计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(7):1−11 [doi: 10.15888/j.cnki.csa.007446]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于度量认证的协同集成可信部署技术①

龙　奔, 孙志宏

(江苏自动化研究所 电子设备事业部, 连云港 222002)
通讯作者: 龙　奔, E-mail: 15339289746@163.com

摘　要: 协同开发技术是进行控制系统软件开发的重要方法, 其中的软件集成部署是协同开发技术的重要一环. 随着软件规模越来越大, 复杂性越来越高, 现有的软件集成部署方法由于缺乏标准统一的部署规范和安全机制, 所以会存在集成部署效率低下和安全性较差的问题. 针对上述问题, 提出了一种基于度量认证的协同集成部署方法, 解决在分布式协同开发环境下集成部署工作带来的安全和效率问题. 本文的主要思想是利用一种安全关联协议, 在软件部署双方之间建立可信的安全传输通道, 继而根据统一的打包部署规范, 完成软件的集成和部署工作. 理论分析和实验结果表明, 该方案统一了软件打包部署规范, 优化了完整性度量算法, 减少了认证和软件部署的时间, 并且由于部署双方在准备阶段已建立了安全关联, 所以提高了整个软件部署过程的安全性. 本文的研究内容能够在分布式开发环境下完成安全且高效的软件集成部署工作.

关键词: 分布式开发环境; 集成部署; 协同开发; 安全可信; 完整性度量

引用格式: 龙奔,孙志宏.基于度量认证的协同集成可信部署技术.计算机系统应用,2020,29(7):1–11. http://www.c-s-a.org.cn/1003-3254/7446.html

Collaborative and Integrated Trusted Deployment Technology Based on Metric Authentication

LONG Ben, SUN Zhi-Hong

(Department of Electronic Equipment, Jiangsu Automation Research Institute, Lianyungang 222002, China)

Abstract: Collaborative development technology is an important method for the development of control system software, and the integrated deployment of software is an important part of collaborative development technology. With the increasing scale and complexity of software, the existing methods of software integration deployment have the problems of low efficiency and low security due to the lack of standardized and unified deployment specifications and security mechanisms. Aiming at the above problems, a collaborative integration deployment method based on metric authentication is proposed to solve the security and efficiency problems brought by integrated deployment work in distributed collaborative development environment. The main idea of this study is to use a security association protocol to establish a trusted secure transmission channel between the software deployment parties, and then complete the software integration and deployment according to the unified package deployment specification. Theoretical analysis and experimental results show that the scheme unifies the software package deployment specification, optimizes the integrity measurement algorithm, and reduces the time for software authentication and deployment. Since the two parties establish security associations, the security of the software deployment process can be improved. The research content of this study can complete safe and efficient software integration deployment in a distributed development environment.

Key words: distributed development environment; integration deployment; collaborative development; trusted; integrity measurement

① 基金项目: 国防基础科研计划 (JCKY2017207C035)
Foundation item: National Defense Basic Research Program (JCKY2017207C035)
收稿时间: 2019-11-05; 修改时间: 2019-11-28; 采用时间: 2019-12-11; csa 在线出版时间: 2020-07-03

1 引言

随着信息技术的快速发展, 大型复杂软件的功能需求不断增加, 规模也越来越大. 在开发环境的构建过程中, 集成部署过程是重要的一环, 其部署效率和安全性也逐渐引起了开发者的重视.

目前, 大型复杂软件的开发工作主要采用分布式协同开发的方式实现. 由各家单位根据自身专业所长, 通过人员沟通、集中试验、集成联调等方式展开协同开发, 并结合过程管理和协议通讯, 实现软件的集成部署. 然而, 实际工作环境对于大型复杂软件的可靠性和安全性存在着极高的要求, 尤其在航空航天、深海探测以及核工业安全等领域, 如果出现质量问题或者安全问题, 将会造成灾难性的后果. 对于开发者而言, 随着软件规模的增大, 构建高效的软件开发环境不仅能够针对软件的开发过程进行有效的控制, 而且能够提高开发过程的规范化, 帮助开发团队开发出高质量的软件.

构建高效的软件开发环境势必需要部署过程, 因此, 部署相关的问题也随之而来. 该类问题主要体现在两个方面:

(1) 传统的软件开发环境需要实施人员进行手动部署, 该方式无法保证软件整体的部署质量. 同时, 在大规模部署时, 由于相互之间通信对象的增多, 测试过程中未暴露的缺陷可能会引起大量的异常, 严重地影响了软件开发的效率.

(2) 大型复杂软件的开发任务逐步呈现高密度态势, 往往存在开发任务重、时间紧、周期长的特点, 同时对软件的正确性、健壮性和安全性的要求越来越高, 这就对集成部署工作提出了更加安全高效的要求.

文献[1]研究了目前主流的几种软件部署方法, 其中, 软件部署工具拥有较为统一的部署流程, 可实现软件的快速部署, 并且满足开发环境多样化的需求. 目前, 国内外已存在不少软件部署工具, 按照分发和部署策略划分, 可归为以下 3 类[2]: 包部署工具、通用产品部署工具和特定供应商产品部署器. 其中, 包部署工具一般适用于开源系统, 基本没有安全措施, 无法保证集成部署过程的安全性. 文献[3]提到的 InstallShield 是一种通用产品部署器, 在软件安装过程中支持多种分发方式, 并且可提供全程的图形功能. 但是, InstallShield 存在封包格式不公开和占用资源较大等问题, 并且安全机制不完善, 无法满足安全需求. 特定供应商产品部署器提供的功能有限, 一般只适用于特定产品的部署更新. 该类部署工具存在的一些身份验证和完整性验证等安全机制, 但是仅针对特定的产品, 不具备通用性.

在软件部署策略研究方面, 文献[4]提出了一种通用的组件部署模型, 该模型基于人工智能中的规划算法实现组件的部署计划. 文献[5]综合多种影响软件服务质量的因素, 并针对软件部署策略提出了一种通用的多目标优化算法, 但是该算法需要借助其他工具支持. 文献[6]提出了一种软件部署优化的策略, 该方法利用 UML 模型对系统进行建模处理, 可有效地提高系统的性能和可靠性. 虽然软件部署策略方面的研究较为丰富, 但是以上研究方案未针对部署环境的复杂性和安全性作出讨论, 其部署效率和安全性未得到验证.

