基于云的 Web 应用性能测试服务平台①

李 萱[1,2], 王 伟[1], 张文博[1], 范国闯[1]

[1](中国科学院软件研究所 软件工程技术研究开发中心, 北京 100190)
[2](中国科学院大学, 北京 100049)

摘 要: 云计算作为一种新型的计算资源交付模型, 为软件性能测试带来了新的机遇. 设计了一种四层架构的性能测试服务平台, 实现传统性能测试软件向云计算环境的迁移, 同时设计一种准入控制和任务调度算法, 实现负载发生资源的共享管理和动态分配, 满足多租户性能测试需求. 基于上述工作, 完成已有性能测试工具 Bench4Q 的迁移.

关键词: 云计算; 多租户; 性能测试; 软件服务化; 虚拟化

Cloud-Based Performance Testing Service

LI Xuan[1,2], WANG Wei[1], ZHANG Wen-Bo[1], FAN Guo-Chuang[1]

[1](China Technology Center of Software Engineering, Beijing 100190, China)
[2](China University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Cloud computing has emerged as a new paradigm for delivery of computing resources. It has brought great opportunities to performance testing. A four layer model for cloud-based performance testing is proposed. At the same time, an admission control and scheduling algorithm is also designed. All this can achieve the flexible management and dynamic dispatch of shared resources, which are pooled for load generation. The requirements of multi-tenancy can also be implemented. Based on the above mentioned proposition, an available tool, Bench4Q is migrated to the cloud.

Key words: cloud computing; multi-tenancy; performance testing; software as a service; virtualization

软件性能测试是通过自动化的测试工具模拟正常、峰值以及异常负载条件来对系统地各项性能指标进行测试, 验证软件系统能否达到用户提出的性能指标, 同时发现软件系统存在的性能瓶颈. 性能测试工具根据测试需求模拟不同规模的负载强度, 需要大量的软硬件投入, 是典型的资源密集型系统. 不同规模负载的测试需求意味着性能测试所需要的软硬件资源规模是动态变化的, 而根据峰值需求配置软硬件资源将会带来大量的资源投入和严重的资源浪费. 例如, 著名社交网站 MySpace 完成一次百万量级并发用户压力测试需要投入上千台计算机资源[1]. 受客观条件限制, 传统性能测试方法通常采用 1: 20 的缩微仿真模拟并发用户. 性能测试不充分, 最终将导致系统上线运行后出现服务失效、甚至系统崩溃.

云计算作为一种新型的计算资源交付模型, 为软件性能测试带来了新的机遇. 基于云的性能测试服务利用云计算的资源动态扩展特性, 将突破本地资源限制, 降低测试资源的投入和管理复杂度. 然而, 将传统软件系统和工具转换成云计算服务, 仍需要解决大量技术问题. 根据云计算提供的服务类型, 通常将云计算分为基础架构即服务(IaaS)、平台即服务(PaaS)和软件即服务(SaaS). 对软件性能测试工作而言, 需要将现有的性能测试工具转换为 SaaS 架构, 提供具有多租户(multi-tenancy)特征的软件服务, 同时, 利用底层 IaaS 架构提供负载发生资源的弹性管理.

本论文针对传统性能测试工具向云计算环境迁移带来体系架构改变和系统需求, 设计实现一种四层架构的性能测试服务平台, 实现传统性能测试软件 Bench4Q[2]

① 基金项目:国家自然科学基金(61173004,61100068);国家高技术研究发展计划(2012AA011204);国家科技支撑计划(2012BAH09F01)
收稿时间:2013-04-01;收到修改稿时间:2013-04-22

© 中国科学院软件研究所 http://www.c-s-a.org.cn

向云计算环境的迁移, 满足多租户性能测试需求. 同时, 设计一种准入控制和任务调度算法, 实现负载发生资源的共享管理和动态分配.

论文组织如下: 第一节介绍相关背景及系统需求, 第二节给出基于云的四层架构性能测试服务平台的设计, 第三节给出平台关键技术, 第四节验证平台的应用效果, 最后是全文结论.

