软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006070]

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

面向操作系统可靠性保障的开源软件供应链*

梁冠宇[1,2], 武延军[1,3], 吴敬征[1,3], 赵 琛[1,3]

[1](中国科学院软件研究所,北京 北京 100190)

[2](中国科学院大学,北京 北京 100190)

[3](计算机科学国家重点实验室,北京 北京 100190)

通讯作者: 武延军, E-mail: yanjun@iscas.ac.cn

摘 要: 软件可靠性是软件工程领域中的研究热点之一,故障率分析是软件可靠性的典型研究方法.然而软件构建模式已从单体模式演进到以开源软件为代表的规模化协作模式,操作系统作为代表性产物之一,所含开源软件之间通过组合关系和依赖关系,形成了一个包含上万节点的供应关系网络.典型方法缺乏对供应关系的考量,无法准确识别和评估因此引入的软件可靠性问题.本文把供应链概念体系拓展到开源软件领域,提出一种基于知识的面向开源协作模式下软件供应可靠性的管理方法:面向开源软件生态进行本体设计,构建开源软件知识图谱,实现知识的提取、存储和管理,以知识为驱动,结合传统的供应链管理方法,提出一组面向开源软件供应链的可靠性管理方法,构成一套开源软件供应链管理系统.实验以 Linux 操作系统发行版的构建为例,展示了开源软件供应链对操作系统可靠性的支撑能力.结果表明,开源软件供应链将有助于理清和评估大型复杂系统软件的可靠性风险.

关键词: 操作系统;软件可靠性;开源软件;供应链;知识图谱

中图法分类号: TP311

中文引用格式: 梁冠宇,武延军,吴敬征,赵琛.面向操作系统可靠性保障的开源软件供应链.软件学报,2020,31(10). http://www.jos.org.cn/1000-9825/6070.htm

英文引用格式: Liang GY, Wu YJ, Wu JZ, Zhao C. Open source software supply chain for reliability assurance of operating systems. Ruan Jian Xue Bao/Journal of Software, 2020,31(10) (in Chinese). http://www.jos.org.cn/1000-9825/6070.htm

Open Source Software Supply Chain for Reliability Assurance of Operating Systems

LIANG Guan-Yu[1,2], WU Yan-Jun[1,3], WU Jing-Zheng[1,3], ZHAO Chen[1,3]

[1](Istitute of Software Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100190, China)

[3](State Key Laboratory of Computer Science, Beijing 100190, China)

Abstract: Software reliability is one of the research hotspots in the field of software engineering, and failure rate analysis is a typical research method for software reliability. However, the software construction mode has evolved from a single mode to a large-scale collaborative model represented by open source software. As one of the representative products, the operating system included open source software connected through combining relationships and dependencies has formed a supply network of tens of thousands of nodes. Typical methods lack consideration of supply relationships and cannot accurately identify and evaluate the software reliability issues introduced as a result. This article extends the concept of supply chain to the field of open source software, proposes a knowledge-based management method for software supply reliability in collaborative model: design the ontological body for the open source software ecosystem firstly, and then construct the knowledge graph of open source software to achieve the extraction, storage and management of knowledge; driven

* 基金项目: 中国科学院战略性科技先导专项(XDC05040100); 国家重点研发计划(2017YFB0801900); 中国科学院前沿科学重点研究计划(ZDBS-LY-JSC038); 国家自然科学基金面上基金(61772507)

Foundation item: Bulletin of Chinese Academy of Sciences (XDC05040100); National Key Research and Development Program of China (2017YFB0801900); Key Research Program of Frontier Sciences, CAS, Grant (ZDBS-LY-JSC038); National Natural Science Foundation of China (61772507)

收稿时间: 2020-02-16; 修改时间: 2020-04-04; 采用时间: 2020-05-09; jos 在线出版时间: 2020-06-10

by knowledge, combined with traditional supply chain management methods, A set of reliability management methods for open source software supply chain is proposed, which constitutes a set of open source software supply chain management system. Taking the construction of a Linux operating system distribution as an example in experiment, it demonstrates how the open source software supply chain will support the reliability of the operating system. Results show that the open source software supply chain will help to clarify and evaluate the reliability risk of large complex system software.

Key words: operating system; software reliability; open source software; supply chain; knowledge graph

1 研究背景

传统的软件可靠性研究重点关注单个软件在规定时间间隔内和规定条件下,系统或部件执行所要求功能的能力[1].单个软件的可靠性研究,通常会将软件拆分为多个独立的模块,之后基于不同的模型,结合软件模块间的关系以及各个模块自身的故障率进行分析,得出软件最终的可靠性评估结果[2].

然而,随着开源协作模式的不断发展,软件的构成和开发过程都发生了巨大的变化.现今软件开发更加关注敏捷和高效,基础功能通常会优先考虑复用相关的开源软件,开发者仅需要进行必要的扩展和改进即可.这种模式可以有效降低软件开发的成本,加速软件迭代,降低开发门槛,加快响应需求的速度.Octoverse[1]年度报告显示,截至 2019 年底,平均有超过 360 万个开源仓库对前 50 的开源项目有依赖,平均每个开源项目包含 180 个包依赖,超过 35 万名开发者参与排名前 1000 的开源项目,总贡献超过 500 万次,此外,有超过 130 万名开发者在 2019 年第一次为开源软件做出贡献.从以上数据不难看出,在开源协作模式下,软件之间的依赖关系将会变得更为普遍和繁杂.

操作系统指管理计算机硬件资源和软件资源的系统程序集合,包括内核及其他系统工具[3].由 Linux 内核衍生出的操作系统发行版（如 Ubuntu、CentOS、Android 等）,统称为 Linux 发行版,它们将众多实现不同功能的开源软件,以软件包的形式和 Linux 内核有机的整合在一起,以满足终端用户不同的使用需求[4].DistroWatch[2]的数据显示,较为常用的 Linux 发行版,仅一个版本就需要维护数以万计的软件包以支撑自身的功能和生态（如 Ubuntu 18.04 涉及 29207 个软件包、Debian Unstable 涉及 32453 个软件包等）,即便是通过剪裁构建而成的较为精简的系统,也包含近百个软件包.在没有工具帮助的情况下,按照正确的顺序和依赖关系安装软件包,需要具备必要的专业知识,以及进行琐碎的准备工作.为应对这一问题,软件包管理工具作为 Linux 发行版的必备组件,依据功能将开源软件拆分或合并为不同的软件包,并维护它们之间的依赖关系,有效减轻用户管理软件的负担.此外,为方便用户构建符合自定义需求的 Linux 发行版,包管理工具作为构建工具的核心组件,帮助用户理清需要添加的软件模块间的依赖关系.除了基于包管理工具实现的构建工具外,Linux 社区发起了 Linux From Scratch（LFS）项目,旨在指导用户如何从零开始构建操作系统,和依靠包管理工具提供包依赖关系并实现构建不同,LFS 衍生出 Automated Linux From Scratch（ALFS）项目,为用户提供自动化构建工具.此外,还出现了 Yocto 等更为高级的 Linux 发行版定制化构建项目,帮助 Linux 发行版版本构建者屏蔽底层硬件架构的差异性.

如上所述,在开源协作模式下,Linux 操作系统发行版蕴含着复杂的软件依赖关系,这使得其可靠性不仅依赖内核本身,而且需要关注其他构筑起整个生态的开源软件.尽管包管理工具能够维护软件包之间的依赖关系,但这些关系的可靠性更多的依赖人工管理和维护,基于这些信息对开源协作模式下的大型复杂基础软件及其生态做出可靠性评估存在较大的局限性.相应的,已有的系统构建工具同样仅关注构建功能本身,少有考虑最终构建生成的操作系统及其生态是否可靠.在已有的研究中,文献[5]同样关注到这样的软件组织形态,称在这种形态下形成的复杂网络为开源供应链,并提出开源供应链所面临的三大挑战:（1）个体开发者学习成本进一步增大;（2）群体协作更加复杂;（3）生态可持续性受到的威胁持续增加.除了这些挑战,当构建一款操作系统发行版时,开源软件供应链可靠性还面临着软件本身特有的挑战,即:（1）软件来源、许可和供应者的多样性;（2）

1 Octoverse: Octoverse (https://octoverse.github.com/)收集 Github 个人开发者活动数据,并从多个维度进行描述;

2 DistroWatch: DistroWatch(https://distrowatch.com/)专注于收集和探讨开源操作系统发行版的最新信息.

软件质量的波动性和传导性;（3）知识产权诉讼、出口管制等导致的供应中断;（4）可持续维护和发展等.这些挑战是影响现今开源操作系统及其供应关系可靠性的重要因素.

针对这些挑战,本文提出一种基于知识的面向开源协作模式下软件供应可靠性的管理方法.首先面向开源软件生态实现本体设计,并构建开源软件知识图谱,实现知识的提取、存储和管理.以开源软件图谱所蕴涵的知识为驱动,结合传统的供应链管理方法,面向开源软件供应链提出一组可靠性管理方法,最终构成一套开源软件供应链管理系统,应对上述挑战.本文第 1 节对整体研究背景做了阐述.第 2 节介绍本文涉及的相关领域成果.第 3 节将详细介绍本文提出开源软件供应链系统,首先会进一步阐述开源软件供应链的定义,之后会描述整个系统架构、数据管理和构建过程,以及可靠性管理方法,此外,还阐述了基于软件供应链实现的操作系统构建工具,相对于已有构建工具（LFS 和 Yocto）有哪些优势.第 4 节重点描述我们为验证本文提出的方法而实现的原型系统,并以面向 RISC-V 构建原生操作系统为例,对当前系统功能进行验证,并对面向该操作系统维护的开源软件供应链进行可靠性评估.最后总结全文,并对未来的改进及研究方向进行探讨.

