主研发了超大规模通用计算操作系统飞天（Apsara）。飞天操作系统可以将遍布全球的百万级服务器连成一台超级计算机，以在线公共服务的方式为社会提供计算能力。作为一个云计算的操作系统，飞天致力于解决跨多个数据中心的云计算的规模、效率和安全问题。飞天的目标是将云计算的三个方向整合起来：提供足够强大的计算能力，提供通用的计算能力，提供普惠的计算能力。

除了云计算操作系统之外，在阿里云 OS 的基础上，阿里巴巴还推出了移动操作系统 AliOS。它以驱动万物智能为目标，可应用于智联网汽车、智能家居、手机、Pad 等智能终端，为行业提供一站式 IoT 解决方案，构建 IoT 云端一体化生态，使物联网终端更加智能。汽车智能也是 AliOS 的一个重要应用领域，正在以汽车作为开始定义一个不同于 PC 和移动时代的物联网操作系统。

4 国内外研究进展比较

与国际先进水平相比，我国的操作系统研究和产业发展还存在相当大的差距。过去20年来，科技部、工信部、产业界和学术界为实现和推进传统计算平台上我国自主操作系统投入了大量努力，这个差距正在逐步缩小，但是我国的操作系统研发依然长期处于隔代追赶的状态。

在桌面和服务器操作系统方面，一方面，国产操作系统基本上基于开源的 Linux 进行研发，然而由于 Linux 内核的复杂性（包括数千万行代码），很难完全脱离 Linux 内核研制全新的操作系统，或者自行维护新的分支；另一方面，国内已经构建了操作系统的研发和产业化团队，能够自主设计满足国内行业需求的桌面和服务器操作系统，并且特定行业中取得了较为广泛的应用。虽然在短时间内无法超越国际先进水平，但是在传统操作系统的领域依然拥有较好的产品替代能力。

在移动操作系统领域，国产智能手机厂商基本是以 Android 操作系统作为基础进行二次开发，然而，虽然 Android 操作系统本身是开源的，其上的 Google 服务框架、Google Play 应用市场等却不属于这个范围，因此在使用 Android 操作系统进行二次开发的时候也会受到相应的制约。虽然国内小米、华为等厂商也针对 Android 进行了深度定制，但主要集中在上层应用框架和用户界面（UI）方面。阿里巴巴等公司也曾经尝试类似于云 OS 这样不同于 Android 的移动操作系统架构，但是并没有取得大范围的应用。智能手机是目前最为广泛使用的计算设备，面向智能手机等移动设备的操作系统也拥有非常重要的战略地位，因此在国产移动操作系统领域还需要进一步规划和加大研发投入。

在面向云计算和数据中心的操作系统方面，在国际上领先的主要是 Amazon、Google、Microsoft 这样的云计算提供商，以及开源的 OpenStack 等云计算系统。近年来，由于拥有具体的大规模应用场景与需求，在云计算和数据中心等相关领域的研究也产生了非常多的成果。国内阿里巴巴、华为、浪潮等领先的云计算服务提供商也研制了自己的云操作系统，并且在应用规模和性能表现等方面达到了国际先进水平。在科研方面，国内科研单位在与云计算和大数据相关的虚拟化技术、大数据处理、分布式图计算、面向 NVM 等新型存储设备的存储系统等领域也拥有国际领先的研究成果。在云计算和大数据操作系统这个新的方向上，国内产业界和学术界的相互促进，在操作系统和相关技术的研发中取得了非常好的研究与产业成果。

总体来看，由于国产的主流操作系统均基于 Linux 等开源操作系统内核，因此国内的研究也主要基于 Linux 开源操作系统进行开展，在很大程度上限制了国内在操作系统研究领域的创新。虽然在 OSDI、SOSP、USENIX ATC、FAST 等操作系统顶级会议上已经可以经常看到中国研究团队的论文，但是与计算机科学的其他领域相比，总体上依然处于相对落后的局面，还需要进一步加大支持力度，促进国内在操作系统这一“卡脖子”关键技术上的投入和研发努力。

在面向传统计算平台的操作系统研制方面，由于国内缺乏完整的操作系统应用链和生态系统，因此大规模发展相对困难。然而，当前计算平台和计算应用正处于一个重要的转折点。近年来出现了以人机物融合为代表的各种新型泛在系统，以实现人类社会、信息空间和物理世界的互联互通，其规模和影响力都将超过服务器、PC 等传统计算平台。国际上针对这类应用的操作系统研究主要面向新型领域进行突破，出现了机器人操作系统、家庭操作系统、智慧城市操作系统物联网操作系统等新的操作系统概念与实现。由于国内在移动互联网和物联网等相关领域拥有巨大的产业规模和应用需求，因此多个研究机构在面向人机物融合的新型操作系统领域也开展了很多研究，具有较好的研究基础和潜力。

在人机物融合时代，硬件设备高度泛化，人机物融合应用刚刚起步。作为全新的计算范式，人机物融合的泛在系统还处于蓄势待发的阶段，目前尚不存在垄断性的操作系统及相关系统软件，为在下一代核心信息技术上摆脱“卡脖子”的困境提供了宝贵的战略机遇。

5 发展趋势与展望

人机物融合泛在应用场景为重新定义操作系统生态提供可能，有可能推动我国操作系统实施换道超车，以这种新型的泛在操作系统为切入点，从基础研究做起，突破换代技术，在人机物融合的泛在系统中实现提前部署，有望从根本上避免信息产业“少魂”的问题。因此，国内研究和产业界应该抓住计算平台发展的转折点，抢占新一代计算平台的操作系统基础研究和关键技术的制高点。

5.1 软件定义与操作系统

“软件定义”（Software-Defined）已成为当今计算机领域最热门的术语之一，从“软件定义的网络”“软件定义的存储”“软件定义的数据中心”，到“软件定义的基础设施”“软件定义的环境”以及“软件定义的安全”，以至于人们使用“软件定义的一切（SDX）”来指代各种“软件定义”。