1 背景和系统需求

1.1 背景

软件性能测试是通过自动化的测试工具模拟正常、峰值以及异常负载条件来对系统的各项性能指标进行测试. 目前, 软件的网络化趋势让系统更加开放、用户规模更难以预计, 需要更多的硬件设备模拟更真实的负载规模, 导致测试成本极高. 例如, 常用商用软件性能测试工具通常按照负载规模、负载协议数量、监控分析功能等收费, 如 LoadRunner[3]. 同时, 性能测试所需的大量软硬件资源也提高了系统管理的复杂度. 受客观条件限制, 现有性能测试通常采用 1: 20 的缩微仿真模拟并发用户. 性能测试不充分容易导致系统上线运行后出现服务失效、甚至系统崩溃.

云计算是一种新型的计算模式, 具有资源池化管理、多租户共享、基于浏览器访问、按使用付费等特性, 为软件性能测试提供了新的发展机遇. 基于云的性能测试服务可以突破本地资源限制, 降低测试资源的投入和管理成本. 用户不必购买数量庞大的测试服务器, 不必购买各类价格昂贵的测试软件, 甚至不必部署复杂的测试环境, 而是通过浏览器使用性能测试服务, 并依据使用时间和资源量进行付费.

1.2 系统需求

云计算分为基础架构即服务(IaaS)、平台即服务(PaaS)和软件即服务(SaaS)等服务类型. 对软件性能测试而言, 需要将现有的性能测试工具转换为 SaaS 架构, 并提供具有多租户(multi-tenancy)特征的软件服务, 即让来自不同组织(租户)的用户共享同一软硬件资源[4,5]; 同时, 利用底层的 IaaS 架构提供基础设施资源的弹性管理. 系统包括以下方面的需求:

(1) 用户自助门户

用户通过自助门户使用性能测试服务, 并提供测试数据的在线展现、测试报告的自动生成、存储以及历史测试记录的查询等功能.

(2) 负载发生资源管理

基于云的性能测试服务带来性能测试应用模式的转变, 服务提供者需要为服务使用者(测试用户)提供负载发生资源, 满足不同规模的性能测试需求, 同时, 需要通过多租户共享提高资源共享程度, 降低资源的运维成本.

(3) 租户性能隔离

对于具有资源密集特征的性能测试服务而言, 租户之间的性能隔离更为关键. 在通过多租户特征最大限度地实现共享负载发生资源的同时, 需要保障每个租户的资源有相应的边界, 使得租户的服务质量不会被其他租户所影响.

(4) 租户数据隔离

SaaS 应用通常采取多租户共享同一数据库、同一数据表结构的数据存储方式. 在这种方式下, 租户之间通过引入租户 ID 实现数据元组之间的隔离. 但是, 由于租户共用相同的数据表, 则降低了租户的数据安全. 特别对于性能测试服务而言, 租户配置的测试计划和测试脚本可能涉及到需要受到保护的企业或个人信息.

(5) 待测系统监测

需要提供待测系统的 CPU 利用率、内存利用率、磁盘读写速度、网络 I/O 等监测信息, 帮助发现和定位系统性能瓶颈.

2 Bench4Q as a Service

2.1 Bench4Q

Bench4Q 是一款面向 Web 应用的性能测试工具, 作为开源项目发布在国际开源软件组织 OW2 社区[6]. Bench4Q 提供性能测试脚本录制、分布式负载发生以及待测系统资源监控等功能. 后文根据云计算环境下性能测试服务平台的系统需求, 将 Bench4Q 工具转换成服务化的形式.

2.2 系统架构

基于云的性能测试服务平台的系统架构如图 1 所示, 分为四层: 门户访问层、业务逻辑层、测试基础架构层、基础平台层.

1) 门户访问层

门户访问层提供用户访问服务的交互界面, 是用户使用测试服务平台的入口. 用户通过注册账号登录并使用测试服务. 用户通过门户可以制定测试计划、

© 中国科学院软件研究所 http://www.c-s-a.org.cn

录制和编辑测试脚本、浏览保存后测试计划和脚本、监测测试过程和待测系统状态、下载历史测试报告等.

