软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(1):22−32[doi: 10.13328/j.cnki.jos.005648]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

系统软件新洞察*

王怀民[1], 毛晓光[2], 丁 博[2], 沈 洁[2], 罗 磊[2], 任 怡[2]

[1](国防科技大学,湖南 长沙 410073)

[2](国防科技大学 计算机学院,湖南 长沙 410073)

通讯作者: 毛晓光, E-mail: xgmao@nudt.edu.cn

摘 要: 系统软件是计算学科的基本概念之一,从系统软件的本质特征、时代特点和发展趋势这 3 个方面给出了关于系统软件的新洞察.洞察 1 认为,通用图灵机和存储程序思想是系统软件的理论源头和技术源头,其本质特征是“操纵计算系统执行”,编码加载和执行管控是两种主要的操纵方式.洞察 2 认为,系统软件在互联网时代的时代特点是持续在线提供基础服务,为“软件即服务”的新型应用模式奠定了基础.洞察 3 认为,系统软件的发展趋势是持续在线演化,在计算系统创新、信息物理融合和智能技术的推动下,将成为未来软件生态的核心.

关键词: 系统软件;洞察;本质特征;操纵;计算系统

中图法分类号: TP316

中文引用格式: 王怀民,毛晓光,丁博,沈洁,罗磊,任怡.系统软件新洞察.软件学报,2019,30(1):22–32. http://www.jos.org.cn/1000-9825/5648.htm

英文引用格式: Wang HM, Mao XG, Ding B, Shen J, Luo L, Ren Y. New insights into system software. Ruan Jian Xue Bao/ Journal of Software, 2019,30(1):22–32 (in Chinese). http://www.jos.org.cn/1000-9825/5648.htm

New Insights into System Software

WANG Huai-Min[1], MAO Xiao-Guang[2], DING Bo[2], SHEN Jie[2], LUO Lei[2], REN Yi[2]

[1](National University of Defense Technology, Changsha 410073, China)

[2](School of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: This paper presents several new insights into system software, which is one of the basic concepts in computing discipline, from three perspectives of essential features, characteristics of the times, and the future development trend. The first insight is that system software stems theoretically and technically from universal Turing machine and the idea of stored-program, with an essential feature of “manipulating the execution of a computing system”. There are two typical manipulation modes: encoding and then loading, executing and controlling. The second insight is that software system is a kind of software, in the Internet age, providing substantial online services continuously, which lay the foundation for the newly emerged “software-as-a-service” paradigm. The final insight is about its development trend: system software will evolve online continuously. Driven by innovations of computing systems, integration of cyber and physical spaces, and intelligence technologies, system software will become the core of future software ecology.

Key words: system software; insight; essential feature; manipulation; computing system

系统软件是计算学科的基本概念之一,早期,此概念是关于操作系统、编译系统等基础性软件的统称.随着计算技术应用范围和应用模式的不断拓展,后续出现的数据库管理系统、网络中间件、云计算基础软件,甚至

* 基金项目: 国家自然科学基金(61432020, 61751208); 国家重点研发计划(2016YFB100100)

Foundation item: National Natural Science Foundation of China (61432020, 61751208); National Key Research and Development Program of China (2016YFB100100)

本文由“软件学科发展回顾特刊”特约编辑梅宏教授、金芝教授、郝丹副教授推荐.

收稿时间: 2018-07-10; 修改时间: 2018-08-21; 采用时间: 2018-09-25; jos 在线出版时间: 2018-11-22

CNKI 网络优先出版: 2018-11-23 07:18:02, http://kns.cnki.net/kcms/detail/11.2560.TP.20181123.0717.003.html

网络浏览器等基础性软件也被归入系统软件的范畴.今天,随着计算技术的持续变迁,系统软件的外延仍在不断拓展.那么,系统软件的本质特征究竟是什么?什么样的软件应该属于系统软件范畴,或者因为什么样的属性使得一种软件应该纳入系统软件的范畴?当今的系统软件具有什么样的时代特点?如何认识和把握系统软件的未来发展趋势?这是软件领域研究者和实践者关注的问题.本文试图从系统软件的本质特征、时代特点和发展趋势这 3 个方面给出关于系统软件的 3 个新洞察.

1 洞察 1:系统软件的本质特征

系统软件是指操纵计算系统(硬件形态或软件系统)有效执行、为上层应用软件提供运行支撑的软件.本文认为,系统软件的本质特征是“操纵计算系统执行”.这一洞察的理论源头和技术源头分别是阿兰·图灵的通用图灵机模型和冯·诺依曼的“存储程序”的通用电子计算机体系结构.

1.1 系统软件的理论基础

在软件领域,系统软件与应用软件的分野并非偶然,而是图灵可计算理论,特别是通用图灵机模型在实现层面上映射的必然结果.阿兰·图灵在 1936 年发表的著名论文“On Computable Numbers, with an Application to the Entscheidungsproblem”[1]不仅奠定了图灵可计算的理论基础,而且也为系统软件提供了理论依据.具体表现在两个方面.

- 第一,图灵提出,把“一个可计算序列γ由计算γ的机器所描述”,这个机器就是图灵构思的图灵机,并且“可以用一个整数(描述数)唯一确定一台机器”[1],即,每个图灵机可被规范表示或编码.理论上讲,此后发展起来的编程和编译系统由此发端;
- 第二,图灵提出能够模拟任何图灵机的机器,被称为通用图灵机(或通用计算机器、通用机),通用图灵机 u 通过在纸带的开头写入任意图灵机 M 的标准描述,可以模拟计算出与 M 相同的序列,即,实现通用计算.理论上讲,此后发展起来的操作系统由此发端.