软件定义的网络（Software Defined Network，SDN）大致可算是“软件定义”热潮的肇始，对其可追溯到 2008 年美国斯坦福大学承担的美国自然基金重大项目“可编程开放移动互联网”（Programmable Open Mobile Internet），目的是为移动互联网提供一种新型的网络架构。该项目的最重要产出是一套原型系统 OpenFlow，其基本思想是将网络设备（如路由器、交换机等）的管理控制功能从硬件中分离出来，形成单独的一个完全由软件形成的控制层，该软件层抽象了底层网络设备的具体细节，为上层应用提供了统一的管理视图和编程接口，从而支持用户通过软件来定义逻辑上的网络拓扑，满足上层应用对网络资源的不同需求，而无须关心底层网络的物理拓扑结构，实现灵活的网络控制。SDN 和 OpenFlow 尽管在下一代互联网的研究中影响较大，但直到 2012 年前后，以 IaaS

(Infrastructure as a Service) 为代表的云计算采用 SDN 实现数据中心网络的灵活管理后，SDN 才开始大规模应用，成为学术界和产业界的关注热点。“软件定义”的概念也开始在云计算的大背景下迅速向存储、数据中心、运行环境等延伸。

分析探究目前各种“软件定义”的技术方案，核心思想都是将传统的“一体式”(monolithic) 硬件设施分解为“基础硬件及其虚拟化 + 管控软件”两部分：基础硬件提供标准化的基本功能，进而在其上新增一个软件层替换“一体式”硬件中实现管控的“硬”逻辑，为用户提供更开放、灵活、智能的系统管控服务。采用这种技术思路的直接动因，主要源自互联网环境下新型应用迫切需要共享各种硬件资源：以云计算为代表的新型互联网应用要求硬件基础设施能够以服务的方式灵活提供计算资源，而目前的主机管理、存储管理、网络管理在很大程度上是与应用业务脱离的，几乎都是手工管理、静态配置、甚少变动、分割运行，难以满足上层应用对计算资源个性定制、灵活调度、开放共享的需求；而要满足上述需求，就必须改变目前应用软件开发和网络化资源管理各自分离的情况，使得各种硬件资源能够根据应用需求自动管理、动态配置，因此，“软件定义”就成为一种自然、也是必然的选择，这也符合计算系统发展的历史规律。

计算系统由硬件和系统软件组成，二者互相协作为用户提供计算服务，支持应用软件的运行。硬件提供基本的计算资源，传统上指计算机的主要部件，包括中央处理器、存储器、外部设备等；随着网络的出现和快速发展，硬件资源开始不局限于单台计算机，可以是一台或多台通过网络连接的、规模更大的分布式计算系统。在硬件之上，系统软件（包括操作系统和编译器、中间件等）负责管理协调硬件资源，并将硬件资源抽象为一个软件实现的“虚拟机”，为用户提供编程接口和访问界面，支持应用软件正确、高效的运行。回顾计算系统的发展历程，从单机计算系统、网络计算系统到今天的互联网计算系统，可以发现，伴随着硬件能力的不断提升和应用需求的复杂多变，硬件的管理配置功能越来越多转而由软件，尤其是系统软件来提供，只有通过软件进行管理调度，硬件的功能和性能才能在具体应用中得以充分发挥。软件实现的“虚拟机”在功能上和硬件机器等价，不仅能够提供更为方便易用的用户界面和开发支撑，同时还具有硬件机器所缺乏的灵活性，可满足不同应用程序对资源的不同需求，这本质上正是“软件定义”的体现。

“软件定义”的核心技术途径是硬件资源虚拟化和管理功能可编程[25]。所谓硬件资源虚拟化，是将硬件资源抽象为虚拟资源，然后由系统软件对虚拟资源进行管理和调度。常见的如操作系统中虚拟内存对物理内存的虚拟、伪终端对终端的虚拟、Socket 对网络接口的虚拟、逻辑卷对物理存储设备的虚拟等等。硬件资源虚拟化带来了如下好处：支持物理资源的共享，提高了资源利用率；屏蔽了不同硬件的复杂细节，简化了对资源的管理和调度；通过系统调用接口对上层应用提供统一的服务，方便进行程序设计；应用软件和物理资源在逻辑上分离，各自可分别进行独立的演化和扩展并保持整个系统的稳定。

管理功能可编程，则是应用对通用计算系统的核心需求。主要表现在访问资源所提供的服务，以及改变资源的配置和行为两个方面。在硬件资源虚拟化的基础上，用户可编写

应用程序，通过系统调用接口，访问资源所提供的服务，更重要的是能够灵活管理和调度资源，改变资源的行为，以满足应用对资源的多样需求。所有的硬件资源在功能上都应该是可以编程的，如此，软件系统才可以对其实施管控，一方面发挥硬件资源的最佳性能，另一方面满足不同应用程序对硬件的不同需求。从程序设计的角度，管理功能可编程意味着计算系统的行为可以通过软件进行定义，成为所谓的“软件定义的系统”。

作为计算系统中最为重要的系统软件，操作系统一方面直接管理各种计算资源，另一方面作为“虚拟机”为应用程序提供运行环境，在此意义上，操作系统体现了“软件定义的系统”技术的集大成。当前出现的所谓软件定义的网络、软件定义的存储等技术，如同设备互联技术、磁盘存储技术之于单机操作系统一样，本质上正反映了“网络化操作系统”对网络化、分布式设备的管理技术诉求，也将成为“网络化操作系统”核心的底层支撑技术，并在操作系统的整体协调下，发挥最佳的功效。同样，未来互联网范围的操作系统，也仍然会通过各种“软件定义”途径，有效管理互联网范围的计算资源。因此，回顾操作系统的发展，有助于从软件的视角更深入地理解“软件定义”的本质。

