연구노트

EMPIRICAL STUDY OF MEMORY SHARING IN VM

2014314920 박수진

2018.05.03

ABSTRACT

- Content-based page sharing 은 가상화 환경에서 서버 메모리를 줄이기 위해 자주 사용되지만, real-world utility 와 sharing potential 을 둘 다 고려한 분석이 많지 않다.
- 우리는 실제 user 머신과 controlled 된 가상화 머신에서의 memory trace 를 분석했다!
  - ① zero page 를 제외한 절대적인 sharing level 을 관측 : 주로 15% 이하
  - ② 전체 sharing 에서 inter-machine sharing 은 낮게 기여하고,
    개별 머신 내의 sharing 이 전체 machine set 의 공유 가능성의 90%를 차지한다.
  - ③ 머신간의 작은 차이는 inter-machine sharing 을 상당히 감소시킨다.
  - ④ address space layout randomization 같은 OS 의 특징은 공유 가능성을 감소시킨다

INTRODUCTION

- 하나의 physical 서버에서 여러 고객을 서비스 하기 위해 가상화를 사용한다.
- memory efficiency 가 중요한데, 단순히 physical memory 를 쪼개서 VM 에 나눠준다.
  → 각 VM 의 메모리 사용량이 감소하면 더 많은 VM 을 돌릴 수 있다.
- **CBPS(Content Based Page Shaing) :** 메모리의 중복된 블록(or page)는 하나의 physical copy 로 합쳐지고 기존의 중복된 페이지들은 하나의 physical page 를 pointing.
  → 쓰지 않는 physical page 는 free 되고, memory footprint 를 줄일 수 있다.
  memory footprint : 프로그램이 실행되는 동안 사용하거나 참조하는 메모리의 양
- 공유를 통해 상당한 메모리 절약을 할 수 있다.
  (ex) Difference Engine : 세 개의 VM 사이에서 absolute memory sharing 50% (page-level)
  (ex) VMware : 열 개의 VM 사이에서 40%의 sharing
- 다양한 환경에서의 **real-world** 페이지 공유 가능성에 대한 연구가 부족하다.
  - 실제로 얼만큼의 sharing 을 얻기를 기대할 수 있는가?
  - primary origin of sharing (공유의 기원)
  - 공유 가능성을 낮추는 요인들:randomizing , security 를 위해 변경되는 memory contents

- 더 어렵다 → 실험을 tightly control + real-world 에서 data 얻기도 힘듬
  그래서 주로 benchmark workload 에서의 성능 비교를 사용한다. (실제 saving 반영 X)

- **가정 : 모든 potential page sharing 은 captured 될 수 있다.**
  특정 공유 시스템을 평가하는 문제와 페이지 공유의 가능성을 평가하는 문제를 분리.
- 공유가 literature 에서 기대되는 것보다 실제로 더 작게 이루어짐(반이나 더 적게)

Self - Sharing : memory duplication within a single OS.

- 전체 공유 가능성의 80% 이상

Inter-VM Sharing : memory duplication across multiple VMs.

- 보통 작고, 서로 다른 platform 에서는 사실상 zero.
- cloned VM 같은 특정 조건에서는 40% 정도..

Sources of Sharing

- Linux Kernel 을 설치하여 데스크탑 환경에서 공유의 기원을 연구
- 우리의 system-aware memory tracing tool 은 어떻게 공유된 페이지가 프로그램에 의해 사용되는지를 보여주는데, 대부분 GUI application 과 display 와 관련된 라이브러리에서 공유됨.

Security features and Sharing

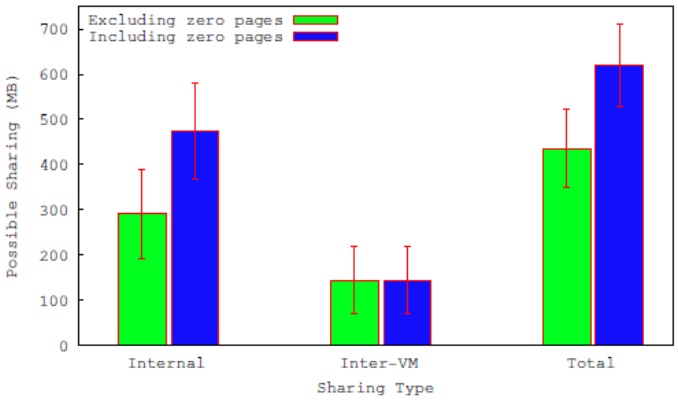
- OS 의 보안을 위한 기능-ASLR(address space layout randomization) 이 공유를 20%까지 감소
- 높은 수준의 homogeneity 인 환경이더라도, inter-VM sharing 이 감소