2 相关工作

2.1 供应链及供应链管理

供应链的概念最早面向企业管理领域提出,最早可以追溯到 20 世纪 60 年代“物料需求计划”（Marterial Requirement Planning,简称 MRP）.由于当时的企业产能较低,供需矛盾主要聚焦在资源上,MRP 的提出主要是为了解决原材料库存与产品零部件投产量之间的计划问题,以最少投入和关键路径作为其基本出发点.随着企业间的竞争越来越激烈,企业对自身资源管理范围也向着更加广阔和精细化的方向发展,MRP 中单纯面向物料的管理已无法满足需求,于是企业进一步将物料和资金、人力、设备等资源关联起来,进行更加全面的计划和控制,使得 MRP 进化为 MRPII,即“制造资源计划”（Manufacturing Resource Planning）.随着 90 年代信息化技术的引入,又提出了新的企业管理计划——“企业资源计划”（Enterprise Resources Planning,简称 ERP）,并成为大型企业管理的标配.供应链管理作为 ERP 的核心,主要用以帮助企业明确业务流程,有效解决传统企业管理中常见的机构人员重叠、资源利用率低等问题.

时至今日,根据目标需求、应用环境等的不同,有很多关于供应链的不同定义.其中,英国供应链管理专家Martin Christopher在1998年给出的定义具有较高的公共认可度,他将供应链定义为“供应链是一种由多个组织参与组成的网络,在这个网络中,组织以上下游的关系相互关联,他们在不同的生产活动或过程中,以产品或者服务的形式为最终客户产品贡献价值”[6].

结合供应链的定义,我们可以更抽象的将供应链看作是一种生产资料的整合方式,它会尽可能全面的收集链上各方的数据,方便进行全局的分析和统筹,而对供应链数据的维护,以及对供应计划的优化等操作,可以统称为供应链管理.Lisa Ellram对供应链管理的定义得到了较为广泛的认同——“供应链管理是供应商和消费者之间集成了控制和计划的材料及产品流”[7].管理供应链系统的任务涉及很多方面,而核心任务主要涉及两个方面,即性能管理和风险管理.David Simchi-Levi准确的描述了供应链性能管理—— “供应链管理是通过一系列方法有效的整合供应商、制造商、仓储、商铺等资源,使企业能够准确控制商品生产和分发的数量,同时确保必要的资源在正确的时间出现在正确的地点,使得供应链系统在满足服务需求的同时,能够最小化整体的开销”[8].通过评估整体花销和最终产出的关系,即可量化的描述一个供应链系统的性能.

供应链风险管理主要是为了应对工业界的发展趋势,这些趋势包括越来越多的外包,更低的供应量基数,更精确控制时间,更短的产品生命周期等,它们都极大的增加企业供应链系统的风险[9,10].为使得供应链系统的风险可控,供应链风险管理将定制一系列策略实现对风险的识别、评估、处理和监控[11,12,13] .其中,风险识别作为首要任务,是开展其他任务的基础[14].目前,各项研究中已经提出很多风险识别相关的策略,并且一部分已经在实际应用中得到验证.风险评估通常基于数据或者专家经验做出判断[15],供应链系统中的风险通常不是孤立出现的,因此需要综合考虑多个风险间的关联关系,并跟据策略模型评定它们的风险级别,即处理的优先级[16].具体的风险处理方式依赖于对于供应链所处的环境,大致可以分为接受、避免、转嫁、分担和缓解[17].由于供应链的风

险并非静态不变,因此需要时刻监控以发现风险的发展动向,并作为调整处理策略的依据,由此也可以看出,风险的监控不仅高度依赖于前三项任务的结果,同时也需要具备一定的实时性[18].

本文的目标是维护可靠的开源软件供应链,为实现自主构建可靠的开源操作系统提供技术支撑.结合上文所述的供应链管理方法,以及第1节中对开源软件供应链可靠性风险特征的描述,本文将更加关注如何实现开源软件供应链的可靠性风险管理.在未来的工作,我们将在风险可控的前提下,提升供应链管理的性能.

2.2 知识图谱

关于知识图谱并没有一个明确的定义,大多数时候它能够与信息科学中的“本体论”这一概念混用[19].最早在 2012 年,Google 公司在其信息检索系统的技术介绍[20]中首先使用“知识图谱”这一名词,之后的一系列相关研究中,“知识图谱”被越来越广泛的使用.一种相对认可度较高的定义是:知识图谱是对实体描述的集合,这些实体具有内部的关联关系,它们可以是真实世界中的对象、事件、某种具体的情况或者抽象的概念等,其中:1)这些描述必须具有统一的结构,确保能够同时允许人类和计算机以高效且明确的方式对其进行处理;2)这些描述组成一个网络,且相互补充,使得每个实体都是其相邻实体描述的一部分[21].

一般情况下,知识图谱都具备以下组成部分:

- 实体:知识网络中具体的实例或对象;
- 类别:符合某一特征或概念的实体的集合;
- 属性:用于描述实体的特征;
- 关系:类别或者实体间的连接方式;
- 事件:当属性或者关系发生改变时产生;

此外,还具备以下特征:

- 支持结构化查询语言访问;
- 以图的形式维护网状数据;
- 所维护的数据具备形式化语义,支持数据理解和推理;
- 具备逻辑形式化,支持生成新的信息、强制一致性以及进行自动化分析;
- 具备动态性,所维护的数据支持自动或人工的持续集成和持续更新;

本文选择以实体图为起点维护开源软件供应链,不断扩充内容,最终构建成一个开源软件领域专用的知识图谱,为后续工作奠定基础.

2.3 常见的构建方法和工具

灵活、可定制是开源软件的一大特性,开源操作系统亦是如此.然而,以 Linux 系统为例,开源操作系统的复杂性使其定制工作的难度大大提升,相应的,也出现了不少开源项目旨在解决这一问题.由于篇幅有限,本文在此小节中将重点介绍两个较为有代表性的项目——LFS 和 Yocto.

2.3.1 LFS

LFS 项目创建的目的,是为用户提供一个从源代码开始,一步一步构建定制 Linux 操作系统的指导,希望在此过程中能够帮助用户理解系统内部的工作机制.本质上,LFS 可以被看作是一个框架,用户可以根据自己的需求,添加和删减功能,最终构建出一个压缩度更高且定制性更强的系统.此外,LFS 强大的灵活性也可以帮助用户加入自定义的安全组件,增强最终构建生成系统的安全性.为了增强整个项目的通用性,LFS 还有其他几个子项目:

- BLFS（Beyond Linux From Scratch）:以 LFS 为基础,指导用户配置和构建包含更丰富功能软件包的 Linux 系统,如图形化界面（LFS 构建生成的系统不包含图形化界面）;
- ALFS:如前文所述,该项目希望提供一个可扩展的系统构建器和包安装器,以实现系统的自动构建;
- CLFS（Cross Linux From Scratch）:在 LFS 的基础上,指导用户通过交叉编译技术实现面向不同体系架构构建 Linux 操作系统;

可以看出,LFS 及其子项目教育和指导的意义远大于工程实用的意义,项目自身也明确声明,用户根据项目指导构建而成的操作系统不是最精简的.要想构建出符合工程需求的系统镜像,用户不仅需要完全掌握 LFS,能够灵活应用其提供的框架,同时需要具备足够的系统知识,才能保证正确的配置构建.虽然存在 ALFS 这样面向工程的项目,但是社区活跃度不高等因素导致其发展并未达到预期.在软件包依赖（供应）关系方面,LFS 仅仅罗列了数量有限的软件包作为示例,而更多的软件包及其之间的关系则需要用户自己去发掘和管理,供应关系可靠性的管理和验证更是基本属于空白,若将构建生成的系统直接投入生产环境使用,存在极大的可靠性隐患.

2.3.2 Yocto

该开源项目的初衷,是为了帮助用户构建基于 Linux 的定制嵌入式操作系统,脱胎于 OpenEmbedded 项目,后者专注于提供构建系统,目前 Yocto 项目中的构建系统仍然延用 OpenEmbedded 项目提供的构建系统,并由双方共同维护.灵活性是 Yocto 项目最为看重的特性之一,项目本身由 3 个重要组件组成,即工具库、发行版构建模板（名为 Poky）以及前文提到的 OpenEmbedded 构建系统.内容丰富的工具库可以帮助用户实现诸如自动化构建和测试、多芯片支持、编译流程验证、嵌入式系统组件信息验证等功能,并支持不同用户之间共享工具.发行版构建模板是一个完整的构建样例,用户可以通过该模板学习 Yocto 的使用方法.此外,通过定制修改该模板的内容,用户也可以构建定制化的系统发行版.

使用 Yocto,用户需要首先定义目标架构、构建策略、补丁等配置细节;之后构建系统会根据配置获取目标源码,并根据策略在本地完成编译构建,最终打包为指定的软件包格式（如 deb、rpm 或者 ipk）;当所有的软件包构建完成,且通过了所有的检测项,构建系统会将所有的软件包打包成定制的软件源;最后,基于该软件源创建目标根文件系统镜像.

从上述流程可以看出,前期用户定义配置对于整个构建流程至关重要.Yocto 抽象出 recipe、 layer 等概念,对不同功能的构建配置实现封装,用于简化特性定制和功能复用,尽管如此,Yocto 仍然存在较为陡峭的学习曲线.此外,值得注意的是,在 Yocto 中,构建配置所描述软件包之间的依赖关系（即本文所称的供应关系）,需要由用户自定义或直接继承其他用户共享的已有配置,但是却缺少有效的工具实现供应关系可靠性的监督和检查.