文献[7]提出的自治计算架构是当前软件管理方面的热点, 该架构提供了一种可实现软件自我管理的机制, 可在尽可能减少人为干预的情况下, 实现系统环境的自主管理. 然而, 由于存在架构规格描述、程序传递、事件记录等技术上的障碍, 自治计算架构还无法在系统环境下实现软件的自我部署功能.

通过上述的研究分析可知, 目前在大型复杂软件开发环境下, 还缺乏高效且安全的方案来解决软件集成部署的问题. 针对该问题, 本文尝试给出一种基于度量认证的协同可信部署 (Cooperative Trusted Deployment based on Metric Authentication, CTDMA) 方案, 其主要思想是: 首先通过一种安全关联协议在软件部署双方之间建立可信的安全传输通道, 然后宿主机会根据统一的打包部署规范, 将部署包通过安全传输通道传送给相应的目标机, 最后目标机经过解析, 并利用一种改进型的完整性度量算法对部署包进行完整性和一致性确认, 确认通过后, 完成集成部署工作. 该方案通过建立统一的打包部署规范, 可保证整个部署过程可复制可预期, 通过优化完整性度量算法, 能够有效地提高软件集成部署的效率. 此外, 安全关联协议的应用以及部署包的度量认证机制可极大地提高数据传输的可信性.

2 相关技术研究

2.1 安全关联协议

Diffie 和 Hellman[8]提出了一种密钥交换算法, 该算法能够帮助通信双方安全地协商出共享密钥, 以便在后续通信过程中使用该密钥对信息进行加解密.

在数学问题上, 计算离散对数的困难性是保证

Diffie-Hellman 算法有效性的基础, 在进行密钥交换前, 通信双方首先需要确定 q 和 α, 其中, α 和 q 是两个公开的整数 (该整数也可由通信双方在密钥交换过程中协商得出), α 是 q 的本原根, 其幂可产生 1 到 q 之间的所有整数.

在分布式协同开发环境下, 利用 Diffie-Hellman 算法进行密钥交换, 可保证宿主机和目标机之间通信的安全性, 如图 1 所示, 其密钥协商的过程如下.

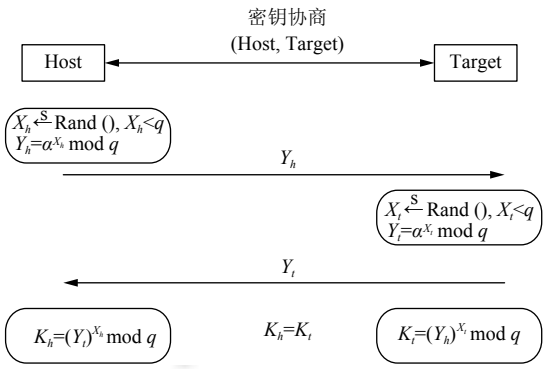

图 1 宿主机和目标机的密钥协商过程

(1) 分布式开发环境下的各计算节点彼此之间可进行身份认证. 宿主机随机选择一个整数 X_h ($X_h<q$) 作为私钥, 计算临时公钥 Y_h, 其中, $Y_h=\alpha^{X_h} \bmod q$, 计算完成后, 将临时公钥 Y_h 发送给目标机;

(2) 目标机接收到信息后, 随机选择一个整数 X_t ($X_t<q$) 作为私钥, 计算临时公钥 Y_t, 其中, $Y_t=\alpha^{X_t} \bmod q$, 计算完成后, 将临时公钥 Y_t 发送给宿主机;

(3) 通信双方分别接收到对方的临时公钥后, 计算共享密钥, 其中, 宿主机端的共享密钥为 K_h, $K_h=(Y_t)^{X_h} \bmod q$, 目标机端的共享密钥为 $K_t=(Y_h)^{X_t} \bmod q$, $K_h=K_t$, 至此, 通信双方完成了密钥交换工作.

在计算共享密钥的过程中, 虽然宿主机和目标机使用的部分参数不同, 但是其计算结果相同, 根据模运算定律, 其证明如下:

$$\begin{aligned} K_h &= (Y_t)^{X_h} \bmod q = (\alpha^{X_t} \bmod q)^{X_h} \bmod q \\ &= (\alpha^{X_t})^{X_h} \bmod q = (\alpha^{X_h} \bmod q)^{X_t} \bmod q \\ &= (Y_h)^{X_t} \bmod q = K_t \end{aligned} \tag{1}$$

密钥交换结束之后, 通信双方可利用共享密钥加密通信信息, 建立一条安全的信息传输通道, 每一次会话结束后, 通信双方需要按照以上过程, 重新协商共享密钥.

2.2 可部署软件建模方法研究

(1) 现有软件描述语言分析

现有的可部署软件建模方法有开放式软件描述 (Open Software Description, OSD)、信息管理格式 (Management Information Format, MIF)、可部署软件描述 (Deployable Software Description, DSD) 3 种. 通过分析, 这 3 种部署软件建模方法的优缺点如表 1 所示.

表 1 部署软件建模方法总结

部署软件建模方法	可扩展性	复杂度	对软件部署生命周期的支持
OSD	部分可扩展	简单	部分支持
MIF	部分可扩展	较复杂	完全支持
DSD	完全可扩展	较复杂	完全支持

通过上述比较可知, 可部署软件描述 (DSD) 对于部署软件规范化描述的可扩展性和支持性较好, 本文提出的方案可采用该种方式.

(2) 可部署软件规范化描述

DSD 将待部署软件和目标机配置信息建模成嵌套的属性集合, 其规范化描述可以由以下 4 方面组成:

1) 部署软件的信息描述

部署软件信息包括软件基本信息和软件特征信息两个部分, 其中, XML 文件格式的属性<Family>描述待部署软件的序列, <Id>描述系统标识, 均位于软件信息描述的外侧, 内部嵌套表示其他信息的标签.

软件基本信息: 软件名称<Name>、软件描述<Description>、软件版本号<Version>、软件生产商<Producter>等.

软件特征信息: 软件特征码<Signature>, 该部分可记录部署包完整性度量值, 用于目标机进行解析验证.