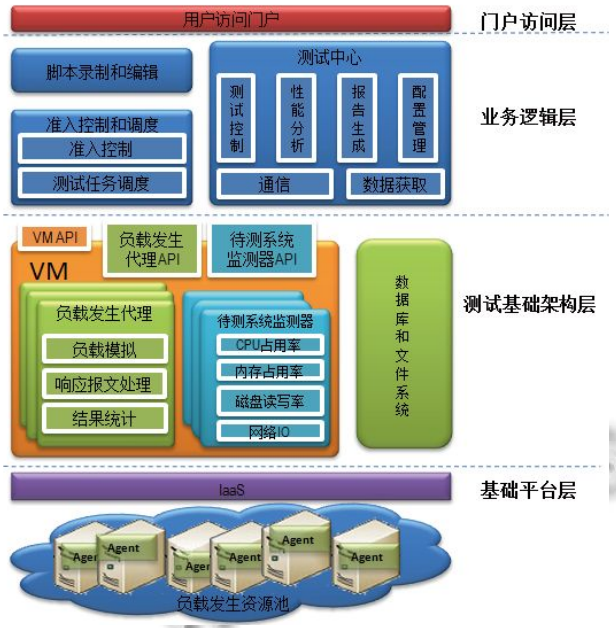

图 1 基于云的性能测试服务平台的系统架构

2) 业务逻辑层

业务逻辑层提供测试服务的配置管理和资源管理, 主要包含 4 个模块:

脚本录制和编辑模块——用于进行性能测试的用例设计. 脚本录制通过设置用户浏览器的代理服务器, 捕获浏览器与服务器端待测系统间的交互通信, 包括浏览器的请求路径及参数、服务器端的响应结果, 并记录为 XML 格式的脚本文件.

准入控制模块——在用户提交测试任务后, 准入控制模块首先检查测试任务等待队列, 判断是否有足够可用的负载发生资源来完成测试任务, 然后决定是否接受该测试任务.

测试任务调度模块——为测试任务分配负载发生资源, 跟踪测试任务的执行进度并调度任务等待队列内的测试任务.

测试中心模块——测试中心由若干子模块组成. 测试控制模块是测试中心的核心模块, 控制整个测试流程; 性能分析模块负责对收集的待测系统性能数据(吞吐率、响应时间等性能度量值)进行处理和分析; 报告生成模块根据测试数据生成可供用户查看和下载的测试报告(图 2 显示了迁移后的Bench4Q的在线测试报告); 配置管理模块提供测试配置的在线编辑接口, 并负责保存用户的测试配置信息; 通信模块提供了基于 RMI、RPC 等协议的远程交互接口, 满足四层架构的通信需求; 数据获取模块负责访问存放测试数据的数据库及文件系统.

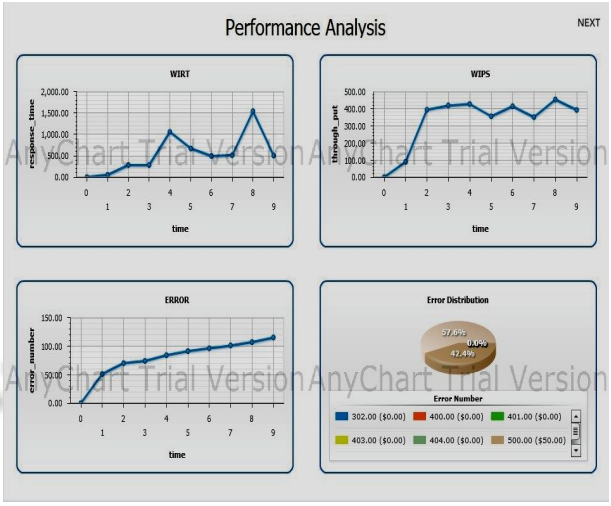

图 2 在线测试报告

3) 测试基础架构层

测试基础架构层包括负载发生代理、待测资源监测器、存储测试配置和结果的数据库及文件系统.

负载发生代理(Agent)负责模拟访问待测系统的并发用户, 产生负载. 对于 Bench4Q 而言, Agent 通过读取用户录制的测试脚本和测试计划, 向待测系统发送 HTTP 请求, 并收集待测系统反馈的性能数据.

待测系统监测器部署在待测系统上, 监测 CPU、内存、硬盘、网络的使用情况.

4) 基础平台层

基础平台层(IaaS)利用基于虚拟机(VM)的虚拟化技术为性能测试服务平台提供基础设施环境服务. 在此基础上, 通过建立基于 VM 的负载发生资源池并提供有效的池化资源调度机制, 实现资源的共享管理和动态分配, 包括负载发生资源的部署、回收、同步配置、负载均衡等.

