第35卷 第22期 计 算 机 工 程 2009年11月
Vol.35 No.22 Computer Engineering November 2009
·开发研究与设计技术· 文章编号：1000—3428(2009)22—0272—03 文献标识码：A 中图分类号：N945

面向瘦客户端的分布式动态二进制翻译系统

林 凌，管海兵，梁阿磊
(上海交通大学软件学院，上海 200240)

摘 要：传统的动态二进制翻译系统不适合直接用于瘦客户端，因为瘦客户端(如手机等)大多存在资源受限的问题，而动态二进制翻译过程会消耗较多的计算和内存资源。针对上述问题，提出一个适用于瘦客户端的分布式动态二进制翻译系统，用远程服务器完成二进制翻译，客户端只要执行翻译好后的代码即可。CPU SPEC 2000 的实验结果表明，在瘦客户端上使用该系统相对于使用传统的动态二进制翻译器可以带来更高的性能和更小的开销。

关键词：动态二进制翻译；分布式系统；瘦客户端；Crossbit 虚拟机

Distributed Dynamic Binary Translation System for Thin Client

LIN Ling, GUAN Hai-bing, LIANG A-lei
(School of Software, Shanghai Jiaotong University, Shanghai 200240)

【Abstract】 Since the binary translation process may consume many computation and memory resources and the thin client is a limited resource device, directly using ported traditional binary translation tools on the thin client may result in high cost and low efficiency. This paper proposes a new distributed dynamic binary translation system designed specifically for the thin client. This system uses a remote server to translate the source binary and executes the translated code in the thin client. Experimental results of CPU SPEC 2000 show that this method works much better than directly using ported traditional binary translation tools, and servers as a promising solution to the execution of different architectures' binaries on the thin client with low overhead.

【Key words】 dynamic binary translation; distributed system; thin client; Crossbit virtual machine

1 概述

不同的处理器有不同的指令集体系结构(ISAs)，其应用领域也不尽相同。例如，ARM 和 MIPS 体系结构经常用于瘦客户端，如手持设备、手机等，而 IA32 体系结构则常用于桌面 PC。这些指令集体系结构互相之间都不兼容，并且为其中一种指令集设计开发的软件往往不能直接运行于其他平台。二进制翻译[1]为上述问题提供了一种很好的解决方案，它不需要重新编译源代码，通过动态翻译二进制代码至目标平台，提供了对异构平台代码的支持。然而，二进制翻译过程需要耗费较多的计算资源以及内存资源。因此，直接将这类动态二进制翻译工具移植到资源受限的瘦客户端可能是不切实际并且低效的做法。

本文提出了一种新的适用于瘦客户端的二进制翻译架构：通过远程服务器翻译异构代码，瘦客户端只需通过向服务器请求翻译好后的代码执行而不需要进行二进制翻译过程就可以完成对异构平台软件的支持了。这种创新的二进制翻译架构将二进制翻译过程和目标代码执行过程作了很好的分离。SPEC 测试结果显示这种分布式的二进制翻译系统比直接移植传统二进制翻译器到瘦客户端的性能要提高很多。

2 相关问题

传统二进制翻译系统有很多，如 Crossbit[2], IA32-EL[3], Daisy[4], UQDBT[5]等，它们的工作方式大都如图 1 所示。通常，二进制翻译系统在一开始会加载源平台的二进制映像，然后从源二进制映像的入口地址开始查找目标代码缓存(TCache)看是否存在已经翻译过的基本块。目标代码缓存保存已经翻译好的目标代码基本块，它使得翻译过的基本块不需要重复翻译即可再次执行。如果二进制翻译器能够在目标代码缓存里找到相应的基本块，控制权将跳转到相应的目标代码块执行。如果没有在目标代码缓存里找到，二进制翻译引擎就需要启动二进制翻译过程：以源平台的程序计数器(PC)地址开始构建一个基本块，继而将代码翻译为目标二进制代码。同时，在翻译完成之后，目标代码块会被放入目标代码缓存中以供下次查找。

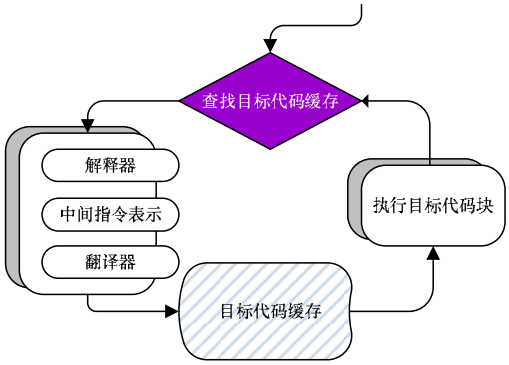

图 1 传统动态二进制翻译器的工作流程

基金项目：国家“973”计划前期研究专项课题基金资助项目(2007CB316506)；国家“863”计划基金资助项目(2006AA01Z169)；国家自然科学基金资助项目(60773093)