2) 目标机开发环境约束描述

目标机的开发环境包含目标机的系统和硬件两个部分, 在打包部署过程中, 为保证待部署软件在目标环境下安全稳定地运行, 需要满足目标机端的软硬件环境需求.

<Assertions>属性位于描述的最外侧, 表示目标机开发环境的硬件和系统的约束集合, 每一对<Assertions>表示一类信息. 该部分包括依赖条件<Condition>、软件描述<Description>等.

其中, <Condition>表示目标机当前的开发环境, 包括操作系统 (OS)、硬盘大小 (Disksize)、内存大小 (Memory) 等.

3) 软件依赖约束描述

<Assertions>属性位于描述的最外侧, 表示目标机开发环境的硬件和系统的约束集合, 每一对<Assertions>表示一类信息. 该部分包括依赖条件<Condition>、软件描述<Description>等.

该部分包括依赖条件<Condition>、软件序列<Family>、软件描述<Description>、解决方案<Resolution>等部分组成, 其中, 每一个<Family>属性表示一个依赖的软件, <Condition>表示目标机环境, 即目标机内是否存在该软件. <Resolution>表示依赖软件需要执行的动作, 如果<Condition>显示该软件尚未安装, 则<Resolution>应取值 Install, 对该软件执行安装操作.

4) 软件部署行为描述

软件部署行为是指在软件部署过程中, 操作系统需要执行的动作. 一般情况下, 软件部署过程中, 系统需要预先关闭一系列进程, 部署结束后, 系统需要重新启动操作系统, 这些均是软件部署行为. 其外部属性为<Activities>, 每一对<Activities>表示一种需要执行的动作, 包含: 动作名称<Name>、动作内容<Action>、判别时间<When>、动作描述<Description>等内容, 其中, <Action>表示需要执行的动作, 如 Restart, 表示系统重启. <When>表示该动作执行的时间, 如 After Deployment, 表示系统重启动作在部署完成后进行.

(3) 完整性度量技术研究

在软件的集成部署过程中, 部署包的完整性度量是影响部署效率的主要因素, 并且部署包越大, 完整性度量计算的耗时就越高, 对完整性度量算法进行优化, 是提高软件集成部署效率的有效方法.

由于散列算法具有不可逆和抗碰撞攻击的能力, 并且输出长度固定、安全性好, 一般采用散列算法作为完整性度量算法. 散列算法主要由预处理过程和散列值计算两个部分组成, 其中, 预处理过程可分为 3 部分: 消息填充、分割已填充消息以及设置散列值计算初始值.

根据组成特点可知, 散列算法存在可并行性, 利用线程级并行和存储级并行技术对算法进行改进, 可提高算法的计算效率. 其改进思想是: 根据散列度量算法的特点, 利用线程级的并行技术将处理任务进行划分, 针对待度量的数据, 将其分成均匀的数据块并交给不同的线程进行流水执行, 通过存储级并行技术实现访问隐藏. 改进的并行算法流程如图 2 所示.

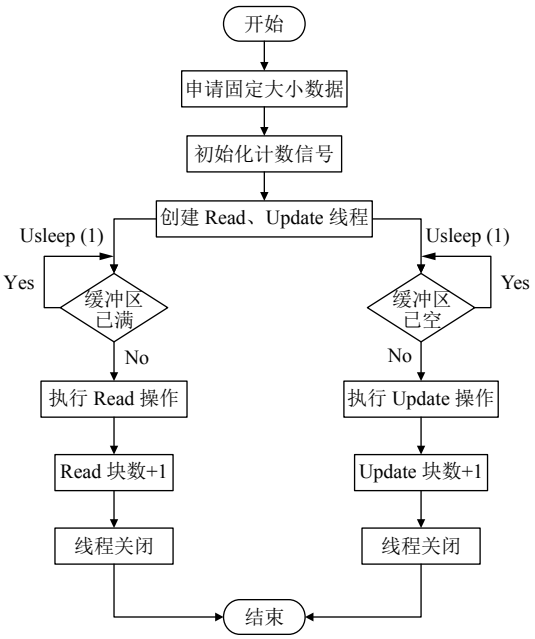

图 2 并行算法流程图

改进后的算法可分为 Read 线程和 Update 线程两类进行并行处理, 其中, Read 线程通过数据预读取[9]的方式读取划分好的数据块, Update 线程则按顺序依次将读取的数据块进行运算处理. 其执行流程如下:

(1) 设置一个共享缓冲区, 其长度为 n, 大小为 $n \times m$ 个字节, 其中, m 值的大小决定并行计算中流水粒度的大小;

(2) 设计一个计数信号, 并初始化为 0, 利用该计数信号保证两个线程之间的同步性;

(3) 创建两个线程, 分别为 Read 线程和 Update 线程, Read 线程以数据预读取的方式将均匀划分的待度量数据块读入共享缓冲区中, Update 线程针对共享缓冲区内的数据进行顺序运算处理;

(4) 随着 Read 线程执行写入缓冲区操作, 计数信号不断累加, 如果缓冲区已满, 则 Read 线程停止写入;

(5) 随着 Update 线程对缓冲区内的数据执行顺序运算处理, 计数信号不断递减, 当缓冲区为空时, 则 Read 线程停止等待, 两个线程之间可通过判断计数信号实现同步.

此外, 改进后的完整性度量算法利用设置私有计数器的方式可有效地减少对临界区的访问次数, 同时也可以避免锁机制引起的死锁事件. 在线程同步过程中, Usleep(1) 函数可减少同步开销, 隐藏访问操作并最终得到度量结果.

3 CTDMA 方案设计

3.1 可部署软件建模方法研究

随着软件规模越来越大, 复杂性也越来越高, 开发者希望优化软件集成部署方法, 提高软件的开发效率, 并保证部署过程的安全性. 鉴于此, 本文的研究思路是基于 Diffie-Hellman 密钥交换算法协商出共享密钥, 在宿主机和目标机端建立一条安全传输通道, 从而在开放式的网络环境下保证部署信息传输的安全性. 在集成部署方面, 本文利用可部署软件描述 (DSD) 语言为待部署软件建立统一的打包部署规范, 可帮助软件进行自动集成部署, 同时利用改进的完整性度量算法, 可有效地提高部署双方的计算效率.