Background and Motivation

- **Hypervisor**
  - 모든 VM 에 걸친 physical memory 에 대한 global view 를 가짐. 호스트 컴퓨터에서 다수의 운영체제를 동시에 실행하기 위한 논리적 플랫폼
  - CBPS 는 가상화 시스템에서 대부분 hypervisor level 에서 수행됨.
  - hypervisor 는 sharable page 를 구별한 다음, 공유한다. (가상머신의 입력이나 협력없이!)
  - 그림 1. 6 개의 VM-level page 들이 4 개의 physical page 를 통해 제공되고 있다.
- 여러 연구에서, 이러한 page 공유 기술을 통해 서버 메모리의 40% 이상이 free 되는 걸 발견
  이 숫자는, VM 을 호스트에서 grouping together → sharing 으로 인해 해제된 페이지 전체 수
  → Self sharing 과 inter VM sharing 을 구별하지 않는다!

[Figure 2]

- 가상화 환경에서 발견되는 sharing 의 상당 부분은 각 VM 의 내부 sharing 때문
  두 개의 VM (Ubuntu 10.10 64-bit / 1.5GB Memory)
  → 2.6.32 리눅스 커널 컴파일(캐싱을 막기 위해 minor ver 는 살짝 다르게)



- (Excluding zero page-초록색) : Total 436MB 중에서 2/3(67%) 가 self-shared page 때문
- (Including zero page-파란색) : 77%가 self-shared 때문
- 대부분의 page sharing 은 OS 라이브러리와 같은 중복된 메모리 영역때문이라는 기존 연구와 반대
  의미 1. 어떻게 memory sharing 이 관리되는가? → 더 많은 VM 을 쓰더라도 공유 커지지 않음
  의미 2. 어떻게 memory sharing 이 구현되는가? → 단일 OS 내에서의 공유는 여러 VM 에서 시도하는 것만큼 효과적일 수 있다.

TYPES OF SHARING

physical memory 가 m, page size 가 s 인 가상 머신 V 는 m/s 개의 page 배열 Pv 를 가진다.

$$P_v = \{p_1, p_2, \ldots, p_{m/s}\}$$

이 때, 두 개의 페이지 p_i 와 p_j 의 contents 가 byte 단위로 모두 같으면 sharable.
만약, 동일한 페이지 p 가 k 개의 복사본이 있다면(p 를 포함해서), k-1 개의 페이지는 제거되고 중복된 페이지들은 single copy p 를 참조하게 된다.

UNIQUE(P_v) = P_V의 unique page 의 집합. V 를 나타내기 위한 최소 페이지 개수.
V 의 self-sharing 은 다음과 같이 계산. self-sharing 은 single VM 에서 측정된 공유.

$$S_{self}(V) = P_V - UNIQUE(P_V)$$

[Inter-VM sharing]

V_1 부터 V_k 의 집합 G 는 $P_G = \{P_{V1}, P_{V2}, \ldots, P_{Vk}\}$ 로 나타낼 수 있고, 다음과 같이 계산.

$$S_{inter}(G) = S_{self}(G) - \sum_{V \in G} S_{self}(V)$$

inter-VM sharing 은 가상머신을 G 안에 함께 두어서 생기는 이득.