作者简介：林 凌(1984－)，男，硕士研究生，主研方向：动态二进制翻译；管海兵，教授、博士、博士生导师；梁阿磊，副教授、博士

收稿日期：2009-06-16 **E-mail：**liangalei@sjtu.edu.cn

目标代码缓存策略的引入使得二进制翻译系统的性能得到很大的提高，这是因为对翻译过的目标代码块可以进行重用。但是，由于目标代码缓存的大小限制，在目标代码缓存满的时候，需要采用替换策略，被替换掉的目标代码块在下一次被查找时就仍旧需要再次翻译了。

通过观察传统二进制翻译系统的工作流程，可以发现一旦出现目标代码缓存缺失，缺失惩罚开销就很高。尤其在目标代码缓存比较小的情况下，这种情况在内存受限的瘦客户端上尤其明显。因为目标代码缓存频繁地刷新从而带来非常高的缺失率。同时，翻译过程要耗费相当多的计算和内存资源。这些原因都使得在瘦客户端上直接使用传统的动态二进制翻译系统不太可行。

为了解决上述问题，本文提出了一种新的二进制翻译框架，它分离了源二进制代码的翻译部分和目标二进制代码的执行部分(在传统二进制翻译里，这 2 个部分是合在一起的)，使得它们分别执行在客户端和服务器端，并且在客户端和服务端采用了分层的目标代码缓存来暂存翻译过的目标代码块。

3　分布式动态二进制翻译系统的设计

本原型系统是基于 C/S 架构的动态二进制翻译器，由 3 个部分组成：翻译服务器，执行瘦客户端以及传输协议。图 2 描述了整个系统的架构。

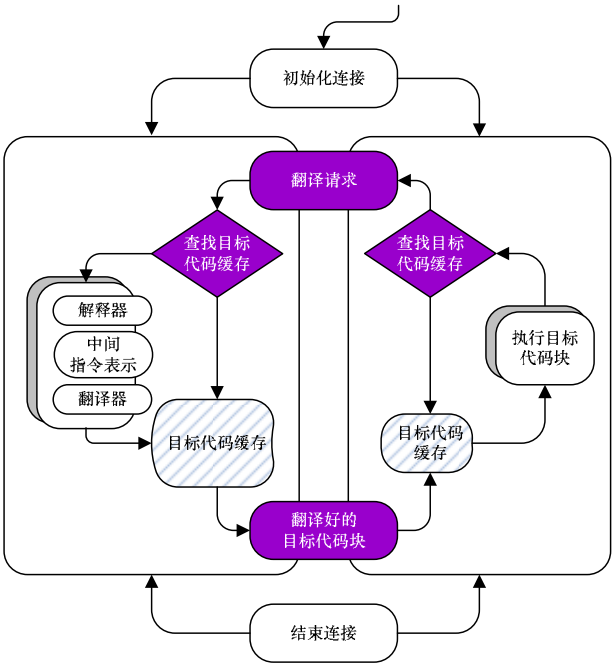

图 2　分布式动态二进制翻译器的框架

翻译服务器是二进制翻译过程真正发生的地方。执行瘦客户端则只需要发送翻译请求至服务器获取相应的目标代码执行。传输协议为服务端和客户端隐藏了传输实现上的细节。

同时，本原型系统还采用了双层目标代码缓存的结构，即在客户端和服务端都有目标代码缓存。

3.1　服务端的设计与实现

翻译服务器主要负责完成二进制翻译请求。由于二进制翻译过程比较耗费 CPU、内存等各类资源，因此假设翻译服务器是台高性能的服务器。翻译服务器的主要工作是在多源多目标的动态二进制翻译框架 Crossbit 的基础上完成的。Crossbit 使用中间指令来支持多个不同的前端机器平台和目标机器平台。目前，它支持有 MIPS、IA32 的前端和 IA32 的后端。

最开始，翻译服务器根据客户端的初始化数据包准备翻译环境，即相应的源平台解释器以及目标平台翻译器。同时，为了方便翻译过程，翻译服务器也会加载源二进制映像。

初始化完成之后，翻译服务器进入主循环等待客户端的翻译请求。翻译请求包含请求代码块的起始地址以及目标代码块的存放地址。由于服务端和客户端可能会将目标代码块放在不同的地址空间，因此翻译后的目标代码块需要根据客户端存放的具体地址作重定向。当接收到翻译请求之后，翻译服务器首先用源二进制代码块的起始地址在目标代码缓存里查找是否存在已经翻译过的代码块。如果找到了，服务器只需要简单地将目标代码块根据客户端的请求信息作一个重定向然后发送回客户端就可以了。如果目标代码缓存里找不到已经翻译过的代码块，翻译服务器就需要启动翻译过程：从请求的起始地址开始创建一个基本块，然后将它翻译成对应的目标代码提交到目标代码缓存。在翻译完成之后，目标代码块会发送回客户端。

