

SIGSS Meeting 2006/06/20

M1

大住裕之

Antfarm: Tracking Processes in a Virtual Machine Environment

Stephen T. Jones et al.

USENIX '06

論文の概要

- VMM層からguest OS内のprocessの挙動を推測する方法(Antfarm)を提案し、評価した
- 提案手法を用いて、processの情報を利用する I/O schedulerをVMMに実装し、性能評価した

Process Inference

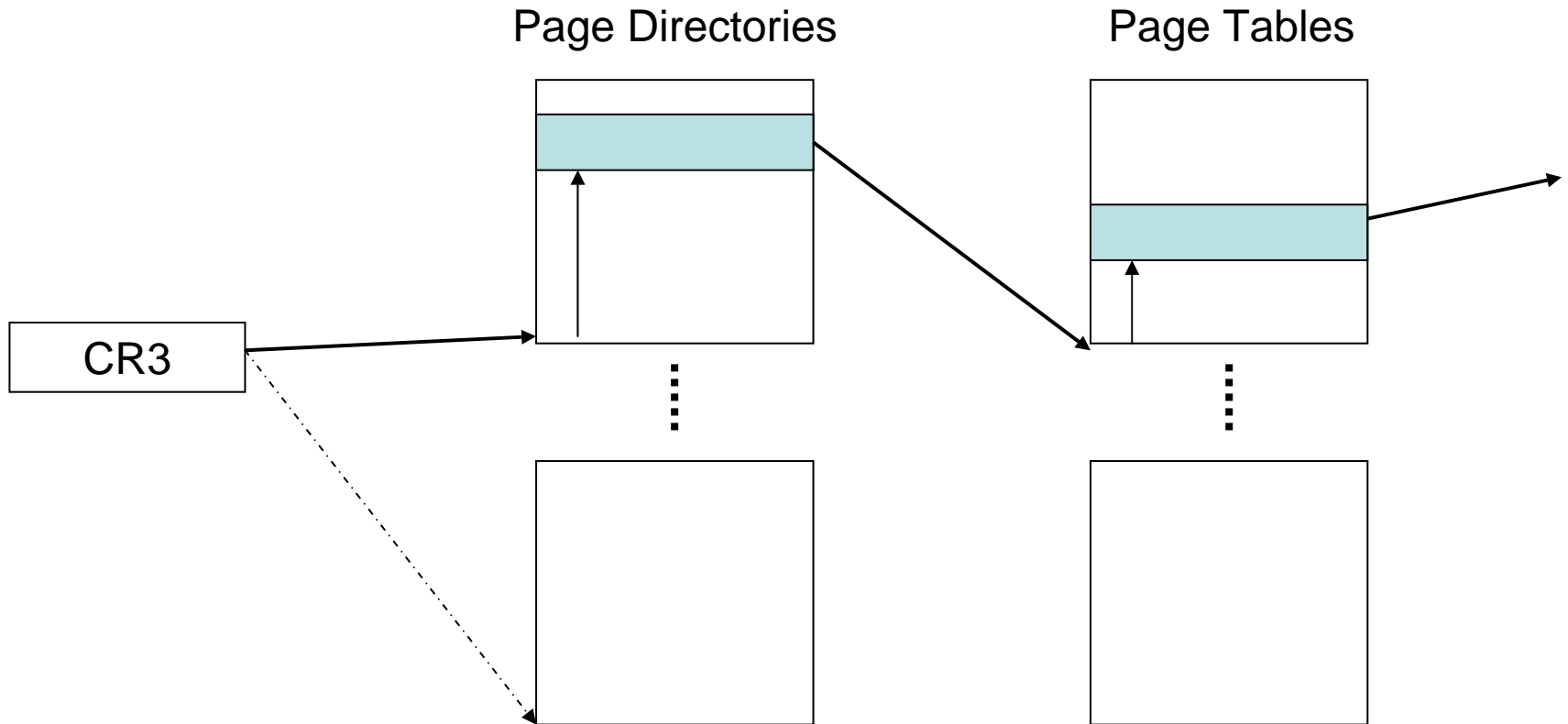
- VMMからprocessを直接見ることはできない
- 仮想アドレス空間への操作を観察することで対応するprocessの挙動を推測する
 - Creation
 - Exit
 - Context Switch

ASID (Address Space Identifier)