하나의 머신 V 에 대해 $S_{inter}(V)$ 는 항상 zero

SHARING PROPERTIES

Self-sharing 은 기존의 시스템에서도 사용될 수 있고, inter-VM sharing 은 가상화환경에서만 적용
→ 공유에 대한 대부분의 연구는 가상 머신에 집중
→ inter-VM sharing 이 sharing 의 중요한 부분이라는 주장
(But, 가상머신일 필요 없다!는 것이 이 논문의 주장)

- 각각의 머신은 대부분 unique 한 페이지를 가지고 있고, 그런 머신이 여럿일 경우, inter-VM 공유↑
  (ex) A/B 가 각각 10 개의 distinct page → inter-VM sharing 은 10 page sharing(50%)
- Self-sharing 의 크기가 커지면 inter-VM sharing 의 상대적인 중요도가 감소한다.
  (ex) A/B 가 각각 서로다른 10 개의 페이지를 3 세트(총 30page) 가지고 있다면 inter-VM 은 그대로 10 page 이지만(17%) , self-sharing 이 각각 20page 이고(총 40) 67%
- inter-VM sharing 의 공유 가능성은 머신의 개수에 제약을 받음
  inter-VM sharing 은 머신 당 최대 한페이지밖에 절약못함. 그 이상 중복된 것은 self 에서 먼저!

➔ **Inter-VM sharing 은 ②내부 중복이 적은 VM 이 ① 서로 비슷하고 ③많을 때, Self-sharing 보다 중요해진다.**

CALCULATING SHARING

- real-system 은 모든 possible sharing 을 포착할 수 없다.
  - memory contents 가 바뀌기 때문에 어떤 공유기회는 수명이 짧다 → miss / 의도적 pass
  - 새로운 system 들은 largest + safest sharing 기회를 식별하려고 지속적으로 시도해왔지만, 우리는 이러한 문제에서 비켜서서, 모든 가능성을 전부 확인하는 방식으로 진행.
    (효율적인 시스템 제안 X, 간단하고 오버헤드가 크지만 효과적으로 sharing 을 측정)
- **Scanning**
  - 연속적으로 memory contents 를 읽고 → 각 page hashing → 중복 검사할 hash list 출력
    중복된 페이지는 중복된 hash 를 가지니까 중복검사하기 쉬움
- **Snapshotting**
  - scanning 하는 동안 메모리가 변하지 않도록 VM 을 pausing.
  - suspend VM → scan the memory snapshot → resume : scan time 이 0 이 됨
  - memory contents 의 raw binary 를 읽어오기 때문에 원하는 크기의 page 로 나눌 수 있다.
- **Use of small deltas (Difference Engine 에서 제안)**

- page 가 조금 달라도 공유. page 크기가 매우 작아지면 거의 정확해지지만, 너무 많은 개수의 page 를 관리하는 오버헤드가 커지게 되기 전까지만 page 크기를 줄일 수 있다.

- **Zero page** 는 무시한다!
  - OS 와 app 이 페이지를 나중에 쓰기 위해서 zero out → zero page 는 많은 편.
  - sharing 에는 좋지 않다. (많긴 하지만, zero page 로 오래 남지 않을 곧 쓰일 애들)
  - Satori : zero page 로부터의 공유가 20 배나 더 많지만 매우 short-lived.
  - zero page 를 포함하면 결과가 매우 뻥튀기된다! 우린 제외할거임!!

EVALUATION

DATA COLLECTION

① Real world machine 에서 측정된 메모리 trace

- Memory Buddies project
- memory tracer 를 UMass(University of Massachusetts)의 다양한 머신에 배치하여 공개

Machine	Task	RAM
MacBook (4 individual user)	Max OS X	2GB
3 Linux Server	Web, Mail, SSH, etc…	1GB
		4GB
		8GB

- 총 7 개의 머신 / 실제 workloads / 30 분마다 memory trace / 150~350 trace per machine / total 1700 trace.

② synthetic experiments 에서 생성된 memory trace

- VM suspend → hash list 를 생성하기 위해 binary memory image 읽기
  memory tracing tool 보다 이득 (tracing 하는 동안 VM 내부의 메모리에 영향 X)
- 10 개의 VM 머신 / 각각 1.5 GB RAM

Machine	OS	총 VM 개수
Linux (32bit / 64bit)	Ubuntu 10.04	6 개
	Ubuntu 10.10	
	CentOS 5.3 (GUI 없음)	