所有翻译过的目标代码块都将会缓存在足够大的目标代码缓存里。一旦客户端请求了一个已经被翻译过的代码块，这种策略就保证了服务端不需要重复翻译目标代码块。

3.2　客户端的设计与实现

由于客户端不需要做二进制翻译，因此实现要简单很多。首先，客户端会加载要执行的源二进制映像，并完成与服务器的初始化。之后，客户端开始执行，它用源机器计数器的值查找本地的目标代码缓存看是否存在相关的目标代码块。如果找到了，客户端就跳转过去执行代码块。如果在本地目标缓存里找不到相应的代码块，客户端就向翻译服务器发送请求。在服务器将目标代码块发回后将其提交到本地的目标代码缓存，然后执行。

当目标代码缓存满时，客户端采用了一种简单的“全清空”策略。这种策略在动态二进制翻译器里被广泛地使用，因为它非常简单，所以不会带来过高的开销。

3.3　通信协议的设计与实现

客户端与服务端通信协议设计的目的是隐藏数据交互时的传输实现细节。此协议用 SOCKET 编程接口构建于 TCP/IP 协议之上。服务端和客户端只需要调用相应的数据结构收发接口函数就可以完成数据结构的发送与接收，数据结构与数据包的相互转换在通信协议函数库里完成。

传输时数据包的格式如表 1 所示。目前一共有 4 种消息类型：INIT_CONNECT, REQUEST_TBLOCK, TARGET_BLOCK 和 END_CONNECT。INIT_CONNECT 数据包用来初始化连接，客户端向服务端请求目标代码块时使用 REQUEST_TBLOCK 数据包，TARGET_BLOCK 数据包则是服务端向客户端发送目标代码块时使用的，END_CONNECT 数据包中止双方的连接。

表 1　传输协议数据包格式

Header	MSG type	MSG length	MSG content	Tail
0xA5	/	/	/	0x5A

4　性能评估

为了模拟瘦客户端的应用场景，选择了 EVOC 嵌入式计算机系统和一台配备四核处理器的 IBM 服务器搭建实验环境。EVOC 嵌入式计算机系统配备有 400 MHz 的 Intel 赛扬处理器以及 128 MB 的内存，运行 Ubuntu Linux 系统。IBM 服

务器配备有 2.00 GHz 的 Intel 至强处理器 E5405 以及 4 GB 的内存，运行 CentOS 5.1 系统。

选用 CPU SPEC 2000 标准测试程序集中的“164.gzip, 181.mcf, 197.parser, 254.gap, 256.bzip2, 300.twolf”等 6 个程序来评估原型系统的性能。由于原型系统使用的是 MIPS 前端和 IA32 后端，因此还需要有相应的交叉编译器来完成 SPEC 程序到 MIPS ELF 可执行映像的编译工作。另外，考虑到瘦客户端一般内存资源受限等实际情况，将 EVOC 系统上的目标代码缓存设定为 512 KB。

