# 存储服务质量保障专题

施展 2020

# 课程信息

- 专题内容
  - 关于虚拟化环境存储系统的研究
    - 一些基础
      - https://github.com/cs-course/vagrant-tutorial
      - https://github.com/cs-course/vagrant-presentation
      - https://github.com/cs-course/minidc-tutorial
    - 需要研究的问题
      - 存储服务质量保障

# 授课目标

- 研究对象
  - 虚拟化环境存储系统
    - 理解虚拟化环境下存储系统的I/O问题
    - 如何在虚拟化环境下保障存储服务质量

# 做个调查

- 单纯只是安装使用过
  - VirtualBox、VMware、Parallels Desktop
  - XEN、KVM、Qemu
- 学习过虚拟化、云计算课程
  - 云计算与虚拟化 24/1.5（4年级）
- 熟悉Linux且买过云主机
  - VPS（Virtual Private Server 虚拟专用服务器）

# 第一部分

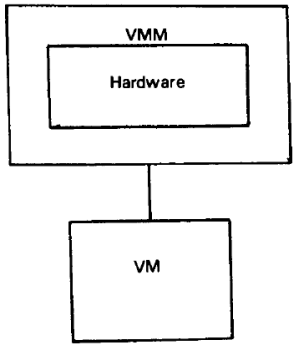
# 虚拟化环境

# 带上疑问

- 何谓虚拟化？
- 有什么用？
- 有什么不足？
- 有什么方法克服那些不足？

# 起源

Fig. 1. The virtual machine monitor.




**"an *efficient, isolated duplicate* of the real machine"**

- **Efficiency**
  - **Innocuous instructions should execute directly on the hardware**
- **Resource control**
  - **Executed programs may not affect the system resources**
- **Equivalence**
  - **The behavior of a program executing under the VMM should be the same as if the program were executed directly on the hardware (except possibly for timing and resource availability)**

## Formal Requirements for Virtualizable Third Generation Architectures

Gerald J. Popek
University of California, Los Angeles
and
Robert P. Goldberg
Honeywell Information Systems and
Harvard University

**Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.**

Communications of the ACM, vol 17, no 7, 1974, pp.412-421

# 历史

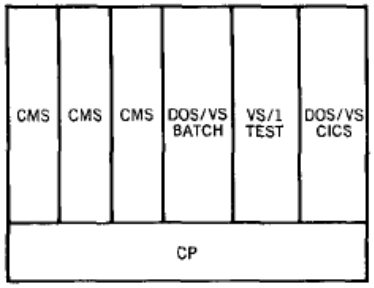
## VM/370—a study of multiplicity and usefulness

by L. H. Seawright and R. A. MacKinnon


The productivity of data processing professionals and other professionals can be enhanced through the use of interactive and time-sharing systems. Similarly, system programmers can benefit from the use of system testing tools. A systems solution to both areas can be the virtual machine concept, which provides multiple software replicas of real computing systems on one real processor. Each virtual machine has a full complement of input/output devices and provides functions similar to those of a real machine. One system that implements virtual machines is IBM's Virtual Machine Facility/370 (VM/370).[1]

**IBM Systems Journal, vol. 18, no. 1, 1979, pp. 4-17.**

Figure 1 A VM/370 environment

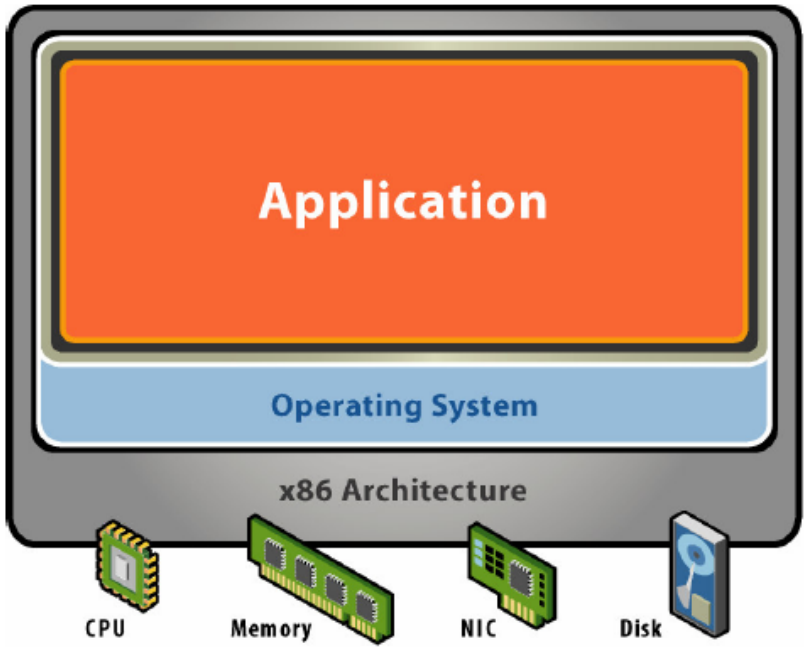



- Concurrent execution of multiple production operating systems
- Testing and development of experimental systems
- Adoption of new systems with continued use of legacy systems
- Ability to accommodate applications requiring special-purpose OS
- Introduced notions of "handshake" and "virtual-equals-real mode" to allow sharing of resource control information with CP
- Leveraged ability to co-design hardware, VMM, and guestOS

# 虚拟化

- Virtualization deals with
  - “extending or replacing an existing interface so as to mimic the behavior of another system”
- Virtual system examples
  - Virtual Private Network
  - Virtual Memory
  - Virtual Machine

# 从物理机开始



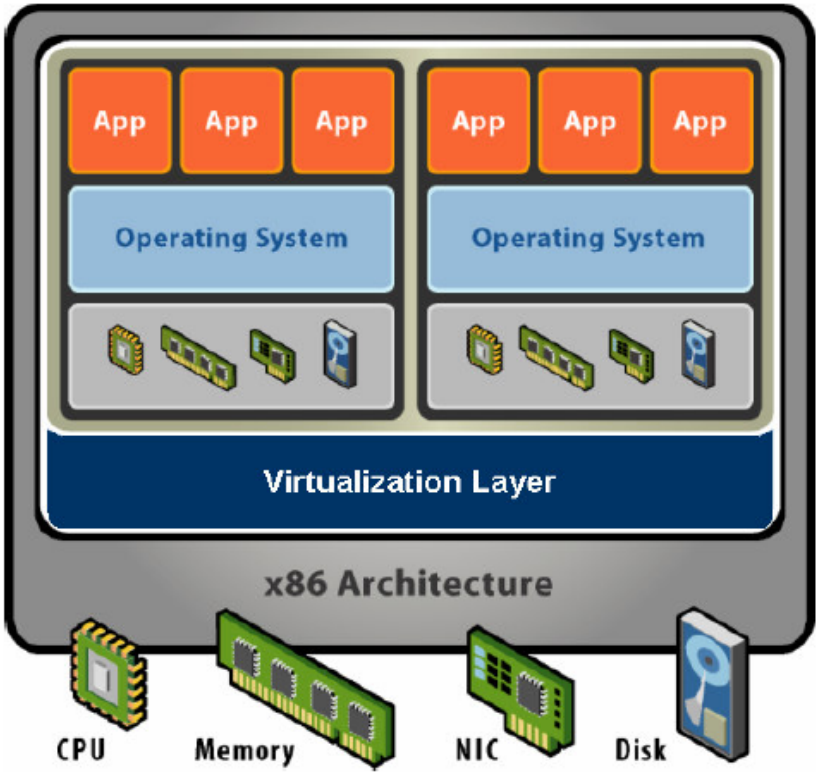

## Physical Hardware

- Processors, memory, chipset, I/O bus and devices, etc.
- Physical resources often underutilized

## Software

- Tightly coupled to hardware
- Single active OS image
- OS controls hardware

# 何谓虚拟机？



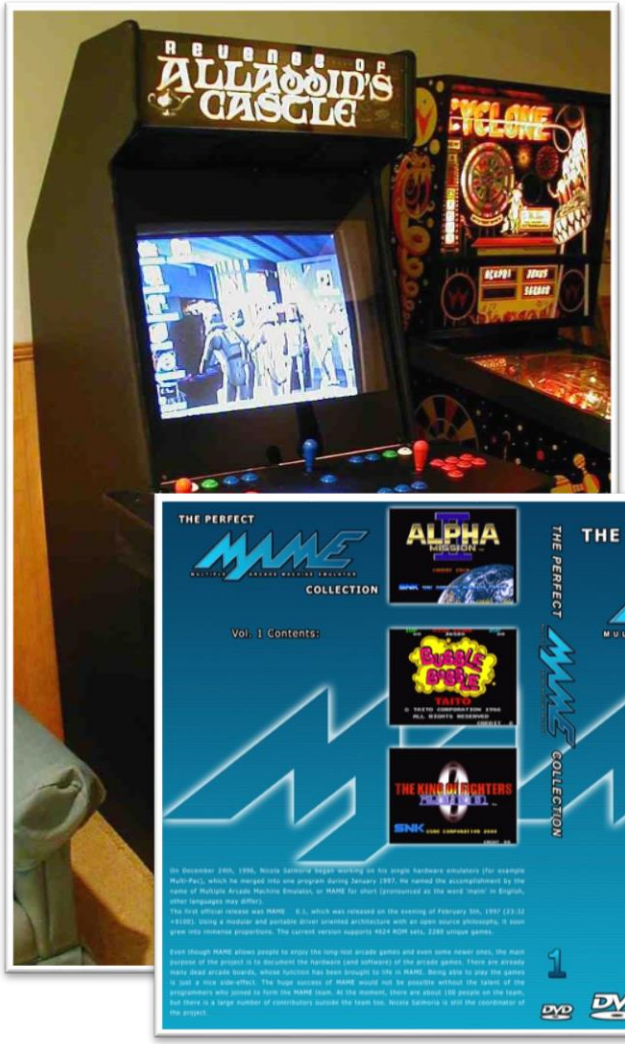

## Hardware-Level Abstraction

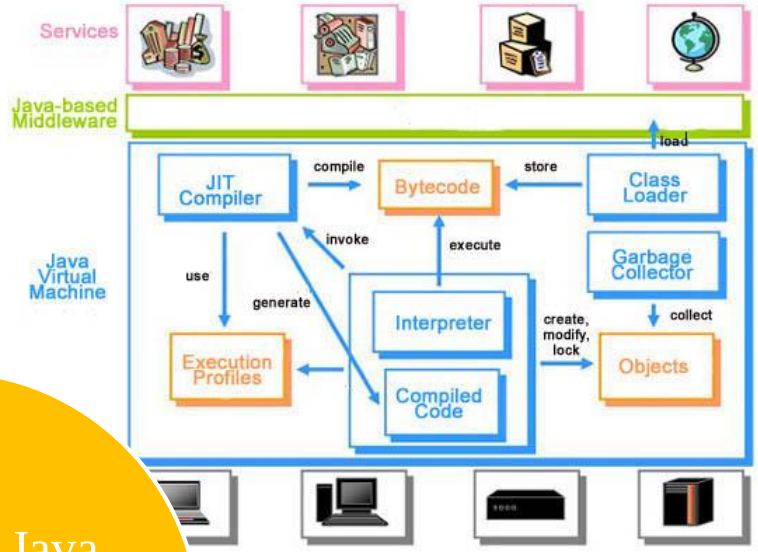
- Virtual hardware: processors, memory, chipset, I/O devices, etc.
- Encapsulates all OS and application state