3.2 CTDMA 方案系统模型

针对分布式协同开发环境的特点, CTDMA 方案的系统模型分为两个部分: 软件协同部署服务器 (Collaborative Deployment Server, CDS) 和软件协同部署客户端 (Collaborative Deployment Client, CDC). CTDMA 方案系统模型交互示意图如图 3 所示.

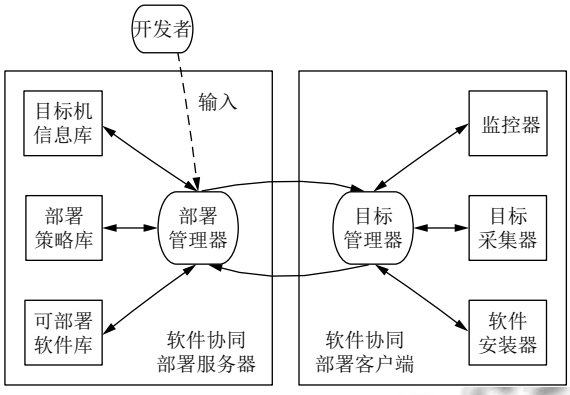

图 3 CTDMA 方案模型交互示意图

软件协同部署服务器 (CDS): 在分布式开发环境下, 软件协同部署服务器主要安装在宿主机端, 其功能主要分为以下 4 部分:

(1) 根据开发者请求, 与待部署的目标机建立安全关联;

(2) 管理目标机信息库、可部署软件库以及部署策略库;

(3) 根据部署策略库提供的规范化描述方法将待部署软件生成部署包和部署描述文件;

(4) 将部署包和部署描述文件发送给目标机端并根据目标机端的部署情况作出响应.

软件协同部署服务器包含以下几个模块:

目标机信息库 (Client Information Base, CIB): 该模块主要存储宿主机所在分布式开发环境下所有目标机节点的软硬件信息, 可根据实际情况及时作出更新.

可部署软件库 (Deployed Software Base, DSB): 该模块存储开发环境构建所需的软件安装包、软件依赖文件以及软件描述信息.

部署策略库 (Policy Base, PB): 该模块可提供软件部署策略, 帮助待部署软件生成软件描述文件, 主要依据包括待部署软件信息、目标机端的软硬件环境以及规范化描述方法.

部署管理器 (Deployment Manager, DM): 该模块是软件协同部署服务器的核心, 负责接收开发者的软件部署请求, 与目标机端建立安全关联, 并调用其他模块完成部署包的生成、发送和安装工作.

软件协同部署客户端 (CDC): 在分布式开发环境下, 软件协同部署客户端主要安装在目标机端, 其功能主要分为以下 4 部分:

(1) 验证宿主机身份, 与宿主机端建立安全关联;

(2) 获取目标机端的软硬件信息, 发送给协同部署服务器;

(3) 接收宿主机端发送的部署包和软件描述文件;

(4) 验证部署包的可信性, 并根据软件描述文件安装相应软件协同部署客户端包含以下几个模块:

监控器 (Monitor): 该模块的主要功能是监控部署包的安装情况并将部署结果返回至服务器, 以便服务器根据安装情况作出响应.

目标采集器 (Target Collector, TC): 该模块可获取目标机端当前的软硬件信息, 并将该信息返回至服务器端.

软件安装器 (Software Installers, SI): 该模块可根据软件描述文件提供的依赖关系安装部署包内的软件.

目标管理器 (Target Manager, TM): 该模块是软件协同部署客户端的核心, 负责验证宿主机的身份, 与宿主建立安全关联, 并调用其他模块完成部署包的解析、验证和安装工作.

3.3 CTDMA 方案详细设计流程

CTDMA 方案的详细设计流程如下:

流程 1. 协同部署服务器接收开发者请求, 与待部署的目标机客户端建立安全关联.

在分布式开发环境下, 协同部署服务器 (CDS) 和

协同部署客户端 (CDC) 的身份认证工作已完成, 并且已知公开的素数 q 及其本原根 α. 根据 Diffie-Hellman 密钥交换算法, CDS 端与 CDC 端需要协商共享密钥, 建立一条安全传输通道.

具体操作如下:

(1) CDS 端选择一个随机整数 X_s 作为自身私钥, 计算临时公钥 Y_s, 并发送给 CDC 端, 表示如下:

$$\begin{aligned} &X_s \xleftarrow{\$} \text{Rand}(), X_s < q \\ &Y_s = \alpha^{X_s} \bmod q \end{aligned} \tag{2}$$

(2) CDC 端随机选择一个随机整数 X_c 作为自身私钥, 计算临时公钥 Y_c, 并发送给 CDS 端, 表示如下:

$$\begin{aligned} &X_c \xleftarrow{\$} \text{Rand}(), X_c < q \\ &Y_c = \alpha^{X_c} \bmod q \end{aligned} \tag{3}$$

(3) 通信双方接收到对方的临时公钥后, 计算共享密钥, 其中, CDS 的共享密钥为 K_s, 目标机端的共享密钥为 K_c, K_s=K_c, 至此, 通信双方完成了密钥交换工作, 其表示方法和证明如下:

$$\begin{aligned} &K_s = (Y_c)^{X_s} \bmod q \\ &K_c = (Y_s)^{X_c} \bmod q \\ &K_s = (Y_t)^{X_s} \bmod q = (\alpha^{X_c} \bmod q)^{X_s} \bmod q \\ &\quad = (\alpha^{X_c})^{X_s} \bmod q = (\alpha^{X_s} \bmod q)^{X_c} \bmod q \\ &\quad = (Y_s)^{X_c} \bmod q = K_c \end{aligned} \tag{4}$$

至此, 通信双方之间的密钥交换工作结束, CDS 端和 CDC 端可利用共享密钥 K 对部署信息进行加解密, 以保证通信信息的安全性.

流程 2. 验证目标机的平台信息.

在进行集成部署工作之前, CDS 端首先需要保证 CIB 内的信息与目标机环境相一致.

具体做法如下:

(1) TM 调用 TC 收集目标机端当前的软硬件信息, 设为 m, 使用共享密钥 K 加密该信息, 生成密文信息 c, 并发送给 DM.

$$c \leftarrow E(m, K) \tag{5}$$

(2) DM 接收到密文信息 c 后, 利用共享密钥 K 解密, 将解密信息与 CIB 内存储的信息进行对比.

$$m \leftarrow D(c, K) \tag{6}$$

(3) 如果信息一致, DM 则执行后续部署, 如果不一致, DM 需要对 CIB 进行更新, 以保持部署双方信息的一致性.