如今,主流的计算机系统从计算模型上讲就是通用图灵机,其上执行的软件,从表示模型上讲,就是完成该软件所描述计算的图灵机编码,而系统软件正是通用图灵机模型在软件实现层面上的具体表现形态:从图灵可计算角度而言,系统软件具备接收、支撑和管理应用层“图灵机”的能力,负责“操纵”应用层“图灵机”的执行;从实现角度而言,系统软件是当今各类计算系统可被编码、可被高效加载和执行的前提和基础.

因此,系统软件的本质特征是“操纵”计算系统有效“执行”的软件.这里的“计算系统”包括了单一应用软件系统,或者分布式系统乃至未来大规模人机物融合系统,而“操纵”则具体有两层含义.

- 编码与加载.也即编码并加载计算系统,实现对其执行生命周期的管理,能够通过编排和协调硬件资源,为计算系统提供良好的通用执行管理;
- 执行与管控.也即在运行时管控硬件资源和计算系统运行时行为,从而实现对计算机资源的高效利用和高效复用.

需要指出的是:依据上述定义,单纯地进行数据查询的数据库管理系统并不属于系统软件范畴,因为它不具备在运行时“操纵”计算系统执行的能力,仅仅是按照一定方式来存储数据的仓库.但是,如果一个数据库管理系统加入了管理、装载和执行“存储过程”的能力,则该数据库管理系统可以划归系统软件.部分浏览器也可以划归系统软件,这是因为,它们具备加载、执行和管控其中所运行的内容或插件的能力.此时,浏览器构成了通用“图灵机”的物化实现,而浏览器内容或插件则可以被视为应用层“图灵机”的物化实现.这也是近年来“Web 操作系统”(如 Google ChromeOS)[2]这一概念出现的背景.

如今,人们讲“软件定义一切”[3]“软件是基础设施”,一方面是指人们利用计算技术解决现实问题的一种信念,即:通用计算平台能够模拟(或解决)物理世界的一切问题,因此,现实世界将运转在通用计算平台上;另一方面是指人们利用计算技术解决问题的一种标准技术途径:部署通用计算平台,其中的系统软件支持编码及其执行.

1.2 系统软件的技术基础

从技术上讲,系统软件并不是与计算机硬件一起诞生的.1946 年,在美国宾夕法尼亚大学问世的世界上第一台通用电子计算机 ENIAC(electronic numerical integrator and computer)还没有操作系统、编译系统等系统软件,计算机程序通过纯硬件实编码,编程采用接线和开关等手工操作方式(如图 1 所示).真正催生系统软件并成为其技术基础的是冯·诺依曼的“存储程序”这一思想[4].1945 年,冯·诺依曼等人在设计电子离散可变自动计算机 EDVAC(electronic discrete variable automatic computer)时提出了“存储程序”这一思想:将程序数据和指令数据储存在同一存储器中.这一思想直接为系统软件奠定了技术基础,使得在单一计算机和分布式系统中装载(存储)和管控计算系统自动执行成为可能.阿兰·图灵本人在 1950 年发表的论文“Computing Machinery and Intelligence”[5]中也明确写到:“数字计算机通常被认为由 3 部分构成:(i) 存储;(ii) 执行部件;(iii) 控制.”因此,通用图灵机模型被 Davis 等人认为是冯·诺依曼“存储程序”思想的源头[6].

Fig.1 Programming ENIAC by a combination of plugboard wiring and portable function tables[7]

图 1 通过插接板接线和“移动式转换装置”进行 ENIAC 编程[7]

(1) 操纵方式 1:“编译加载”应用软件

随着“存储程序”思想的提出,应用层“图灵机”的具体编码能够以数据的形式存储.这催生了最早的汇编语言,一种作为机器体系结构抽象的符号化语言.然而,计算机能读懂的只有机器指令,此时就需要一个能够将汇编指令转换成机器指令的翻译程序,这就是最早的编译器——汇编器.1947 年,伦敦大学伯贝克学院的 Kathleen Booth 为 ARC2 计算机开发了第一个汇编语言及其机器汇编器[8].之后不久,开始出现带有支持代码“库”的计算机,预先记录在打孔卡片和磁带上的支持代码“库”能够在装载器(loader)的支持下与用户程序进行连接,从而支持输入/输出等操作.这就是编译器(或编译系统)这种系统软件类型最早的雏形.

虽然汇编语言相对于机器语言提高了程序的可读性,但编写、阅读和理解汇编语言的难度依然较高,而且汇编代码很难在不同指令集机器间移植.为了提高编程效率和程序可移植性,出现了高级程序设计语言和相应的编译系统.20 世纪 50 年代为 IBM 704 大型机设计的 FORTRAN 语言是第一个高级程序设计语言,1957 年发布的 FORTRAN 编译器则是第一个功能比较完备的高级程序语言编译器[9].它也是第一个具备优化能力的编译器,引入了循环优化、寄存器分配等技术,使编译器翻译的机器代码能够与当时人工编写的汇编程序性能相当,从而使更多的用户接受.20 世纪 70 年代,B 语言、C 语言等高级语言和相应的编译技术快速发展,并在 Unix 操作系统[10]核开发中发挥了重要作用.同时,面向对象思想也在这一时期应用到高级语言中,出现了 SIMULA67、Smalltalk 等多个面向对象语言.受到 C 语言和 SIMULA67 语言的影响,Bell 实验室在 20 世纪 80 年代开发了 C++,随后开发了支持 C++的 Cfront 编译器,逐渐成为主流编程语言[11].1995 年,Sun 公司发布了 Java 语言和 Java 编译器.Java 具有简单性、动态性和独立于平台的特点,至今应用广泛.进入网络时代,编程语言又进入发展的活跃期,出现了 Python、C#、Go 等编程语言与编译系统.