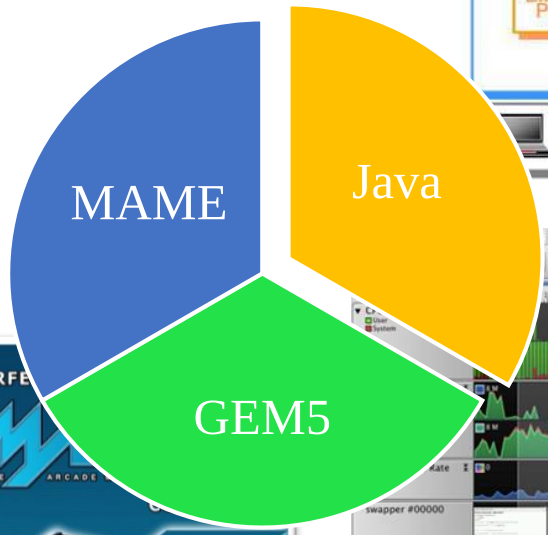
## Virtualization Software

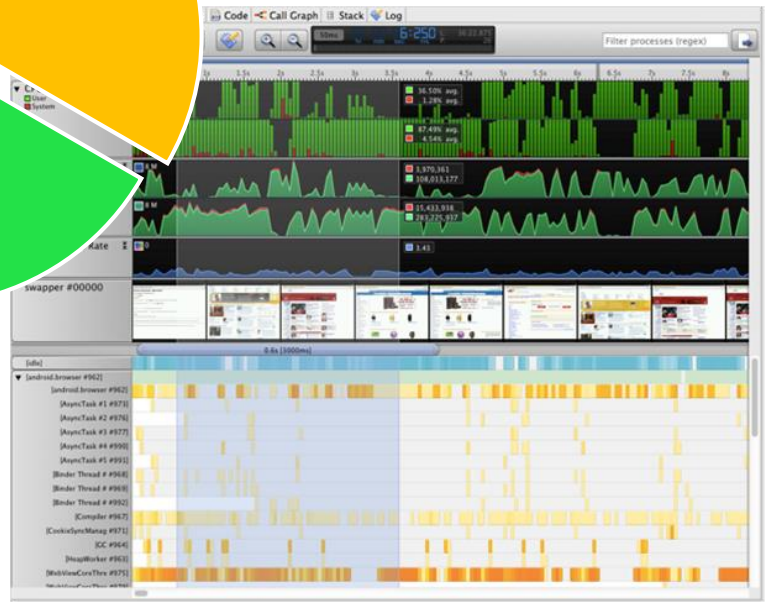
- Extra level of indirection decouples hardware and OS
- Multiplexes physical hardware across multiple “guest” VMs
- Strong isolation between VMs
- Manages physical resources, improves utilization

# 何谓虚拟机？


REVENGE OF
ALLADDIN'S
CASTLE
THE PERFECT
MAME
COLLECTION
Vol. 1 Contents:
ALPHA
BUBBLE
BOBBLE
TAITO
THE KING OF FIGHTERS
SNK
DVD


Services
Java-based
Middleware
load
JIT
Compiler
compile
Bytecode
store
Class
Loader
Java
Virtual
Machine
invoke
execute
use
generate
Interpreter
Garbage
Collector
collect
create,
modify,
lock
Execution
Profiles
Compiled
Code
Objects


MAME
Java
GEM5


Code
Call Graph
Stack
Log
swapper #00000

# 隔离

## Secure Multiplexing

- Run multiple VMs on single physical host
- Processor hardware isolates VMs, *e.g.* MMU

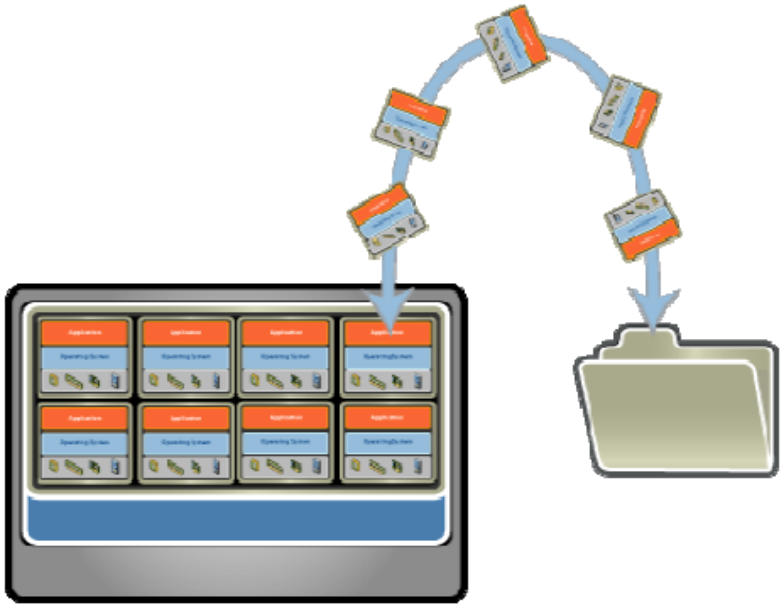
## Strong Guarantees

- Software bugs, crashes, viruses within one VM cannot affect other VMs

## Performance Isolation

- Partition system resources
- Example: VMware controls for reservation, limit, shares

# 封装



Entire VM is a File

- OS, applications, data
- Memory and device state

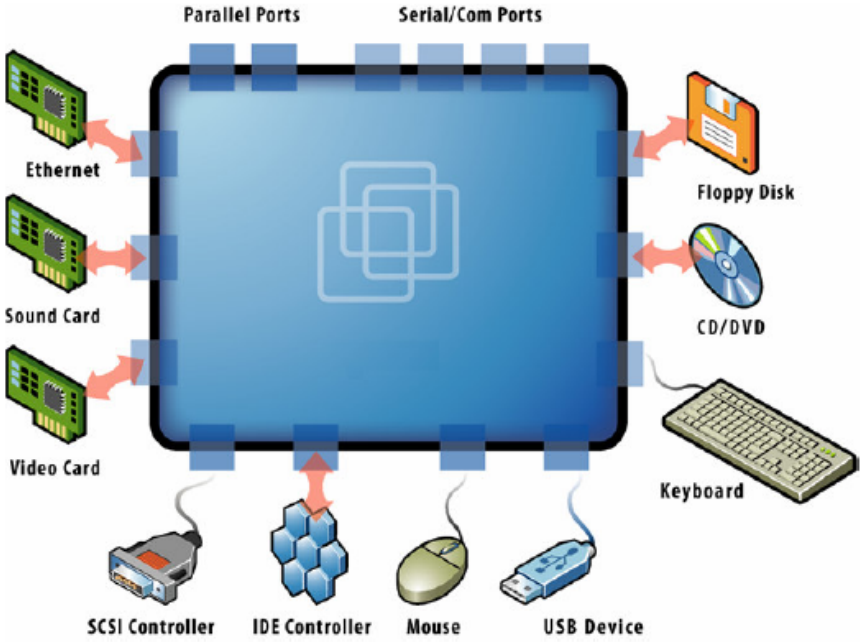
Snapshots and Clones

- Capture VM state on the fly and restore to point-in-time
- Rapid system provisioning, backup, remote mirroring

Easy Content Distribution

- Pre-configured apps, demos
- Virtual appliances

# 兼容




## Hardware-Independent

- Physical hardware hidden by virtualization layer
- Standard virtual hardware exposed to VM

## Create Once, Run Anywhere

- No configuration issues
- Migrate VMs between hosts

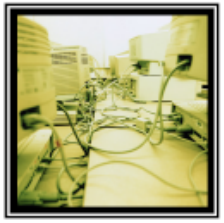
## Legacy VMs

- Run ancient OS on new platform
- *E.g.* DOS VM drives virtual IDE and vLance devices, mapped to modern SAN and GigE hardware

# 用途

**Test and Development** – Rapidly provision test and development servers; store libraries of pre-configured test machines

**Business Continuity** – Reduce cost and complexity by encapsulating entire systems into single files that can be replicated and restored onto any target server



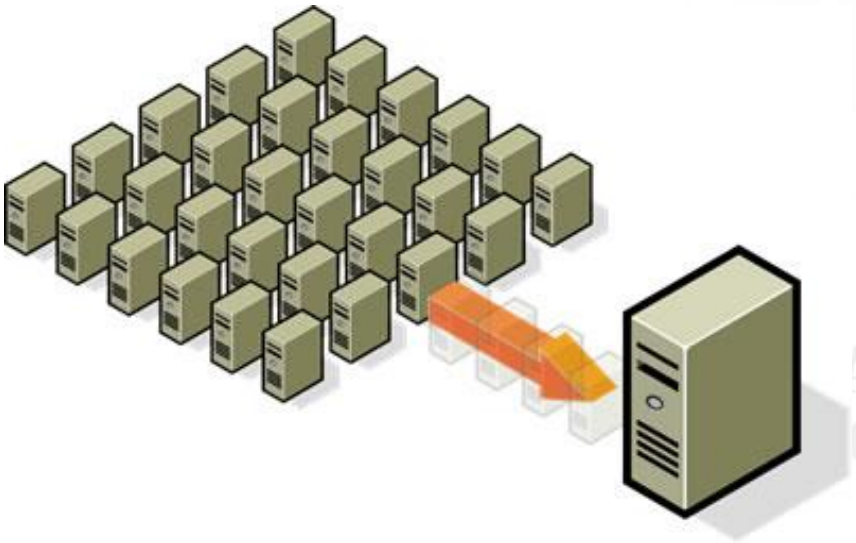
**Enterprise Desktop** – Secure unmanaged PCs without compromising end-user autonomy by layering a security policy in software around desktop virtual machines

# 用途

- Run legacy software on non-legacy hardware
- Run multiple operating systems on the same hardware
- Create a manageable upgrade path
- Manage outages (expected and unexpected) dynamically

# 用途

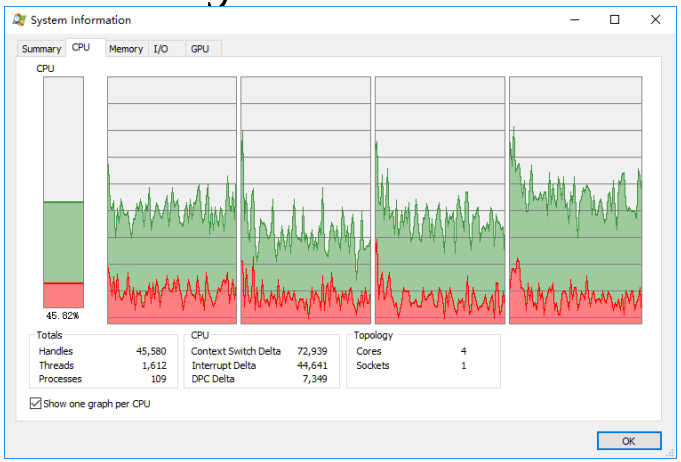
- Reduce costs by **consolidating 整合** services onto the fewest number of physical machines

