应用程序稳态测试系统的改进

计55 王逸松 2015011369

2018.10.10

目录

* 选题背景
* 已有工作
* 设计方案
* 相关工作

选题背景

* 评测任务的介绍
* 稳态评测的需求

评测任务

* 在程序设计竞赛等场景中，选手要完成的任务：编写一段程序来实现一个特定的功能。

* 如：编写一个函数 `void sort(int *a, int n)`，功能是将 a 数组中的 n 个元素进行排序。

* 评测系统收到选手程序后，将测试：
  - 运行时间是否符合要求（时间限制）
  - 运行空间是否符合要求（内存限制）
  - 是否正常结束，不产生运行时错误
  - 运行结果是否正确

* 若不满足这些限制，得0分。

* 此外，不得访问除该程序和输入输出之外的系统资源。

稳态测试的需求

* 现有评测系统多数在 Linux 中运行，用 time 来测量用户态时间，用 ulimit 来限制时间和内存，用 ptrace 来限制系统调用。

* 主要缺点是时间的测量不够精确：
  - 中断、系统调用、进程切换导致TLB和缓存缺失带来的不确定性（1s内可影响数十ms）
  - 虚拟内存与物理内存的对应关系不确定，导致缓存缺失率的不确定性（可影响数倍时间）
  - 其他设备和程序（如DMA设备、虚拟机宿主）占用CPU、缓存等资源带来的不确定性（1s内可影响数十到数百ms）

** 在 Linux 上运行一段以 2^21 字节为步长访问内存的程序，时间波动可达6倍。

稳态测试的需求

* 现有评测系统主要缺点是时间的测量不够精确。

* 我们希望能构建一个稳态测试系统，特性包括：
  - 被测程序在运行时不受中断、异常、系统调用和进程切换的干扰；
  - 被测程序能够得到一段连续的物理内存，以减少缓存的不确定性；
  - 被测程序不会受到其他设备和程序的影响。

* 即，从被测程序开始运行到结束运行，一直在用户态，没有其他程序和设备的干扰，并且每次运行的物理内存布局是一致的。

已有工作

* 应用程序稳态测试系统

应用程序稳态测试系统

* 基于 MIT JOS：微内核、页式内存管理、多进程。
* 评测前分配连续物理内存，保证不缺页，如有必要则整理内存。
* 评测时关闭中断，只保留用于检测超时的单次时钟中断。
* 关闭其他处理器和DMA设备，且不得运行在虚拟机里。
* 暂不支持任何系统调用，因此不支持libc。

* 另外实现了简单的网卡驱动，可作为在线评测系统的后端。

应用程序稳态测试系统

* 比 Linux 稳定，比其他在线评测系统稳定。

std::sort 1e7 test	duck-libstdduck	Linux (static, user)	Linux (dyn, user)	BZOJ	UOJ	LOJ	Vijos
ratio	**1.0**	**34.3**	**30.7**	**15.4**	**46.5**	**37.7**	**448.0**
rel. uncertainty	**0.015%**	**0.508%**	**0.455%**	**0.227%**	**0.689%**	**0.558%**	**6.636%**
uncertainty (ms)	**0.107**	**3.619**	**3.224**	**2.780**	**5.559**	**4.124**	**65.336**
Type B uncertainty (ms)	0.0005	2	2	2	0.5	0.5	0.5
time resolution (ms)	0.001	4	4	4	1	1	1
95% conf. interval (ms)	0.107	3.016	2.528	1.931	5.537	4.093	65.334
stddev (ms)	0.228	4.216	3.534	2.700	11.830	9.917	91.331
avg (ms)	**719.093**	**712.000**	**708.400**	**1222.800**	**807.200**	**739.440**	**984.600**
n	**20**	**10**	**10**	**10**	**20**	**25**	**10**

** 不仅是 std::sort，其他程序的测试结果也类似。

应用程序稳态测试系统

* 不够完善：
  - 不支持“系统调用”
  - 不支持 libc 和 C++（libstdc++）
  - 网络性能不够（因收发包需要中断，目前传文件速度仅为 1500KB/s）

设计方案

* 第一阶段：改进作为在线评测系统后端的特性。
  - 实现 libc 和 libstdc++ 的支持
  - 改进网络性能
* 第二阶段：实现基于多核（SMP）的稳态测试系统。
  - 支持“系统调用”
  - 更高的测时精度

libc/libstdc++

* libc 可使用轻量级项目 musl-libc。
* 由于不支持系统调用，可修改 libc 的实现：
  - 将 read, write 等系统调用重定向到内存。
  - 重新实现 malloc，使之不依赖 mmap 系统调用。
  - 去除其他可能的系统调用。