- 仮想メモリ空間を区別するための値
 - VMMから取得できる値
 - アーキテクチャによって異なる
 - Page directory tableの物理アドレス (x86)
 - Virtual address context ID (SPARC)

Techniques for x86

- x86でのページング



Creation and Context Switch

- ASID = page directory の物理アドレス
 - Antfarmは現在有効なASIDのリストを保持
- Creation
 - CR3レジスタに新しい値が書き込まれたとき
- Context Switch
 - CR3レジスタに既知の値が書き込まれたとき

Exit

- Page tableの有効なentry (mapping)数が0になったとき
 - OSはpage directory / page table pageを再利用する前に、clearする/しなければならない
- and
- TLB flushがかかったとき
 - 存在しないアドレス空間のentryが残っていてはならない
 - CR3レジスタへの書き込みを監視

Techniques for SPARC

- ASID = Context ID
- Creation and Context Switch
 - Context ID操作は特権命令なので監視している
- Exit
 - Context demap operationを監視するだけでよい

評価

- Completeness
 - Processのcreation, exit, context switchを正しく検知できるか
- Lag
 - Process で上記のeventが起こった時間とVMMがそれを検知した時間の差
- Overhead

x86 Evaluation

- 以下のOSについて実験
 - Linux 2.4.30 on Xen 2.0.6
 - Linux 2.6.11 on Xen 2.0.6
 - Windows NT 4 on Simics/x86
- ハードウェア構成
 - CPU: Pentium 4 2.4 GHz
 - RAM:512 MB
 - 128 MB / virtual machine
 - 245 MB / Simics

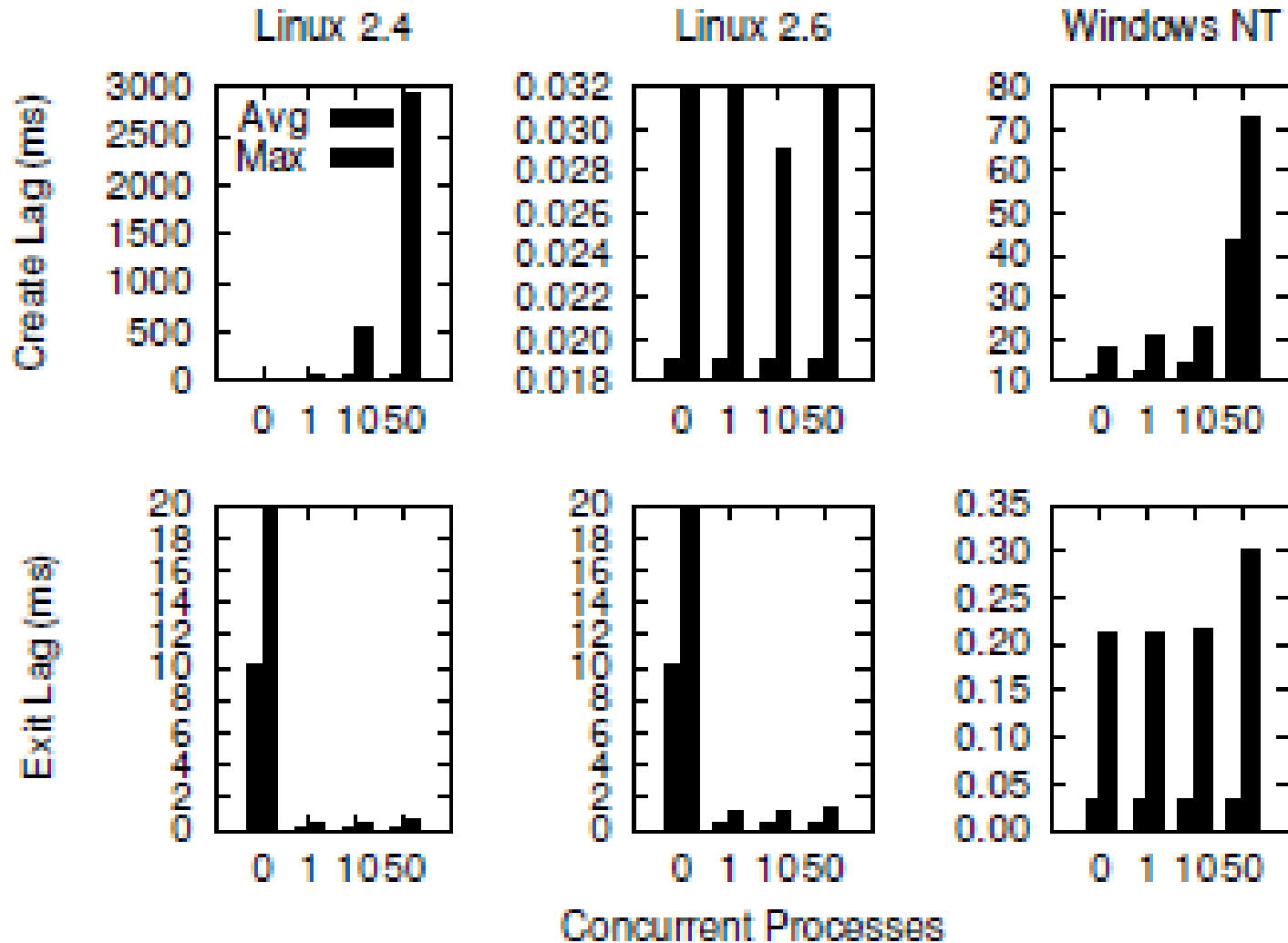
Completeness [Table 1]