3 开源软件供应链

3.1 定义

正如前文所述,当下开源模式盛行,各大IT行业巨头都在开源方面投入了大量的人力和财力,这也为他们赢得了诸如市场认可度、技术发展主导权等诸多益处.例如微软公司,曾经以 Windows、Office 等闭源软件而知名,近两年已在开源社交网站 Github 创建超过 3200 个开源仓库,共吸引了超过 4300 名开发者参与其中,产出多达 20 款成熟的开源软件工具.在这种趋势下,软件开发的模式也发生变化,从之前的单人/单组织独立开发,变成现在的多人/多组织协作.在软件模块化设计的加持下,基于他人已发布的开源软件模块,构建自己定制软件的开发模式逐渐成为主流,开源操作系统就是这种模式的典型产物.因此,基于开源软件这种特殊的生产方式,同时结合 2.1 小结中描述的供应链定义,可以得出,开源软件供应链的核心即是将开源软件模块间的依赖关系,看作是一种对操作系统构建的供应关系.为了更直观的描述,可以进行如下概念映射:

- 商品:交付给最终用户的软件,本文即特指大型复杂操作系统及相关软件包;
- 制造商:设计实现开源软件模块的个人或组织;
- 供应商:提供开源软件服务的公司或个人;
- 仓储:集中托管开源软件的仓库;

从概念映射可以看出,相较于传统的软件依赖关系模型,开源软件供应链能够更全面的描述软件的构成,除去构成软件本身的各个模块外,还引入了模块的开发者、维护者,以及获取源等维度.构建开源软件供应链的初衷,即是为了解决在当前模式下,如何保证开源软件可靠性的问题.以供应链的视角看待开源软件,我们能够以一种新的视角来审视大型复杂软件的构成,使我们更有可能发现软件潜在的可靠性风险,以应对文献[5]中提出的挑战.

定义 1(面向操作系统可靠性保障的开源软件供应链). 一个扩展的有限自动机 $A=(Q,O,\Sigma,\rho,\delta,q_0,F)$,其中,

- $Q=\{S_1,S_2\}$,表示有限的状态集合,其中,S_1表示可靠状态,S_2表示不可靠状态;
- $O=O_1 \cup O_2 \cup \ldots \cup O_n$,表示各 Linux 发行版提供的软件包集合,其中O_n表示某一种 Linux 发行版;
- $\Sigma=\{\omega_i \mid \omega_i \in O_j\}$,表示有限的输入集合,其中,$\omega$表示属于发行版$O_j$的开源软件包;
- 可靠性管理方法 $\rho:\Sigma \to \{0,1\}$,判断软件包是否可靠;
- 状态转换函数 $\delta: Q \times \rho(\Sigma) \to Q$,状态转换关系如图 1 所示;
- $q_0=S_1$,表示初始状态;
- $F=\{S_1\}$,表示接受状态集合;
- $\mathcal{L}(A)=\{\omega \mid \rho(\omega)=1, \omega \in O_j\}$,表示面向某一 Linux 发行版$O_j$,可靠的开源软件供应链;

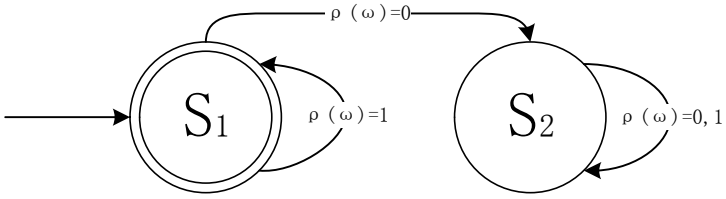

Fig.1 The state diagram of supply chain system of open source software

图 1 开源软件供应链状态图

基于定义 1,本文研究的问题转化为设计一种可靠性管理方法 ρ,从软件包集合 Σ 中筛选出符合输出状态集合 F 的软件包子集,组成面向 Linux 发行版的可靠的开源软件供应链.为实现这一目标,我们结合传统供应链管理积累的经验,以及知识图谱对数据丰富的表达能力,构建了一套以开源软件图谱为核心的开源软件供应链系统,整体架构如图 2 所示,各组件职责如下:

- 图形化界面（GUI）:直观的展示开源软件供应链系统,方便用户获取丰富的信息,包括开源软件之间的依赖关系图、开源软件的可靠性分析图表等;
- 应用编程接口（API）:以服务接口的形式对外提供开源软件供应链系统的能力,如供应关系获取、开源软件可靠性分析等;
- 数据源（Data Sources）:系统以软件包管理器作为数据源,提取开源软件间的供应关系,同时通过 Github 等上游仓库抽取开源软件的补充信息,如软件自身的演化信息、作者信息等,以丰富供应链系统中的数据信息;
- 数据源驱动程序（Data Source Drivers）:为采集不同的数据源的数据,提供不同驱动程序,验证采集到的数据,并转换为统一的格式方便系统存储和管理;

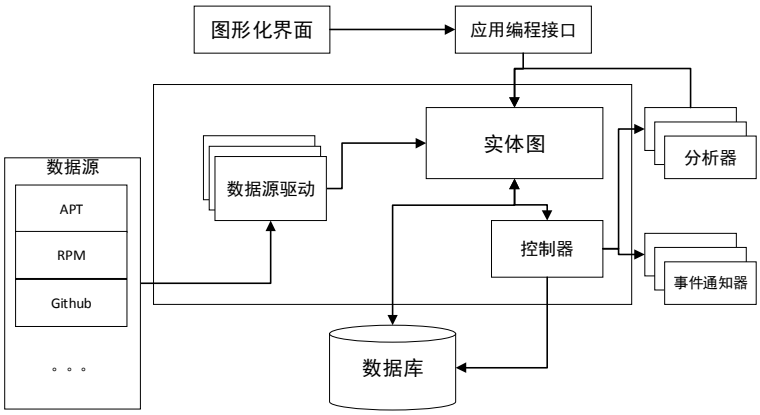

Fig.2 The architecture of supply chain system of open source software

图 2 开源软件供应链系统架构图

- 实体图（Entity Graph）:系统的核心数据结构——以实体图的形式维护所有数据信息,实体种类除了开源软件外,还包括人员、公司（或组织）等多种其他类型,实体间的关系也丰富多样,包括供应、开发、维护、从属等,以直观的形式建模整个开源世界的供应关系,本文将在 3.2 小节详细介绍这部分内容;
- 控制器（Controller）:负责协调和分发后台任务,包括数据分析和状态监控;
- 分析器（Analyzers）:以插件的形式实现,负责执行指定的分析任务,如软件可靠性分析等;
- 事件通知器（Notifiers）:以插件的形式实现,负责当系统生成事件时（如监控发现异常或者存在潜在风险）,发送通知至指定的位置;

3.2 数据管理

本小节重点介绍开源软件供应链系统的数据管理实现.如前文所述,系统以实体图的形式组织和维护供应链相关信息,相比于其他数据组织形式,实体图能够更加直观反应实体间的关系,蕴含更加丰富的语义信息,便于理解数据本身并推导出新的事实.随着数据体量的不断增大和数据内容的不断丰富,最终目标是将其建设成为一个关于开源软件供应链的知识图谱.

3.2.1 本体设计

本体设计是知识图谱构建的首要步骤,在该步骤中,我们需要抽象定义图谱中所包含的实体类型,并梳理它们之间的关系,最终以图的形式将它们组织起来.为了限制知识图谱的规模和复杂性,在本文中,我们限定知识图谱在开源软件领域.如图 3 所示,当前我们共定义了 5 大实体类型:

- 软件（Software）:软件是开源软件图谱的核心实体类型,根据开源软件的协作模式,我们进一步分为包（Package）和操作系统（OS）两个实体子类型,和软件实体类型是“isA”的关系,其中,包是软件中最小粒度的实体,而操作系统是由多个包组装而成,因此它们之间存在“hasComponent”的关系,而包之间的供应关系通过“hasSupplier”描述,包自身的演化关系通过“hasPreversion”描述;

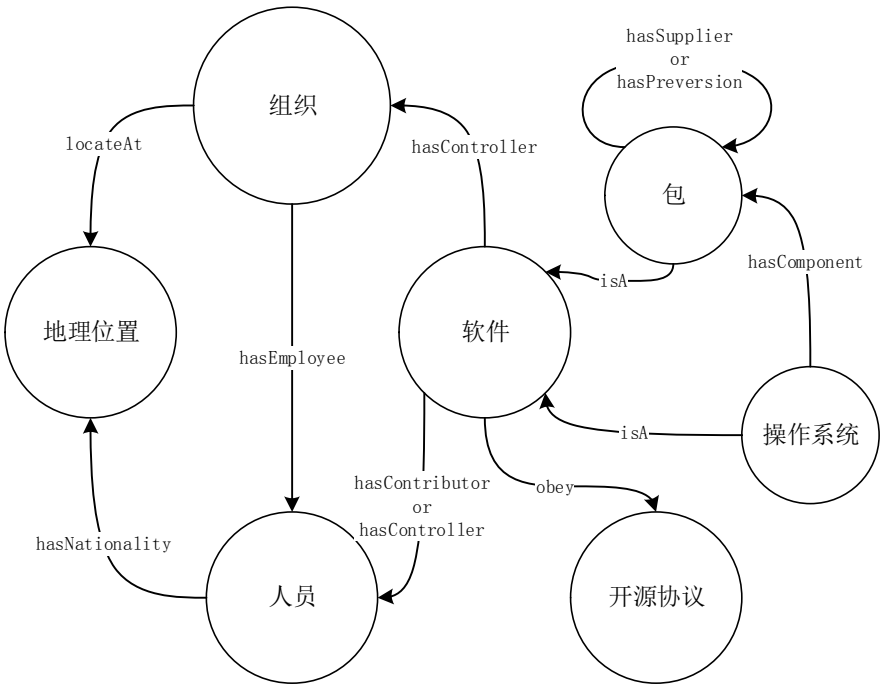

Fig.3 The ontology design of knowledge graph of open source software

图 3 开源软件知识图谱本体设计

- 开源协议（License）:该实体类型描述某一种具体的开源协议,软件必须“obey”一种或多种开源协议,它限制了开源软件的使用方式,对开源软件的可用性存在潜在的影响,因而使得开源软件供应链存在潜在的可靠性风险;
- 地理位置（Location）:该实体类型描述某一个具体的地理位置,和组织或者人员是一对多的关系,由于任何一个软件都至少由一个组织或个人维护,进而使得软件包本身带有地理位置的属性,通常情况下,组织或个人需要遵守当地的法律,而当由于政治等因素导致某个国家通过法律手段,限制某个组织或个人对其所拥有开源软件的开放等级时,开源软件间现有的供应关系极可能被破坏;
- 组织（Organization）:该实体类型描述某一个具体的组织,可以是开源组织,或者是提供开源软件的公司,它们对某些开源软件有实际的控制权,因此存在“hasController”的关系,同时,它们“locateAt”某一个具体的地理位置,需要遵守当地的法律法规;
- 人员（Human）:该实体类型描述某一个具体的开发或维护人员,它们既可以是以个人名义,亦或者属于（“hasEmployee”）某一个具体的组织为开源软件做出贡献,因此他们和软件之间存在“hasContributor”的关系,同时某个人员可能以个人名义对某个开源软件有实际控制权,因此他们和软件也可能存在“hasController”的关系;

为了简洁,图 3 中仅展示了基本的实体类型和它们之间的关系,省略了每个实体类型所包含的属性,该部分将在 4.1 小结中做进一步的描述.