这里有一个误区:既然编译器是系统软件,那么其他以程序为操作对象的工具软件(例如程序分析或程序验证工具)是否也可以视为系统软件?这里的界限在于该工具软件的直接目的是否服务于操纵软件执行,这是系

统软件的本质特征所决定的.软件工程领域中大量的工具软件以产生高质量的软件为直接目的,并不关注软件的运行态行为以及它们是如何被执行的,而编译器(当然也被认为是工具软件)的直接目的是产生可加载执行的代码并为操纵软件的执行奠定基础.所以,一般以程序为操作对象的工具软件通常不能纳入系统软件范畴,而编译器则被纳入系统软件范畴.无论是以高级语言、汇编指令、机器指令二进制编码甚至是纸带形式表示的代码,它们都是应用层“图灵机”的一种物化实现形式.编译器则是在上述形式之间进行变换的系统软件,并且这种变换通常是从高层次的高级语言向可加载执行的机器指令集的变换,并最终映射为可加载执行的形式.在与编译系统密切相关的装载器(通常内置于操作系统/语言等运行环境)[12]的协助下,应用层“图灵机”的具体编码被载入到主内存,成为准备就绪、随时可以执行的状态.

(2) 操纵方式 2:“执行管控”应用软件和资源

早期的计算机虽然具备加载应用程序的能力,但需要由操作人员手工装入包含程序的纸带或磁带.计算机在加载完程序之后,运行控制权就完全交给了应用程序,只有程序运行完并取走计算结果之后,才能再由人手工装入下一个纸带或磁带.这种手工操作、排他运行的方式具有效率低下、系统资源利用率低的特点.最早的操作系统就是为了解决这一问题而应运而生的[13]:1956 年的 GM-NAA IO 系统具备单道批处理能力,能够自动地从磁带顺序读入并执行应用程序.这种“执行管控”避免了两个作业之间的空闲时间,但在作业执行过程中,由于只有单个应用程序在排他运行,资源利用率低的问题并没得到彻底解决.因此,以当时主流的大型主机为载体,操作系统围绕两种使用模式展开了探索.

(1) 多道批处理(batch processing):即对单道批处理进行扩展,允许单一计算机同时载入多个批处理作业,使之可以以恰当的调度策略交替占用 CPU,从而提高资源利用率.其典型案例是 20 世纪 60 年代由 Dijkstra 所领导研发的 THE 操作系统[14];

(2) 分时(time sharing):其思想几乎与操作系统同时出现,目标是“通过分时共享,使得一台大的计算机能够像多台小型计算机一样使用”[15].从 1961 年的首个 CTSS(compatible time-sharing system)分时操作系统[16]开始,此类系统均通过 CPU 时间的划分来支持多个终端用户.

无论是多道批处理还是分时,其本质都是“管控”应用软件执行,也即通过引入恰当的应用程序调度和行为调控机制,使得单个通用“图灵机”可以同时读入和运行多个应用层“图灵机”,最终实现计算机资源的高效利用和高效复用的目标.这一思路直接推动了多任务多用户的 Unix 操作系统的出现,为现代意义上的操作系统奠定了基础.

除了“执行管控”应用软件,操作系统的另一核心职能是“管控”资源,也即通过管理和协调执行部件、内存、磁盘、网络等计算机资源,实现高效协同,形成具备通用图灵机能力的基础执行环境.操作系统这一职能的发展经历了两个阶段.

(1) 早期的操作系统往往与特定计算机硬件和特定应用绑定:即使是同一个生产商,不同型号计算机的操作系统也会有不同的使用方式,甚至很多操作系统就是计算机用户根据应用场景定制的;

(2) 其后,操作系统逐渐与具体硬件解耦,力求对应用软件提供一致的通用“图灵机”物化形态.

20 世纪 60 年代,IBM 为 System/360 系列计算机开发了统一的操作系统 OS/360[17],首次为应用软件提供标准化、与特定计算机型号和应用场景解耦的运行环境;其后,1972 年发布的 VM/370 操作系统扩展了分时共享的思想,首次提供了成熟的虚拟化能力[18].20 世纪 80 年代,IEEE 和 ISO 组织制定了一组可移植操作系统接口 POSIX(portable operating system interface of UNIX)标准,在某种程度上,它定义了一个通用图灵机物化形态所应具备的接口,应用软件通过该接口与操作系统交互,再由操作系统操纵软/硬件资源.无论是大型主机还是微型机,这种解耦和抽象都直接推动了软/硬件技术的快速发展,奠定了操作系统作为整个计算机生态链的基础性地位.

相对于其先驱者,今天的操作系统能力已经得到极大丰富,诸如图形用户界面、多媒体播放器等各类公用程序等都被内建到操作系统中,其运行场景也扩展到了移动终端、物联网设备、无人系统等多样化环境,但上述“管控”应用软件和资源的核心职能并没有发生变化.

(3) 操纵方式 3:“联接协调”分布计算系统

“中间件(middleware)”的概念最早出现于 20 世纪 60 年代,1968 年,北大西洋公约组织的软件工程报告[19]所给出的软件结构倒金字塔中(如图 2 所示),中间件被代指位于应用软件之下、控制程序/服务例程之上的一层软件,被用来弥补底层系统软件通用功能和上层应用特殊需求之间的鸿沟.虽然也被纳入系统软件范畴,但此时的中间件还没有“管控”分布计算系统的职能.