- Linux 2.4 と Windows では全て検知
- Linux 2.6 では false positive が多発
 - fork, execシステムコールの実装に因る
 - Execしたとき、メモリ空間を一度破棄して新たに確保する
 - Antfarmからは2つの異なるプロセスとして見えることになる
 - User processの動きとしては、全て捉えられている (アドレス空間が存在しない間にuserの命令が実行されることはない)

Lag

- システムの負荷を変えて、以下の環境で実験
 - Linux 2.4
 - Linux 2.6
 - Windows NT 4
- 測定基準
 - Create: forkが呼び出されたとき
 - Exit: exitが呼び出されたとき

Lag vs. System Load



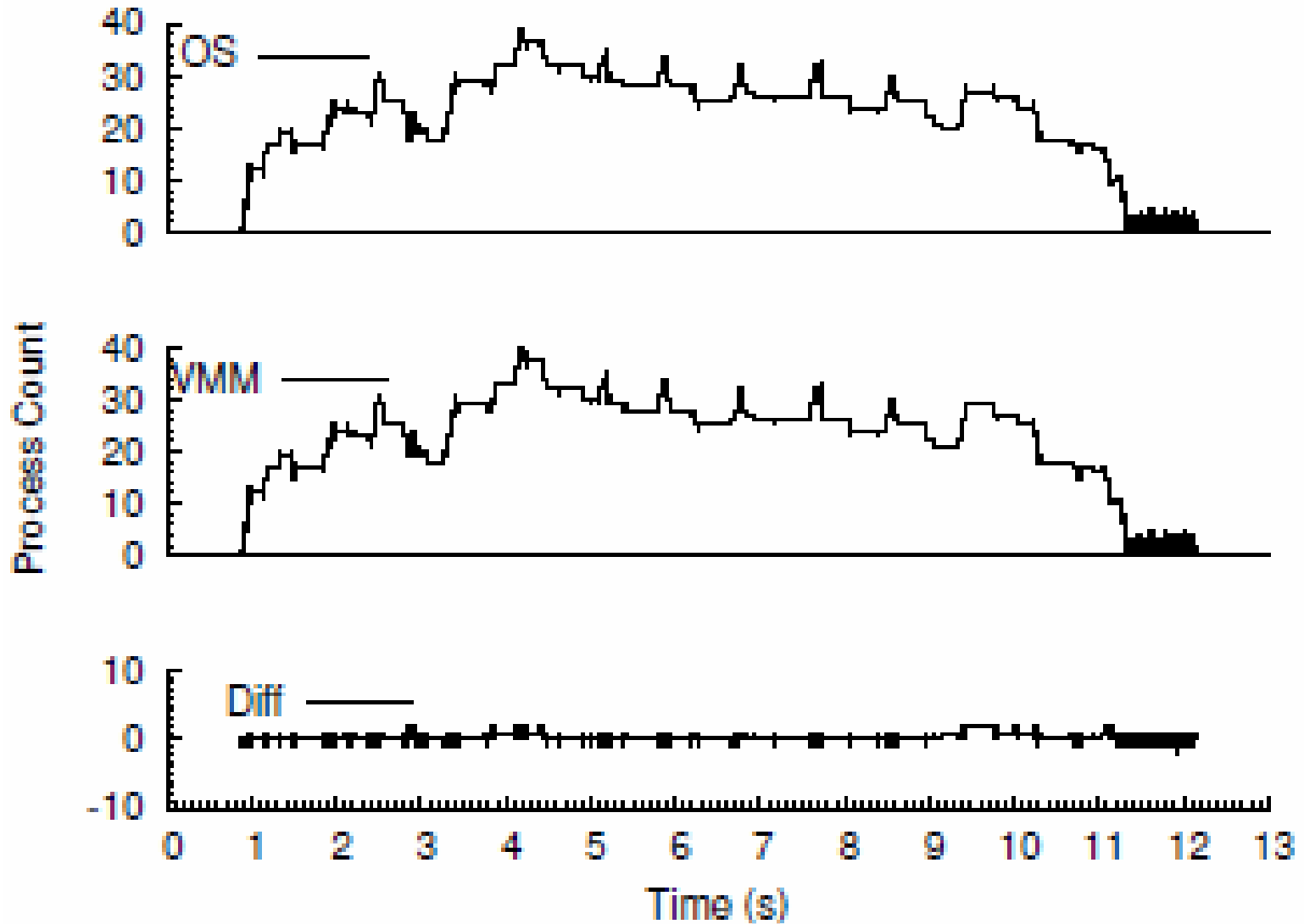
Lag

- Create lagはシステム負荷に強く影響される
 - forkシステムコールの呼び出しから、仮想メモリ空間生成までの遅延が大きくなるため