3.2.2 构建过程

构建本文提出的开源软件知识图谱,主要分为两个步骤,如图 4 所示,具体如下:

1） 基础图谱构建

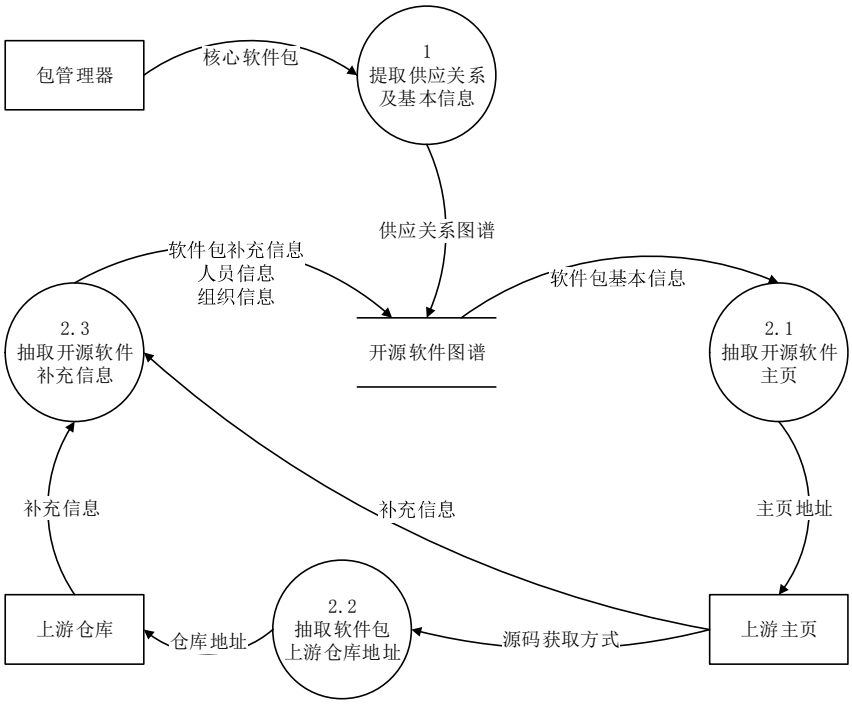

Fig.4 The data flow diagram for contructing the knowledge graph of open source software

图 4 开源软件知识图谱数据流图

此步骤以现有包管理器（RPM[1]、APT[2]等）作为数据源,以目标构建的操作系统中提供核心功能的软件包作为“种子”,通过挖掘“种子”软件包的供应链条,即可得到基本的供应关系图谱,同时为每个软件包实例填充基本的属性信息.需要注意的是,在此过程中,会判断某一软件实体是否已经存在,若存在则不会重复添加.该基础图谱仅包含软件包基本信息,如“name”、“size”等,而“features”、“maintainers”等属性的信息则无法获取,因此需要通过第二步来完善和扩展整个图谱的“知识量”.

2） 扩展补充信息

此步骤基于第一步生成的基础图谱,分三个子步骤完成信息扩充.

a) 从基础图谱中,遍历每个 Package 类型实体,提取上游（upstream）软件的主页地址;

b) 访问每个上游软件的主页,抽取源码获取方式,即开源代码仓库地址,如 GitHub 中的仓库地址等;

c) 访问上游代码仓库,抽取软件的历史版本信息,包括每个版本的版本号、累计提交信息（即自身演化历史）、参与开发人员信息、版本特性、已知问题等,若个别信息不全,则需要在上一步中从主页中抽取补充.

正如2.2小节所述,知识图谱的重要特性之一即是具备动态性.对开源软件知识图谱而言,图4所描述的流程可以理解为一次完整的更新过程,随着新“种子”不断补充,图谱的“知识量”也会不断扩充,为整个开源软件供应链的可靠性管理提供充分的数据支持,同时为未来工作打下坚实的基础,如提供智能化的操作系统自主构建服务.

1 RPM:常用包管理工具,Red Hat、Fedora、openSUSE、CentOS 等主流操作系统发行版的核心组件;

2 APT:常用包管理工具,Debian、Ubuntu 等主流操作系统发行版的核心组件;

Table 1 reliability metrics of open source software

表 1 开源软件可靠性度量

度量			描述
软件包分析	活跃程度	更新频率	软件提交更新的频率
		问题响应速度	响应反馈问题或提议的速度
		贡献者数量	参与贡献人员数量
	贡献者地理分布		参与贡献人员所述国籍的分布
	可替代性		可被类似功能软件包替换的程度
	开源协议类型		所遵循的开源协议
	归属	性质	归属于组织或个人
		国籍	组织或个人的国籍
供应链分析	可靠性分布		供应链中软件包的可靠性分布
	软件包地理分布		供应链中软件包所属国籍的分布
	开源协议分布		供应链中遵守开源协议的分布

3.3 可靠性风险管理

如前文所述,本文关注的可靠性并非一般意义上单体软件的可靠性,而是在当前开源软件模式下,由于文献[5]所描述的 3 大挑战而引入的开源软件生态可靠性风险.结合供应链风险管理的四个主要任务——即识别、评估、处理和监控（我们认为在开源软件供应链风险管理中,识别和评估两个任务界限较为模糊,故本文将其合并为一个任务）,我们在系统中设计并实现了相关的策略和功能,实现可靠性风险管理的同时,在一定程度上解决开源软件供应链面临的 3 大挑战.

3.3.1 识别和评估

为识别供应链中的可靠性风险,我们提出一种开源软件可靠性的度量模型,使系统能够量化的描述软件可靠性,具体度量项及描述如表 1 所示.在度量模型中,主要分为两大部分,即软件包本身和其所依赖的供应链,这一模型也是为了直观表述我们对现代软件构成的认识,即可靠性不仅依赖软件本身的一些指标,同时也和它复杂的依赖项密切相关.在模型中,我们除了关注常规的度量项,如社区活跃度、开源协议等,还重点关注了开源软件的地理属性,包括参与贡献人员、维护（或拥有）者的国籍分布,以及供应链中其他依赖项的国籍分布,这是为了能够更直观的描述,由于政治因素而引入的潜在可靠性风险.同时,我们还关注软件包的可替代性,显而易见,无法被替代的软件包将成为整个供应链中较为脆弱的一环.

在本文提出的系统中,结合 2.2.2 小节中描述的本体设计可以看出,度量模型中的基础项,如软件包的归属、所遵循的开源协议等,已经存在于开源图谱中,而其他进阶数据如可替代性、各类分布等则需要基于基础数据加工得到.最终,基于度量模型提供的数据,我们通过可靠性评分的形式,量化的描述任一软件的可靠性.就我们所知,目前并没有相关公开的基准数据集.因此,为了保证我们系统对开源软件可靠性评价的客观性和公正性,在当前系统中,我们采用多种方法综合的方式得到最终可靠性评分,并基于该评分实现开源软件供应可靠性风险的识别和评估.

基于开源图谱和度量模型,我们设计并实现了图 5 所描述的数据流程,以得到对软件的可靠性评分.在目前的系统中,我将可靠性评分限定在 0 至 5 分,以小数表示中间分值,并保留小数点后两位,如 3.47.可靠性评分的计算过程如下:

$$P = f(e) \tag{1}$$

$$P_{init} = similar(attr, cluster(G)) \tag{2}$$

$$P = \frac{\sum_i^n a_i p_i}{n} \tag{3}$$

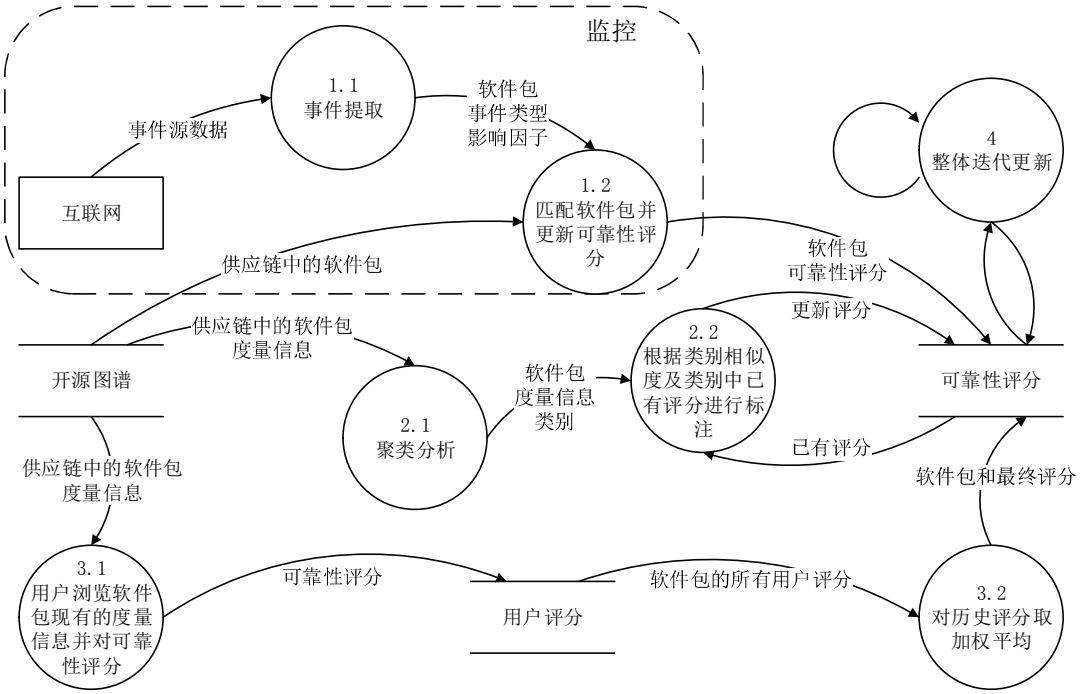

Fig.5 The data flow diagram for generating evaluation of software's reliability

图 5 可靠性评分数据流图

$$P_{new} = \frac{P_{old}+P_{updated}}{2} \tag{4}$$

1） 更新途径 1 如公式(1)所示,e 表示通过从互联网上抽取相关事件信息,包括软件直接相关的信息（如明确表示了限制使用或不再维护,或者在事实上已经处于无人维护的状态）、软件归属国籍的政治因素（如存在出口管制,明确限制某些国家的组织或个人使用,或者明确限制某些用途）等,f 表示具体的分值计算方法,因计算过程可与监控任务复用,故在 3.3.3 小结中具体描述,此外,由公式(4)可以看出,当某一软件其供应关系中包含了该受事件影响的软件包,其可靠性评分也会受到相应的影响;

2） 显然,仅仅通过途径 1 获取的可靠性评分依据将十分有限,对于尚未获得初始评分的软件包,我们基于度量模型提供的数据对软件包进行聚类,如公式(2)所示,这使得具有类似“可靠性特征”的软件包能够归为一类,之后即可基于相似度为尚未获得评分的软件赋予初始化评分;

3） 在途径 1 能够获取的数据极端稀少的情况下,通过途径 2 赋予的初始化评分,其参考价值十分有限,因此在途径 3 中,我们允许用户基于系统提供的度量信息,对任一软件的可靠性进行评分,并最终综合全部用户的评分,得到某一个软件的可靠性评分,如公式(3)所示,可以看出,考虑到用户不同的专业水平,我们在综合所有评分时,使用加权平均的方式,用户的专业水平和其评分的权值成正相关,通过这一途径,我们能够从大众对生态可靠性的认知中,获取符合大多数人标准的可靠性评价,提升整个数据集的参考价值;

4） 基于度量模型,软件自身的可靠性依赖于其供应链中软件的可靠性,因此,当其他途径更新了软件的可靠性评分后,需要对该软件所属供应链上的软件进行更新,我们以评分发生更新的软件为起点,沿着供应关系逆向遍历开源图谱并更新途经软件的可靠性评分,考虑到可靠性风险并不随供应路径的加长,而呈现衰减或增强趋势,故我们以算数平均的方式计算新的可靠性评分,如公式(4)所示.这样不仅可以保证计算的简洁性,同时也可以直观的表达其影响的传播,进一步的,还可以反应出多种风险叠加的后

果.为了保证每一次更新过程能够收敛,我们规定只有前三种更新途径会触发该更新过程;

3.3.2 处理

正如前文所述,供应链风险处理大致可以分为接受、避免、转嫁、分担和缓解.考虑到开源软件供应链的特殊性,我们主要从可靠性风险的等级出发,针对接受、避免和缓解这 3 类提出风险处理方案.具体的风险等级划分如表 2 所示,我们将风险等级划分为低、中、高三个等级.评分在 3.5 分以上的软件包风险较低,处理优先级较低,可以采用暂时接受的处理方式;评分在 1.5 至 3.5 之间的软件包属于中等风险,需要采用必要的措施缓解风险;评分不足 1.5 的软件存在高可靠性风险,必须以最高优先级进行处理,做到完全避免.

Table 2 risk level of reliability

表 2 可靠性风险等级

可靠性评分	风险
(3.5, 5]	低
(1.5, 3.5]	中
[0, 1.5]	高

本文提出的系统中,不会直接采取措施处理捕获到的可靠性风险,而是会为改善软件及其供应链的可靠性提供决策支持.具体的,系统在展示软件可靠性度量信息的同时,会依据可靠性评分,由低到高推荐该软件的供应链中需要被优化风险的软件包.配合 2.3.1 小结中的评分机制和 2.3.3 小结描述的监控机制,系统将根据优化的结果重新评估软件的可靠性,最终使得被优化软件包所在供应链的可靠性风险降低.

3.3.3 监控

供应链系统是一个动态的系统,链上各个节点的状态会不断动态变化,相比一般的产品,软件天然具有快速迭代的特性,开源软件供应链的变化周期将进一步缩短,不确定因素更多.因此,可靠性风险管理的时效性显得尤为重要,若不能及时捕获对可靠性风险有影响的事件,会导致系统对风险产生错误的评估.结合 3.1 小结中描述的系统架构,我们基于多个组件配合实现可靠性风险监控流程,如图 5 所示（数据流程如图 5 中虚线部分所示）.整个流程分为 3 个阶段,在数据采集阶段,面向不同数据源（表 3）实现相应的驱动程序,分别负责采集相关数据,并以时序数据的形式保存;在数据分析阶段,分析器程序基于历史数据和最新捕获的数据进行事件提取（在下文中详述）;若发现有效事件,会触发事件处理器分发至相应的处理流程,在目前的系统中,会触发可靠性评估子流程,评估结果将反馈给监控时序数据,用于优化事件提取子流程的精确度.

在当前系统中,我们基于规则实现事件提取.结合我们提出的度量信息模型,我们总结了如表 3 所示的 6 种风险,表中同时展示了每种风险对应的度量信息模型中相关的属性,以及相关的数据来源.如前文所述,我们根据不同风险的不同数据源,实现相应的数据驱动,因此也可以方便的为每种风险数据信息打上标签,用于标识风险类型.事件提取子流程如图 6 所示,首先基于标签判断风险事件类型,若是表 3 所示的前两种风险类型,则基于关键字实现信息提取,主要包括软件名称和情感词,对照系统预制的情感词表即可得到影响影子;若为后四种风险,同样通过关键字匹配提取软件名称,之后通过对比历史信息或计算得到的统计数据,即可得到风险的影响因子,具体的统计指标见表 3,需要说明的是,当前指标是根据我们的经验指定的,可以认为是经验初始值,在未来的工作中,我们将根据系统收集的数据对指标进行调整.影响因子的意义是为了描述,监控到的事件对于软件可靠性风险起正向影响还是负向影响,若事件结果能够帮助降低风险,则认为影响是正向的;否则,认为影响是负向的.为简化计算,当前正、负向因子均取值为 1,未考虑不同事件的影响力,随着数据的积累和系统的完善,在未来工作中我们会通过引入权重的方式,进一步客观的描述每一事件对软件可靠性风险的影响.

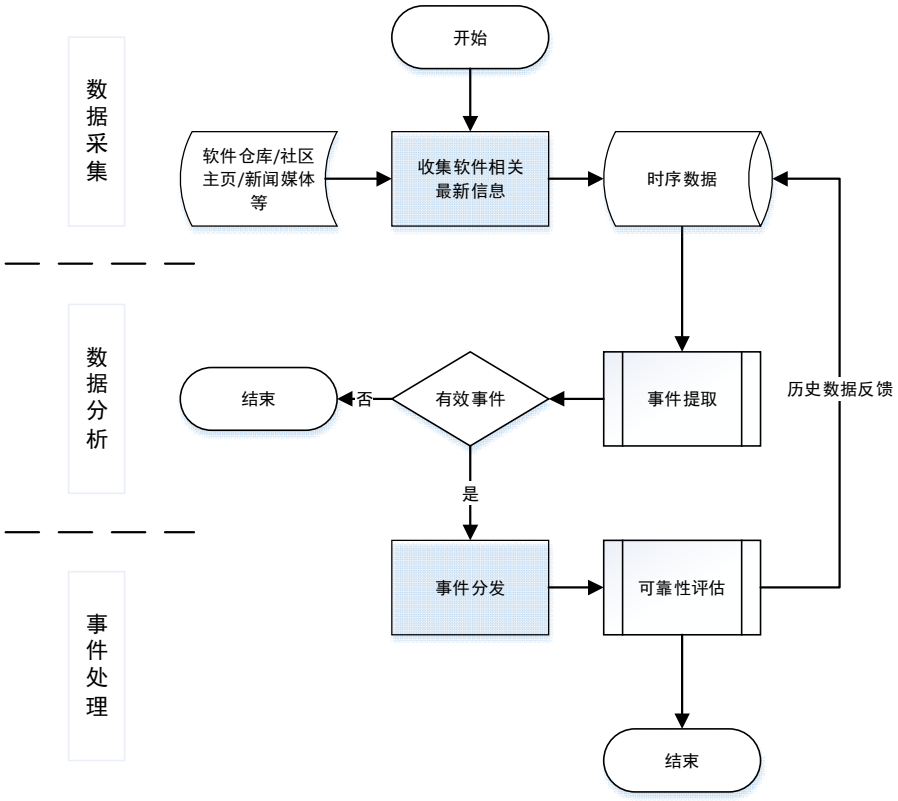

Fig.6 The flow diagram for monitoring risks of reliability

图 6 可靠性风险监控流程

如图 7 所示,事件提取流程最终输出的结果包括软件包的名称、事件类型,以及影响因子.图 6 中的事件处理阶段接收到以上信息后,按照以下公式更新可靠性评分:

$$v_i' = v_i + e \tag{5}$$

$$p = \begin{cases} \sum_{i=1}^{n} v_i * \frac{P_{max}}{N}, & N > 0 \\ 0, & N = 0 \end{cases} \tag{6}$$