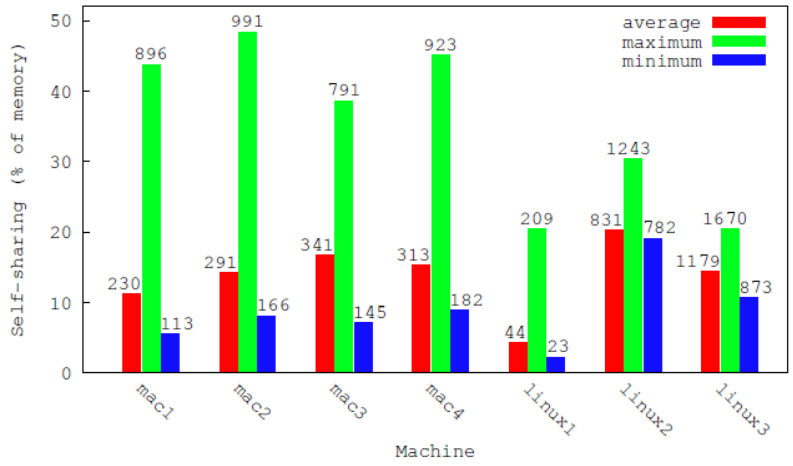
Windows	Windows XP (x86)	3 개
	Windows 7 x86	
	Windows 7 x64	
Mac	Max OS X 10.6 Server (Snow Leopard)	1 개

- 3 가지 application setups
  - No applications : 막 부팅해서 어떤 software 도 실행하지 않음
  - Server applications : 전형적인 LAMP(Linux, Apache2, MySQL5, PHP 5)
    웹이나 서버 운영에 자주 같이 쓰이는 소프트웨어
    static / dynamic page 를 섞어서 제공
  - Desktop applications : 웹브라우저(firefox), 오피스(OpenOffice), 이메일, 미디어 플레이어…
    웹 페이지 열고 문서도 열고 media file 재생하고…
- 부팅 → workload 실행 → memory snapshot
- 총 28 개의 trace (CENTOS 는 GUI 가 없어서 desktop 못돌림)

SHARING IN REAL-WORLD TRACES

- 각각의 7 개의 머신에 대해 sequential trace 를 진행 → 공유 가능성의 min/max/avg 계산
- zero page 는 배제되었음.
  - zero page 로부터의 sharing 은 전체의 5% 미만

① Self Sharing



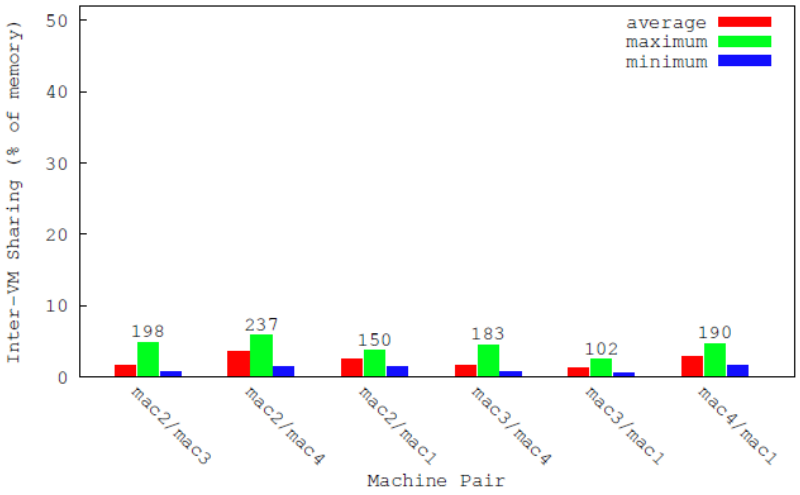
① 평균 14%의 공유 메모리 → 하나의 머신 내에서도 상당한 중복 메모리가 존재한다

② 최대 공유와 최소 공유의 차이가 큼 → 시간에 따른 공유의 변화폭이 크다

③ 평균은 max 보다 min 에 더 가깝다 → 짧고높은 공유 가능성이 길고 낮은 공유가능성 사이에있다.

④ 리눅스보다 MAC 의 공유 variation 이 더 크다
→ 고정된 목적의 서버보다 desktop 머신이 더 넓은 application 범위를 가지기 때문이다.