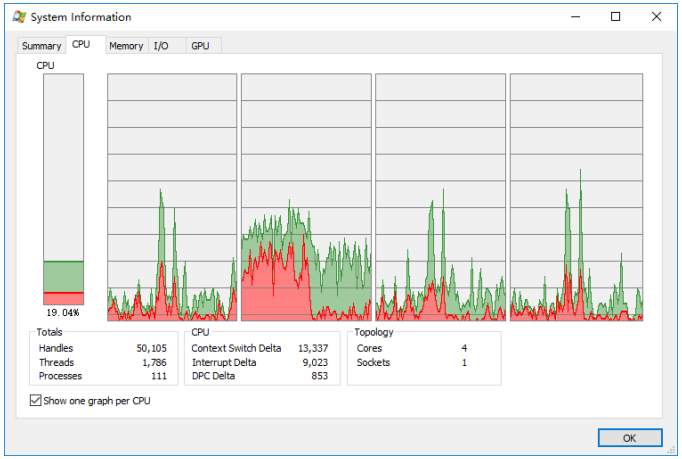


http://www.vmware.com/img/serverconsolidation.jpg

# What IF Non-Virtualized

- Too many servers/cores for too little work



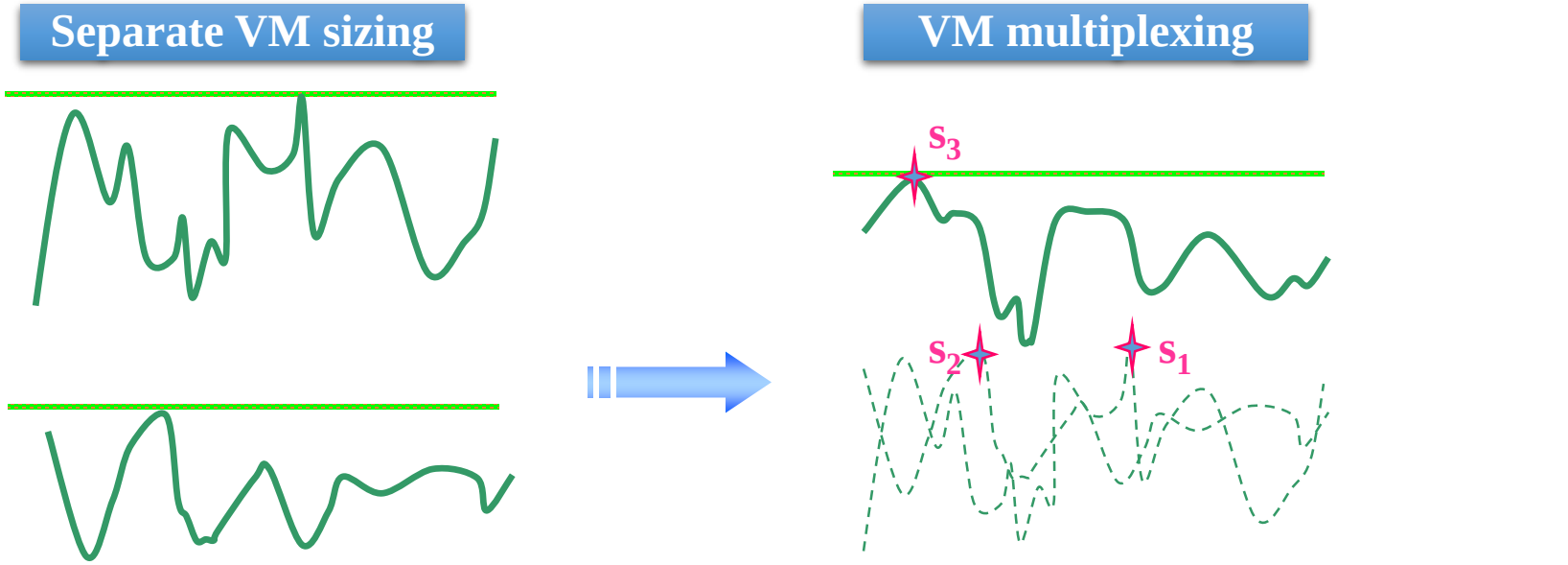




- High costs and infrastructure needs
  - 维护 Maintenance
  - 网络 Networking
  - 空间 Floor space
  - 冷却 Cooling
  - 能耗 Power
  - 容灾 Disaster Recovery

# Dynamic

- Virtualization helps us break the "one service per server" model
- Consolidate many services into a fewer number of machines when workload is low, reducing costs
- Conversely, as demand for a particular service increases, we can shift more virtual machines to run that service
- We can build a data center with fewer total resources, since resources are used as needed instead of being dedicated to single services

# VM workload multiplexing




*We expect $s_3 < s_1 + s_2$. Benefit of multiplexing !*

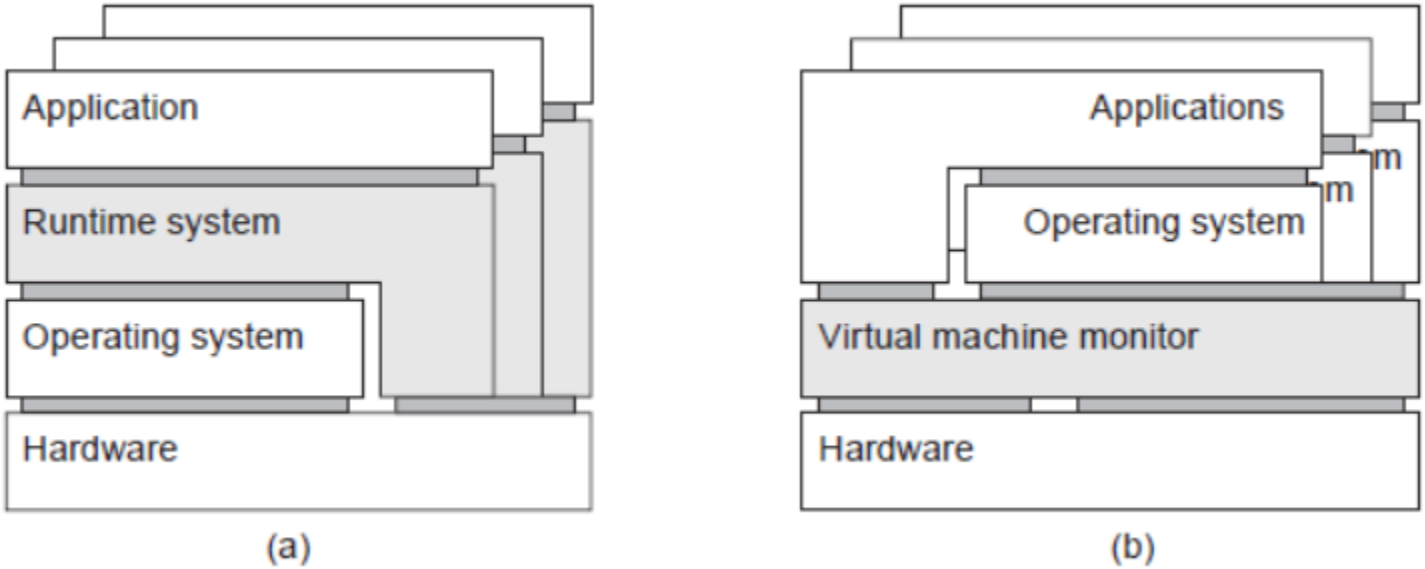
- 工作负载复用
  - Multiplex VMs' workload on same physical server
  - Aggregate multiple workload. Estimate total capacity need based on aggregated workload
  - Performance level of each VM be preserved

# 虚拟机管理器

A virtual machine is taken to be an *efficient, isolated duplicate* of the real machine. We explain these notions through the idea of a *virtual machine monitor* (VMM). See Figure 1. As a piece of software a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.


- Java 虚拟机？
- 显然不是，Java 虚拟机以及各类高级语言虚拟机目标不在于模拟各类机器部件设备
- 虚拟机管理器需要从 CPU 指令层面准确模拟各种硬件

# 两种虚拟机



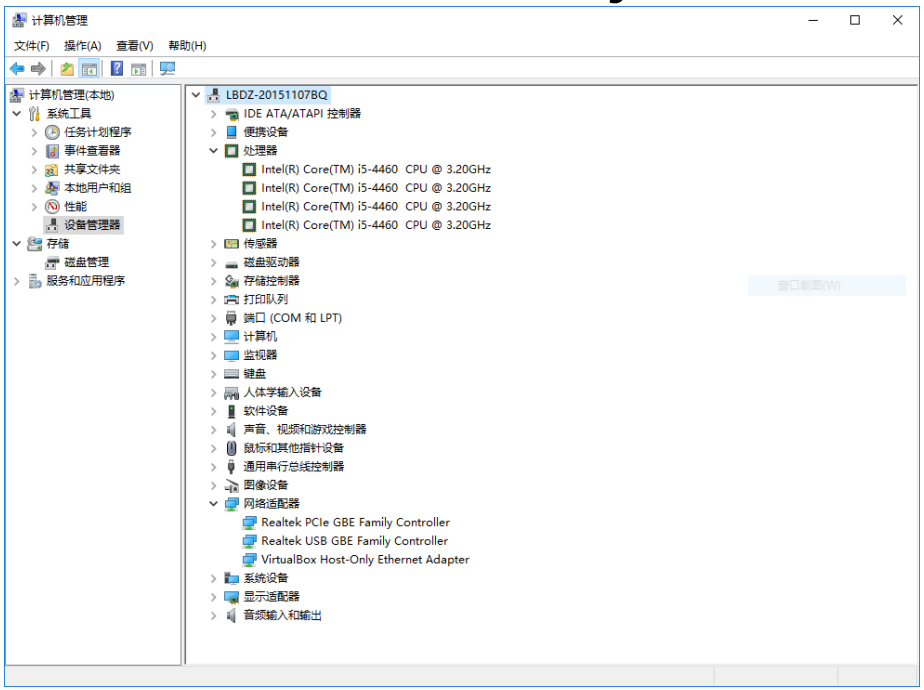

- Process VM 进程虚拟机
  - 程序编译为 Intermediate Code 中间代码交由“Runtime运行时”执行
  - 为编程语言提供“跨平台/平台无关”特性
    - Java VM, Python VM
- VM Monitor 虚拟机管理器
  - 用于模拟另外一套硬件指令集的特殊软件层
  - 足以支持一整套操作系统及其应用软件运行
    - VMWare, VirtualBox, Hyper-V

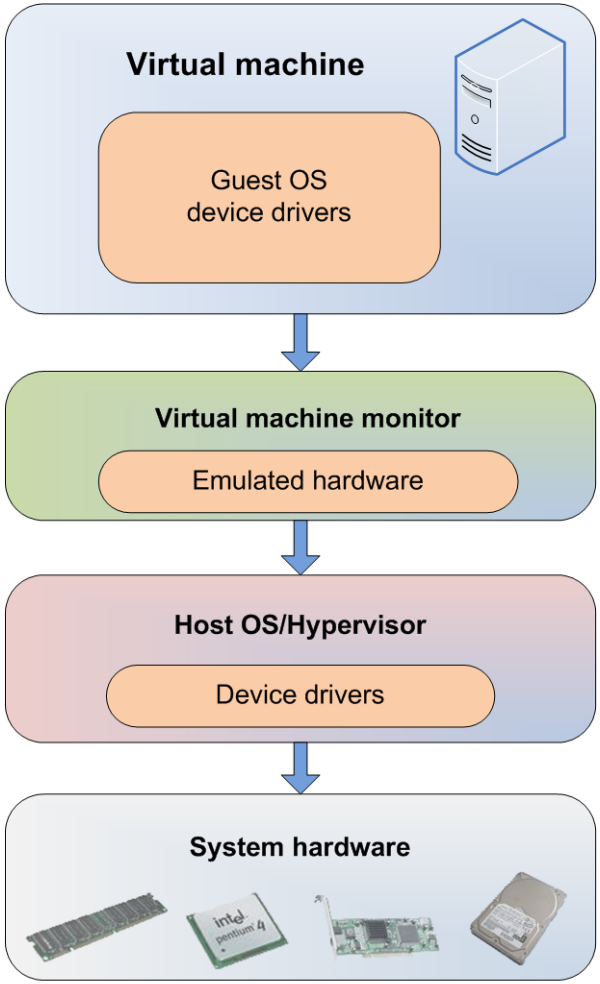
# 三种虚拟化方法

- 全虚拟化 Full Virtualization
- 半虚拟化 Paravirtualization
- 硬件辅助虚拟化 Hardware-Assisted Virtualization

# Full Virtualization

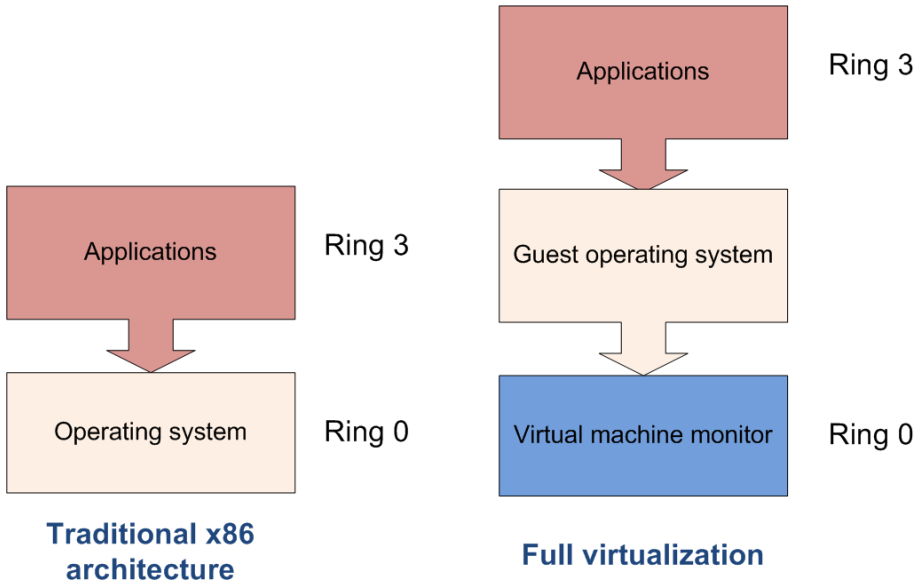
- Everything is virtualized
- Full hardware emulation
- Emulation = latency