近年来随着网络的迅速发展和不断延伸，以互联网为主干，融合其他多种异构网络（包括电信网、广电网、传感网）的复杂网络计算环境正在形成。相较于传统单机计算环境下的操作系统而言，运行于复杂网络计算环境下的互联网操作系统的主要功能面临若干挑战[3]，需要在现有单机系统和中间件的基础上，实现若干新的特点[2]，包括结构化（不再一次加载软件的全部功能，而是按需使用部分功能）、服务化（用户在不拥有软件代码的前提下，随时随地访问和使用软件所提供的功能，“只求使用，不求拥有”）、可伸缩（统一管理虚拟化的计算和存储资源，根据用户需求弹性、透明地调度和分配）和可定制（支持对共性资源的个性化定制，以更加灵活地构建面向公众、部门、行业、家庭或个人的网络操作系统），为了支持上述特点，需要通过软件手段对互联网操作系统的能力进行扩展、提升甚至重新定义。

5.2　面向人机物融合的泛在操作系统

进入21世纪以来，随着互联网的快速发展和普及，几乎所有的计算机系统和其上的操作系统都提供了方便的网络接入和访问能力。然而，虽然传统操作系统提供了基本的网络访问功能，但是主要的管理目标依然是单台计算机上的资源。如果把互联网当作一台巨大的计算机（Internet as a computer），那么如何能够管理好互联网平台上的海量资源，为用户提供更好的服务，成为互联网时代操作系统亟须解决的问题。

近年来，面向不同的互联网计算与应用模式，国内外都提出了许多面向云计算和数据中心的云操作系统。根据Techopedia的定义，“云操作系统（cloud operating system）是设计来管理云计算和虚拟化环境的操作系统”。一般来讲，云操作系统是指构架于服务器、存储、网络等基础硬件资源和单机操作系统、中间件、数据库等基础软件之上的、管理海量的基础硬件、软件资源的云平台综合管理系统。云操作系统管理的对象包括虚拟机的创建、执行和维护，虚拟服务器和虚拟基础设施，以及后台的软硬件资源。

除此之外，随着移动互联网和物联网的发展，出现了面向不同领域的操作系统的概念和实现，包括物联网操作系统、机器人操作系统、企业操作系统、城市操作系统、家庭操作系统等等。究其本质，这些操作系统都是面向新型互联网应用而构建的支持这些应用的开发和运行的网络化操作系统，都提供与传统操作系统类似的功能，即：向下管理资源，向上为软件开发和运行提供支撑。

基于操作系统的内涵和外延及其发展趋势，结合人机物融合应用场景的特征，梅宏院士等人在2018年*IEEE Computer*杂志发表了题为“走向泛在操作系统：一种软件定义的视角”的文献[21]，首次提出“泛在操作系统”（Ubiquitous Operating System）的概念，用来指代面向新型泛在计算设备和提供人机物互联和泛在资源调度的一类新型操作系统，即运行在不同规模、不同功能的计算系统之上的一层系统软件，提供管理底层资源，以及支撑上层应用开发和运行支撑。

泛在操作系统是传统操作系统概念的进一步扩展与泛化，不再把操作系统的概念局限于像Linux和Windows这样的单机操作系统。因此，泛在操作系统的形式更加灵活、多样，更多地强调操作系统的“操作”和管理的功能本质，支持灵活多样的资源虚拟化机制与异构管理功能，侧重于新型应用模式下的应用开发与管理支撑。另一方面，泛在操作系统也秉承泛在计算（Ubiquitous Computing）的基本思想，提倡多样化的操作系统实现机制与形态，为操作系统相关技术的研究提供了一条新的思路。

泛在操作系统的理念既涵盖了当前人机物融合操作系统发展的一些雏形和趋势，比如，物联网节点的操作系统（例如TinyOS、Contiki、Google Brillo等）、智能手机操作系统（Android和iOS）等，也涵盖了新应用模式下的典型的面向领域的操作系统，比如，云操作系统（Amazon AWS、Google Cloud、Microsoft Azure、OpenStack、AliCloud等）、机器人操作系统（ROS）、基于浏览器的Web操作系统（Google Chromium、Facebook OS等）等。这些都是面向人机物融合的泛在操作系统将走向主流的标志。

另一方面，泛在操作系统建立在“软件定义”的理念上，也通过软件定义这种特有的技术手段，达到管理泛在资源并支撑人机物融合应用的目的。目前，以软件定义网络为代表的“软件定义”，在很多领域都得到蓬勃发展，例如，软件定义网络、软件定义存储、软件定义卫星、软件定义雷达、软件定义安防、软件定义的自动驾驶等等。这些软件定义技术的背后，蕴藏了人机物融合操作系统发展的重大机遇。

5.3 未来发展趋势

纵观操作系统的发展史，我们发现操作系统的出现与流行和计算机硬件的发展以及应用模式的演化紧密相关。每一代计算机硬件的出现都会驱动新的操作系统技术以及不同的系统形态；同时每一次应用模式的转变也会改变操作系统的形态以及背后的支撑技术。国内过去的主要努力都集中在取代国外操作系统的垄断，主要针对的是PC或服务器上的Windows和Linux等系统。但事实上，自从Linux系统在30年前出现之后，面向传统计算设备（PC或服务器）的操作系统就没有再发生大的变革，因此也没有提供相应的

新机遇。

在互联网时代，网络化操作系统的管理对象变成了互联网平台上的海量存储和计算资源，而用户可以通过 PC、智能手机、平板电脑等多种互联网终端访问互联网平台上的资源和服务。为了适应这样的变化，互联网时代的网络化操作系统需要为海量数据资源和大规模计算提供高效的虚拟化支撑，使应用和服务可以通过透明的方式访问互联网上的资源；同时以服务化作为主要技术把云和端进行统一管理，为服务器端（云）和客户端（浏览器或智能手机）提供“云－端”融合的解决方案。