② Inter-VM Sharing



- (M1, M2) 고르기 : 7 개의 Machine 이 있으니까, 21 가지 set
- (Max/Linux) 와 (Linux/Linux) 조합에서 inter-VM sharing 은 항상 1% 미만(보통 0.1% 미만)
- (Mac/Mac)의 경우에도, 2-3%정도이고, 6%를 넘지 않는다.
- 제일 큰 average inter-VM sharing 에서, Mac2 랑 Mac4 는 self-sharing 이 평균 10-15%이다.
  → inter-VM 이 가장 클 때에도, inter-VM 은 전체 공유의 20%정도 뿐이고 self-sharing 이 80%

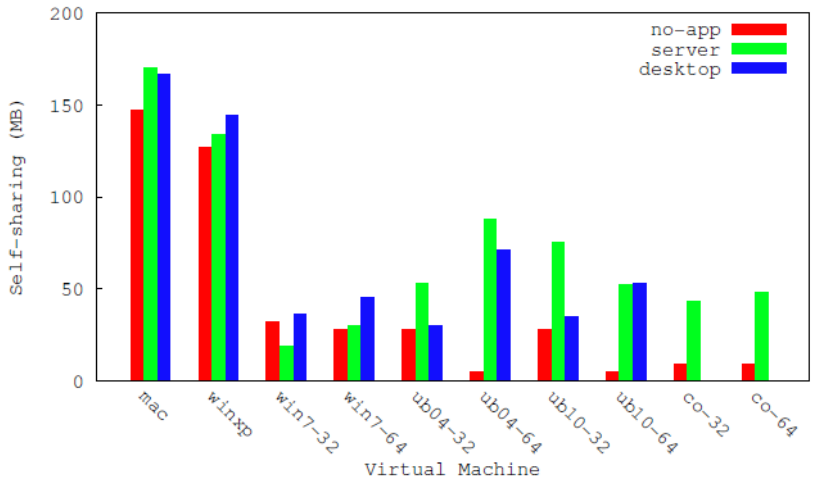
- 머신 개수가 많아지면 inter-VM sharing 이 많아진댔는데, 7 개를 동시에 돌려도 전체 메모리의 15%가 공유되고, 그 중 90%가 self, 10%미만이 inter-VM sharing.
  → memory sharing 을 위해 7 개를 동시에 돌려봤자 매우 적게 절약된다.
- page sharing 은 Xen 이나 VMware 같은 hypervisor-level 시스템에 의해서만 관리될 것이 아니라, Window/Linux 같은 상용 운영체제에서 구현되면 더욱 유용할 것이다.
  → regular home user 가 이득 + VM 이 memory 사용도 더 잘 관리할 수 있다.

PLATFORM HOMOGENEITY

Sharing : Homogeneous platform > Heterogeneous platform

- Degree of Homogeneity
  - Windows 7 은 Ubuntu Linux 보다는 Windows XP 와 더 비슷할 것이다.
  - Windows 7 X86 은 Windows XP x86 과 Windows 7 X64 중 뭐랑 더 가깝나?
  - Operating system 과 User application 중 뭐가 더 중요한가?

① Self-Sharing



- Max OS X 와 Windows XP 가 다른 것들보다 sharing 이 매우 크다.
  → 내부 메모리 중복이 큰 편이고, 이러한 중복을 잘 이용하고 있음
- No-app 에 application 을 더해도 크게 늘어나지는 않음
  → base system 이 많은 양의 self-sharing 을 제공하고, user-level application 에서는 작은 증가
- Ubuntu04 와 Ubuntu10 둘 다 32→ 64 로 바뀌면서 base-system(no-app) 의 sharing 감소
  → 64bit system 의 library 가 32bit 에 비해 redundancy 가 낮다.
- CentOS 는 32/64 둘다 낮은데, GUI 가 없기 때문이라고 추측

② Inter-VM Sharing

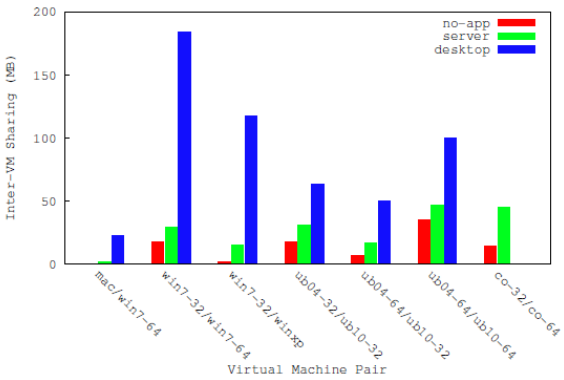


Figure 8: Inter-VM sharing between VM pairs.