 - Linux 2.6では小さく、かつ一定となっている
- Exit lagは大体小さいといえるが、OSの実装によって例外がある
 - Linux でのkernel task

Overhead

- Microbenchmark
 - 100 MBのメモリをallocate, すべてのpageにアクセスしてpage table管理部分に負荷をかける
 - 2.4 %の速度低下
- Bashのビルド
 - 多数の平行processが生成・消滅する例
 - 0.6 %の速度低下

Linux 2.6 Bash Compile



SPARC Evaluation

- OS
 - Linux 2.4.14 on Simics
- 仮想マシン構成
 - CPU: UltraSPARC II 168MHz
 - RAM:256 MB

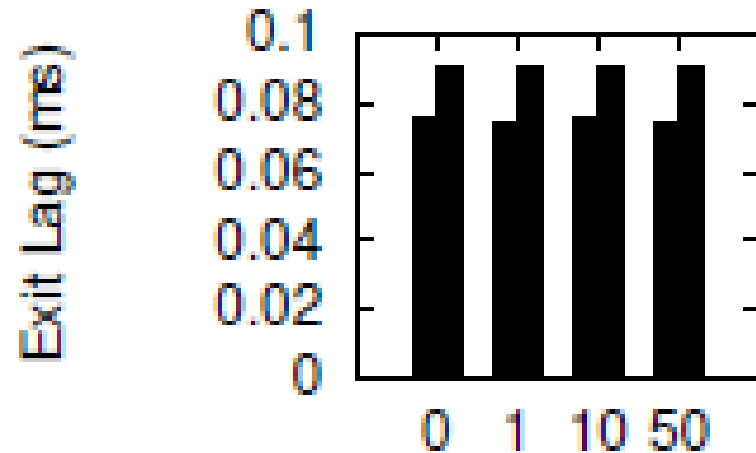
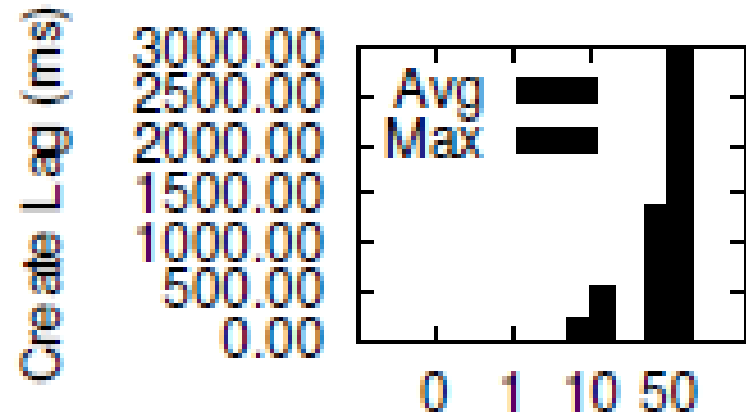
Completeness [Table 2]