为了更好地支持人机融合计算环境下的新型应用模式，管理海量的泛在计算和数据资源，并为用户提供充分个性化的服务，人机物融合操作系统的发展呈现如下趋势：

- **结构化**：庞大的单体化的操作系统（或中间件）难以满足个性化定制需求，操作系统和中间件的结构会呈现构件化的形态。以构件化技术为基础，实现类似基于库的操作系统，能够更好地支持以构件为单元的“按需使用”的总体目标，不仅可以支持应用软件的按需加载，而且对操作系统等基础软件也可以做到以构件为单元的按需定制、按需加载和使用。
- **服务化**：互联网时代的应用模式的一个重要特点是服务化。服务与本地应用的最大区别在于“只求使用、不求拥有”，由于服务不需要在本地执行，因此可以达到“按需使用，即用即丢”的目标。随着互联网操作系统的进一步发展，互联网上将会提供无处不在的软件服务，用户可以随时随地通过任何设备访问这些服务。另外，由于网络化操作系统要支持包括桌面计算机、智能手机、上网本（平板电脑）在内的多种终端，因此通过应用的服务化（和标准化），可以比较容易地解决应用的多终端适配和“云－端”协同的问题。
- **易伸缩**：互联网正在逐步发展成为一个大型虚拟计算平台，为其上应用或终端用户提供分布式的数据存储和计算能力。人机物融合的泛在操作系统以统一的方式管理分布式的软硬件资源，根据应用或用户需求提供易伸缩的资源访问和计算能力。
- **可定制**：集中计算模式和大量共性沉淀使得互联网操作系统可能成为一个包含巨大资源和应用的管理系统，但是，对于面向用户的操作系统来说，由于每个用户的需求不同，因此都不可能用到所有的资源。另外，互联网上的资源太多，也使得用户可能难以找到自己所需的特定资源。因此，人机物融合的泛在操作系统需要支持基于强大共性资源的个性化定制功能，使之不仅可以支持公共的软件和服务，还可以通过个性化定制方便地构建面向公众、部门、行业、家庭或个人的泛在操作系统。

5.4　重要问题和研究方向

为了支撑人机物融合应用场景，操作系统在理论、方法和技术面临包括异质资源泛在应用、操作系统架构与构造方法、新型应用的开发和运行支持、多维融合安全可信等

一系列挑战，归结为如下的科学问题和重要研究方向：

- **人机物融合泛在计算的建模理论与软件定义方法**：面向人机物融合的泛在操作系统是在人机物融合环境下服务于多样化泛在计算应用的开发运行和适应演化的系统软件平台。作为一种操作系统，其本质上仍然是提供资源虚拟化和应用编程接口的一个软件层，核心是如何提供好的抽象。因此，构建人机物融合操作系统首先要研究的科学问题是如何认识人机物融合的计算环境、该环境中各种实现感知、计算、通信、执行、服务等能力的异构资源，以及领域场景下的泛在应用的本质特征，进而分别予以有效抽象，并配以相应的软件定义方法，以更好地凝练应用共性，更有效地管理资源，并适应动态多变的应用场景。
- **人机物融合操作系统的体系结构与运行机理**：操作系统是计算平台和计算应用的桥梁，落实到人机物融合操作系统上，它一方面需要面向新型泛在计算设备，提供人机物互联和泛在资源调度；另一方面需要支撑不同泛在应用场景的应用需求，以及这些应用场景中对效能、时延、安全性、可靠性、隐私保护等具体要求。这凸显了对操作系统的可定制、柔性化、可演化的需求，这是对传统操作系统在体系结构和运行机理上的重大挑战。需要针对硬件软件数据等泛在异构资源进行建模和管理，以及场景驱动的泛在应用抽象的基础上，探索人机物融合操作系统的新型体系结构模型，应用场景驱动的按需求构造、全栈资源和性能优化机制、操作系统结构演化机制等。
- **人机物融合操作系统的可信与安全保障**：在人机物融合时代，操作系统的基础设施地位使得其可靠性和安全性保障面临着前所未有的科学挑战。操作系统支撑人机物融合应用，需要为应用提供信息安全（security）和隐私保护（privacy），免受恶意用户或代码的攻击以及数据泄露。这其中包括如何确保数据的保密性和完整性、如何在主机系统和应用层面保证基础设施的安全；如何降低数据的安全风险、如何规避数据隐私风险以及与隐私相关的法律法规等；人机物融合操作系统的信任边界、用户管理、身份认证机制、访问控制模型等；人机物融合泛在计算的安全性和可用性管理、访问控制、安全漏洞与保护机制等。

6 结束语

本文主要围绕面向人机物融合计算的新型操作系统的国内外最新进展进行了总结和分析。与国外先进水平相比，虽然国内在操作系统的基础研究和产业应用方面还存在较大差距，但是人机物融合的泛在计算给国内操作系统的研究和产业发展提供了新的机遇。通过研究人机物融合计算和软件定义的基础理论和方法、面向人机物融合的操作系统的构造方法，构建人机物融合操作系统的新生态，可以在软件定义的人机物融合操作系统关键理论和技术上取得突破性进展，并有望在人机物融合操作系统的研究和产业应用上占据领先地位。

参考文献

[1] 张效祥. 计算机科学技术百科全书[M]. 2 版. 北京: 清华大学出版社, 2005.

[2] 梅宏, 郭耀. 网络化操作系统——现状与挑战[J]. 中国科学信息科学, 43(3), 2013.

[3] 梅宏, 刘譞哲. 互联网时代的软件技术: 现状与趋势[J]. 科学通报, 2010, 55(31): 1214-1220.