为计算事件影响的可靠性评分,我们为每个软件包维护了两个变量——V 和 N.其中,V 为数组,记录每种风险事件捕获后的影响因子累计结果,每一项初始值为 1（为确保公式（6）计算结果在没有任何负面影响时为满分）,公式（5）描述了累计结果更新规则,v_i和v_i'分别表示某一种风险事件累计值更改前和更改后的值,e 表示本次捕获到的该事件的影响因子.另一个变量 N 表示捕获事件的次数,公式（6）描述了受事件影响的可靠性分值更新规则,式中,p 表示最终分值,v_i与（5）式中含义相同,n 表示风险类型的数量,P_{max}表示满分的分值（即当前所设置的 5）.当 N 为 0 时,即未捕获过相关事件,显然该途径对可靠性分值没有任何贡献,故 p 值为 0;当 N 大于 0 时,通过累加所有风险的影响因子累计值得到总的可靠性评分,为保证最后的计算分值在区间[0, P_{max}]内,添加正则项$\frac{P_{max}}{N}$.显然,当捕获到负面事件时,根据公式（6）计算得到的结果将小于P_{max},且最终分值与捕获到负面事件的数量呈负相关.结合 3.3.1 小结更新途径 4 所描述的更新规则,监控任务捕获到的面向不同风险类型的事件,最终会对软件的可靠性评分产生正面或者负面的影响.

Table 3 risks to be monitored

表 3 风险监控

风险	数据源	相关度量模型属性	说明
政治风险	新闻网站 社交媒体	贡献者地理分布 归属/性质 归属/国籍	因政治因素引入的可靠性风险:如中美关系紧张,美政府要求 Google 公司限制华为手机安装并使用其提供的服务软件.
可替代性差	专用网站 用户补充	可替代性	不存在功能类似的软件或类似软件成熟度不足.
协议变动	开源软件的主页、社交媒体及仓库	开源协议类型	通过更换开源协议来限制使用:如各大开源厂商通过替换开源协议限制云厂商使用.
更新频率低	开源软件仓库	社区活跃度/更新频率 归属/性质	超过 1 周无任何更新,若性质为个人项目,则风险更大.
问题响应速度慢	开源软件社区	社区活跃度/问题响应速度 归属/性质	反馈的问题超过 3 天无人响应,若性质为个人项目,则风险更大.
贡献者数量低	开源软件仓库及社区	社区活跃度/贡献者数量 归属/性质	个人项目或贡献者数量少于 3 人,使得软件存在潜在的不可用风险.

3.4 基于可靠供应链构建操作系统

虽然以 LFS 和 Yocto 为代表的面向操作系统构建的开源项目,已经在一定程度上满足了用户构建定制化操作系统的需求,但仍然存在很多不足和需要改进的地方,尤其是在软件生态可靠性方面.为解决可靠性问题,我们提出一种基于开源软件供应链实现的构建方法[22],该方法使用本文提出的开源软件供应链替换现有构建工具中软件依赖关系管理方法.具体的,开源软件供应链主要有以下贡献:

- 降低学习门槛:开源软件供应链管理系统维护了软件之间的供应关系,包括不同软件之间的供应关系,以及从上游源码到具体打包格式的供应关系.通过系统提供的检索功能,用户可以轻松获取供应关系信息,不再需要构建人员花费大量精力去记忆这些信息,进而能够保证用户快速、准确、高效的完成工作;
- 降低维护成本:开源软件供应链管理系统能够以图形化的方式展示供应链信息,帮助用户更直观的管理供应关系,提升工作效率;
- 提升生态可靠性:开源软件供应链管理系统通过内部实现的可靠性风险管理相关机制,为用户提供识别、评估、处理和监控可靠性风险的功能,有效降低目标构建系统软件生态的可靠性风险,提升可靠性.

4 实验验证及结果

4.1 系统实现

基于图 1 所示的系统架构图,我们将系统按照功能组件进行拆分并分别实现,本小节将从中挑出一些重要的实现细节加以描述.为便于管理和维护,各组件以容器的形式部署在 Kubernetes 集群上,我们使用 4 台配置相同的服务器搭建该集群,单台服务器的配置如表 4 所示.

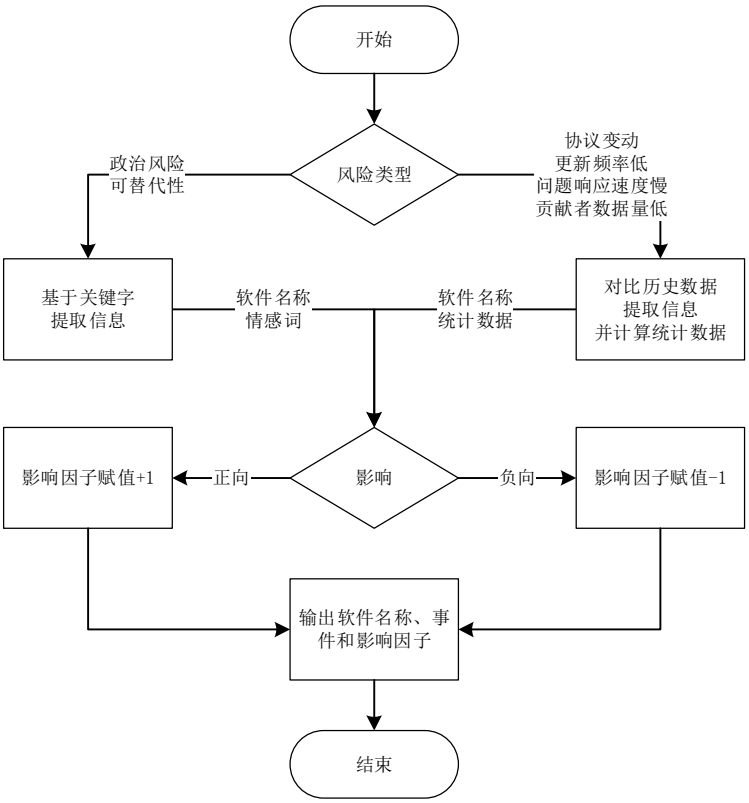

Fig.7 The flow diagram for extracting events

图 7 事件提取流程

实体图是系统实现的重要组成部分,基于图 3 描述的本体设计,我们将其映射为图 8 所示的数据结构.具体的,我们严格按照面向对象的设计思想,将每个实体视作一个类,并为每个类定义相关的属性和类型.在图中,我们以不同形式的箭头描述了类间的关系.其中,Entity 和 Relation 是基础类型;空心箭头表示继承关系,即 Software、License、Organization、Location、Human 继承实现 Entity,而 Package 和 OS 继承实现 Software;虚线单箭头表示依赖关系,描述类属性对其他类及其属性的依赖关系;实线单箭头表示关联关系,

Table 4 Hardware Configuration

表 4 硬件配置

CPU	E5 2690 V4(2*28 线程) 2.6 GHz
内存	128 GB
磁盘	38.3 TB

即图中 Relation 通过 from 和 to 属性分别指向关联 Entity 的 key 属性,将两个不同的 Entity 实例连接起来,以表示两个 Entity 之间存在某种 Relation.

在我们的实现中,使用图数据库保存知识图谱数据.根据 G2[1]的推荐,在众多候选的图数据库中,我们锁

[1] G2:当用户寻找最优解决方案时,G2(https://www.g2.com/)通过多维数据分析给出客观的建议,帮助用户做出决策;

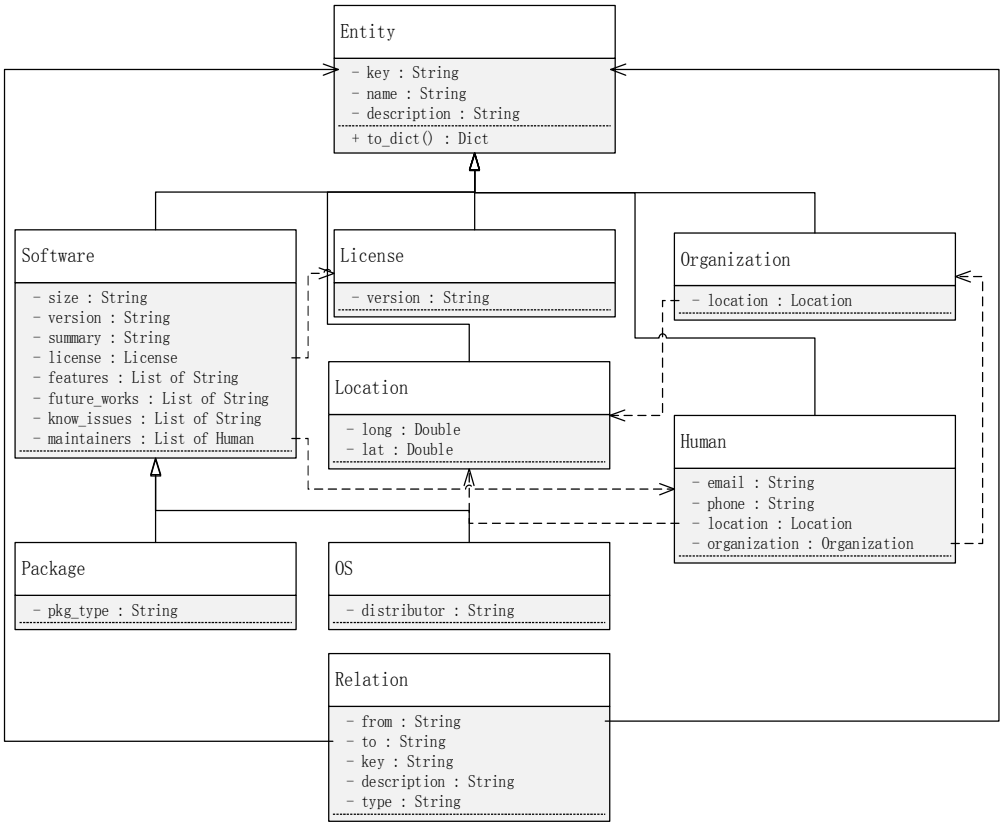

Fig.8 The data structure of knowledge graph of open source software

图 8 开源软件知识图谱数据结构

定在排名前二的 Neo4j[1]和 ArangoDB[2]两款工具上,通过进一步比较,虽然 Neo4j 的工程采用度更高,但是在开放性方面,前者的社区版本(Neo4j 分为商用版本和社区版本)在横向扩展能力上严重受限;相比之下,后者完全开源,因此我们选择更具开放性的 ArangoDB.基于 ArangoDB 提供的数据访问接口,我们根据实际的业务需求,参照 RESTful[23]标准实现一套接口,用于其他组件检索和更新实体图.