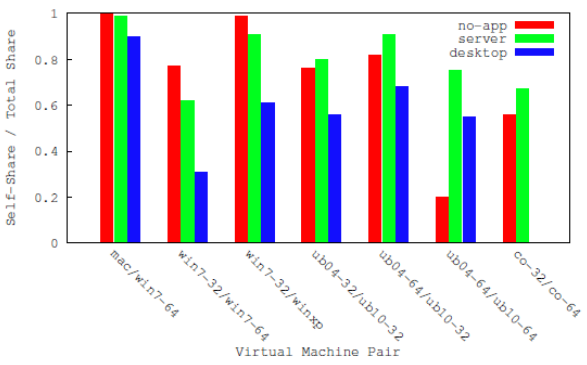


Figure 9: Self-sharing over total sharing among VM pairs.

- (Figure 8) : Inter-VM sharing 의 절대적인 크기(MB)
  (Figure 9) : 전체 sharing 에서 self sharing 이 차지하는 비율

[Major OS mixes]

- OS 자체를 섞은 것(맥, 윈도우, 리눅스)
  : base system 에서는 inter-VM sharing 이 없고, application setup 시에 조금 생겨남
  → shared resource file 때문(그래서 desktop 이 더 큼). 그 조차 10% 미만

[OS version mixes]

- base system 에서는 공유가 상당히 낮아졌지만, common application 에 대해서는 크게 감소 X
- (ex) win7-32/winxp : base system 에서는 inter-VM sharing 이 5% 미만
  application setups 시에는 상당히 증가하여, 40% 정도.
  → 많은 application 들이 같은 버전을 돌리면서 상당한 sharing 이 생김
- (ex) Ub04-32 / Ub10-32 : base system 에서는 self-sharing 이 75% 차지. app 에서 좀 감소

[Architecture mixes]

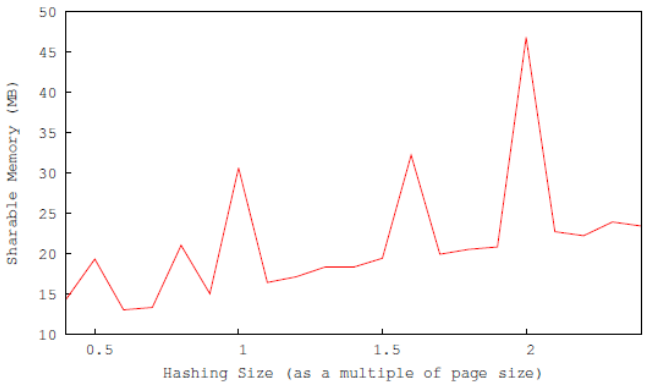
- OS 버전을 바꾸는 거랑 비슷한 경향을 보인다. major OS 버전을 바꾸는 것(대부분의 공유를 없애버림)보다는 나은 편.
- (ub04-64 / ub10-64) : 80%의 sharing 이 self 가 아니라 inter-VM 때문!!
  → 둘 다 애초에 self-sharing 이 작았기 때문이다.

[Application types]

- GUI desktop application 이 서버 application 보다 공유가 많다.
- 더 많은 memory footprint + GUI 와 관련된 라이브러리가 메모리 중복을 높인다.

VARIABLE SIZED HASHING

- 일반적으로 공유는 page 를 기준으로 한다(sharing 의 granularity 는 page 하나 전체)
- 다른 granularity 로 공유를 할 수 있다.
- sharing potential 과 efficiency 를 trade-off 하면 됨
  → 더 작은 granularity 는 sharing potential 을 높이지만, efficiency 는 더 떨어지기 때문
- sharing granularity 를 0.4 에서부터 2.4 까지 0.1 단위로 다양하게 바꿔가며 trace



- granularity 가 증가하면 sharing 도 증가한다.
- 가로 축은 hash per page.
- sharable memory / hash_per_page 는 1-page granularity 가 최대

SOURCES OF SELF-SHARING

- self-sharing 의 중요성을 보았는데, 이러한 self-sharing 은 어떻게 생기는지.