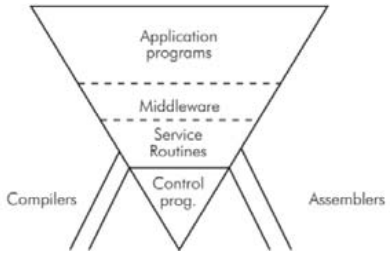

Fig.2 Inverted pyramid of software structure in NATO software engineering report[19]

图 2 NATO 软件工程报告中的倒金字塔软件结构[19]

20 世纪 80 年代,TCP/IP 协议等计算机网络技术走向成熟,计算机应用模式迈入网络计算时代.与冯·诺依曼体系结构中的“存储程序”思想类似,网络环境下的分布式应用通过二进制网络消息对指令和数据进行统一编码和传递.在现代中间件出现以前,开发人员只能通过操作系统自带的 SocketAPI 等原生接口,以完全手工的方式实现从网络消息到程序行为的双向映射,直接后果是开发繁琐、复杂和低效.另一方面,研究者也逐渐认识到:相对于单机软件,分布式软件有着特有的理论和技术问题(如负载均衡、时间同步等),而传统的操作系统并未提供相应的管控能力.

为了应对上述挑战,早期主要围绕两种思路展开了探索.

- 一是以 Amobea[20]等分布式操作系统为代表的自顶向下方法,其核心思想是将单机操作系统的概念放大到网络环境.例如,文献[21]指出:“分布式操作系统对用户而言与单机操作系统无二,只是运行在多个独立处理器上,它的核心概念是透明性.”但是,网络资源具有分布自治、高度异构、持续演化等特点,其上软件很难通过严格的以自顶向下的方法精确设计出来,而更多的是通过互联互通、以自底向上的方式成长式构造出来的[22];
- 后一种思路以 Birrel 等人于 1984 年实现的首个远程过程调用(remote procedure call,简称 RPC)框架[23]为代表,催生了现代意义上的中间件.

中间件运行于操作系统之上,聚焦分布式应用中的“连接”问题,沉淀此类软件开发、运行与集成的共性机制.在冯·诺依曼体系结构的单个计算机中,系统软件会将所存储的程序映射为 CPU 指令执行动作,这是其操纵能力的最基本体现.与之类似,中间件通过将网络消息映射为分布式应用程序模型基本构造单元的行为,实现对分布计算系统的操纵.依据应用程序模型的不同,中间件可以分为面向过程中间件、面向对象中间件、面向构件中间件、面向消息中间件等,其操纵能力见表 1.

Table 1 Manipulation capability of different types of middleware

表 1 不同中间件所具备的操纵能力

类型	程序模型	基本操纵能力	典型实例
面向过程中间件	远程过程	(1) 网络消息与过程调用之间的映射	Sun RPC,DCE,Apache Thrift 等
面向对象中间件	分布式对象	(1) 网络消息与对象方法之间的映射	CORBA,Java RMI,DCOM 等
面向构件中间件	分布式构件	(1) 网络消息与构件行为(功能接口)之间的映射 (2) 通过构件的管理接口管控生命周期、交互关系等	EJB,OSGi Bundle 等
面向消息中间件	异步事件	(1) 网络消息与消息回调函数之间的映射	MQSeries,Kafka,DDS 等

除了支撑应用程序模型的通用中间件外,部分中间件仅关注“协调”分布计算系统过程中的一些特定场景.

事务处理中间件即为一个典型案例,它通过控制服务端多个软件实体的行为,保证分布式事务处理过程中的(全部或部分)一致性.早在大型主机时代,IBM 的 CICS(customer information control system)系统就已提供了跨计算节点事务支持.针对美国电话公司需要的联机事务处理能力,AT & T 贝尔实验室于 20 世纪 80 年代初发布了跨平台的 Tuxedo 事务处理中间件,相关研究与实践对后续 X/Open 组织的分布式事务处理规范、OMG(Object Management Group)组织的对象事务服务规范等产生了较大影响[24].

2 洞察 2:系统软件的时代特点

系统软件是持续在线提供基础服务的软件,这是系统软件的时代特点.这一洞察的时代背景是互联网发展成为当今世界的信息基础设施.系统软件是当今云计算平台之所以成为公共服务平台的关键,是“软件即服务”应用模式的基石.

2.1 系统软件的基本属性

系统软件是基础性软件,其诞生之初就具有不间断持续运行的基本属性.例如:操作系统与计算机裸机紧密结合,共同向应用软件提供通用图灵机能力抽象,其生命周期与硬件上电运行时间基本一致;编译系统中一个重要的内容是运行时库(runtime library),程序由何种编程语言编写、运行在何种运行环境上,就需要相应的编译系统提供相应的运行时库,它不仅仅本身是动态运行的,更为上层应用程序的持续运行提供了有力支撑;无论是以静态/动态链接库,还是以独立服务等形式存在,中间件的主要实现部件(如互操作协议引擎)的生命周期都至少与所支撑的分布式应用生命周期相同.

然而,早期系统软件的“持续运行”是相对于其所操纵的计算系统而言的.换言之,系统软件的生命周期通常与微观层面上的具体计算系统一致,所操纵计算系统一旦停止运行或重启,系统软件的状态也随之发生变化.此外,系统软件能力提供的方式是“购买/授权、本地安装和配置、后期维护”的模式,与下文阐述的按需服务模式也存在很大差异.