[4] Ritchie O M. & Thompson K. The UNIX time-sharing system[J]. Bell System Technical Journal, The, 1978, 57(6): 1905-1929.

[5] Boyd-Wickizer, S Chen, H Chen, R Mao, Y Kaashoek, M F Morris, R, Zhang Y. , et al. Corey: An Operating System for Many Cores[J]. In OSDI, 2008, 8: 43-57.

[6] Wentzlaff D & Agarwal, A. Factored operating systems (fos): the case for a scalable operating system for multicores[J]. ACM SIGOPS Operating Systems Review, 2009, 43(2): 76-85.

[7] Bernstein P A. Middleware: a model for distributed system services[J]. Commun. ACM. 1996, 39: 86-98.

[8] https://www. techopedia. com/definition/26867/cloud-operating-system-cloud-os.

[9] Peterson, Larry, Scott Baker, Marc De Leenheer, Andy Bavier, Sapan Bhatia, Mike Wawrzoniak, Jude Nelson, and John Hartman. Xos: An extensible cloud operating system[C]. In Proceedings of the 2nd International Workshop on Software-Defined Ecosystems, 2015: 23-30.

[10] Zhang, Yaoxue, and Yuezhi Zhou. TransOS: a transparent computing-based operating system for the cloud[C]. International Journal of Cloud Computing 1, no. 4(2012): 287-301.

[11] M. Weiser, The Computer for the 21st Century[J], Scientific Am. , 1991, 265(3): 94-105.

[12] Tan L, Wang N. Future internet: The internet of things[C]//2010 3rd international conference on advanced computer theory and engineering (ICACTE). IEEE, 2010, 5: V5-376-V5-380.

[13] Android Things, https://developer. android. com/things.

[14] Quigley M, Conley K, Gerkey B, et al. ROS: an open-source Robot Operating System[C]//ICRA workshop on open source software. 2009, 3(3.2): 5.

[15] Koubâa, Anis, ed. Robot Operating System (ROS)[M]. Vol. 1. Springer, 2019.

[16] Quigley, Morgan, Brian Gerkey, and William D Smart. Programming Robots with ROS: a practical introduction to the Robot Operating System[M]. O'Reilly Media, Inc. ', 2015.

[17] Hart, Stephen, Paul Dinh, and Kimberly Hambuchen. The affordance template ROS package for robot task programming[C]. In 2015 IEEE international conference on robotics and automation (ICRA), 2015: 6227-6234.

[18] Roldán, Juan Jesús, Elena Peña-Tapia, David Garzón-Ramos, Jorge de León, Mario Garzón, Jaime del Cerro, and Antonio Barrientos. Multi-robot systems, virtual reality and ROS: developing a new generation of operator interfaces[M]. In Robot Operating System (ROS), pp. 29-64. Springer, Cham, 2019.

[19] Hu Chi, Wei Dong, Yonghui Yang, Hao Shi, and Ge Zhou. Runtime Verification on Hierarchical Properties of ROS-Based Robot Swarms[C]. IEEE Transactions on Reliability (2019).

[20] Dixon C, Mahajan R, Agarwal S, et al. An operating system for the home[C]//Presented as part of the

9th USENIX Symposium on Networked Systems Design and Implementation ({NSDI} 12). 2012: 337-352.

[21] Hong Mei, Yao Guo, Toward Ubiquitous Operating Systems: A Software-Defined Perspective[J], Computer, 2018, 51(1): 50-56.

[22] Hong Mei, Yao Guo, Operating Systems for Internetware: Challenges and Future Directions[C], 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, 2018: 1377-1384.

[23] Hong Mei, Understanding "software-defined" from an OS perspective: technical challenges and research issues[C]. Perspective, SCIENCE CHINA Information Sciences, Dec 2017.

[24] 梅宏, 郭耀. 网构操作系统: 发展与现状[J], 科技导报, 2016, 34(14).

[25] 梅宏, 黄罡, 曹东刚, 郭耀, 张颖, 熊英飞. 从软件研究者的视角认识"软件定义"[J]. 中国计算机学会通讯, 2015, 11(1).

[26] Bomberger A C, Frantz W S, Hardy A C, et al. The KeyKOS nanokernel architecture[C]. Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures, Seattle, WA, USA, 27-28 April 1992s. DBLP, 1992.

[27] Hille M, Asmussen N, Bhatotia P, et al. Semperos: A distributed capability system[C]//2019 USENIX Annual Technical Conference (USENIX ATC 19). 2019: 709-722.

[28] Asmussen N, Völp M, Nöthen B, et al. M3: A hardware/operating-system co-design to tame heterogeneous manycores[C]. Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems. 2016: 189-203.

[29] Gabber E, Small C, Bruno J L, et al. The Pebble Component-Based Operating System[C]. USENIX Annual Technical Conference, General Track. 1999: 267-282.

[30] Ford B, Back G, Benson G, et al. The Flux OSKit[J]. ACM SIGOPS Operating Systems Review, 1997, 31(5): 38-51.

[31] Fassino J P, Stefani J B, Lawall J L, et al. Think: A Software Framework for Component-based Operating System Kernels[C]. USENIX Annual Technical Conference, General Track. 2002: 73-86.

[32] D R Engler, M F Kaashoek and J. O'Toole Jr. Exokernel: An Operating System Architecture for Application-level Resource Management[C]. In: Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles. Copper Mountain, Colorado, USA, 1995: 251-266.

[33] Madhavapeddy A, Scott D J. Unikernels: Rise of the virtual library operating system[J]. Queue, 2013, 11(11): 30-44.

[34] Schatzberg, Dan, James Cadden, Han Dong, Orran Krieger, and Jonathan Appavoo. EbbRT: a framework for building per-application library operating systems[C]. In 12th USENIX Symposium on Operating Systems Design and Implementation ({OSDI} 16), 2016: 671-688.