流程 3. CDS 端生成部署包及软件描述文件.

CDS 端需要针对待部署软件的特点和目标机的软硬件环境, 按照可部署软件描述方法生成部署包和软件描述文件.

具体做法如下:

(1) DM 根据开发者的部署请求查询 DSB, 如果 DSB 内存在待部署软件, 则继续执行, 否则, 停止部署工作;

(2) DM 查询 CIB 内的目标机信息和 DSB 内的软件配置信息, 并根据 PB 提供的软件部署策略, 写入软件约束依赖描述和平台信息约束描述, 生成软件部署包和软件描述文件;

(3) DM 利用改进后的完整性度量算法对部署包进行完整性度量计算, 并将度量结果写入软件描述文件的相应属性标签内;

(4) DM 利用共享密钥 K 加密软件描述文件 M, 并将加密文件 C 和部署包 P 和发送给 TM.

$$C \leftarrow E(M, K) \tag{7}$$

流程 4. CDC 端验证加密文件, 执行部署过程.

CDC 端接收到 CDS 端发送的加密文件后, 需要执行解密文件、验证部署包的一致性、安装部署包等一系列操作, 具体做法如下:

(1) TM 接收到部署包 P 和加密文件后, 使用共享密钥 K 解密该文件, 得到软件描述文件 M:

$$M \leftarrow D(C, K) \tag{8}$$

(2) TM 利用改进后的完整性度量算法计算部署包的完整性度量值, 并与软件描述文件中的记录进行对比, 如果一致, 则进行后续操作, 否则停止部署过程, Monitor 向 CDS 端反馈错误信息;

(3) TM 解析部署包, 验证待部署软件版本号是否与软件描述文件相一致, 如果一致, 则进行后续操作, 否则停止部署过程, Monitor 模块向 CDS 端反馈错误信息;

(4) TM 解析软件描述文件, 验证部署包内是否存在完整的依赖文件, 如果存在, 则进行后续操作, 否则停止部署过程, Monitor 模块向 CDS 端反馈错误信息;

(5) TM 根据软件依赖约束描述和平台信息约束描述, 调用 SI 模块, 进行软件安装工作, Monitor 模块监控整个部署过程, 并将部署结果通过 CDC 端发送给 CDS 端.

至此, 一轮完整的软件部署过程结束, 其流程如图 4 所示. 在进行下一次部署工作之前, 宿主机和目标机需要重新进行密钥交换, 协商出新的共享密钥.

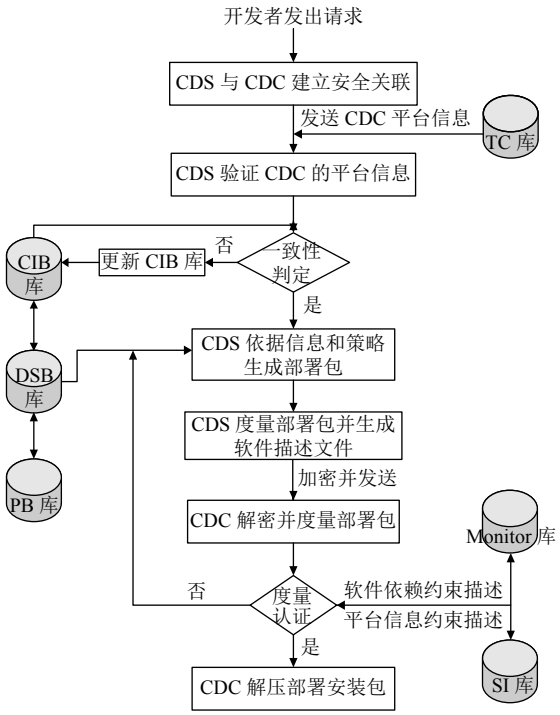

图 4 CTDMA 方案设计流程图

4 安全性证明和性能分析

4.1 安全性证明

宿主机和目标机暴露在开放的网络环境下, 其通信信息的安全性无法得到保证. CTDMA 方案采用 Diffie-Hellman 密钥交换算法, 在每一轮会话过程中, 共享密钥 K 是保证通信安全性的关键.

假定在宿主机 H 和目标机 T 所处的网络环境中存在一个敌手 A, A 观察到 H 和 T 进行了密钥交换过程, 并且期望得到它们协商出的共享密钥 K. 由于 X_h 和 X_t 是会话双方随机选择并且私有的, 因此, A 只能通过公开的参数 q 和 α (q 和 α 不一定需要公开, 可通过修改算法由通信双方协商决定), 以及 H 和 T 的临时公钥 Y_h 和 Y_t 进行攻击. 由于 $K=(Y_t)^{X_h} \bmod q$, 如果 A 想获得共享密钥 K, 首先需要计算 H 的私钥 X_h. 由于 $Y_h=\alpha^{X_h} \bmod q$, $X_h=d\log_{q,a}(Y_h)$, A 必须通过离散对数计算才能求得私钥 X_h, 继而求得共享密钥. 然而, Diffie-Hellman 密钥交换算法的安全性建立在以下事实之上: 求素数的模素数幂运算相对容易, 而计算离散对数却非常困难; 对于大素数, 求离散对数被认为是不可行的. 因此, A 无法通过求离散对数获得共享密钥 K.

在集成部署过程中, 散列算法的安全性较好, 部署包数据发生微小的改变即能引起完整性度量值发生明显的变化. 因此采用散列算法可保证传输过程中部署包的完整性和一致性. 此外, 在集成部署之前, 目标机端还需要验证软件版本号、依赖软件的完整性以及平台信息, 进一步地提高了集成部署过程的可信性.

CTDMA 方案与传统的集成部署方式的安全性对比如表 2 所示.

表 2 安全性对比

对比项目	CTDMA 方案	传统集成部署方案
传输安全性	建立安全关联	无
部署包认证	存在解析对比等机制	一般缺乏认证机制
完整性认证部署策略	改进的完整性度量算法规范化的软件描述	散列算法人工部署或其他部署策略

4.2 性能分析

(1) 理论分析

1) 密钥协商及通信信息加解密分析

在 CTDMA 方案中, Diffie-Hellman 密钥交换算法的计算周期主要分为两个阶段, 一是共享密钥的协商阶段, 二是部署双方利用共享密钥对通信信息进行加解密的阶段. 在实际处理过程中, 这两个阶段均会带来一定的计算开销.