2.2 从基本属性到时代特点

互联网的高速发展改变了计算系统应用模式,同时也对系统软件产生了重要影响,推动着系统软件“不间断持续运行”这一属性跳出单个计算系统的微观范畴,在整个信息空间的宏观尺度上,成为持续在线提供基础服务、支撑信息空间不间断运行的核心基础设施.

系统软件的上述时代特点与 20 世纪 90 年代中后期出现的服务化思想密切相关:互联网资源具有开放动态、成长自治的特点,但人们同时又希望软件能够持续在线、高度可用,二者之间的矛盾催生了“服务”的概念.从 2000 年左右 W3C 组织所制定的一系列 Web Service 标准[25]开始,“服务”一词即被用来抽象可通过网络按需访问、无需关心实现细节的能力.以此为核心,面向服务体系结构通过服务化封装和“服务提供者-消费者”之间的动态发现,以解决开放动态与持续在线之间的矛盾.

作为互联网时代的产物,服务化思想首先推动了中间件技术外延的拓展.Apache Tomcat、IBM WebSphere 等 Web 服务中间件以支撑面向服务体系结构为目标,封装了服务注册、服务发现、服务访问等共性基础设施,完成对服务化软件的操纵、实现网络消息与服务实现者之间的映射;以企业服务总线(enterprise service bus,简称 ESB)[26]为代表的服务集成中间件通过服务封装、服务组合、服务协同等技术实现自治子系统之间的通信,进而实现复杂业务系统跨域集成;微服务[27]则聚焦单个业务系统内部,通过系统内部组件的彻底服务化,实现开发阶段“分而治之”和运行阶段的灵活应变,这一思路催生了 Spring Cloud 等微服务框架.

近年来,作为服务化思想的集大成者,云计算模式被广泛接受.如同工业革命让水、电、交通等成为公共服务一样,云计算正在通过后端资源集约化和前端按需服务的方式,推动信息技术实现社会化和专业化,使得信息服务成为公共基础设施[28].这一趋势推动着整个系统软件生态链的变革:操作系统、编译系统、中间件、数据库等逐渐融合,以云计算操作系统、“平台即服务(platform as a service,简称 PaaS)”[29]软件基础设施等形式打包.此类系统软件是各大云服务提供商的核心竞争力所在,其典型代表包括 Google AppEngine、Microsoft Azure、

Amazon EC2、阿里飞天平台等,其特点如下.

- 在资源的管控方面,此类系统软件要应对的资源规模是数据中心甚至多数据中心级别的,管控的核心在于通过计算、网络、存储等资源虚拟化,实现资源池中资源的有效聚合和弹性扩展,在不同尺度上形成具有通用图灵机能力的环境.例如,互联网虚拟计算环境 iVCE[30]通过资源的“自主协同、按需聚合”,将静态统一视图的资源组织模式扩展为相对稳定视图、按需可伸缩的资源组织模式,从而实现面向互联网计算的高效管控;
- 在应用软件的执行管控维度上,此类系统软件面向大规模云服务这类分布计算系统,提供了任务调度与执行、负载均衡、安全管控、在线监控和升级演化、分布式数据存储访问等一系列基础设施和相应能力.这些能力以“Pay as you go”[27]的服务化方式提供,用户无需像传统系统软件那样自行安装、部署和维护.部分“平台即服务”基础设施还内建了编译系统和调测试环境,支持云端开发和编译构建;
- 在自身生命周期方面,此类系统软件 7×24 小时持续提供基础服务,已经与单个应用软件的生命周期解耦,是支撑整个信息空间持续运行的核心基础设施.其突出表现为一旦发生故障,会产生较大的社会经济影响.例如,文献[31]指出:如果一个主要的云平台宕机 3~6 天,美国经济会产生 30 亿~150 亿美元的损失;而在现实生活中,2017 年 2 月底,Amazon EC2 中的对象存储服务出现故障近 4 个小时,导致数千个互联网服务中断,产生了重大的社会影响.

3 洞察 3:系统软件的未来趋势

系统软件是持续在线演化的软件,这是系统软件的发展趋势.这一洞察的依据是:因为软件正在成为未来社会运行的基础设施,系统软件不仅持续在线提供基础服务、支撑上层应用软件的持续演化,而且本身也必须因适应变化而持续在线演化,系统软件是未来“软件生态”的核心.

3.1 系统软件演化的动力与特点

软件正在成为未来社会运行的基础设施,这主要受 3 个方面因素的驱使.

(1) 软件自身能力的不断提升.表现为软件规模的持续增长、软件实体之间连接形式的多样化、连接紧密程度的深化等,使得软件可以在许多关键领域代替原来传统的非信息化手段,并表现出显著优势;

(2) 软件正在驱动信息空间与物理空间的深度融合,提高了人类社会生产和生活的效率.以物联网软件为例,其应用不断融入智能工业、智能交通、环境保护、公共管理、智能家庭、医疗保健等多个领域,已经成为世界经济社会发展的热点;

(3) 计算技术的普适化,使得人类社会产生了深刻变革.在移动和可穿戴设备、移动互联网、智能助理等功能的支持下,人们有可能接受“无时无处不在而又不可见”的软件服务.

在上述背景下,软件将与其所在的社会和物理环境紧密融合在一起,相互作用、相互影响.这与传统的、完全在信息空间运行的软件有着质的区别.具体到基础性的、“操纵计算系统执行”的层面,体现为系统软件需要能够不断地适应技术、物理世界和人类社会的变化,也即需要具备演化能力[22].事实上,系统软件的演化并非新现象,以 Linux 内核为例,据统计,从 1994 年 3 月~2008 年 8 月一共发布了 810 个版本;为了适应硬件的急剧发展和变迁,Linux 内核中与体系结构和外设有关的目录(arch 和 drivers 目录)下的文件数从最早的寥寥数个扩展到了将近 10 000 个[32].在 Linux 演化过程中,产生了大量分支和发行版本,这些“副本”以类似于生物系统演化的方式“优胜劣汰”,推动着 Linux 逐渐走向成熟.