[35] Porter, Donald E, Silas Boyd-Wickizer, Jon Howell, Reuben Olinsky, and Galen C. Hunt. Rethinking the library OS from the top down[C]. In Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems, 2011: 291-304.

[36] Peter S, Li J, Zhang I, et al. Arrakis: The operating system is the control plane[J]. ACM Transactions on Computer Systems (TOCS), 2015, 33(4): 1-30.

[37] Shan Y, Huang Y, Chen Y, et al. Legoos: A disseminated, distributed OS for hardware resource disaggregation[C]//13th USENIX Symposium on Operating Systems Design and Implementation (OSDI

18). 2018: 69-87.

[38] Marvin, Simon, and Andrés Luque- Ayala. Urban operating systems: Diagramming the city [C]. International Journal of Urban and Regional Research 41, no. 1(2017): 84-103.

[39] Vögler, Michael, Johannes M Schleicher, Christian Inzinger, Schahram Dustdar, and Rajiv Ranjan. Migrating smart city applications to the cloud[C]. IEEE Cloud Computing 3, no. 2(2016): 72-79.

[40] Abdelzaher, T B Blum, D Evans, J George, S George, C Huang, P Nagaraddi, P Sorokin, and J Stankovic. SwarmOS: A Distributed Computing System for Sensor Networks[D]. University of Virgina.

[41] Lee, Edward A, Björn Hartmann, John Kubiatowicz, Tajana Simunic Rosing, John Wawrzynek, David Wessel, Jan Rabaey et al. The swarm at the edge of the cloud[C]. IEEE Design & Test 31, no. 3 (2014): 8-20.

[42] Liu, Rose, Kevin Klues, Sarah Bird, Steven Hofmeyr, Krste Asanovic, and John Kubiatowicz. Tessellation: Space-time partitioning in a manycore client OS[C]. In Proceedings of the First USENIX conference on Hot topics in parallelism, pp. 10-10. 2009.

[43] Levis P, Madden S, Polastre J, et al. TinyOS: An operating system for sensor networks[M]. Ambient intelligence. Springer, Berlin, Heidelberg, 2005: 115-148.

[44] Levis, Philip. Experiences from a decade of TinyOS development[C]. In Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12), pp. 207-220. 2012.

[45] Pianese, Fabio, Peter Bosch, Alessandro Duminuco, Nico Janssens, Thanos Stathopoulos, and Moritz Steiner. Toward a cloud operating system[C]. In 2010 IEEE/IFIP Network Operations and Management Symposium Workshops, pp. 335-342. IEEE, 2010.

[46] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems[C]. In Proceedings of the 26th Symposium on Operating Systems Principles (SOSP'17). Association for Computing Machinery, New York, NY, USA, 1-18.

[47] Zuoning Chen, Kang Chen, Jinlei Jiang, Lufei Zhang, Song Wu, Zhengwei Qi, Chunming Hu, Yongwei Wu, Yuzhong Sun, Hong Tang, Aobing Sun, Zilu Kang: Evolution of Cloud Operating System: From Technology to Ecosystem[J]. J. Comput. Sci. Technol. 32(2): 224-241(2017).

[48] 卢锡城, 王怀民, 王戟. 虚拟计算环境 iVCE: 概念与体系结构[J]. 中国科学 信息科学. 2006, 36 (10): 1081-1099.

[49] 史佩昌, 王怀民, 郑子彬, 尹浩. 面向云际计算的自主对等协作环境[J]. 中国科学: 信息科学, 2007, 9: 2.

[50] Yang, Xuejun, Huadong Dai, Xiaodong Yi, Yanzhen Wang, Shaowu Yang, Bo Zhang, Zhiyuan Wang, Yun Zhou, and Xuefeng Peng. micROS: a morphable, intelligent and collective robot operating system [C]. Robotics and biomimetics 3, no. 1(2016): 1-9.

[51] 张尧学, 周悦芝. 一种云计算操作系统 TransOS: 基于透明计算的设计与实现[J]. 电子学报. 2011, 5(39).

[52] Zhang, Mingxing, Yongwei Wu, Kang Chen, Xuehai Qian, Xue Li, and Weimin Zheng. Exploring the hidden dimension in graph processing[C]. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), pp. 285-300. 2016.

[53] Zhu, Xiaowei, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. Gemini: A computation- centric distributed graph processing system[C]. In 12th {USENIX} Symposium on Operating Systems Design and

Implementation ({OSDI} 16), 2016: 301-316.

[54] Ji Yu, Youhui Zhang, Wenguang Chen, and Yuan Xie. Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler[C]. In Proceedings of the Twenty- Third International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 448-460. 2018.

[55] Mei H, Huang G, Xie T. Internetware: A Software Paradigm for Internet Computing[J]. Computer. 2012, 45(6): 26-31.

[56] 杨芙清, 吕建, 梅宏. 网构软件技术体系: 一种以体系结构为中心的途径[J]. 中国科学 E 辑. 2008, 38(6): 818-828.

[57] Mei H, Huang G, Zhao H, et al. A software architecture centric engineering approach for Internetware [Z]. Science China Press, co-published with Springer, 2006: 49, 702-730.

[58] X Chen, Y Zhang, X Zhang, Y Wu, G Huang, and H Mei, Towards runtime model based integrated management of cloud resources[C], in Proceedings of the 5th Asia-Pacific Symposium on Internetware, ser. Internetware '13. New York, NY, USA: ACM, 2013, pp. 1: 1-1: 10.

[59] Pengfei Yuan, Yao Guo, Xiangqun Chen. Towards an operating system for the campus[C]. Proceedings of the 5th Asia-Pacific Symposium on Internetware. ACM, 2013.