在密钥协商阶段, CDS 端和 CDC 端需要计算各自的临时公钥和共享密钥, 由通用的运算公式 $K=\alpha^x \bmod q\ (0 \le x \le q-1)$ 可知: 在此过程中, 最大的计算开销来源于大整数的幂运算和模运算.

在通信信息加解密阶段, AES 是一种对称分组密码算法, CDS 端和 CDC 端可利用前一阶段协商的共享密钥作为对称密钥, 通过 AES 密码算法对通信信息进行加解密. 在有限域 $GF(2^8)$ 上的多项式模运算是 AES 密码算法重要的数学基础, 其中涉及到的运算包括字节替代正逆变换、列混淆正逆变换和轮密钥扩展等, 整套算法设计起来较为复杂. 在本方案中, 共享密钥作为加解密的对称密钥, 可扩展为 128 位, 对应的轮密钥扩展为 10 轮.

2) 完整性度量算法分析

CTDMA 方案对于完整性度量散列算法的优化主要在于将原算法的计算方式由串行计算改为了并行计算, 可通过加速比衡量并行优化带来的性能变化.

根据 Amdahl 定律, 加速比的计算为:

$$S(p,f)=\frac{p}{1+(p-1)f} \tag{9}$$

其中, p 为运算的线程数, f 为串行计算在整个计算过程中占的比重.

设系统 I/O 速率为 s_r, s_r=$f_s(m)$, 计算速率为 s_u, 考虑算法并行部分访存无依赖, 由式 (1) 可推得并行优化的完整性度量算法 (以下简称为并行算法) 加速比为:

考虑到通信、同步以及空闲等待时间等因素, 设 T_0 为并行算法的额外开销时间, T 为串行算法执行总时间, 并行算法的实际加速比为:

$$S(p_r,f)=\frac{p_r+1}{1+p_r\cdot f_s(m)/(s_u+f_s(m))} \tag{10}$$

在并行算法中, P_r 为 Read 线程数, P_u 为 Update 线程数.

假设部署包大小为 S, 并行算法将其分块处理后, 单块文件大小为 m, Read 线程耗时记为 T_r, Update 线程耗时记为 T_u, 串行算法总时间记为 T, 并行算法总时间记为 T_p. 并行算法的额外开销记为 T_0.

计算公式如下:

$$T_r=f_r(m,s_r)=m/s_r=m/f_s(m) \tag{11}$$

$$T_u=f_u(m,s_u)=m/s_u \tag{12}$$

$$T_0=T_p-\frac{S}{s_u} \tag{13}$$

3) 自动集成部署能力分析

在 CTDMA 方案中, 自动集成部署阶段的一个周期包括软件描述文件生成、软件部署包集成以及部署包解压安装几个部分. 其中软件部署包内又包含待部署软件、依赖的软件和库以及相应的配置文件. 因此, 相对于单一的待部署软件来讲, 其依赖的软件和库越多, 软件描述信息的结构就越复杂, 生成的软件部署包就越大, 部署过程耗时就越长.

其中, 软件描述文件生成和软件部署包集成部分均为前期工作, 对于同一批次的部署任务来讲, 前期工作只需要进行一次, 便可以应用在多个目标机客户端上, 同时, 协同部署客户端接收到服务器端发送的软件描述文件和软件部署包后, 便可同时进行软件部署工作. 因此, 待部署的目标机数量越多, 整个部署任务的耗时就越短.

自动集成部署适用于大规模、复杂情况下的集成部署任务.

(2) 实验结果

1) 实验平台

本文的实验方案主要针对 Diffie-Hellman 密钥交换算法、AES 加解密算法、改进的完整性度量算法以及自动集成部署能力测试 4 部分, 根据方案的特点设计实验用例, 并利用开源的安全加密软件库 Openssl 实现以上密码算法的功能, 并度量其计算性能. 实验平台的软硬件信息如表 3 所示.

表 3 实验环境

具体配置		运算服务器
硬件平台	服务器型号	浪潮英信 NF5280MA
	CPU	Intel Xeon CPU E5-2660 v4 @2.00 Hz
	Memory	16 GB
软件平台	操作系统	Centos
	Linux 版本	Linux version 2.6.32
	完整性度量算法	SHA-1 算法

2) 测试方案

① Diffie-Hellman 密钥交换算法的实现和性能分析

本方案利用 Openssl 提供的接口实现了 Diffie-Hellman 密钥交换算法的基本功能, 包括部署双方密钥参数集合的生成、公私钥对的生成以及共享密钥的生成, 其中, 按照 AES 密码算法的标准 (在部署双方安全通信的过程中, 共享密钥将作为 AES 算法的私钥使用), 将共享密钥扩展为 128 位.

在实验平台下, 通过多次测量, 每个阶段的计算开销如表 4 所示.

表 4 Diffie-Hellman 密钥交换算法各阶段耗时

功能阶段	时间开销 (μs)
密钥参数集合生成	<1
公私钥对生成	258
共享密钥生成	169
全过程耗时	427

通过表 4 可知, 密钥参数的生成阶段耗时极小, 几乎可以忽略不计, 主要的计算开销在于公私钥对和共享密钥的生成部分, 该部分需要进行幂运算和模运算, 因此存在比较复杂的计算量, 但是考虑到后续的实验数据, Diffie-Hellman 密钥交换算法的计算开销只占整个部署过程的一小部分.

② AES 密钥算法的实现和性能分析

在本方案中, AES 密钥算法主要用于在部署双方之间对通信信息进行加解密计算. 通信信息主要包括软件部署包和软件描述文件, 由于软件描述文件和软件部署包之间存在一致性, 只需要对软件描述文件进行加解密即可保证通信信息的安全性, 并且也可以有效地提高加解密过程的计算效率. Openssl 提供的加解密接口 AES_set_encrypt_key()、AES_set_decrypt_key() 和 AES_cbc_encrypt() 可实现 AES 密钥算法的基本功能.