但是与传统的离线、阶段式演化不同,互联网时代的系统软件演化强调持续和在线的特点:在整个信息空间尺度上,这种演化并非停止所有信息服务、离线更新版本,而是随着社会和物理环境的变化在持续地、细粒度地演化.在单个平台化的系统软件运行实例中,由于其需要 7×24 小时持续提供基础服务,其适应变化的过程多以部件的单独更新、整体服务并不停止的在线形式表现出来.

3.2 系统软件持续演化的方向

在驱动软件成为社会基础设施的 3 个因素的作用下,本文认为,未来系统软件自身的持续成长和在线演化主要表现在如下一些方面.

(1) 适应硬件和软件技术创新,向平台化、体系化拓展.

系统软件是“操纵计算系统执行”的软件,它们与底层硬件一起,向上层应用提供通用图灵机能力抽象.因此,系统软件的演化首先体现在适应硬件体系结构的变化上.以编译系统为例,21 世纪以来,编译系统的发展主要集中在面向多核和众核处理器体系结构的编译优化上[33].在可预见的未来,众核和宽向量将成为体系结构的发展方向,计算部件和存储层次的异构性和复杂性将不断增加,因此,面向体系结构的编译优化技术仍将持续发展,如何在不对程序员造成过度负担的条件下尽可能高效地优化程序性能是其中的一大挑战.同时,新型计算机架构,如量子计算机、生物计算机等,很有可能解决经典计算机难以解决的问题.这些新型器件和设备将与经典计算机并存,如何设计面向新型设备的编程模型和编译系统,也是未来的一个研究难点.

作为加载、支撑和管控应用层“图灵机”的基础设施,系统软件的演化还体现在适应应用软件形态的变革上.例如:随着信息空间内部连接的进一步深化,如何实现多个云服务提供者之间的无缝连接、实现资源的跨云共享,是云际计算(joint cloud)[34]这一新兴计算模式的焦点,相应的系统软件技术也开始崭露头角,典型实例包括 IBM 的 Altocumulus[35]、欧盟的 mOSAIC[36]等中间件;“软件即服务”模式对软件敏捷交付能力和轻量级维护能力提出了新需求,开发运维一体化(DevOps)[37]及相关的自动化测试部署、智能运维(AIOps)[38]等技术也将沉淀到系统软件中;随着区块链等技术的发展,智能合约、共识机制、信任机制等将在系统软件层面上得以体现.此外,各类应用软件正在成为未来社会运行的基础设施,它们本身具有迫切的在线演化需求,作为“操纵”软件的软件,系统软件成为驱动上层应用实施这种演化操作[39]的理想载体,相关“操纵”机制的沉淀是未来系统软件的重要发展方向.

在适应软/硬件技术变革的过程中,操作系统、中间件、编译系统等系统软件各个分支将持续融合.用户的选择重心将从产品本身聚焦到自身能获得的应用和服务支持上,因此,系统软件在产品形态上将表现出依托开源社区等新兴力量形成有机统一的整体解决方案,在运营形态上从传统销售获利转变为平台化的持续服务提供,在生态建设方面呈现出技术、平台和实践三者良性互动螺旋式上升的格局.此外,基于“云+端”和边缘计算等体系结构,在后端提供具有丰富资源的服务化平台,在前端提供多样化应用环境,实现前后端的优势互补、有机融合,是未来系统软件体系化发展的重要方向.

(2) 通过感知和操纵物理空间,成为“软件定义一切”的基石.

信息技术向物理世界迈出的第一步是实现环境“感知”.自 20 世纪末以来,随着相关硬件技术的逐渐成型,系统软件领域开始关注对物理空间情境数据的处理,出现了传感器网络操作系统/中间件[40]、情境感知中间件[41]等一系列系统软件.近年来快速发展的物联网中间件[42]在上述概念的基础上进一步针对“万物互联”的大规模异构环境进行了拓展,其核心关注点是:在资源受限、高度异构、规模庞大、网络动态变化条件下,如何实现互联互通互操作,满足物联网应用对海量传感器数据或事件实时处理的需求.在“操纵”物理世界方面,系统软件领域也已经展开了初步探索.例如,机器人操作系统/中间件面向机器人计算环境高度异构、资源受限、常态失效、实时性要求高等特点,自 2000 年左右展开探索,包括 Miro、CLARAty、OpenRTM-aist、Pyra、Orca、MARIE 等在内的一系列实践推动了无人系统中间件技术的发展[43],并直接为今天广泛使用的开源机器人操作系统 ROS(robot operating system)[44]奠定了理论和技术基础.

未来,作为信息物理融合软件系统的基础性软件,系统软件将继续增强感知和操纵物理世界的能力,正在演化成为“软件定义一切”的核心基础设施.文献[45]提出了普适操作系统的概念,指出操作系统正在将其软件定义能力扩展到物理世界和人类社会,这正是上述思想的体现.

(3) 与智能技术融合,为软件无缝融入人类社会提供支撑.