# Privileged Instructions

- Privileged instructions:
  - OS kernel and device driver access to system hardware
- Trapped and emulated by VMM

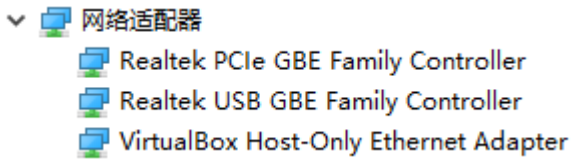


# Full Virtualization *Pros and Cons*

- **Pros**
  - Disaster recovery, failover
  - Virtual appliance deployment
  - Legacy code on non-legacy hardware
- **Cons** – LATENCY of core four resources
  - RAM performance reduced 25% to 75%
  - Disk I/O degraded from 5% to 20%
  - Network performance decreased up to 10%
  - CPU privileged instruction dings nearing 1% to 7%

# Paravirtualization

- OS or system devices are virtualization aware
- **Requirements**:
  - OS level -- recompiled kernel
  - Device level – paravirtualized or “enlightened” device drivers

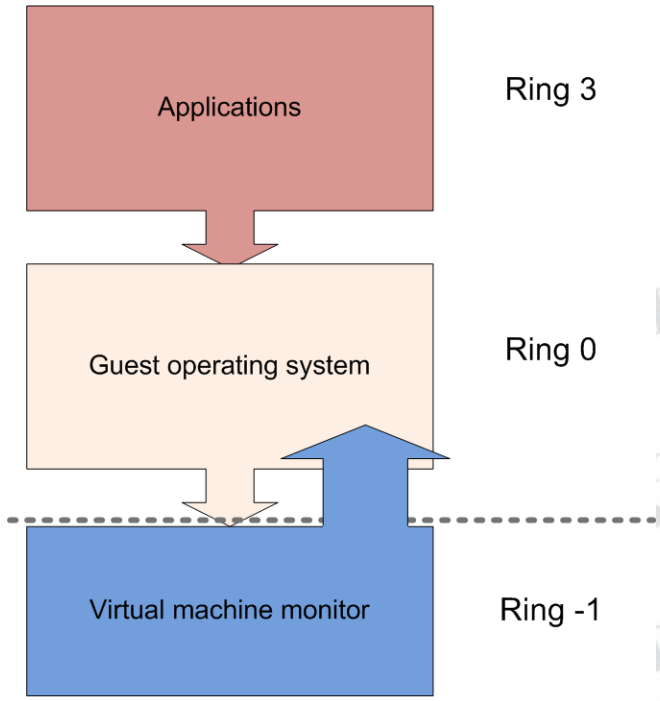


# Paravirtualization *Pros and Cons*

- **Pros**: fast!
- **Cons**: requires ***a specially modified guest OS***, thus precludes the ability to run off-the-shelf and legacy OS in para-virtual environments.

# Hardware-assisted Virtualization

- Server hardware is virtualization aware
- Hypervisor and VMM load at privilege Ring-1 (firmware)
- Removes CPU emulation bottleneck
- Memory virtualization coming in quad core AMD and Intel CPUs



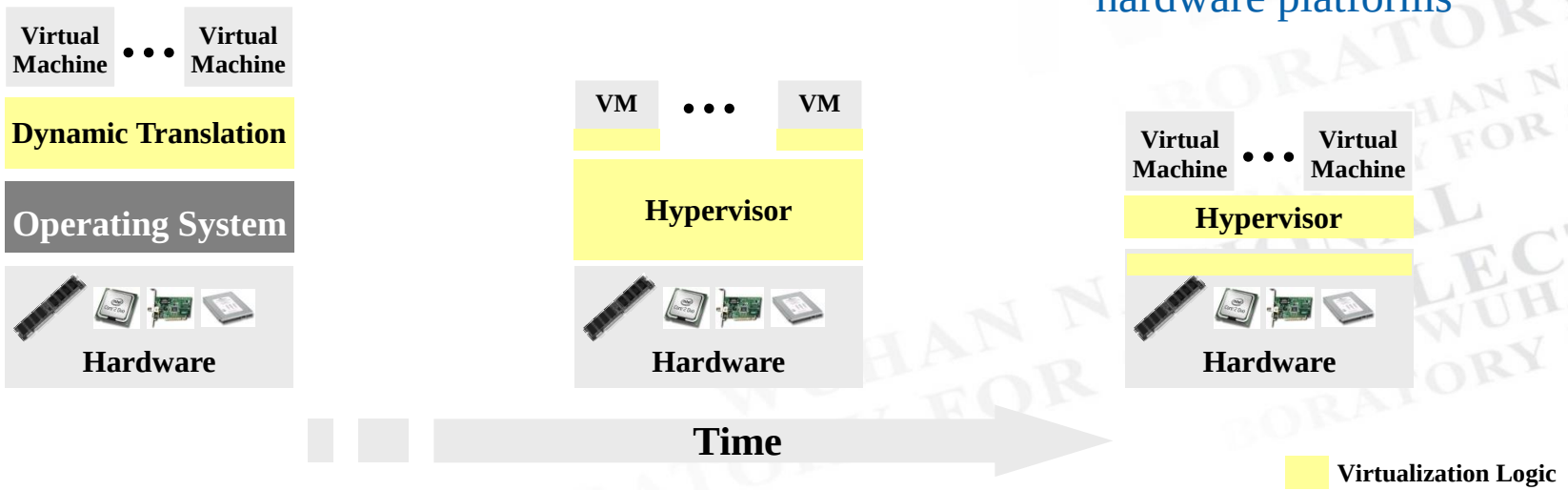

Hardware-assisted virtualization

# Evolution of Software solutions*

1st Generation: Full virtualization (Binary rewriting)
- Software Based
- VMware and Microsoft

2nd Generation: Para virtualization
- Cooperative virtualization
- Modified guest
- VMware, Xen

3rd Generation: Silicon-based (Hardware-assisted) virtualization
- Unmodified guest
- VMware and Xen on virtualization-aware hardware platforms




*This slide is from Intel® Corporation

# 带上疑问

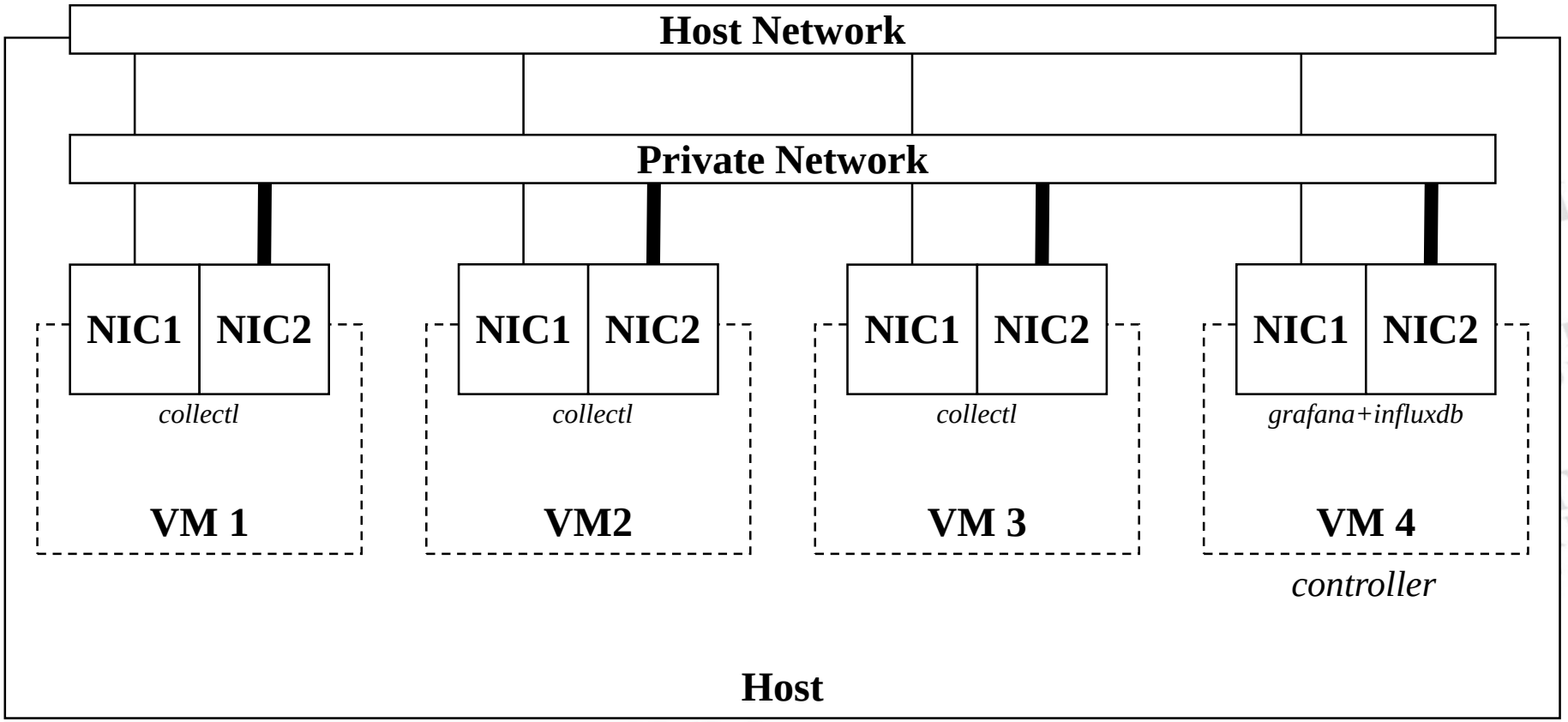
- 如何虚拟化？
- 该怎么用？
- 怎样观察分析？
- 具体怎么处理？

# 初步实践

- 所需材料
  - VirtualBox(All)/HyperV(Windows10)/Parallels(MacOS)
  - Vagrant
  - https://github.com/cs-course/vagrant-tutorial

# 小型系统

四机操演


Host Network
Private Network
NIC1
NIC2
collectl
VM 1
NIC1
NIC2
collectl
VM2
NIC1
NIC2
collectl
VM 3
NIC1
NIC2
grafana+influxdb
VM 4
controller
Host

# 性能观测


vm dashboard
Zoom Out
Last 30 minutes
Refresh every 5s
disk-read
1.0
0.5
0
-0.5
-1.0
11:30
11:32
11:34
11:36
11:44
11:46
11:48
11:50
11:52
11:54
11:56
11:58
2019-09-24 11:36:52
node1.diskinfo.readkbs.sda.mean: 0
node2.diskinfo.readkbs.sda.mean: 0
node3.diskinfo.readkbs.sda.mean: 0
node1.diskinfo.readkbs.sda.mean
node2.diskinfo.readkbs.sda.mean
node3.diskinfo.readkbs.sda.mean
disk-write
40
30
20
10
0
11:30
11:32
11:34
11:36
11:38
11:40
11:42
11:44
11:46
11:48
11:50
11:52
11:54
11:56
11:58
node1.diskinfo.writekbs.sda.mean
node2.diskinfo.writekbs.sda.mean
node3.diskinfo.writekbs.sda.mean

# 找出问题

- 开监控

```
for i in 21 22 23
do ssh -i insecure-key vagrant@192.168.33.$i \
   'nohup \
      collectl \
         -i1 -scdmnstxyCDMNTXY --dskfilt sd -f /vagrant_data/collectl --rawtoo \
         --export graphite,192.168.33.127:2003,d=1,s=cdmnstxyCDMNTXY > /dev/null &'
done
```