[60] G Huang, X Liu, X Lu, Y Ma, Y. Zhang, and Y Xiong, Programming situational mobile web applications with cloud-mobile convergence: An internetware-oriented approach[C], IEEE Transactions on Services Computing, vol. PP, no. 99, pp. 1-1, 2016.

[61] Xu, Fengwei, Ming Fu, Xinyu Feng, Xiaoran Zhang, Hui Zhang, and Zhaohui Li. A practical verification framework for preemptive OS kernels[C]. In International Conference on Computer Aided Verification, pp. 59-79. Springer, Cham, 2016.

[62] Wang, Jiawei, Ming Fu, and Lei Qiao. Formalizing SPARCv8 Instruction Set Architecture in Coq[J]. In Dependable Software Engineering. Theories, Tools, and Applications: Third International Symposium, SETTA 2017, Changsha, China, October 23-25, 2017, Proceedings, vol. 10606, p. 300. Springer, 2017.

[63] Wang, Jiawei, Ming Fu, Lei Qiao, and Xinyu Feng. Formalizing SPARCv8 instruction set architecture in Coq[C]. Science of Computer Programming 187(2020): 102371.

[64] Liu, Yutao, Tianyu Zhou, Kexin Chen, Haibo Chen, and Yubin Xia. Thwarting memory disclosure with efficient hypervisor-enforced intra-domain isolation[C]. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1607-1619. 2015.

[65] Li, Wenhao, Yubin Xia, Haibo Chen, Binyu Zang, and Haibing Guan. Reducing world switches in virtualized environment with flexible cross-world calls[J]. ACM SIGARCH Computer Architecture News 43, no. 3S(2015): 375-387.

[66] Mi, Zeyu, Dingji Li, Zihan Yang, Xinran Wang, and Haibo Chen. Skybridge: Fast and secure inter-process communication for microkernels[C]. In Proceedings of the Fourteenth EuroSys Conference 2019, pp. 1-15. 2019.

[67] Gu, Jinyu, Xinyue Wu, Wentai Li, Nian Liu, Zeyu Mi, Yubin Xia, and Haibo Chen. Harmonizing Performance and Isolation in Microkernels with Efficient Intra-kernel Isolation and Communication[C]. In 2020 USENIX Annual Technical Conference (USENIX ATC 20), 2020: 401-417.

[68] Du, Dong, Zhichao Hua, Yubin Xia, Binyu Zang, and Haibo Chen. XPC: architectural support for

secure and efficient cross process call[C]. In Proceedings of the 46th International Symposium on Computer Architecture, 2019: 671-684.

[69] Zhang, Fengzhe, Jin Chen, Haibo Chen, and Binyu Zang. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization[C]. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, 2011: 203-216.

[70] Mi, Zeyu, Dingji Li, Haibo Chen, Binyu Zang, and Haibing Guan. (Mostly) Exitless VM Protection from Untrusted Hypervisor through Disaggregated Nested Virtualization[C]. In Proceedings of 29th Usenix Security Symposium. Boston, MA, USA. August 12-14, 2020.

[71] Hua, Zhichao, Jinyu Gu, Yubin Xia, Haibo Chen, Binyu Zang, and Haibing Guan[C]. vTZ: Virtualizing ARM TrustZone. In 26th USENIX Security Symposium (USENIX Security 17), 2017: 541-556.

[72] Liang, Liang, Rong Chen, Haibo Chen, Yubin Xia, KwanJong Park, Binyu Zang, and Haibing Guan. A case for virtualizing persistent memory[C]. In Proceedings of the Seventh ACM Symposium on Cloud Computing, 2016: 126-140.

[73] Gu, Jinyu, Zhichao Hua, Yubin Xia, Haibo Chen, Binyu Zang, Haibing Guan, and Jinming Li. Secure live migration of SGX enclaves on untrusted cloud[C]. In 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2017: 225-236.

[74] Song, Xiang, Jicheng Shi, Ran Liu, Jian Yang, and Haibo Chen. Parallelizing live migration of virtual machines[C]. In Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, 2013: 85-96.

[75] Wu, Mingyu, Ziming Zhao, Yanfei Yang, Haoyu Li, Haibo Chen, Binyu Zang, Haibing Guan, Sanhong Li, Chuansheng Lu, and Tongbao Zhang. Platinum: A CPU-Efficient Concurrent Garbage Collector for Tail-Reduction of Interactive Services[C]. In 2020 USENIX Annual Technical Conference (USENIX ATC 20), 2020: 159-172.

[76] Dong, Mingkai, and Haibo Chen. Soft updates made simple and fast on non-volatile memory[C]. In 2017 USENIX Annual Technical Conference (USENIX ATC 17), 2017: 719-731.

[77] Wu, Mingyu, Ziming Zhao, Haoyu Li, Heting Li, Haibo Chen, Binyu Zang, and Haibing Guan. Espresso: Brewing java for more non-volatility with non-volatile memory[C]. In Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, 2018: 70-83.

[78] Gu, Jinyu, Qianqian Yu, Xiayang Wang, Zhaoguo Wang, Binyu Zang, Haibing Guan, and Haibo Chen. Pisces: a scalable and efficient persistent transactional memory[C]. In 2019 USENIX Annual Technical Conference (USENIX ATC 19), 2019: 913-928.

[79] Dong, Mingkai, Heng Bu, Jifei Yi, Benchao Dong, and Haibo Chen. Performance and protection in the ZoFS user-space NVM file system[C]. In Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019: 478-493.

[80] Liu, Ran, Heng Zhang, and Haibo Chen. Scalable read-mostly synchronization using passive reader-writer locks[C]. In 2014 USENIX Annual Technical Conference (USENIX ATC 14), 2014: 219-230.

[81] Chen, Haibo, Heng Zhang, Ran Liu, Binyu Zang, and Haibing Guan. Fast consensus using bounded staleness for scalable read-mostly synchronization[J]. IEEE Transactions on Parallel and Distributed Systems 27, no. 12(2016): 3485-3500.