- x86と同様にfalse negativeはない
- forkのみでもfalse positiveが発生している
 - context demap operationが誤ってexitとして解釈されている?
 - forkを呼ぶ度に1つ多くprocess数を推定してしまうことになる
 - GNU make, gccはvforkを呼ぶのでCompileではそれほど影響がでていない

Lag

- Createはsystem loadに強く影響される
- Exitはsystem loadに関係なくほぼ一定
- x86と同様の傾向

SPARC-Linux 2.4



ここまでのまとめ

- Antfarmはprocessの生成・消滅・切替というイベントを正確に捉えられている
- False positiveが発生しているが、パターンは決まっていて、processのカウント精度にほとんど影響を与えない

Case Study: Anticipatory scheduling

- 概要

- XenにAnticipatory schedulingを実装し評価
- VMMに実装するのが有効であり、かつguest OSのprocessの挙動を推定することで可能になる例

Anticipatory Scheduling

- Disk I/O scheduler
- 従来のschedulerはrequestを到着順に実行
 - 複数のプロセスが連続的にアクセスするとき、seekの占める時間が増大してしまう
- そこで、次のrequestを処理する前に、「少し」待ってみる
 - Seekを減らして、結果的に速く処理できる可能性

Anticipatory Scheduling

- Anticipatory schedulerは各processの過去の履歴を元に次のアクセスを予想する
- したがってVMM上の複数のVMが存在する環境下ではうまく機能しない
 - 他のVMのアクセスは検知できず、混在すると無意味