近年来,人工智能技术的巨大进步迅速反映到软件领域.“人工智能就是新的电能,正如 100 年前的电力改变了工业界,现在人工智能在做着同样的事情.”智能化软件正在将计算以人为中心、在人类社会中“随时随地而又

不可见”的理想变成现实.反映到系统软件层面:

- 一方面,系统软件本身需要具备操纵智能化软件执行的能力.例如:操作系统可能需要提供对深度学习加速硬件的支持,在运行时提供深度学习等共性抽象;针对大数据和深度学习应用,编译系统需要开发相应编程模型和接口、针对多硬件后端的可定制优化模块,以弥合高层次深度学习框架与底层硬件后端的性能差距;
- 另一方面,现代操作系统都提供了人机交互界面,而这一界面的智能化也是实现软件无缝融入人类社会的前提和基础.苹果 iOS 操作系统内建的 Siri 语音助手、谷歌 Android 操作系统内建的 Google Assistant 在这一方面已经做了有益的尝试,中间件在情境智能(如智能家居、智能城市)等方面也已取得一些初步成果[46].长远而言,系统软件将变得越来越智能,并且这种智能是在与人和人类社会交互过程中逐渐累积、以在线演化的方式成长起来的.

4 总 结

1936 年,阿兰·图灵提出了能够模拟任何图灵机的通用图灵机模型;1945 年,冯·诺依曼等人提出了“存储程序”的思想,二者分别奠定了系统软件的理论和技术基础.本文认为:系统软件是指操纵计算系统有效执行、为上层应用软件提供运行支撑的软件,其本质特征是“操纵计算系统执行”,这就是本文的第 1 个洞察.系统软件与底层硬件一起,向上层应用提供通用图灵机能力抽象,其操纵计算系统的主要方式有 3 种:编译加载应用软件、执行管控应用软件和资源、联接协调分布计算系统.洞察 2 认为:系统软件具有不间断持续运行的基本属性,在互联网时代的时代特点是持续在线提供基础服务,是当今云计算平台之所以成为公共服务平台的关键,是“软件即服务”新型应用模式的基石.洞察 3 认为:系统软件的发展趋势是持续在线演化,计算系统软/硬件技术创新、信息物理空间融合和智能技术是系统软件持续成长和在线演化的主要动力,这也决定了系统软件的未来发展趋势.

References:

[1] Turing AM. On computable numbers, with an application to the Entscheidungsproblem. Proc. of the London Mathematical Society, 1937,2(1):230−265.

[2] Lawton G. Moving the OS to the Web. Computer, 2008,41(3):16−19.

[3] Mei H, Huang G, Cao DG, *et al*. Perspectives on “software-defined” from software researchers. Communications of CCCF, 2015, 11(1):68−71 (in Chinese with English abstract).

[4] Burks AW. From ENIAC to the stored-program computer: Two revolutions in computers. In: A History of Computing in the 20th Century. Elsevier, 1980. 311−344.

[5] Turing AM. Computing machinery and intelligence. Mind, 1950,59(236):433−460.

[6] Davis MD. Engines of Logic: Mathematicians and the Origin of the Computer. W.W.Norton & Company, 2001.

[7] Columbia University. Programming the ENIAC. Retrieved 2018-05. http://www.columbia.edu/cu/computinghistory/eniac.html

[8] Campbell-Kelly M. The development of computer programming in Britain (1945 to 1955). Annals of the History of Computing, 1982,4(2):121−139.

[9] Backus JW, Beeber RJ, Best S, *et al*. The FORTRAN automatic coding system. In: Proc. of the Western Joint Computer Conf.: Techniques for Reliability. ACM Press, 1957. 188−198.

[10] Ritchie DM, Thompson K. The UNIX time-sharing system. Bell System Technical Journal, 1978,57(6):1905−1929.

[11] Stroustrup B. The Design and Evolution of C++. Pearson Education India, 1994.

[12] Salomon D. Assemblers and Loaders. Ellis Horwood, 1992.

[13] Tanenbaum AS. Modern Operating System. Pearson Education, Inc, 2009.

[14] Dijkstra EW. The structure of the “THE” multiprogramming system. In: Origin of Concurrent Programming. Springer-Verlag, 1968. 139−152.

[15] Bauer WF. Computer design from the programmer's viewpoint. In: Proc. of the Eastern Joint Computer Conf.: Modern Computers: Objectives, Designs, Applications. ACM Press, 1958. 46−51.

[16] Lee J. Time-Sharing at MIT: Introduction. IEEE Annals of the History of Computing, 1992,1:13−15.

[17] Mealy GH. The functional structure of OS/360, Part I: Introductory survey. IBM Systems Journal, 1966,5(1):3−11.

[18] Creasy RJ. The origin of the VM/370 time-sharing system. IBM Journal of Research and Development, 1981,25(5):483−490.

[19] Buxton JN, Randell B. Software engineering techniques: Report on a Conf. sponsored by the NATO science committee. In: Proc. of the NATO Science Committee; Available from Scientific Affairs Division. NATO, 1970.

[20] Tanenbaum AS, Van Renesse R, Van Staveren H, *et al*. Experiences with the Amoeba distributed operating system. Communications of the ACM, 1990,33(12):46−63.

[21] Tanenbaum AS, Van Renesse R. Distributed operating systems. ACM Computing Surveys (CSUR), 1985,17(4):419−470.

[22] Wang HM, Wu WJ, Mao XJ, Ding B, Guo CG, Li W. Growing construction and adaptive evolution of complex software system. Science China: Information Sciences, 2014,44(6):743−761 (in Chinese with English abstract).

[23] Birrell AD, Nelson BJ. Implementing remote procedure calls. ACM Trans. on Computer Systems (TOCS), 1984,2(1):39−59.

[24] Emmerich W, Aoyama M, Sventek J. The impact of research on the development of middleware technology. ACM Trans. on Software Engineering and Methodology (TOSEM), 2008,17(4):19.