- 上负载

```
# 分别在node{1..3}上，同时执行(tmux同步输入命令)，观察仪表盘，分析资源隔离情况
dd if=/dev/zero of=/dev/null bs=$(( 1024*1024*1024 )) count=2
```

```
# 分别在node{1..3}上，同时执行(tmux同步输入命令)，观察仪表盘，复现前述资源冲突
dd if=/dev/zero of=test_2GB bs=$(( 1024*1024*1024 )) count=2
```

avg disk write

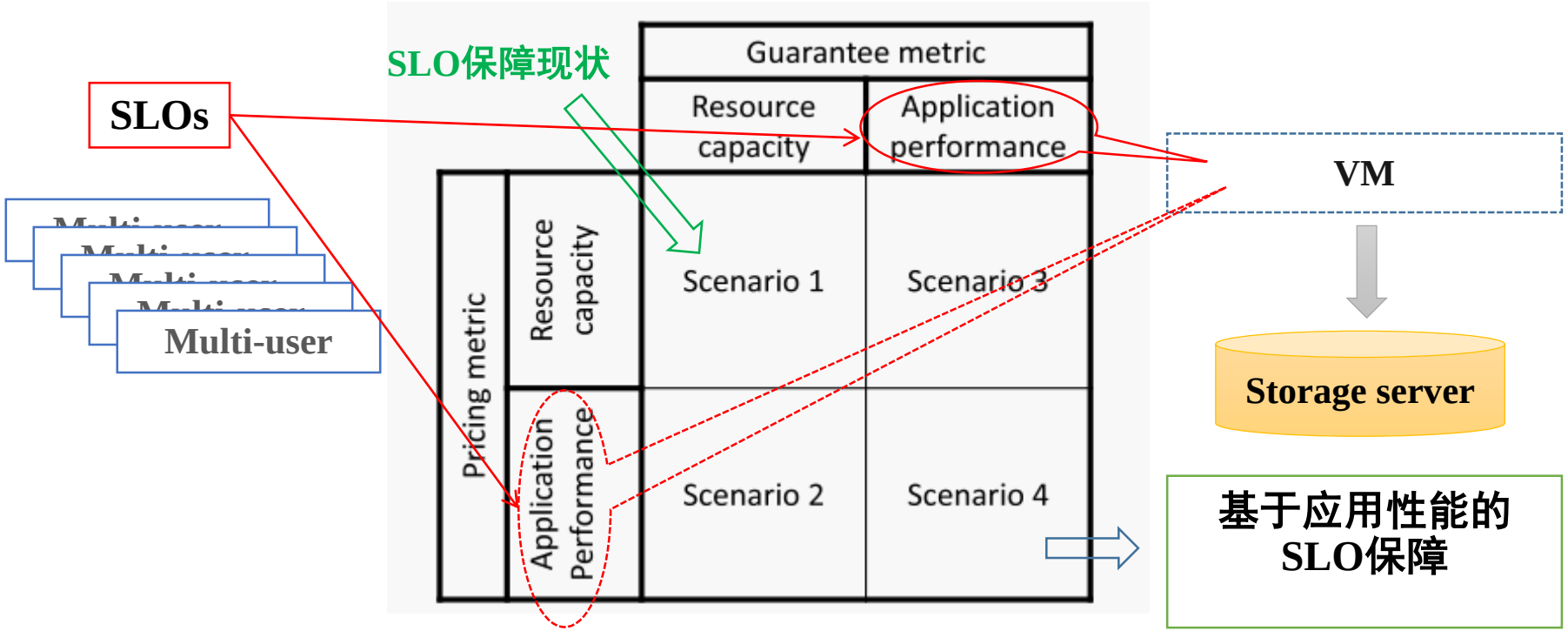
| Time | node1.diskinfo.writekbs.sda.mean | node2.diskinfo.writekbs.sda.mean | node3.diskinfo.writekbs.sda.mean |
|---|---|---|---|
| 2019-09-24 12:05:00 | 1.03 | 455.42 | 1.18 |
| 2019-09-24 12:00:00 | 1.01 | 1.14 | 1.14 |
| 2019-09-24 11:55:00 | 1.00 | 1.14 | 1.13 |
| 2019-09-24 11:50:00 | 0.99 | 1.11 | 1.14 |
| 2019-09-24 11:45:00 | 1.01 | 1.14 | 1.14 |
| 2019-09-24 11:40:00 | 1.01 | 1.14 | 1.15 |

# 意味着什么

- 直观认识
  - 物理机开一台就是一台
  - 虚拟机开一台 … 不见得有一台的资源
- 严格描述： 服务等级协议SLA *Service Level Agreement*
  - SLA含多个服务等级目标SLO *Service Level Objective*，每个SLO规定一项性能需求的量化指标。
  - 越来越多的企业级用户将应用部署在大规模的虚拟机集群中，虚拟机的SLO保障水平对用户应用的服务质量而言至关重要。

# 进一步剖析

现有的企业级数据中心或私有云的解决方案主要还是依据应用性能峰值（最坏情况下）的资源需求量作为参考来进行资源供给以保障应用的SLO




不足：

- 造成SLO保障中的资源过量供给
- 无法充分挖掘现有资源优化VM性能

# 因此

- 如何尽可能避免性能保障中的资源过量供给，用必需而不过量的资源保障虚拟机性能达到用户要求的SLO，即SLO精确保障的问题。
- 在SLO 精确保障的基础上怎么进一步利用可用的资源优化虚拟机性能，即SLO约束下的性能优化问题。

# 本讲小结

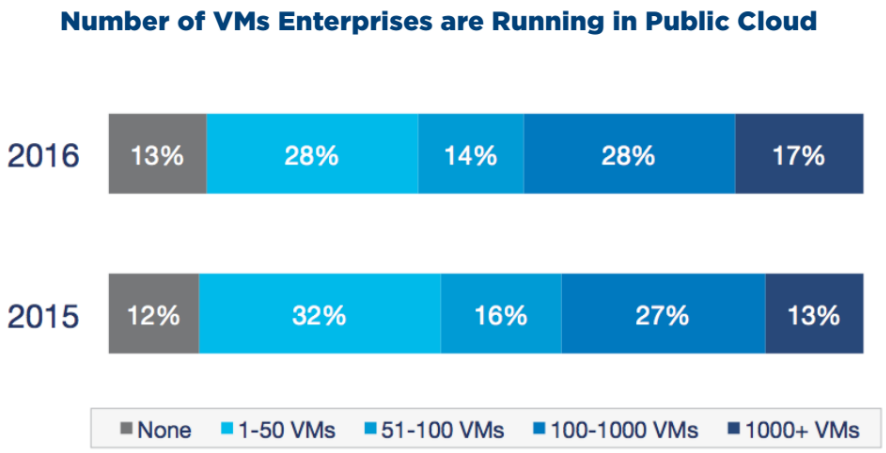
- 虚拟化环境存储系统基础知识
- 实验环境
- 服务质量保障问题

# 第二部分

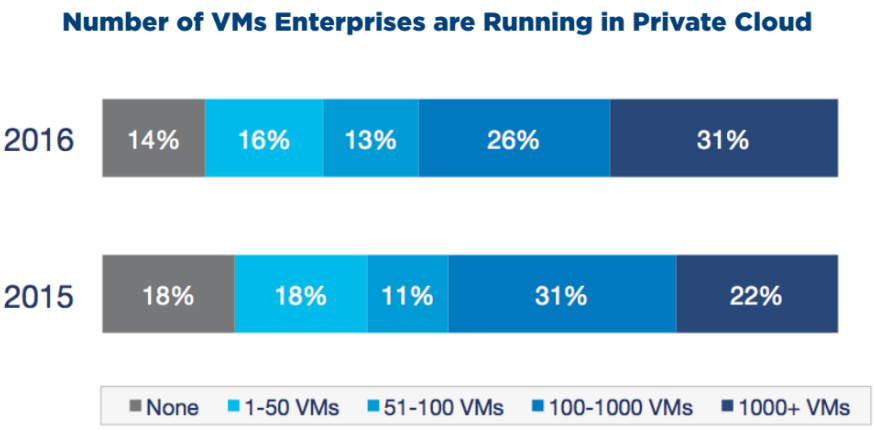
# 存储服务质量精确保障

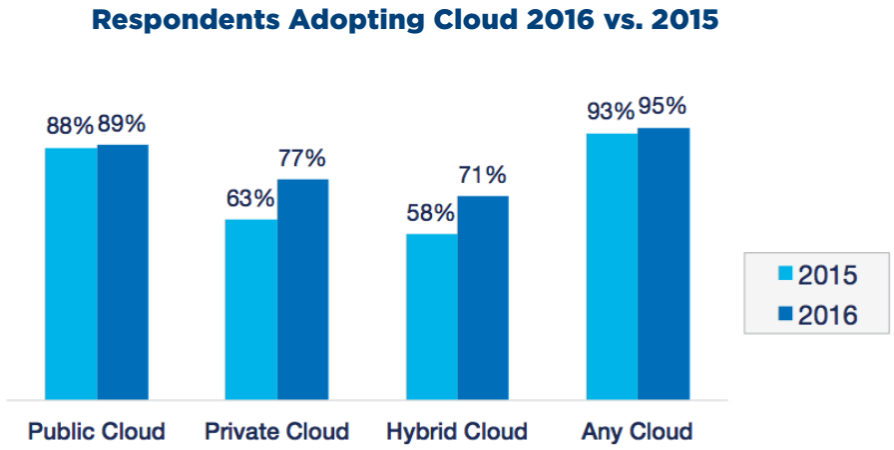
1. Customizable SLO and Its Near-Precise Enforcement for Storage Bandwidth[J]. ACM Transactions on Storage (TOS), 2017.
2. SASLO: Support User-Customized SLO Policy via Programmable End-to-End VM-Oriented IO Control[C]. International Conference on Cloud Computing and Big Data (CCBD), 2015.

# 背景




- 各类云服务应用比例逐年提升
- 云中租赁大规模VM集群的企业级用户的比例增长较快




2016年 RightScale “State of the Cloud Report”

# 概念


DUBLIN
DRUPALCON
SLO, SLA & SLI Terminology
→ SLO - Service level objective is agreed as a means of measuring the performance of the Service Provider.
→ SLA - Service Level Agreement specifies what service is to be provided, how it is supported, times, locations, costs, performance, and responsibilities of the parties involved. SLOs are specific measurable characteristics of the SLA such as availability, throughput, frequency, response time, or quality.
→ SLI - Service Level Indicator is a measure of the service level provided by a service provider to a customer. SLIs form the basis of Service Level Objectives (SLOs), which in turn form the basis of Service Level Agreements (SLAs).