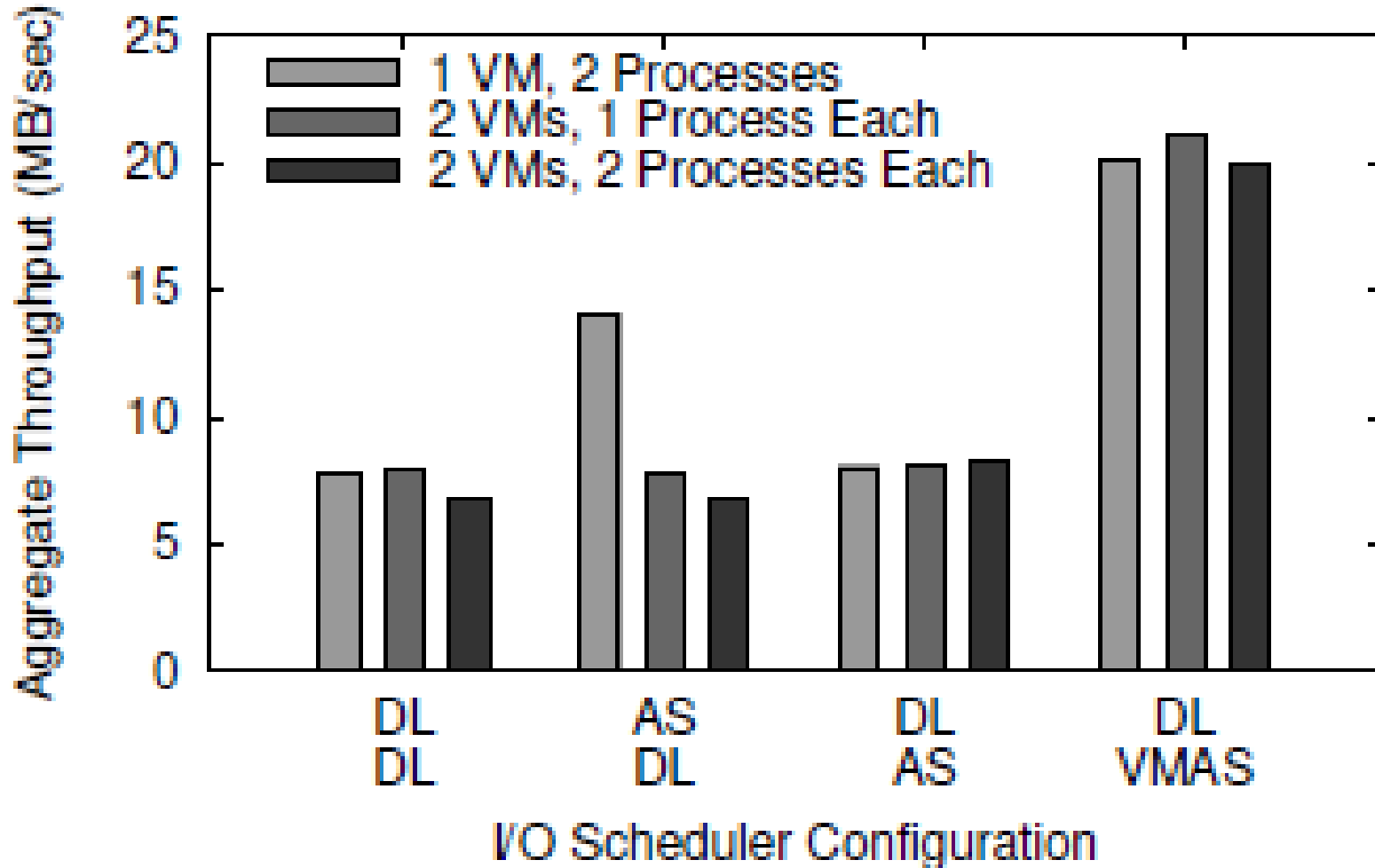
Anticipatory Scheduling

- ならばVMMの層で行ってやればよい...が、processを特定することが必要になる
- そこでAntfarmでprocessを推定することで実装
 - diskへのアクセスとprocessとの対応をつける

Evaluation

- 1GBのファイルから200MBずつsequential readするプログラム
- プロセス数、VM数、VMのI/O scheduler, VMMのI/O schedulerを変更して、全体のスループットを測定

Comparison of VM Layer I/O Schedulers



Conclusion

- VMMにOSのような機能を実装するのは困難
 - OSやアプリケーションといったhigh-levelの情報を得ることができない
- AntfarmはVM上のOS内のprocessの挙動を正確に推定する
 - 専用interfaceをつくる方法に比べて、guest OSのバージョン等への依存がない

References

- Simics: A Full system Simulation Platform
 - Peter S. Magnusson et al. IEEE Computer 35(2), 2002
- Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O
 - Sitaram Iyer, Peter Druschel SOSP '01
- 図表は論文より引用

Table 1-a

	Process Create	Addr Spc Create	Inferred Create
Linux 2.4 x86			
Fork only	1000	1000	1000
Fork + Exec	1000	1000	1000
Vfork + Exec	1000	1000	1000
Compile	815	815	815
Linux 2.6 x86			
Fork only	1000	1000	1000
Fork + Exec	1000	2000	2000
Vfork + Exec	1000	1000	1000
Compile	748	1191	1191
Windows			
Create	1000	1000	1000
Compile	2602	2602	2602

Table 1-b

	Process Exit	Addr Spc Exit	Inferred Exit
Linux 2.4 x86			
Fork only	1000	1000	1000
Fork + Exec	1000	1000	1000
Vfork + Exec	1000	1000	1000
Compile	815	815	815
Linux 2.6 x86			
Fork only	1000	1000	1000
Fork + Exec	1000	2000	2000
Vfork + Exec	1000	1000	1000
Compile	748	1191	1191
Windows			
Create	1000	1000	1000
Compile	2602	2602	2602

	Context Switch	CS Inferred
Linux 2.4 x86		
Fork only	3331	3331
Fork + Exec	3332	3332
Vfork + Exec	3937	3937
Compile	4447	4447
Linux 2.6 x86		
Fork only	3939	3939
Fork + Exec	4938	4938
Vfork + Exec	3957	3957
Compile	2550	2550
Windows		
Create	74431	74431
Compile	835248	835248

Compile:

a parallel compile
of the bash shell
source using
“make -j 20”

Table 2-a

	Process Create	Addr Spc Create	Inferred Create
SPARC/Linux			
Fork Only	1000	1000	2000
Fork & Exec	1000	1000	3000
Vfork	1000	1000	1000
Compile	603	603	1396

Table 2-b

	Process Exit	Addr Spc Exit	Inferred Exit
SPARC/Linux			
Fork Only	1000	1000	2000
Fork & Exec	1000	1000	3000
Vfork	1000	1000	1000
Compile	603	603	1396

	Context Switch	CS Inferred
SPARC/Linux		
Fork Only	3419	3419
Fork & Exec	3426	3426
Vfork	4133	4133
Compile	1678	1678