负载发生资源池由一组建立在同一个或多个物理机上的多个 VM 组成. 每个 VM 定义了物理资源的边界, 并包含一个 Agent 模块. 经过任务调度模块分配的用户测试任务最终通过 VM 资源产生测试负载, 并在测试任务执行过程将VM节点作为准入控制和调度的最小单位.

图3中展示的是Bench4Q性能测试平台的测试流程: 1)用户通过访问门户层提供的交互界面, 指定待测系统的访问地址、负载规模、测试时长等配置信息; 2)提交测

© 中国科学院软件研究所 http://www.c-s-a.org.cn

试任务到性能测试服务平台; 3)执行测试任务, 负载发生代理向待测系统发送负载; 4)收集待测系统反馈的性能数据并为用户提供数据的在线显示界面; 5)测试结束后, 为用户生成测试报告.

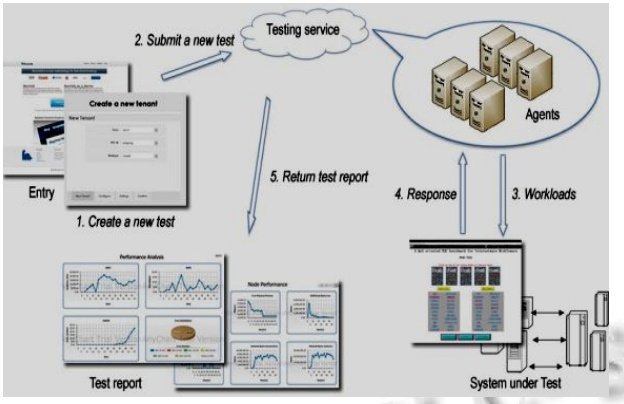

图 3　使用 Bench4Q 性能测试服务的流程

3　关键技术

在将 Bench4Q 工具迁移到云计算环境时, 需要克服多租户带来的负载发生资源调度和隔离问题.

3.1 负载发生资源调度

文献[7]提出一种面向多 SaaS 应用的 VM 分配算法. 该算法提供了对资源请求响应的策略, 通过在单一 VM 节点中处理不同用户请求, 提高资源利用率. 这种场景和性能测试服务有一定程度的相似. 不同的是, 对于每一个性能测试任务, 都可能需要性能测试服务平台分配多个 VM 同时协作完成测试任务. 我们对文献[7]的算法进行修改, 提出一种针对于性能测试服务的负载发生资源池管理算法, 利用有限的 VM 资源服务尽可能多的租户. 算法涉及两个阶段: 准入控制和测试任务调度.

准入控制的具体算法流程如下:

判断某个 VM 的配置是否满足用户的负载生成需求. 如果满足, 判断如果把当前测试任务放在这个 VM 的等待队列的队尾, 能否在用户指定的完成时间内完成测试.

如果不能完成, 则判断能否将当前测试任务插入到队列的某个位置, 并且保证每个任务都能按时完成.

如果现有负载发生资源不能满足上述需求, 在不能满足需求的 VM 中进行遍历, 检查是否存在某两个 VM 能够满足用户的负载生成需求, 且可以通过插入到队尾的方式在规定时间内完成测试任务.

如果上述三个条件都不能满足, 则为该任务初始化新的 VM. 如果初始化 VM 成功, 接受该任务. 否则拒绝接受该任务.

算法伪代码如下:

Algorithm 1 Admission Control

Input:

Expected configuration of objective VM: $Conf_{new}$

Duration of the new test: Dur_{new}

Deadline of the new test: Dl_{new}

Output:

Whether to accept the new request: *accept* or *reject*

```
1.  if there is initiated VM then
2.    //VMs are sorted by Conf asc
3.    find VM_k where Conf_k > Conf_new ;
4.    for i=k to n do
5.      if !canWait(Dur_new, Dl_new,VM_i) then
6.        if !canInsert(Dur_new, Dl_new,VM_i) then
7.          continue;
8.        else
9.          mark VMi ;
10.         return accept ;
11.       end if
12.     else
13.       mark VMi ;
14.       return accept ;
15.     end if
16.   end for
17.   for i=1 to k do
18.     for j=i+1 to k do
19.       if!canWait(Dur_new, Dl_new,VM_i) then
20.         continue;
21.       else
22.         mark VMi and VMj ;
23.         return accept ;
24.       end if
25.     end for
26.   end for
27. end if
28. if !canInitNew(Dur_new, Dl_new,Conf_new) then
29.     return reject;
30. else
31.     return accept;
32. end if
```