[25] Ferris C, Farrell J. What are Web services? Communications of the ACM, 2003,46(6):31.

[26] Schmidt M, Hutchison B, Lambros P, *et al*. The enterprise service bus: Making service-oriented architecture real. IBM Systems Journal, 2005,44(4):781−797.

[27] Dragoni N, Giallorenzo S, Lafuente AL, *et al*. Microservices: Yesterday, today, and tomorrow. In: Proc. of the Present and Ulterior Software Engineering. Springer-Verlag, 2017. 195−216.

[28] Armbrust M, Fox A, Griffith R, *et al*. A view of cloud computing. Communications of the ACM, 2010,53(4):50−58.

[29] Mell P, Grance T. The NIST definition of cloud computing (draft). NIST Special Publication, 2011,800:145.

[30] Lu X, Wang H, Wang J, *et al*. Internet-Based virtual computing environment: Beyond the data center as a computer. Future Generation Computer Systems, 2013,29(1):309−322.

[31] Llyod's. Cloud Down: Impacts on the US Economy. Llyod's, 2018.

[32] Israeli A, Feitelson DG. The Linux kernel as a case study in software evolution. Journal of Systems and Software, 2010,83(3): 485−501.

[33] Hall M, Padua D, Pingali K. Compiler research: The next 50 years. Communications of the ACM, 2009,52(2):60−67.

[34] Wang H, Shi P, Zhang Y. Jointcloud: A cross-cloud cooperation architecture for integrated Internet service customization. In: Proc. of the 37th IEEE Int'l Conf. on Distributed Computing Systems. IEEE, 2017. 1846−1855.

[35] Maximilien EM, Ranabahu A, Engehausen R, *et al*. IBM altocumulus: A cross-cloud middleware and platform. In: Proc. of the 24th ACM SIGPLAN Conf. on Companion on Object Oriented Programming Systems Languages and Applications. ACM Press, 2009. 805−806.

[36] Munteanu VI, Sandru C, Petcu D. Multi-Cloud resource management: Cloud service interfacing. Journal of Cloud Computing, 2014, 3(1):3.

[37] Httermann M. DevOps for Developers. Apress, 2012.

[38] Pei D, Zhang SL, Pei CH. AIOps based on machine learning. Communications of CCCF, 2017,13(12):68−72 (in Chinese with English abstract).

[39] Ding B, Wang HM, Shi DX. Constructing software with self-adaptabilit. Ruan Jian Xue Bao/Journal of Software, 2013,24(9): 1981−2000 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4432.htm [doi: 10.3724/SP.J.1001.2013.04432]

[40] Wang M, Cao J, Li J, *et al*. Middleware for wireless sensor networks: A survey. Journal of Computer Science and Technology, 2008,23(3):305−326.

[41] Baldauf M, Dustdar S, Rosenberg F. A survey on context-aware systems. Int'l Journal of Ad Hoc and Ubiquitous Computing, 2007, 2(4):263−277.

[42] Wu QY. Network computing middleware. Ruan Jian Xue Bao/Journal of Software, 2013,24(1):67−76 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4296.htm [doi: 10.3724/SP.J.1001.2013.04296]

[43] Elkady A, Sobh T. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. Journal of Robotics, 2012.

[44] Quigley M, Conley K, Gerkey B, *et al*. ROS: An open-source robot operating system. In: Proc. of the ICRA Workshop on Open Source Software. Kobe, 2009. 5.

[45] Mei H, Guo Y. Toward ubiquitous operating systems: A software-defined perspective. Computer, 2018,51(1):50−56.

[46] Ding B, Wang HM, Shi DX. Pervasive middleware technology. Journal of Computer Science and Frontiers, 2007,1(3):241−254 (in Chinese with English abstract).

附中文参考文献:

[3] 梅宏,黄罡,曹东刚.从软件研究者的视角认识“软件定义”.中国计算机学会通讯,2015,11(1):68−71.

[22] 王怀民,吴文峻,毛新军,丁博,郭长国,李未.复杂软件系统的成长性构造与适应性演化.中国科学:信息科学,2014,44(6):743−761.

[38] 裴丹,张圣林,裴昶华.基于机器学习的智能运维.中国计算机学会通讯,2017,13(12):68−72.

[39] 丁博,王怀民,史殿习.构造具备自适应能力的软件.软件学报,2013,24(9):1981−2000. http://www.jos.org.cn/1000-9825/4432.htm [doi: 10.3724/SP.J.1001.2013.04432]

[42] 吴泉源.网络计算中间件.软件学报,2013,24(1):67−76. http://www.jos.org.cn/1000-9825/4296.htm [doi: 10.3724/SP.J.1001.2013.04296]

[46] 丁博,王怀民,史殿习.普适计算中间件技术.计算机科学与探索,2007,1(3):241−254.

王怀民(1962－),男,江苏南京人,博士,教授,博士生导师,CCF 会士,主要研究领域为分布式系统,云际计算,群体智能.

沈洁(1987－),女,博士,助理研究员,主要研究领域为高性能计算,并行计算.

毛晓光(1970－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为可信软件,软件维护与演化.

罗磊(1984－),男,博士,讲师,CCF 专业会员,主要研究领域为计算机视觉,机器学习,系统软件.

丁博(1978－),男,博士,副研究员,CCF 专业会员,主要研究领域为分布式计算,软件维护与演化.

任怡(1977－),女,博士,副研究员,CCF 高级会员,主要研究领域为系统软件,云计算与虚拟化,软件工程,分布计算.