由于开源软件体量巨大,可以预见,开源软件知识图谱将包含海量的数据.为便于用户探索和获取相关知识,并进行直观的数据分析工作,我们通过图形化界面实现开源软件供应链可视化,降低使用门槛.在实现供应链可视化时,我们需要同时兼顾全局信息的完整性和局部细节的明确性,然而对开源操作系统发行版而言,供应链通常包含几百甚至上万个节点,这与有限的屏幕尺寸形成一组尖锐的矛盾.考虑到每一个开源软件供应链其实都可以看作是一个有向无环图(Directed Acyclic Graph,以下简称为 DAG)[24].因此,在可视化供应链时,我们利用 DAG 的特性,基于拓扑排序算法[25]实现供应链分层,即每屏只需要展示一层供应链节点的细节,有效克服屏幕尺寸带来的可视化供应链容量限制.同时,为保留供应链的全局信息,我们会为每个供应链提供分层结构的全局缩略图,在用户查看某一层供应链细节时,在全局缩略图中做相应标识,保证用户获取完整的信息.

1 Neo4j:图数据库,2007 年发布第一个版本,使用 Java 语言开发实现;

2 ArangoDB:图数据库,2011 年发布第一个版本,使用 C++和 Javascript 语言开发实现;

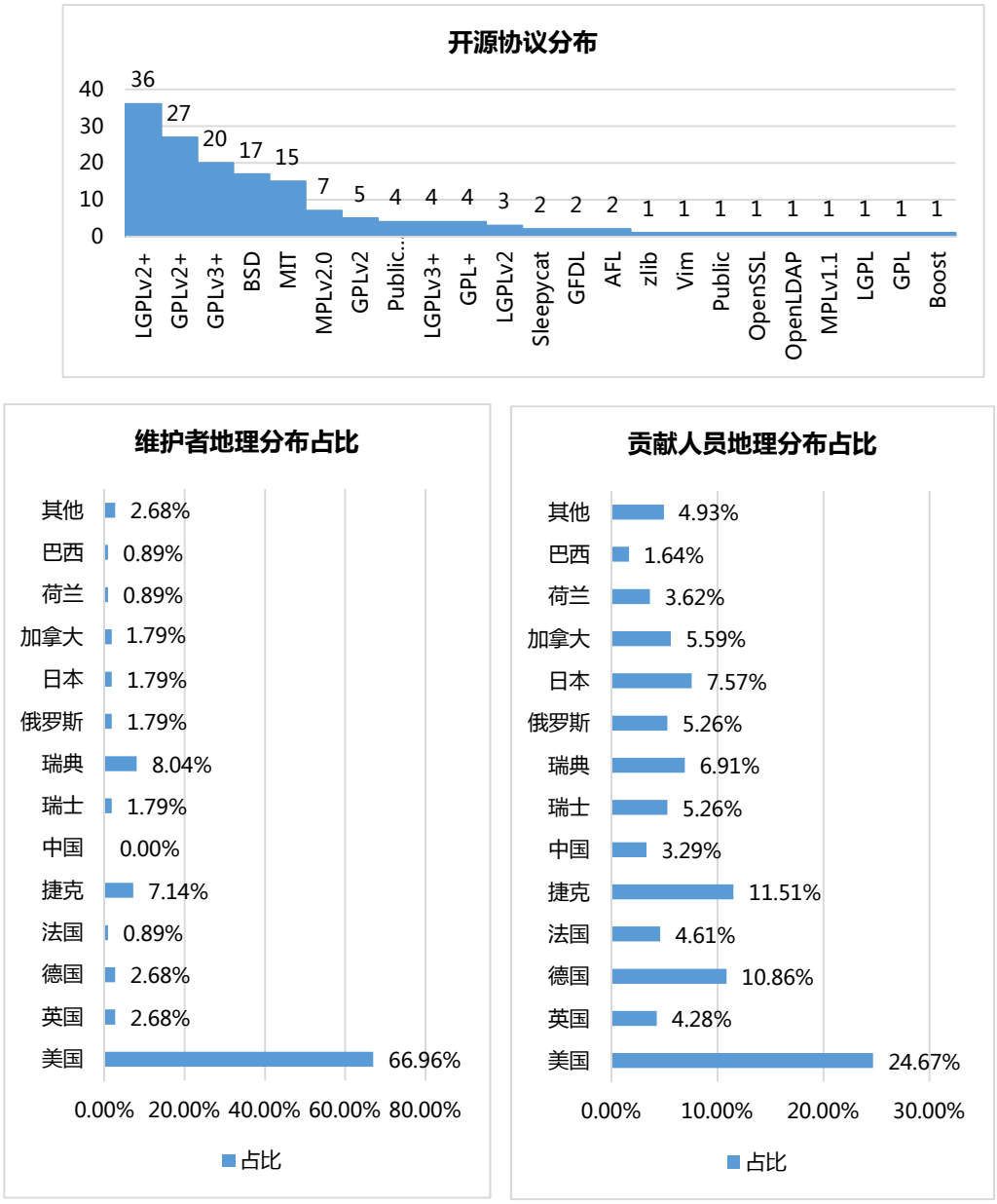

Fig.9 Statistics of open source supply chain

图 9 供应链统计数据

此外,系统中的控制器、分析器和事件处理器均采用插件式实现,允许以插件的形式扩展系统相关功能,保证系统的灵活性和可扩展性.

4.2 可用性验证

为验证系统可用性,我们以一个面向 RISC-V[1]构建的精简 Linux 操作系统为例,为其构建并维护开源软

[1] RISC-V:一种基于精简指令集（RISC）原则的开放指令集架构（ISA）,由加州大学伯克利分校发起,常解释为“开源硬件”;

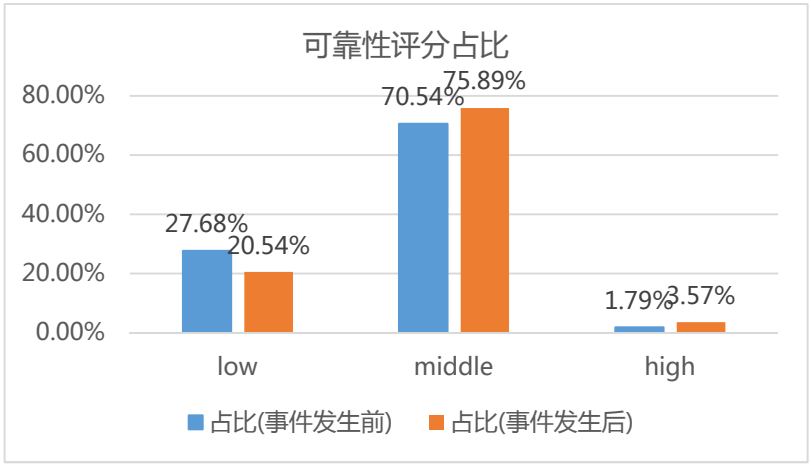

Fig.10 The proportion of reliability score

图 10 可靠性评分占比

件供应链.该精简操作系统以基本的文本编辑为使用场景,经过剪裁,确认仅需要保留 bash、coreutil、passwd、systemd-udev、util-linux、vim-minimal 这 6 个核心软件包.为便于检索,类似 1.2.3 小节所描述的构建过程,我们为该操作系统加入一个特殊的软件实体,并将这 6 个软件包作为其供应依赖实体,将这些信息输入系统即可将该操作系统的软件实体加入到开软软件图谱中.提取该例所构建系统的软件供应链时,只需要以该实体为起点进行检索即可,最终生成开源软件供应链共包含有 112 个软件包,相关统计数据如图 9 所示,包括开源协议分布、软件维护者地理分布,以及贡献人员地理分布.

通过开源协议分布数据可以看出,在该供应链中,LGPLv2+、GPLv2+、GPLv3+、BSD、MIT 这几种协议的采用率较高,占到总数的 73.2%.其中,采用率最高的是 LGPLv2+,该种类型协议约束下的开源软件允许商业软件直接使用,但是不允许修改源代码;GPLv2+和 GPLv3+的采用率次之,总体来说,GPL 类协议约束下的开源软件,允许使用和修改源代码,但不允许修改后的代码作为闭源商业软件的一部分发布和销售;BSD 和 MIT 两种协议约束更为宽松,既允许使用和修改,也允许商业发售.整体来看,目前大部分开源软件所采用的协议以鼓励开放和共享为主,同时,对于无条件索取的行为也在进一步加以限制.对于国内大部分软件产品的供应链可靠性来说,若不尽快调整我们对开源的认知和态度,构建更加可靠的软件生态,这些限制无疑将在未来给开源软件供应可靠性带来巨大的挑战.