[82] Liu, Nian, Binyu Zang, and Haibo Chen. No barrier in the road: a comprehensive study and optimization of ARM barriers[C]. In Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2020: 348-361.

[83] Zou, Mo, Haoran Ding, Dong Du, Ming Fu, Ronghui Gu, and Haibo Chen. Using concurrent relational logic with helpers for verifying the AtomFS file system[C]. In Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019: 259-274.

[84] Wei, Xingda, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. Fast in-memory transaction processing using RDMA and HTM[C]. In Proceedings of the 25th Symposium on Operating Systems Principles, 2015: 87-104.

[85] Chen, Haibo, Rong Chen, Xingda Wei, Jiaxin Shi, Yanzhe Chen, Zhaoguo Wang, Binyu Zang, and Haibing Guan. Fast in-memory transaction processing using rdma and htm[C]. ACM Transactions on Computer Systems (TOCS) 2017, 35(1): 1-37.

[86] Chen, Yanzhe, Xingda Wei, Jiaxin Shi, Rong Chen, and Haibo Chen. Fast and general distributed transactions using RDMA and HTM[C]. In Proceedings of the Eleventh European Conference on Computer Systems, 2016: 1-17.

[87] Wei, Xingda, Zhiyuan Dong, Rong Chen, and Haibo Chen. Deconstructing RDMA-enabled distributed transactions: Hybrid is better![C]. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), 2018: 233-251.

[88] Kenji Yamanishi and Yuko Maruyama. Dynamic Syslog Mining for Network Failure Monitoring[C]. In ACM SIGKDD, 2005.

[89] Wei Xu, Ling Huang, Armando Fox, David Patterson and Michael I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs[C]. In SOSP, 2009.

[90] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Online system problem detection by mining patterns of console logs[C]. In ICDM, 2009.

[91] Wei Xu, Ling Huang, Armando Fox, David Patterson and Michael I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs (Invited Applications Paper)[C]. In ICML, 2010.

[92] 柳永坡, 吴际, 金茂忠, 杨海燕, 贾晓霞, 刘雪梅. 基于贝叶斯统计推理的故障定位实验研究[J]. 计算机研究与发展, 2010, 47(4): 707-715.

[93] Jian-Guang Lou, Qiang Fu, Sheng-Qi Yang, Ye Xu and Jiang Li. Mining Invariants from Console Logs for System Problem Detection[C]. In USENIX ATC, 2010.

[94] Shi-Lin He, Jie-Ming Zhu, Pin-Jia He and Michael R. Lyu. Experience Report: System Log Analysis for Anomaly Detection[C]. In ISSRE, 2016.

[95] Min Du, Fei-Fei Li, Gui-Neng Zheng and Vivek Srikumar. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning[C]. In CCS, 2017.

[96] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning[C]. ACM Workshop on Hot Topics in Networks, 2016: 50-56.

[97] Richard S Sutton. Policy gradient methods for reinforcement learning with function approximation[C]. Advances in Neural Information Processing Systems, 1999, 12: 1057-1063.

[98] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms[J]. arXiv preprint arXiv: 1707.06347, 2017.

[99] Tim Kraska, Alex Beutel, Ed H Chi, Jeff Dean, Neoklis Polyzotis. The case for learned index structures

[C]. Proceedings of the International Conference on Management of Data (SIGMOD), 2018.
[100] Michael Mitzenmacher. A model for learned Bloom filters and related structures[J]. arXiv preprint arXiv: 1802.00884, 2018.
[101] Alluxio. http://www.alluxio.org/.
[102] 星云 Clustar. http://36kr.com/p/5126024.html.
[103] Hao Zhang, Zeyu Zheng, Shizhen Xu, et al. Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters[C]. Proceedings of USENIX Annual Technical Conference (ATC), 2017.
[104] Eric X, Qirong H, Xie PT, Wei D. Strategies and principles of distributed machine learning on big data [J]. Engineering, 2016, 2(2): 179-195.
[105] Kim J K, Ho Q, Lee S, et al. STRADS: a distributed framework for scheduled model parallel machine learning[C]//Proceedings of the Eleventh European Conference on Computer Systems. ACM, 2016: 5.
[106] https://zh.wikipedia.org/wiki/Chrome_OS.
[107] https://www.google.com/chromebook/chrome-os/.
[108] https://fuchsia.dev/.
[109] https://zh.wikipedia.org/wiki/Google_Fuchsia.
[110] https://techcrunch.com/2019/12/19/facebook-operating-system/.
[111] https://developer.apple.com/tvos/.
[112] https://en.wikipedia.org/wiki/TvOS.
[113] https://microsoft.github.io/Win-RoS-Landing-Page/#.
[114] Gao, Xiang, Mingkai Dong, Xie Miao, Wei Du, Chao Yu, and Haibo Chen. {EROFS}: A Compression-friendly Readonly File System for Resource-scarce Devices[C]. In 2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19), 2019: 149-162.

作者简介

郭 耀 北京大学教授、计算机科学技术系副主任。主要研究方向为操作系统、移动计算、可信计算、程序分析等。现任 CCF 系统软件专委会委员。

陈海波 上海交通大学教授，上海交通大学并行与分布式系统研究所所长，教育部领域操作系统工程研究中心主任。主要研究领域为操作系统和系统安全。现任 CCF 系统软件专委会副主任。

胡春明 北京航空航天大学教授、计算机学院副院长。主要研究方向包括分布式系统、计算系统虚拟化、大规模分布式数据处理、分布式图计算系统、移动计算与云端融合等。现任 CCF 系统软件专委会常务委员。

卜 磊 南京大学教授。主要研究领域为软件工程与形式化方法，包括模型检验技术，实时混成系统，信息物理系统等。现任 CCF 系统软件专委会秘书长。