算法 1　准入控制算法

© 中国科学院软件研究所 http://www.c-s-a.org.cn

测试任务调度依据准入控制算法产生的结果进行资源分配和调度，具体算法流程如下：如果采取的是等待策略，新的测试任务需要进入所分配的 VM 的等待队列队尾．如果采取的是插入策略，新的测试任务会被插入到准入控制算法分配给的 VM 的等待队列前端．否则，新的 VM 将被初始化纳入待分配的负载发生资源池中．

算法伪代码如下：

Algorithm 2 Test Scheduling

Input:

Strategy: *wait*, *insert* or *initNew*

Marked VMs : VM_i (, VM_j)

Inserted index: *flag*

Expected configuration of objective VM: $Conf_{new}$

1. **if** *strategy*=*wait* **then**
2. put $test_{new}$ to the end of the waiting queue of VM_i (, VM_j) ;
3. **end if**
4. **if** *strategy* = *insert* **then**
5. insert $test_{new}$ to position *flag* of the waiting queue of VM_i;
6. **end if**
7. **if** *strategy* = *initNew* **then**
8. initiate a new VM with $Conf_{new}$;
9. **end if**

算法 2 测试任务调度算法

3.2 多租户隔离

文献[8]将 VM 定义为物理机的一种高效、隔离的复制，由于虚拟机之间是逻辑隔离的，因此可以基本避免 VM 之间的干扰和影响．我们将 VM 作为负载发生资源池的基本组成单元，并在测试任务执行过程以 VM 节点作为准入控制和调度的最小单位，从而在实现资源共享管理的同时，满足租户性能隔离要求．

同时，为了实现测试用户的数据隔离，除了在共享数据库中使用租户 ID 实现数据元组之间的隔离，还为租户分配专有文件目录保存租户历史测试记录和测试脚本，并通过设置文件读写权限来控制租户信息的访问．

4 实验验证

我们将部署在 Tomcat 服务器和 MySQL 数据库上的 J2EE 网上书店应用作为待侧系统，对比 Bench4Q 工具在迁移前后的表现．图 4-5 分别显示使用传统 Bench4Q 工具和迁移后的 Bench4Q 测试服务平台的测试结果，其中横轴表示每秒的并发用户数(单位为个)，左侧纵轴表示吞吐量 WIPS(Web Interactions Processed Per Second，每秒交互数，单位为次)，右侧纵轴表示平均响应时间 WIRT (Web Interaction Response Time，单位为毫秒)．

如图 4 所示，在使用传统 Bench4Q 工具进行测试时，使用单个 VM 执行测试任务，当并发用户数超过 500 后，由于负载发生资源成为瓶颈，所产生的实际负载低于测试配置值．在此情况下，当并发用户数超过 500 后，系统吞吐量的增长速度降低，甚至出现下降(如 700 并发用户时)．该测试结果容易误导用户得出待测系统支持的最大并发用户数在 700 至 900 之间的测试结论．

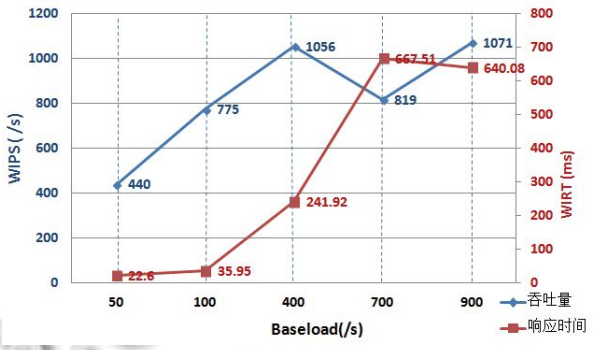

图 4 使用传统 Bench4Q 进行性能测试

图 5 显示使用迁移后的 Bench4Q 性能测试服务进行测试的结果．与传统 Bench4Q 工具相比，迁移后的 Bench4Q 突破了负载发生资源限制，提供了更真实的大规模测试负载．如图 5 所示，随着负载规模的增加，系统吞吐量和响应时间也持续增加，当并发用户达到 2000 时，待测系统出现服务失效．由此判断待测系统支持的最大并发用户数在 1600 至 2000 之间．．

5 结束语

在将传统性能测试工具迁移到云计算环境需要解决一系列技术问题．本文设计实现了一种四层架构的性能测试服务平台，实现 Bench4Q 开源性能测试工具

© 中国科学院软件研究所 http://www.c-s-a.org.cn

的迁移，同时，设计一种准入控制和任务调度算法，实现负载发生资源的共享管理和动态分配，满足多租户管理需求.