在测试用例的设计过程中, 设置密钥长度为 128 位 (16 字节), 相应的密钥扩展轮数为 10 轮. 由于明文的分组长度为固定的 128 位, 所以软件描述文件中的数据越长, 分组就越多, 相应的加解密复杂度就越高, 耗时就越长.

本方案的测试用例以软件描述文件的数据长度为测试变量, 验证数据长度大小对于通信信息加解密的计算开销的影响. 测试流程如下:

a. 生成不同长度的软件描述信息, 依次写入不同的文件中;

b. 测试用例对不同的软件描述文件进行加密操作, 然后将密文信息依次写入加密文件, 统计执行时间;

c. 测试用例对不同的密文文件进行解密操作, 然后依次还原软件描述文件, 统计执行时间.

通过多次测量, 计算开销如表 5 所示.

表 5 AES 算法下不同大小的文件加解密耗时比较

加密算法	数据大小	加密耗时 (μs)	解密耗时 (μs)
AES-128	16 Byte	2	2
AES-128	128 Byte	11	10
AES-128	1 KB	85	78
AES-128	8 KB	634	629
AES-128	16 KB	5298	5312

通过表 5 可知:

a. 在相同的数据长度下, AES 测试用例的加解密耗时几乎相同, 这是因为 AES 密钥算法的加解密过程几乎是相同的, 解密过程是加密过程的逆变换, 测量数据符合实际情况;

b. 随着测试数据长度的增加, 加解密的计算开销几乎呈指数型增长, 这是由于明文和密文分组增多, 加解密过程中的计算量增大导致的.

在实际应用中, 由于软件部署包的复杂性不同, 描述文件的大小范围一般固定在 1 KB 到 10 KB 之间, 在这个区间内, 加解密的计算耗时主要控制在 1000 μs 以下, 与 Diffie-Hellman 密钥交换算法的测试数据相差不大, 仅占整个部署过程的一小部分.

③ 完整性度量算法的改进及性能分析

本方案采用 SHA-1 算法为实验对象 (SHA-1 算法为一种常用的完整性度量算法), 原算法记为算法 1, 采用串行处理的方式进行计算, 改进算法记为算法 2, 采用并行方式进行计算, 其实验平台软硬件信息如表 6 所示.

表 6 算法 1 和算法 2 耗时比较 (单位: μs)

S(MB)	10	20	30	40	50	60	70	80	90	100
串行计算耗时	58 537	115 313	181 456	233 564	298 243	351 582	407 712	467 338	523 281	597 413
并行计算耗时	57 517	114 531	173 745	225 072	283 818	330 495	391 863	433 820	500 271	561 525
额外开销	6317	12 131	20 145	20 272	27 828	23 295	33 463	24 220	39 471	49 525

算法 2 通过对算法 1 源码进行并行化处理以达到优化计算效率的目的, 分为访存阶段和度量运算阶段两个部分, 访存阶段使用 Read 线程进行处理, 计算阶段使用 Update 线程进行处理. 其中, m 值的大小与同步开销存在着直接的联系, 可影响并行计算的效率. 由文献[10]可得结论: 当 m 值的大小与文件系统 block 大小相同时, 程序的读写效率最高, 串行计算的耗时最小. 因此, 根据实验环境, m 值大小应选择 4 KB, 这样既能占用较小的缓存, 也能保证算法的性能.

在测试环境下, 本方案主要从以下两个方面进行性能测试:

a. 在算法 2 中, 以部署包大小为变量, 测试系统访存速率和度量算法运算速率之间的变化, 并计算加速比;

b. 以部署包大小为变量, 测试算法 1 和算法 2 度量耗时的变化, 并计算加速比.

在算法 2 中, 图 5 显示了针对不同大小的部署包, 访存速率和算法运算速率的变化情况, 根据式 (4)、式 (5) 以及表中数据可知, s_u=2.048×108 (Byte/s), s_r=$f(m)$ 2.048×109(Byte/s). 根据式 (2) 可知, 其理论加速比为 S≈1.05.

结论:

a. 在算法 2 中, 访存阶段的速率和计算阶段的速率波动很小, 几乎不随部署包大小的改变而改变;

b. 访存阶段的速率远小于计算阶段的速率, 在程序设计过程中, 应尽量减少对同一缓冲区进行 I/O 操作.

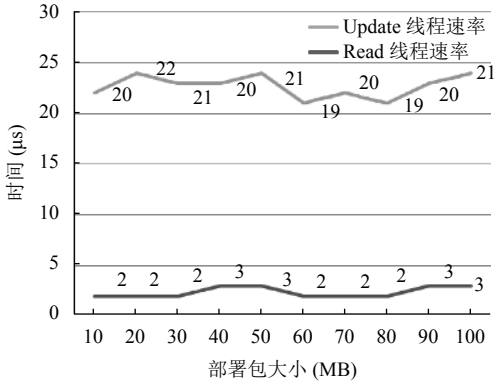

图 5 访存阶段耗时与计算阶段耗时

表 6 和图 6 记录了针对不同大小的部署包, 算法 1 和算法 2 消耗的总时间以及算法 2 产生的额外开销, 根据式 (3) 和式 (6) 可知, 算法 1 和算法 2 的实际加速比是 S≈1.01.

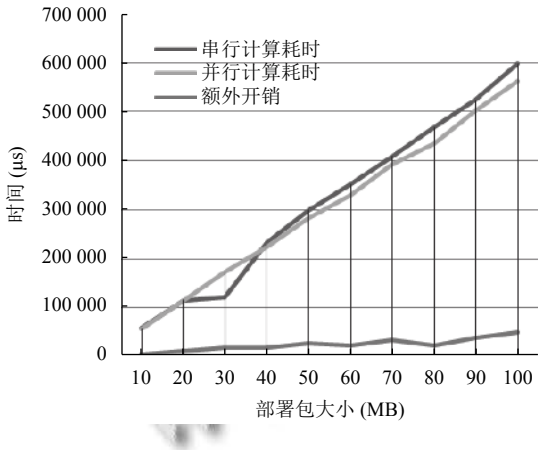

图 6 串行计算耗时与并行计算耗时

通过分析上述数据可知:

a. 针对不同大小的部署包, 算法 2 的耗时小于算法 1. 与算法 1 相比, 算法 2 利用并行运算实现了访存的隐藏, 因此提高了计算效率;

b. 考虑到额外消耗的时间, 理论加速比和实际加速比几乎一致. 由于实验选择了合理的分块大小, 所以通信、同步和空闲等待时间等因素带来的额外开销在整个计算过程中所占比例较小.

c. 在一般的部署任务中, 部署包大小范围一般在 100 MB 左右, 部署包越大, 算法 2 相较于算法 1 的效率提升就越明显, 减少的计算耗时可抵消部署双方通信加解密带来的计算开销.