此外,从图 8 的两个地理分布相关的图例中,可以明显的看出,在构建操作系统这类基础软件所需的开源软件包中,无论是项目维护者和还是直接参与贡献的人员中,我国的比例都相对较低.造成这种现象的原因,一方面可能是因为国内开源文化起步较晚,大部分从业人员或相关公司都对开源的理解有限,与此同时,基础软件开发存也在门槛相对较高,且投入成本较大等现实问题,这些因素综合起来导致整体参与度不高;另一方面,也可能是我国的开源贡献人员在这些开源软件中缺少话语权,很多建议和贡献都很难被采纳,因而导致贡献度较低.显然,较低的主导权和参与度会对开源软件供应链可靠性带来巨大的风险.

根据 2.3.1 小节描述的可靠性评分生成过程,在本实验中,由于缺少途径 1 相关数据,我们主要依赖途径 3,为了尽可能保证数据的客观有效,我们邀请了 4 位长期从事基础软件可靠性和安全性研究的博士,10 位具备硕士学位或 5 年以上工作经验的相关专业从业者参与实验,以及 10 位有 2 年以上开源软件使用经验的在读研究生,所有评分参与者均依据系统提供的统计数据和自己的经验进行打分.考虑到 3 类人员不同的专业能力,我们为 3 类人员分别分配了 0.5、0.3 和 0.2 的权值.配合途径 2 和 4 的机制,系统最终收敛得到该供应链的可靠性评分.最终评分占比如图 10 中蓝色柱形(“事件发生前”)所示,其中低风险 27.68%,中等风险 70.54%,高风险 1.79%.

结合供应链统计数据,我们发现,参与人员的评分结果更加关注地理分布和开源协议类型这两个指标的数据,评分较低的软件包均归属于来自美国的组织或个人,同时采用的开源协议类型约束相对较大.这说明参与的评分的人员对政治风险,以及开源协议约束相对关注.进一步观察发现,活跃程度和可替代性等指标起到指导作用有限,是因为当前实验用例中,目标系统的供应链在这些指标的数据并无明显差异.

为验证系统关于途径 1 以及事件监控相关的能力,我们特意在系统评分稳定后,通过事件注入的方式,注入事件“GNU 组织限制中国企业使用 bash”.该事件中,“GNU”、“bash”均为系统用于事件提取的关键字,“限制”是负向影响事件的关键字,当系统捕获该事件后,系统对整个供应链的可靠性评分做出调整,结果如图 10 中橙色柱形(“事件发生后”)所示.从结果可以看出,由于负向事件的捕获,系统对供应链中“bash”软件包节点的可靠性评估做出相应的调整,同时,由于系统的更新机制,依赖“bash”的软件包,其可靠性评分也相应做出调整.

5 总结

Linux 发行版、Android 等大型复杂操作系统的核心功能和生态都建立在大量开源软件的基础上,这些开源软件构成了操作系统构建的供应链.事实证明开源软件也存在“断供”风险,如果供应链出现问题,那么操作系统构建和运行的基础不再可靠.因此,开源操作系统能够可持续、高质量发展的前提是保证开源软件供应链可靠.本文通过借鉴传统供应链方法,引入开源软件供应链管理体系,建立开源软件知识图谱,从多角度分析评估开源软件包及供应链的可靠性程度,更加全面的风险评价和消除建议,为大型复杂操作系统找出构建的关键环节,也暴露出生态的脆弱环节.本文提出的方法和系统也可以用于大数据、云平台等其他复杂基础设施软件,并为这些软件的定制化、智能化奠定供应链基础.总结来看,本文有以下几点贡献:

- 开源软件知识图谱:基于本文描述的构建方法,我们构建了一个面向开源软件的知识图谱,随着知识的积累,该图谱将维护海量开源软件供应关系及软件相关实体信息,为相关分析提供有力的数据支撑,同时也可以为大众了解开源软件世界开辟一条新的路径;
- 一种软件供应可靠性管理方法:面向开源协作模式,我们通过构建开源软件知识图谱,以知识为核心,结合传统的供应链管理方法,提出一组面向开源软件供应链的可靠性管理方法;
- 一种可靠性评估数据集的构建方法:不论学术界还是工业界,就我们所知,当前尚不存在大规模基准数据集可用于评估开源软件供应链的可靠性,基于本文提出的度量模型和评分方法,系统在使用过程中,将积累生成一个可以用于可靠性评估的有效数据集.

虽然软件工程管理和供应链管理两个研究领域已有大量的研究成果,但是作为交叉领域的软件供应链管理尚处于萌芽阶段,受限于已有的成果,本文提出的系统仍有很多可以改进的地方.在未来的工作中,我们希望能够以下几个方面做进一步的探索和研究:

- 改进事件提取方法:当前系统采用基于关键字的事件提取方法,在提取效率和准确性方面都存在一定限制,在未来的研究中,可以考虑引入更为先进的自然语言处理模型提升效果;
- 改进可靠性评估方法:当前系统在使用中,能够积累大量有效的标注数据,在未来的研究中,可以基于这些数据挖掘度量模型中不同维度对于可靠性评估的贡献值,进一步提升评估模型的准确性,提升可靠性评估的智能化程度;
- 扩展开源软件知识图谱:当前系统构建的知识图谱重点关注开源软件供应关系可靠性,在未来的研究中,可以考虑和已有的面向其他开源软件领域知识的知识图谱进行融合,使开源软件知识图谱具有更强的通用性.

References:

[1] Luo Bin, Ding Eryu., Liu Qin. Software Engineering and Computing: Technical Basis for Software Development. Volume 2: China Machine Press. 2012. 308-309

[2] Goševa-Popstojanova K, Trivedi KS. Architecture-based approach to reliability assessment of software systems. Performance Evaluation, 2001,45(2-3):179−204.

[3] Tanenbaum AS, Woodhull AS. Operating Systems Design and Implementation: Pearson Education; 2011. 4-5

[4] “Linux Operating Systems: Distributions” [EB/OL]. http://swift.siphos.be/linux_sea/whatislinux.html#idm3548300567744

[5] ZHOU M, ZHANG Y, TAN X. Software digital sociology[J]. SCIENTIA SINICA Informationis, 2019.

[6] Christopher M. Logistics and Supply Chain Management: Journal of Marketing; 1998. 4-5

[7] Ellram Lisa M. Supply Chain Management, Partnership, and the Shipper – Third Party Relationship. The International Journal of Logistics Management. 1990;1(2):1-10.

[8] David Simchi-Levi PK, Edith Simchi-Levi. Designing and Managing the Supply Chain: Concepts, Strategies, and Case Studies: McGraw Hill Professional; 2003. 354 p.

[9] Colicchia, C. & Strozzi, F., Supply chain risk management: a new methodology for a systematic literature review, Supply Chain Management: An International Journal, 2012,17(4):403-418.

[10] Trkman, P., De Oliveira, M.P.V. & Mccormack, K., Value-oriented supply chain risk management: you get what you expect, Industrial Management & Data Systems, 2016,116(5):1061-1083.

[11] Neiger, D., Rotaru, K. & Churilov, L., Supply chain risk identification with value-focused process engineering, Journal of Operations Management, 2009, 27(2):154-168.

[12] Tummala, R. & Schoenherr, T., Assessing and managing risks using the supply chain risk management process (SCRMP), Supply Chain Management: An International Journal, 2011,16(6): 474-483.

[13] Ho, W., Zheng, T., Yildiz, H. & Talluri, S., Supply chain risk management: a literature review, International Journal of Production Research, 2015, 53(16):5031-5069.

[14] Kern, D., Moser, R., Hartmann, E. & Moder, M., Supply risk management: model development and empirical analysis, International Journal of Physical Distribution & Logistics Management, 2012,42(1): 60-82.

[15] Cohen, M.A. & Kunreuther, H., Operations risk management: Overview of Paul Kleindorfer's contributions, Production and Operations Management, 2007, 16(5):525-541.

[16] Gaudenzi, B. & Borghesi, A., Managing risks in the supply chain using the AHP method, International Journal of Logistics Management, 2006,17(1):114-136.

[17] Fan Y, Stevenson M. A review of supply chain risk management: definition, theory, and research agenda[J]. International Journal of Physical Distribution & Logistics Management, 2018, 48(3): 205-230.

[18] Zsidisin, G.A., A grounded definition of supply risk, Journal of Purchasing and Supply Management, 2003,9(5): 217-224.

[19] Ehrlinger L, Wöß W. Towards a Definition of Knowledge Graphs[J]. SEMANTiCS (Posters, Demos, SuCCESS), 2016, 48.

[20] Singhal, Amit (May 16, 2012). "Introducing the Knowledge Graph: Things, Not Strings". Google Official Blog. Retrieved September 6, 2014.

[21] “What is a Knowledge graph?” [EB/OL]. https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/

[22] Peng Zhou, Guanyu Liang, Jiageng Yu, Jianmin Wang, Yanjun Wu and Chen Zhao. “Customized operating system for RISC-V fragmentation based on software supply chain”. 2019.

[23] Fielding RT, Taylor RN. Architectural styles and the design of network-based software architectures: University of California, Irvine Irvine; 2000.

[24] Thulasiraman, K.; Swamy, M. N. S. Graphs: Theory and Algorithms, John Wiley and Son, 1992. 118

[25] Kahn ABJCotA. Topological sorting of large networks. 1962,5(11):558-62.

附中文参考文献:

[1] 骆斌, 丁二玉, 刘钦. 软件工程与计算: 软件开发的技术基础. 卷二: 机械工业出版社. 2012. 308-309

[5] 周明辉, 张宇霞, 谭鑫. 软件数字社会学. 中国科学: 信息科学, 2019, 49: 1399{1411, doi: 10.1360/N112018-00319.

[22] 周鹏, 梁冠宇, 于佳耕, 王建民, 武延军, 赵琛. 基于软件供应链的应对 RISC-V 碎片化的操作系统定制平台. 2019.