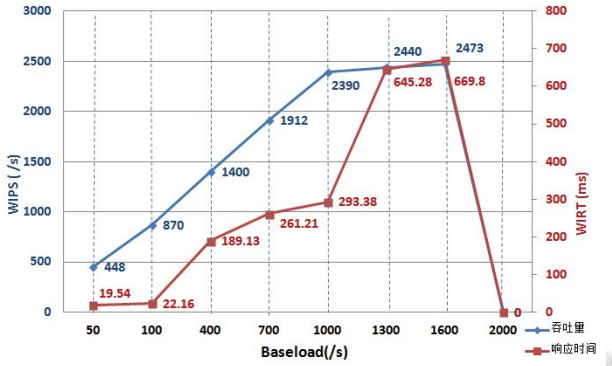

图 5 使用迁移后的 Bench4Q 进行性能测试

参考文献

1 Reuters. (2009)SOASTA Leverages the Cloud to Test 1,000,000 Users on MySpace. http://cn.reuters.com/ article/ pressRelease/idUS121904%2B17-Nov-2009%2BMW20091117.

2 Zhang WB, Wang S, Wang W, Zhong H. Bench4Q: A QoS-oriented E-commerce benchmark. Proc. of 35th Annual IEEE International Computer Software and Applications Conference (COMPSAC 2011). Munich, Germany, 2011: 38–47.

3 Wikipedia.(2012) HP LoadRunner. http://en.wikipedia.org/wiki/HPLoadRunner.

4 Zhang Q, Cheng L, Boutaba R. Cloud computing: State-of-the-art and research challenges. Journal of Internet Services and Applicatons, 2010, 1(1): 7–18.

5 林海略,韩燕波.多租户应用的性能管理关键问题研究.计算机学报,2010,33(10): 1881–1895.

6 OW2,(2012)Bench4Q.:http://forge.ow2.org /projects/jaspte.

7 Wu LL, Garg SK, Rajkumar Buyya. SLA-based admission control for a software-as-a-service provider in cloud computing environments. Journal of Computer and System Sciences, 2012, 78(5): 1280–1299.

8 Popek GJ, Goldberg RP. Formal requirements for virtualizable third generation Architectures. Commun. ACM, 1974, 17: 412–421.

(上接第 7 页)

June1998:14–19.

7 Ghiasi S, Casmira J, Grunwald D. Using IPC variation in workloads with externally specified rates to reduce power consumption. Workshop on Complexity Effective Design. Wisconsin. 2000. 96–101.

8 Weissel A, Bellosa F. Process cruise control: event-driven clock scaling for dynamic power management. CASES. Grenoble. France. October 2002. 234–246.

9 Choi K, Soma R, Pedram M. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. Automation and Test in Europe Conference and Exhibition, Proceedings. 2004, 1: 4–9.

10 Chantem T, Hu XS, Dick RP. Online work maximization under a peak temperature constraint. ISLPED'09, August 2009:19–21.

11 Wang L, von Laszewskiy G, Dayaly J, Hey X, Andrew J, Younge Y, Furlani TR. Towards thermal aware workload scheduling in a data center. Pervasive Systems, Algorithms, and Networks(ISPAN). 2009 10th International Symposium. 14-16 Dec. 2009.

12 Mukherjee T, Tang Q, Ziesman C, Gupta SKS, Cayton P. Software architecture for dynamic thermal management in datacenters.COMSWARE. Bangalore, India. 2007. 1–11.

13 Pathak A, Hu YC, Zhang M. Fine-grained power modeling for smartphones using system call tracin. EuroSys'11. Salzburg, Austria. 2011.

14 Burguera I, Zurutuza U, Nadjm-Tehrani S. Crowdroid: behavior-based malware detection system for android. SPSM'11. Chicago, Illin. October 17, 2011.

15 Sherwood T, Perelman E, Hamerly G, Sair S, Calder B. Discovering and exploiting program phases. IEEE Micro, 2003, 23(6): 84–93.

© 中国科学院软件研究所 http://www.c-s-a.org.cn