* libstdc++ 可使用 GCC 中的 libstdc++，只要 libc 的实现正确就能够运行。

改进网络性能

* 内核将网卡相关的数据结构映射到用户态，再由用户态程序直接进行操作。

* 需要实现用户态网络协议栈。由于本项目中只使用 UDP 协议，完全重新实现的难度或不高于移植。

多核稳态测试系统

* 每个核只有两种状态：内核态、用户态。
* 核与核之间使用共享内存来进行通信。
* 只有0号核可以长时间处于内核态，用于提供系统服务。
* 其他核可运行被测程序或“裁判程序”。
* 评测开始前，内核核将裁判程序和被测程序载入内存。
* 被测程序等待裁判程序宣布开始，然后开始运行。
* 被测程序声称运行结束，或超时后，内核发送中断杀死被测程序。
* 被测程序若需“系统调用”，可（用共享内存）向内核发送请求；若需交互函数的调用，可向裁判程序发送请求。

相关工作

* C10M问题
* DPDK

C10M问题

* 在一台服务器上保持10M(1e7)个长连接。该问题在2013年被提出。
* 主要解决方法如下：
  - 用户态网络协议栈，每个包无需经过内核，只需200时钟周期（原先30k）
  - 每个进程绑定一个CPU核，且保证不发生中断和进程调度。
  - 通过使用2MB或1GB的页大小，来减少缺页中断的发生次数。
  - 进程间通信使用无锁数据结构（基于CPU的原子指令）

** Ref: http://c10m.robertgraham.com/p/blog-page.html

Data Plane Development Kit

* 主要思想：将网络设备直接映射到用户态，并提供接口。

* 可用于开发用户态网络协议栈。

* 也有基于该思想的 SPDK (Storage Performance Development Kit)。

** Ref: https://www.dpdk.org/

为何要开发稳态测试系统

* 现有解决方案不完整
* Linux 系统过于复杂
* 自主知识产权

感谢倾听，欢迎提问！

关于稳定性

loop 4e9 test	duck-libstdduck	duck-musl	Linux (static, user)	BZOJ	UOJ	LOJ	Vijos
ratio	**1.0**	**1.0**	**3791.9**	**2598.8**	**1931.7**	**4979.9**	**70388.6**
rel. uncertainty	**0.000%**	**0.000%**	**0.282%**	**0.194%**	**0.144%**	**0.371%**	**5.244%**
uncertainty (ms)	**0.001**	**0.001**	**2.000**	**3.568**	**1.200**	**2.695**	**44.577**
Type B uncertainty (ms)	0.0005	0.0005	2	2	0.5	0.5	0.5
time resolution (ms)	0.001	0.001	4	4	1	1	1
95% conf. interval (ms)	0.000	0.000	0.000	2.955	1.091	2.649	44.574
stddev (ms)	0.000	0.001	0.000	4.131	2.331	6.417	62.310
avg (ms)	**685.716**	**705.884**	**708.000**	**1843.200**	**833.800**	**726.560**	**850.100**
n	**20**	**25**	**10**	**10**	**20**	**25**	**10**

putchar+fflush 1e6 test	duck-libstdduck	duck-musl	Linux (static, user)	BZOJ	UOJ	LOJ	Vijos
ratio	**1.0**	**44.4**	**623.3**		**60.5**	**69.7**	**628.1**
rel. uncertainty	**0.016%**	**0.689%**	**9.684%**		**0.940%**	**1.082%**	**9.758%**
uncertainty (ms)	**0.001**	**0.445**	**12.183**		**5.939**	**6.378**	**24.395**
Type B uncertainty (ms)	0.0005	0.0005	2		0.5	0.5	0.5
time resolution (ms)	0.001	0.001	4		1	1	1
95% conf. interval (ms)	0.000	0.445	12.017		5.918	6.359	24.390
stddev (ms)	0.001	0.951	25.677		12.645	15.404	34.095
avg (ms)	**4.491**	**64.587**	**125.800**	**TLE**	**631.900**	**589.280**	**250.000**
n	**20**	**20**	**20**		**20**	**25**	**10**