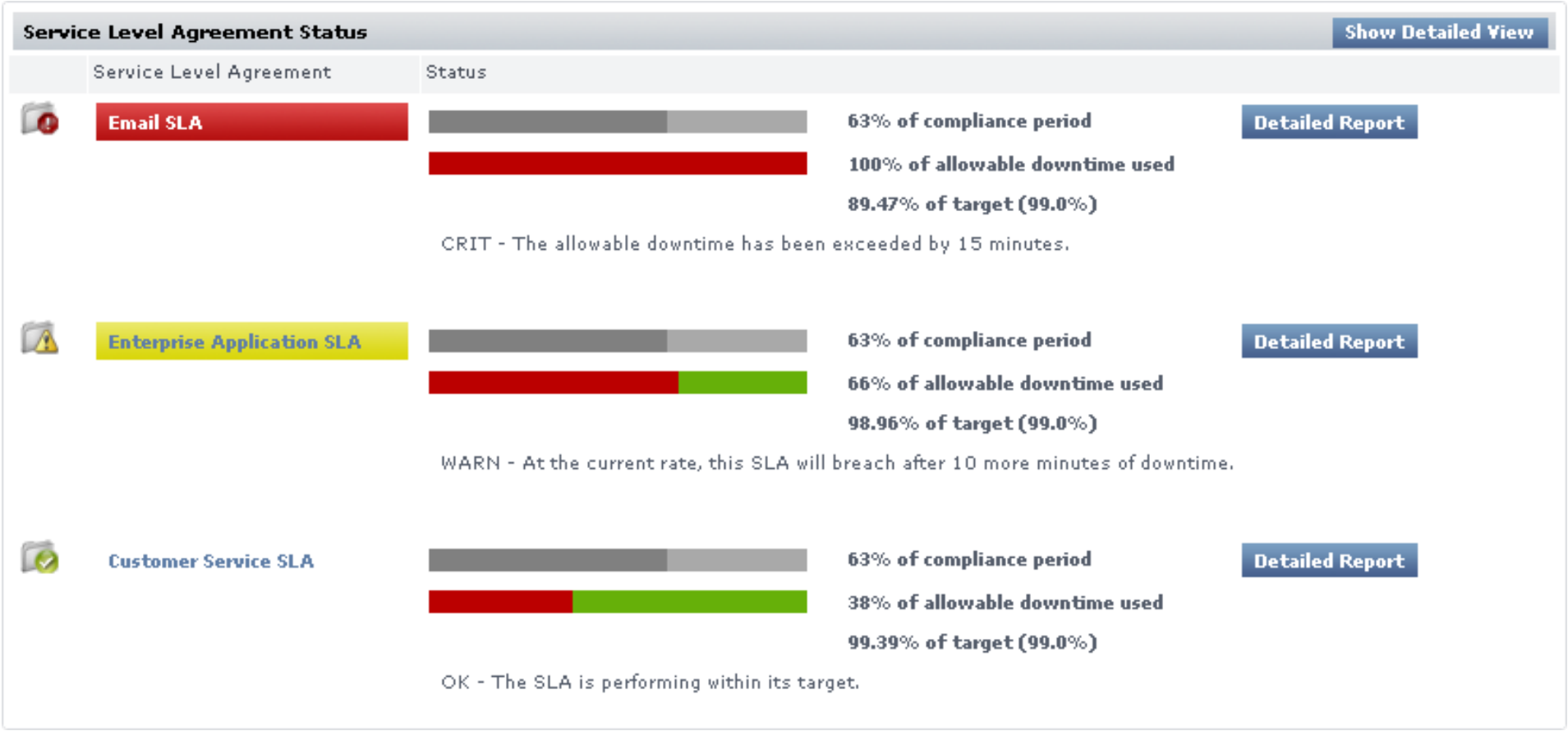
# 性能评价标准

| 性能指标 | 评价标准 | 性能评价标准 | 数学定义 |
|---|---|---|---|
| I/O延时($L$) | 均值误差($E$) | I/O延时均值误差($E_L$) | $E_L = (L_{ave} - L_{SLO})/L_{SLO} * 100\%)$ |
| | 服从率($SCrate$) | I/O延时SLO服从率($SCrate_L$) | $SCrate_L = F(\delta_1 \leq L \leq \delta_2) * 100\%$ |
| | 百分位性能($PC$) | I/O延时的百分位性能($PC_L$) | $PC_L = F^{-1}(X\%), 0 \leq X \leq 100$ |
| I/O吞吐率($T$) | 均值误差($E$) | I/O吞吐率均值误差($E_T$) | $E_T = (T_{ave} - T_{SLO})/T_{SLO} * 100\%)$ |
| | 服从率($SCrate$) | I/O吞吐率SLO服从率($SCrate_T$) | $SCrate_T = F(\delta_1 \leq T \leq \delta_2) * 100\%$ |
| | 性能波动($PFratio$) | I/O吞吐率性能波动($PFratio_T$) | $PFratio_T = \frac{SD(T_{actual}(k))}{T_{SLO}} * 100\%$ |
| I/O带宽($B$) | 均值误差($E$) | I/O带宽均值误差($E_B$) | $E_B = (B_{ave} - B_{SLO})/B_{SLO} * 100\%)$ |
| | 绝对性能误差($AE$) | I/O带宽绝对性能误差($AE_B$) | $AE_B = (\sum_{i=1}^{T} \lvert G_B^{(t_i)} \rvert)/T * 100\%)$ |

# 举例

| Example Service Level Objectives | |
|---|---|
| **General Comparisons** | **Service levels** (*X* percent answer/*Y* seconds) |
| Emergency services | 100/0 |
| Service level objectives "high" | 90/20, 85/15, 90/15 |
| Service level objectives "moderate" | 80/20, 80/30, 90/60 |
| Service level objectives "modest" | 70/60, 80/120, 80/300 |

# 应用场景


Service Level Agreement Status
Show Detailed View
Service Level Agreement
Status
Email SLA
63% of compliance period
100% of allowable downtime used
89.47% of target (99.0%)
Detailed Report
CRIT - The allowable downtime has been exceeded by 15 minutes.
Enterprise Application SLA
63% of compliance period
66% of allowable downtime used
98.96% of target (99.0%)
Detailed Report
WARN - At the current rate, this SLA will breach after 10 more minutes of downtime.
Customer Service SLA
63% of compliance period
38% of allowable downtime used
99.39% of target (99.0%)
Detailed Report
OK - The SLA is performing within its target.

# 研究现状

- 目前的I/O吞吐率/带宽SLO保障和优化方案没有充分考虑I/O请求队列的突发性波动对I/O延时性能的影响，导致I/O吞吐率/带宽和I/O延时指标之间的SLO保障行为和性能优化未实现有效隔离。
- 现有的I/O延时SLO保障方案主要基于冗余资源或最坏情况下的资源供给模式，难以支持精确I/O延时，尤其是高百分位尾延时SLO的执行。

# 吞吐率/带宽SLO精确保障方法

## SASLO系统的设计 Stable and Accurate SLO

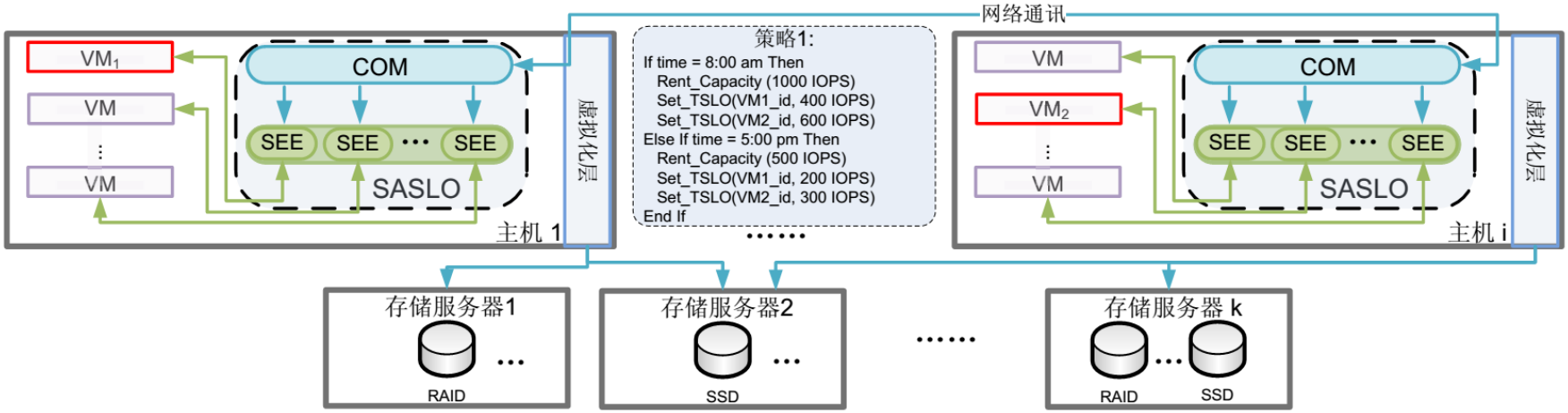


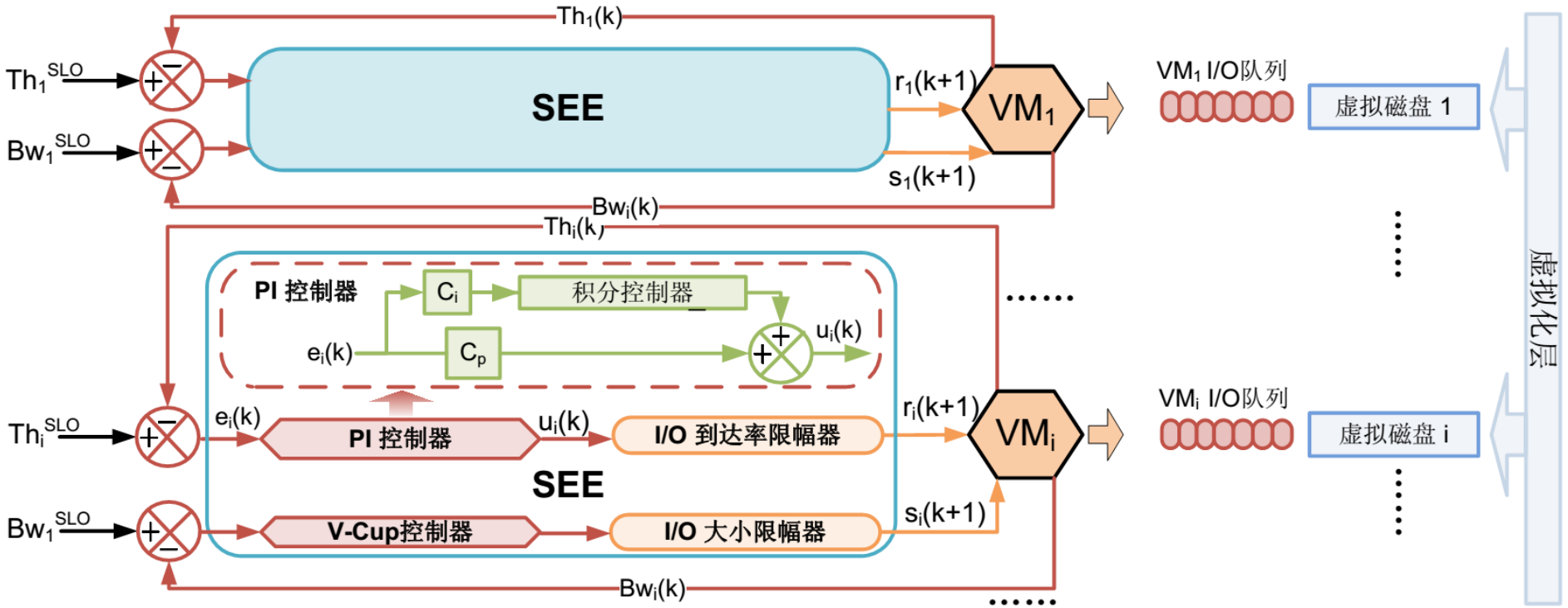

表 2.1 面向用户订制SLO的API

| # | API |
|---|---|
| 1 | **Set_TSLO (VM_id i, Throughput_slo t)**<br>将ID为*i*的虚拟机的I/O吞吐率SLO目标值设定为*t* |
| 2 | **Set_BSLO (VM_id i, Bandwidth_slo t)**<br>将ID为*i*的虚拟机的I/O带宽SLO目标值设定为*t* |

SASLO主要包括通讯层(COM) 和SLO执行引擎 (SEE)

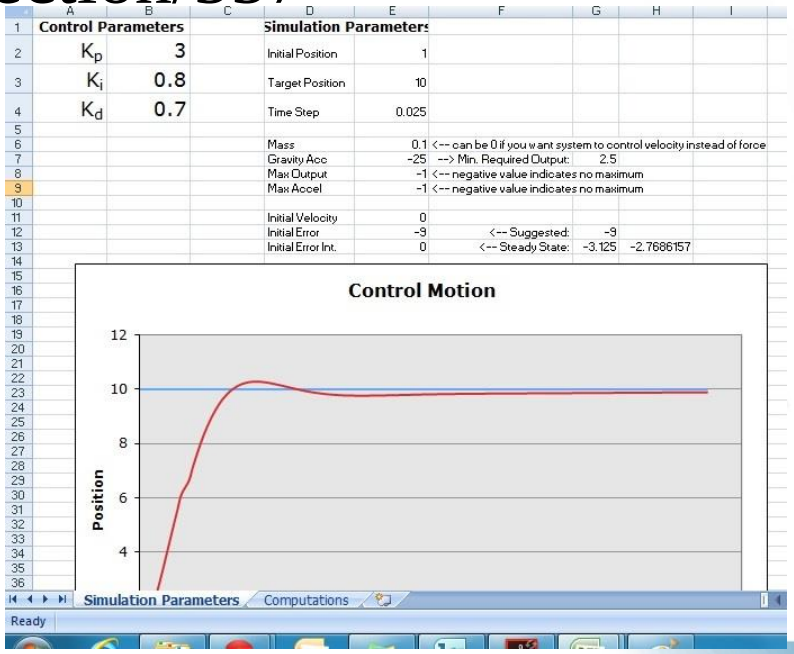
# 吞吐率/带宽SLO精确保障方法

- **比例积分(PI) 控制器**: 将实际虚拟机的I/O吞吐率收敛于SLO目标值。
- **V-Cup控制器**: 通过I/O请求大小波动控制进行I/O带宽SLO精确保障。