AN EXTENDED MEMORY TRACER

- Real-world Linux trace 를 모으기 위해 memory tracer 를 확장했다.
- (기존) : 단순하게 memory 의 page 에 가서, page contents 를 기준으로 hash 를 계산
  (수정) : 두 개의 정보를 추가로 더 수집
  ① 보통의 프로그램의 address space 부분(stack, heap) 인지, shared library 의 맵핑된 페이지인지
  ② 페이지를 사용하는 프로세스. shared library page 는 그 페이지를 쓰고 있는 프로세스중 어떤 것이어도 되지만, 그 외의 페이지는 하나의 프로세스만 있을 것이다.
- (ex) [libc-2.12.so 000b6000 r-xp] : sshd apache2 → libc 페이지를 SSH 와 Apache2 가 씀
  [heap] : mysqld → MySQL 의 힙 영역
- 공유 가능성의 양을 측정할 뿐 아니라, 어떤 프로세스와 라이브러리가 실제 공유에 관여하는지!

CASE STUDY : DESKTOP APPLICATIONS

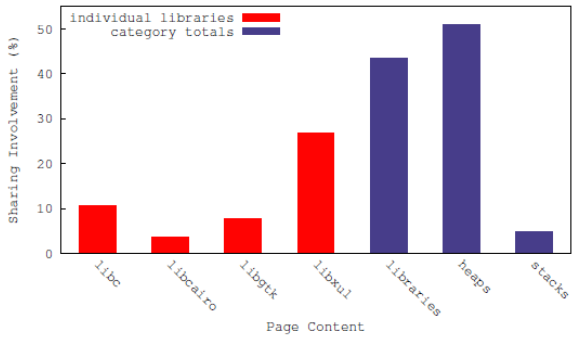


Figure 11: Sharing by content as a percentage of total sharing.

- 주로 검사한 페이지는 스택, 힙, 공유 라이브러리.
- 여러 페이지 타입이 같은 내용을 가질 수도 있다.
- 가장 공유가 많이 되는 부분은 heap page, 전체 sharing 의 50%를 차지.
  라이브러리 페이지도 바로 다음으로 43%의 공유를 차지한다.
  Stack 페이지는 5% 미만의 관여.
- Single 라이브러리는, libxul (파이어 폭스에서 사용되는 user interface)
  libc 는 더 널리 쓰이지만, 공유에 상당히 작게 포함된다.

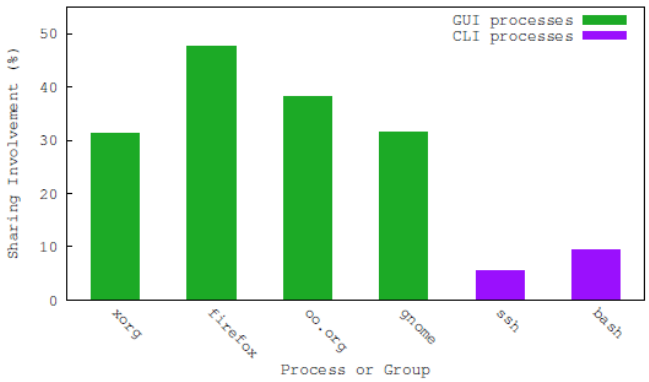


Figure 12: Sharing by process as a percentage of total sharing.

- 여러 프로세스가 하나의 페이지를 사용할 수 있으므로 not additive
- Firefox : 공유의 거의 절반에 관여. (libxul 의 중요성도 보았음)
- OpenOffice(38%)와 같은 다른 GUI application 이나 Xserver(31%)와 같이 시스템 전반에 걸친 GUI 프로세스와 Gnome-관련 프로세스(31%).