比较移植到瘦客户端的传统二进制翻译器 Crossbit 和分布式的二进制翻译系统原型，实验结果如图 3 和图 4 所示。

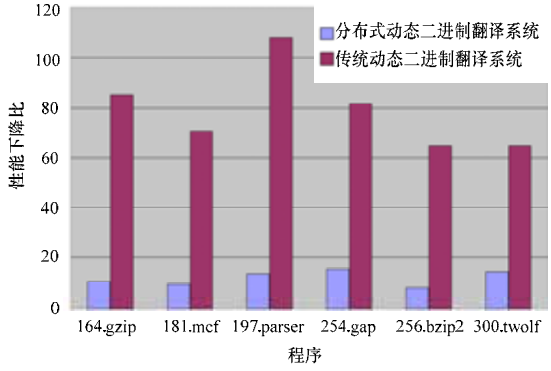

图 3　CPU SPEC 2000 部分程序测试结果

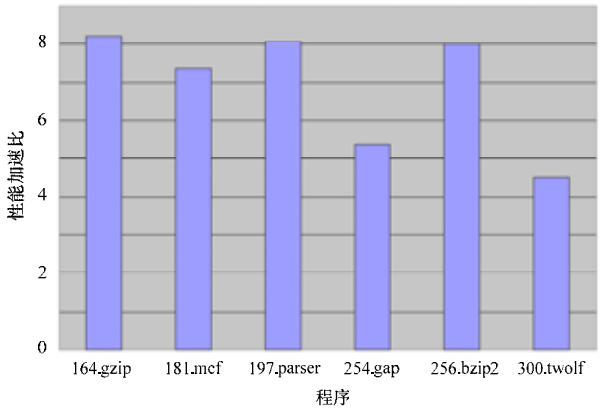

图 4　分布式动态二进制翻译系统相对于传统系统的性能加速比

图 3 的纵坐标是通过二进制翻译器执行程序相对于本地执行的性能下降比。图 4 则是分布式二进制翻译系统原型相对于传统二进制翻译器的性能提升。通过实验结果可以看到，分布式的二进制翻译系统原型比直接移植到瘦客户端上的传统二进制翻译系统性能要提高 4~8 倍。因为此分布式原型系统不需要在客户端上进行动态二进制翻译，所以向服务端请求翻译好的目标代码块即可。而且在目标代码缓存较小的情况下，传统二进制翻译器由于目标缓存的替换策略，需要经常重复翻译一些已经翻译过的代码块，而在分布式的二进制翻译系统原型里，客户端不需要重复翻译目标代码，只需要向服务端重新请求目标代码块。服务端由于采用的是足够大的目标代码缓存，也不需要重复翻译，只需将找到的目标代码块发送回客户端即可。而且值得指出的是，本原型系统还没有经过任何形式的优化，相信在运用一些性能优化方法之后，实验结果将会更加理想。

5　相关工作

本文介绍的是一种分布式的动态二进制翻译系统原型。现在有很多传统的二进制翻译系统，包括由上海交通大学设计的 Crossbit 系统、Intel 公司设计开发的 IA32-EL 系统、IBM 公司设计的 Daisy 二进制翻译系统，以及昆士兰大学设计的 UQDBT 系统和佛吉尼亚大学的 Strata 系统[2-5]等。所有这些系统都使用了动态二进制翻译的方法兼容不同的指令集体系结构。在这些系统里，翻译和执行过程是交替进行的，这种交织在一起执行的方式带来的开销对于瘦客户端来说是不可接受的。相反，本文介绍的系统采用了显示分离翻译和执行的过程，使得客户端的动态二进制执行变得更加容易实现。同时，传统的二进制翻译器使用的是统一的目标代码缓存，而本文介绍的分布式系统原型则采用了分层结构：服务端的目标代码缓存足够大，而客户端的缓存则考虑到受限系统的实际情况设计为限定大小。

Java 虚拟机(JVM)的实时编译(JIT)技术和微软的.NET 架构采用的中间语言运行时(CLR)是 2 个支持中间语言级别移植的运行时系统。这些系统需要将程序编译成特定的中间语言格式(如 Java 的字节码、微软的中间语言)才能完成程序的移植。而本文介绍的系统则是通过动态二进制翻译的方式，无须重新编译源代码即可支持异构的代码。

6　结束语

本文介绍了一种创新的分布式动态二进制翻译系统，该系统可用于计算资源及内存资源受限的瘦客户端，如手机等。通过此分布式的模型，瘦客户端可以在较低开销的情况下支持多种体系结构的二进制映像。

本系统仍然处在原型开发阶段，还有很多优化方法可以采用，如在客户端使用轻量级的代码块链接技术，服务端运用更有效的翻译策略以生成更高效的目标代码等。未来的工作将主要研究这些优化方法，从而使整个系统更加实用。

参考文献

[1] Altman E R, Kaeli D, Sheffer Y. Welcome to the Opportunities of Binary Translation[J]. IEEE Computer, 2000, 33(3): 40-45.

[2] 包云程, 梁阿磊, 管海兵. 动态二进制翻译基础平台 CrossBit 的设计与实现[J]. 计算机工程, 2007, 33(23): 100-101, 134.

[3] Baraz L, Devor B L, Etzion T O, et al. IA-32 Execution Layer: A Two Phase Dynamic Translator Designed to Support IA-32 Applications on Itanium(R)-based Systems[C]//Proc. of the 36th Int'l Symp. on Microarchitecture. San Diego, USA: IEEE Computer Society, 2003: 191-202.

[4] Ebcioglu K, Altman E. DAISY: Dynamic Compilation for 100% Architectural Compatibility[C]//Proc. of the 24th Int'l Symp. on Computer Architecture. Denver, USA: ACM Press, 1997: 26-37.

[5] Ung D, Cifuentes C. Machine-adaptable Dynamic Binary Translation[C]//Proc. of the ACM Workshop on Dynamic Optimization Dynamo. Boston, USA: ACM SIGPLAN, 2000: 30-40.

编辑　任吉慧