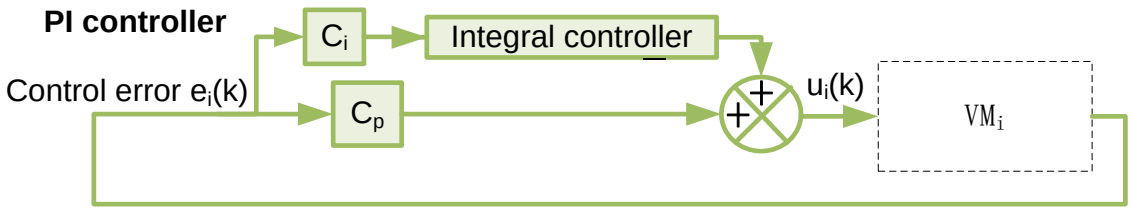
# 吞吐率/带宽SLO精确保障方法

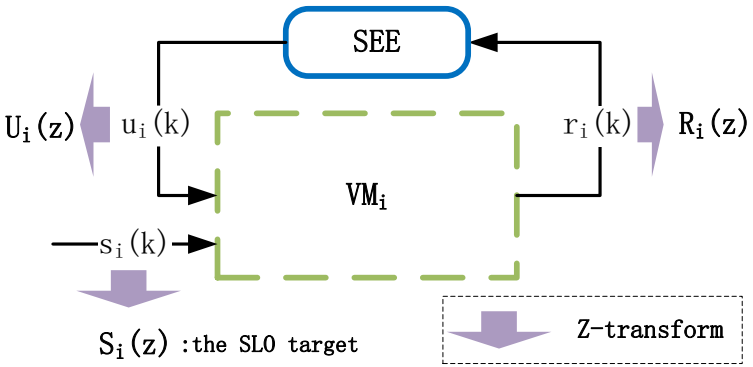
- 关于PI控制器
  - https://automationforum.in/t/pid-simulator-free-tools-collection/557



# 吞吐率/带宽SLO精确保障方法

## 比例积分(PI)控制器




$$u_i^p(k) = C_P \times e_i(k)$$

$$u_i^I(k) = u_i^I(k-1) + C_I \times e_i(k)$$

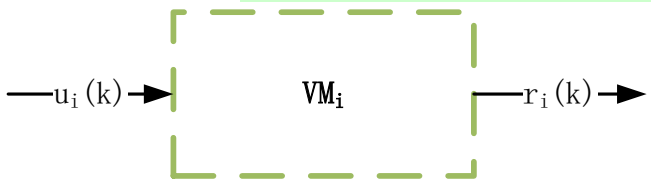
$$u_i(k) = u_i^p(k) + u_i^I(k)$$

比例积分控制模型

闭环传递函数

采用自回归移动平均 (ARMA)模型描述VM输入ui(k)对VM输出ri(k)的影响。

$$F_i(z) = \frac{R_i(z)}{S_i(z)} = \frac{[(C_p + C_I) \times z - C_p] \times G_i(z)}{(z-1) + [(C_p + C_I) \times z - C_p] \times G_i(z)}$$




$$r_i(k+1) = a_i^k \times r_i(k) + b_i^k \times u_i(k)$$

传递函数

根据控制理论，当闭环系统稳定的时候，PI控制具有0稳态误差

$$G_i(z) = \frac{R_i(z)}{U_i(z)} \Rightarrow G_i(z) = \frac{b_i^k}{z - a_i^k}$$

$$E_{ss} = \lim_{z \to 1}(z-1)(S_i(z) - R_i(z))$$

$$= \lim_{z \to 1} \frac{(z-1)^2 \times (z - a^k) \times S_i(z)}{z^2 + [(C_P + C_I) \times b_i^k - (1 + a_i^k)] \times z + a_i^k - C_P \times b_i^k}$$

# 吞吐率/带宽SLO精确保障方法

## V-Cup控制器

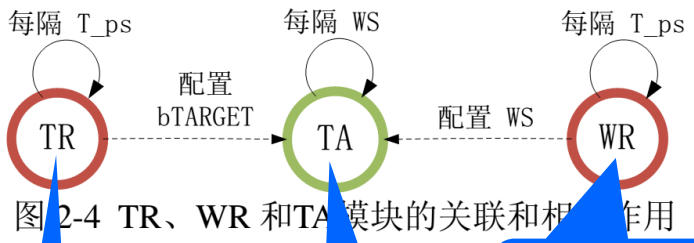



图2-4 TR、WR 和TA模块的关联和相互作用

目标调整模块 (TR)

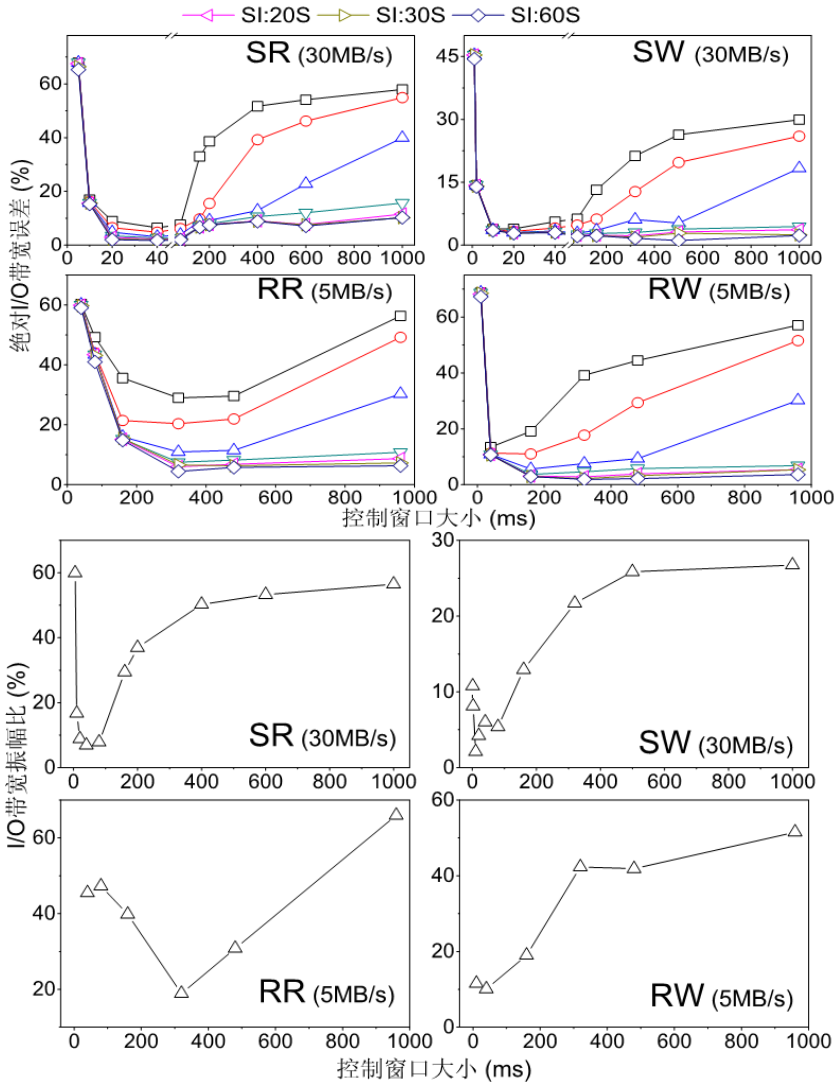
I/O带宽控制模块(TA)

控制时间窗口调整模块 (WR)

负责优化TA的控制窗口大小

I/O带宽的细粒度调整

目标调整模块（TR）通过动态修正TA模块采用的目标值来控制I/O带宽误差。



# 吞吐率/带宽SLO精确保障方法

## V-Cup控制器

**ALGORITHM 1:** 控制时间窗口调整(WR) 算法

**Input**: 统计时段$T_ps$得出的I/O带宽振幅比($V\_ps$), 统计时段$T_ps$得出的绝对I/O带宽误差($\beta\_ps$) 和I/O带宽SLO ($Bw_{SLO}$).

**Output**: 控制窗口大小$WS$的次优解($ws$).

```
1  ws_max = 1024 ms; ws_min = 32 ms; ws = ws_max;
2  repeat
3      if (V_ps − U ≤ last_V) and (β_ps ≤ B_threshold);
4      then
5          ws_max = ws;
6      else
7          ws_min = ws;
8      end
9      ws = (ws_min + ws_max) / 2;
10 until (ws_max − ws_min ≤ UN_WIN) or ((V_ps ≤ BEST_V) and
   (β_ps ≤ B_threshold));
11 return ws
```

假设在算法第一次迭代时，控制窗口大小WS的初始搜索空间大小为n，则该值在第二次迭代时为n/2。

以此类推，在后续的迭代中，搜索的空间大小分别为n/4, n/8, ... , $n/(2^k)$。其中，k是最大的迭代次数。

所以，WR算法的时间复杂度可以表示为O(logn)。

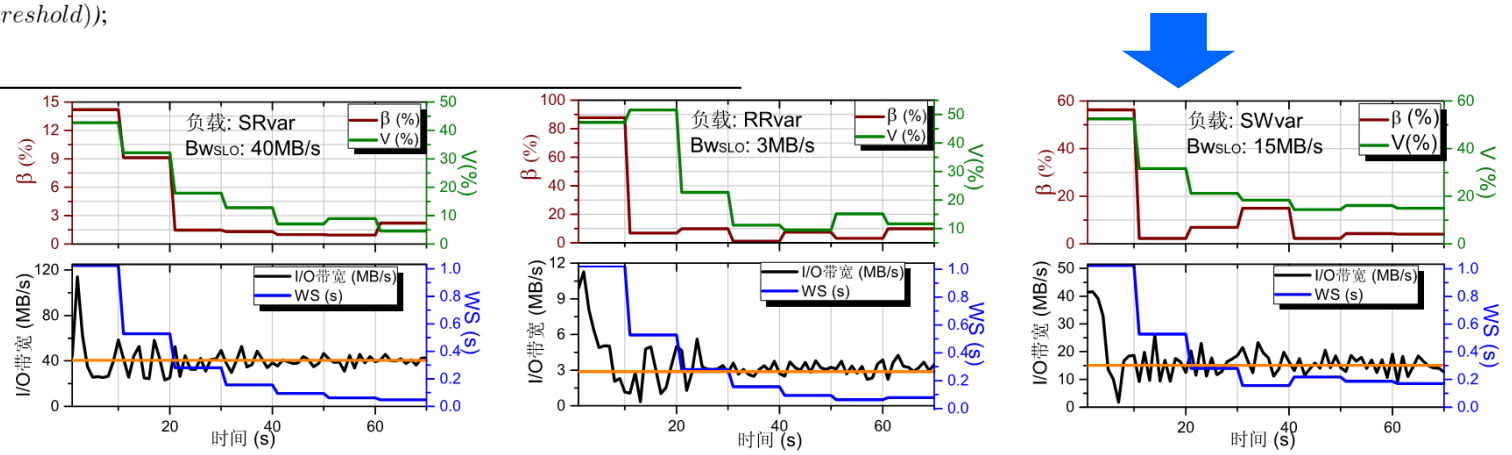



图 2-6 三个典型负载的控制窗口大小调整过程

# 吞吐率/带宽SLO精确保障方法

## 性能测试

测试平台介绍：

主机: 2台 PowerLeader PR2760T servers
*) 2 Intel Xeon E5620 quad-core 处理器,
*) 12GB 内存,
*) 10Gbps NIC (Intel 82598EB).