④ 自动集成部署能力分析

Cube-1.3 是一款基于 Linux 操作系统的可信网络连接软件, 在安装过程中, 需要依赖的软件包括 libtool、autoconf 和 automake, 同时需要在配置 3 条环境变量.

本方案以 Cube-1.3 软件的安装部署为例, 设计测试用例, 模拟自动部署过程, 并以目标机的软件装机量为实验变量, 分析自动集成部署和手动部署之间的差异.

在手动部署过程中, 需要进行环境配置、依赖软件安装、待部署软件安装 3 部分. 针对单个装机量来讲, 环境配置平均需要 30 s, 依赖软件安装需要 25 s, 待部署软件安装需要 10 s, 一共需要 65 s 即可完成一台的装机量. 按照以上时间计算, 手动部署 10 台目标机的时间是 650 s.

在自动集成部署过程中, 测试用例主要测试软件部署包集成和软件部署包解压安装两个部分, 软件描述文件主要使用脚本替代. 其中, 软件描述文件的生成大约需要 60 s, 软件部署包集成工作大约需要 28 s, 软件部署包解压安装需要 5 s, 单台目标机客户端的软件自动部署时间一共大约需要 93 s. 由于软件描述文件和软件部署包的生成部分都是前期工作, 因此, 在部署多台目标机客户端的过程中不需要重复生成, 只需要通过网络传输给相应客户端即可. 按照单台目标机自动集成部署的时间计算, 部署 10 台目标机的时间为: 软件描述文件生成 60 s+软件部署包集成 28 s+软件部署包解压安装 8 s (10 台目标机同步自动安装, 由于网络传输和目标机的差异性, 最终全部安装结束的时间具有差异性), 一共约需要 98 s.

通过分析上述数据可知:

a. 自动集成部署需要进行前期准备工作, 耗时较长, 在软件部署规模较小和复杂度较低的情况下, 不如手动部署配置灵活, 速度快;

b. 在软件规模大、结构复杂的情况下, 自动集成部署优势明显, 并且由于实现了标准化部署, 出错概率较低, 后期维护成本小.

(3) 实验总结

1) 相较于整个部署过程, Diffie-Hellman 密钥交换算法和 AES 加解密算法带来的计算开销所占比重较

小, 但是对于通信信息的安全性提升很大;

2) 关于软件部署包的度量, 本方案利用硬件环境多核多线程的特点, 对完整性度量算法进行了并行化处理, 部署包越大, 线程数越多, 计算效率的提升就越明显, 可有效地抵消方案中部分安全设计带来的计算开销;

3) 在目标机规模大, 任务复杂的情况下, 采用自动集成部署的设计方案对于部署效率提升显著.

5 结束语

在分布式协同开发环境下, 软件规模越来越大, 其复杂性也越来越高, 针对现有的软件集成部署方法缺乏安全机制以及工作效率较低的问题, 本文提出了一种安全高效的解决方案. 该方案利用 Diffie-Hellman 密钥交换算法为部署双方创建安全关联, 通过可部署软件描述方法为待部署软件建立规范化的软件描述, 并且改进完整性度量算法, 进一步提高了软件的部署效率. 最后, 本文通过安全性分析和性能测试实验, 说明了该方案的研究价值.

在后续研究过程中, 本方案还需改善以下两点: 1) 引入公钥证书和数字签名等密码学机制, 进一步提高本方案的安全性; 2) 本方案提供的集成部署方法还处于模型阶段, 后续过程中可在 Eclipse、QtCreator 等开放式平台的基础上, 实现部署过程的图形化、智能化, 构建一套界面友好、使用方便的集成开发环境.

参考文献

1 Talwar V, Milojicic D, Wu QY, *et al*. Approaches for service deployment. IEEE Internet Computing, 2005, 9(2): 70–80. [doi: 10.1109/MIC.2005.32]

2 Ajmani S. Automatic software upgrades for distributed systems[Ph.D. Thesis]. Cambridge, Massachusetts: Massachusetts Institute of Technology, 2003.

3 姜军银, 侯立刚, 柴凯. 用 InstallShield 制作商品化软件安装程序. 电脑编程技巧与维护, 2004, (4): 10–12. [doi: 10.3969/j.issn.1006-4052.2004.04.004]

4 Kichkaylo T, Ivan A, Karamcheti V. Constrained component deployment in wide-area networks using AI planning techniques. Proceedings International Parallel and Distributed Processing Symposium. Nice, France. 2003.

5 Malek S, Medvidovic N, Mikic-Rakic M. An extensible framework for improving a distributed software system's deployment architecture. IEEE Transactions on Software Engineering, 2012, 38(1): 73–100. [doi: 10.1109/TSE.2011.3]

6 Balasubramanian K, Schmidt DC. Physical assembly mapper: A model-driven optimization tool for QoS-enabled component middleware. Proceedings of 2008 IEEE Real-Time and Embedded Technology and Applications Symposium. St. Louis, MO, USA. 2008. 123–134.

7 Chandra S, Parashar M, Yang JM, *et al*. Investigating autonomic runtime management strategies for SAMR applications. International Journal of Parallel Programming, 2005, 33(2–3): 247–259.

8 Diffie W, Hellman M. New directions in cryptography. IEEE Transactions on Information Theory, 1976, 22(6): 644–654. [doi: 10.1109/TIT.1976.1055638]

9 陈伟, 杜凌霞, 陈红. 多核架构下的数据处理算法优化策略综述. 计算机科学与探索, 2011, 5(12): 1057–1075. [doi: 10.3778/j.issn.1673-9418.2011.12.001]

10 李梦雨. 文件 I/O 函数和标准 I/O 库函数的读写效率研究. 软件导刊, 2010, 9(6): 5–7.