Memory Security and Sharing

- key library function 같은 알려진 주소에 있는 메모리가 덮어쓰여지는 경우 발생
- **ASLR(Address space layout randomization) :** 프로그램 자원(heap, stack, …)과 key library 의 주소가 랜덤하게 결정되는 것. → 공격자가 알려진 주소를 사용하는 것을 방지.

- memory 의 content 를 변경하기 때문에 page 의 공유 가능성에 영향.
  똑같은 software 를 돌리는 똑 같은 VM 은 ASLR 이 켜있으면 다른 결과를 보인다.

CURRENT ASLR SUPPORT

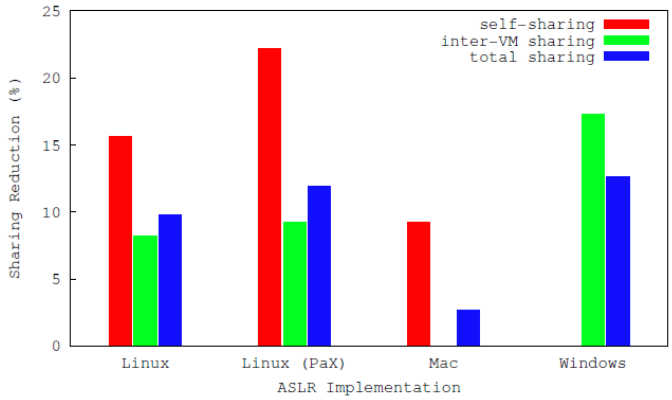
- 4 개의 implementations : Linux 2 개, Windows 7, Mac OS X Lion
- **Linux**
  - 프로세스의 중요한 부분(라이브러리 위치, stack, heap, code) 을 랜덤화
  - /proc 인터페이스를 통해서 시스템의 ASLR 을 끄고 켤 수 있다.
  - heap 랜덤화만 안 일어나고 나머지는 일어나게 할 수 도 있다.
- **PaX**
  - ASLR for Linux 는 PaX 를 통해 제공된다. (보안을 위해 패치됨)
  - 스탠다드 리눅스 구현에서는 제공되지 않는 몇가지 기능(커널 자신의 랜덤화)을 제공한다
- **Windows**
  - Windows vista 부터. stack, heap, DLLs 을 랜덤화
  - application 마다 실행할 수 있고, MS 는 최근 특정 프로세스에 대해 강제로 ASLR 을 켜고 끌 수 있는 유틸리티를 제공함.
  - default 로 opt-in 되지 않은 application 에 enable 해도 문제는 없다.
- **Mac OS X**
  - randomization 을 끌 방법이 없다. 특정 POSIX flag 설정을 사용해야 한다.

EVALUATING ASLR'S SHARING IMPACT

- Randomization 의 영향과 그 정도를 알아보자.
- 여러 VM 이 동일한 base image 로부터 부팅
  → Host : Ubuntu 10.10 64-bit (Standard / PaX , 2.6.32), Windows7 64bit, Max OS X 10.7

① ASLR 끄기(Mac 은 system 의 randomization 못끔)
② 재부팅-메모리를 안정적 상태로 리셋
③ application list 열기(웹 브라우저, 텍스트 에디터, 오피스, 음악 등등) by shell script 나 bat
④ 메모리가 안정되면, 메모리 스냅샷 → NOT-randomized 스냅샷 찍기
⑤ 다시 ASLR 켜지
⑥ 재부팅
⑦ 스냅샷 절차 반복 → randomized 스냅샷 찍기

총 4 번 반복, 평균.



- 모든 경우 감소했다.
  Windows 7 에서, Total(파란색) 13% 가 감소했고, Mac 은 3% 감소했지만, 시스템 자체의 randomization 을 끄지 못했으니 실제보다 작게 잡힌 셈.
  Linux 에서는 10%감소 했고, PaX 에서는 12% 감소했다.

- Linux 에서는 self 와 inter 모두 감소했지만,
  Window 에서는 Self 는 그대로이고, inter-VM sharing 만 감소했다. → 가상화가 아닌 개별 머신에서 윈도우는 ASLR 에 전혀 영향을 받지 않는다는 뜻.
  Mac 에서는 반대로, 전부 self-sharing 때문. → Mac 은 self-sharing 이 많기 때문이라고 추측