存储: 2类存储子系统
*) 16-disk (7200RPM, 250GB) RAID 0 磁盘阵列
*) Fusion-io ioScale 2, 825GB Multi Level Cell (MLC) SSD

虚拟化软件: Xen 4.2

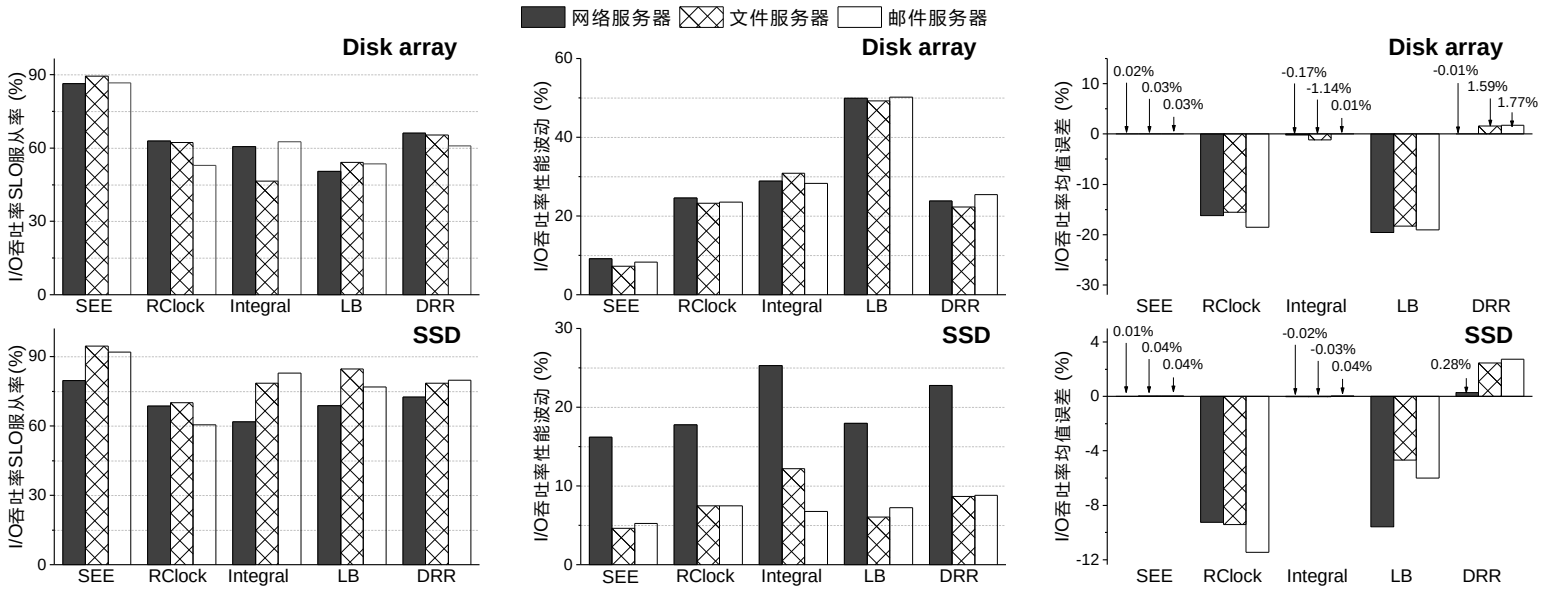
测试目标:

*) 验证SASLO框架支持下I/O吞吐率/带宽的SLO保障精度；
*) 验证SASLO响应SLO秒级变化的能力；
*) 验证SASLO对用户不同SLO策略的支持能力；
*) 考察SASLO对不同数量的聚合虚拟机进行SLO保障时产生的精度差异；
*) 考察SASLO框架支持下I/O吞吐率SLO保障对I/O延时性能的影响。

# 性能测试

## I/O吞吐率SLO保障精度比较

| 简称 | 参考算法 | 说明 |
|---|---|---|
| RClock | Time-stamp based IO control | 按请求的时间戳顺序进行I/O调度。 |
| DRR | Deficit round-robin | 一类基于GPS公平调度算法的改进版，它有助于降低控制误差。 |
| LB | Leaky bucket | 一种用于IO 节流控制的调度算法 |
| Integral | Integral control | 一种反馈控制算法，它保证控制输出的变化和控制误差的积分成比例。 |

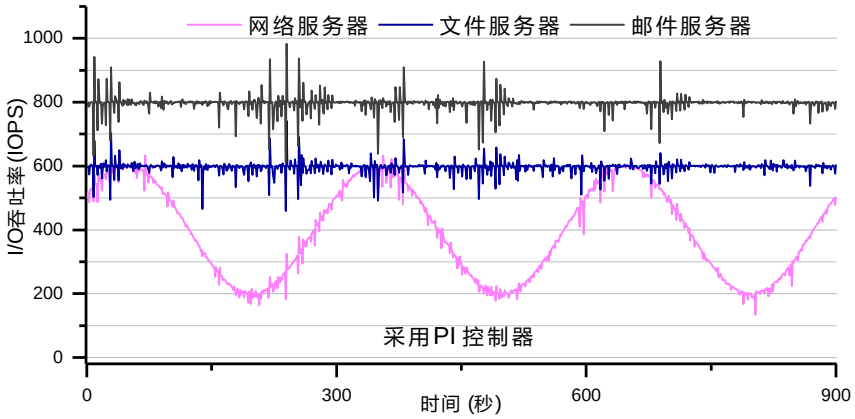


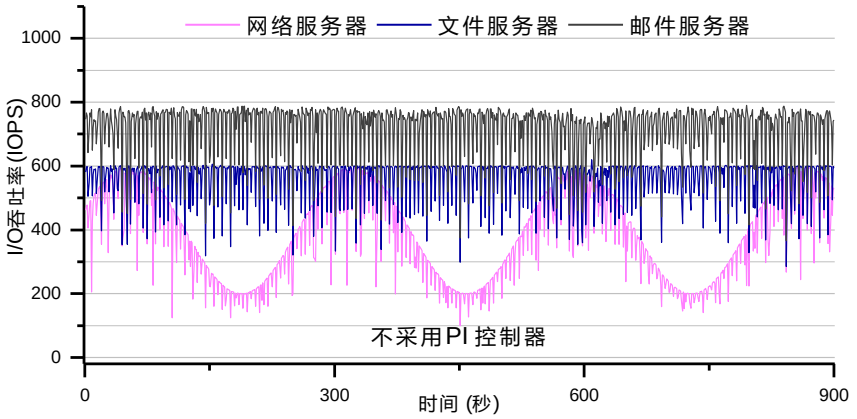

SLO执行引擎（SEE） 在磁盘阵列和SSD作为存储设备时均有较好的SLO保障精度和稳定性。

# 性能测试

响应SLO秒级变化

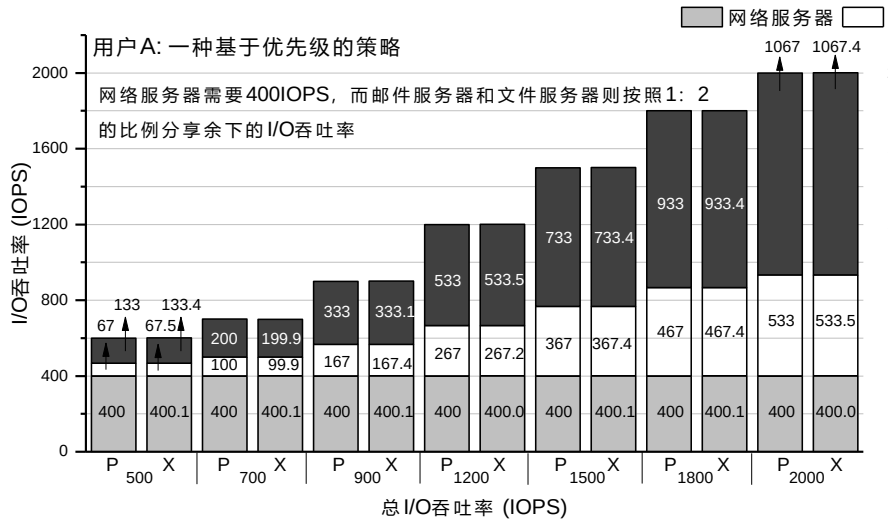
采用比例积分(PI)模型的SLO执行引擎（SEE）对变化的SLO序列具有较好的时间响应。

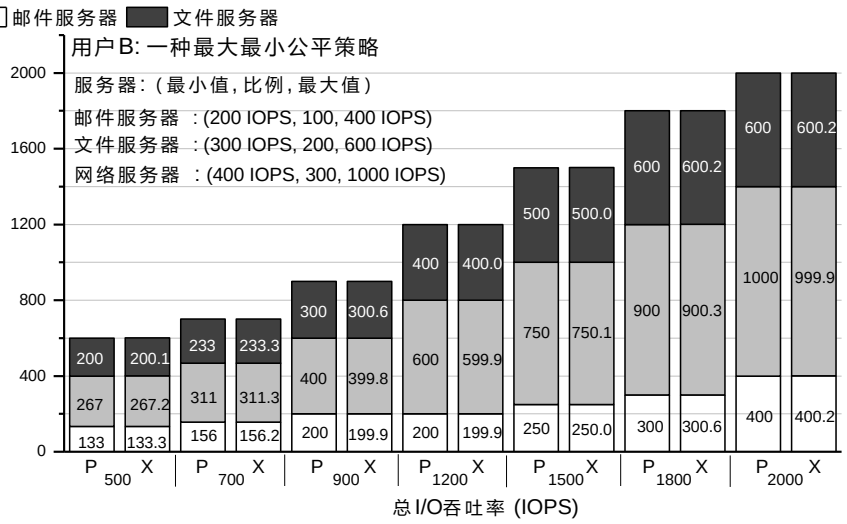



对用户不同SLO策略的支持

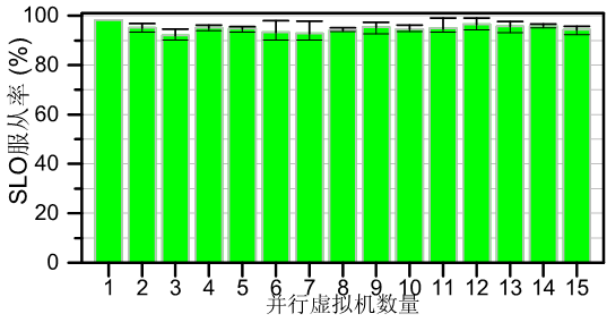
能准确的执行由不同用户策略发出的SLO序列

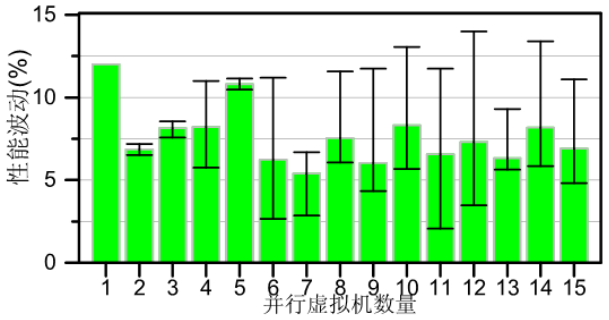


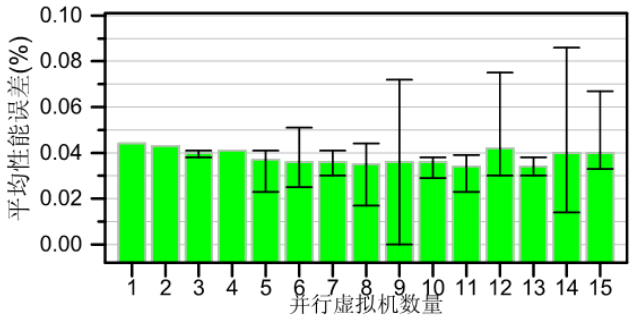



# 性能测试

对不同数量的并行虚拟机进行SLO保障时产生的精度差异

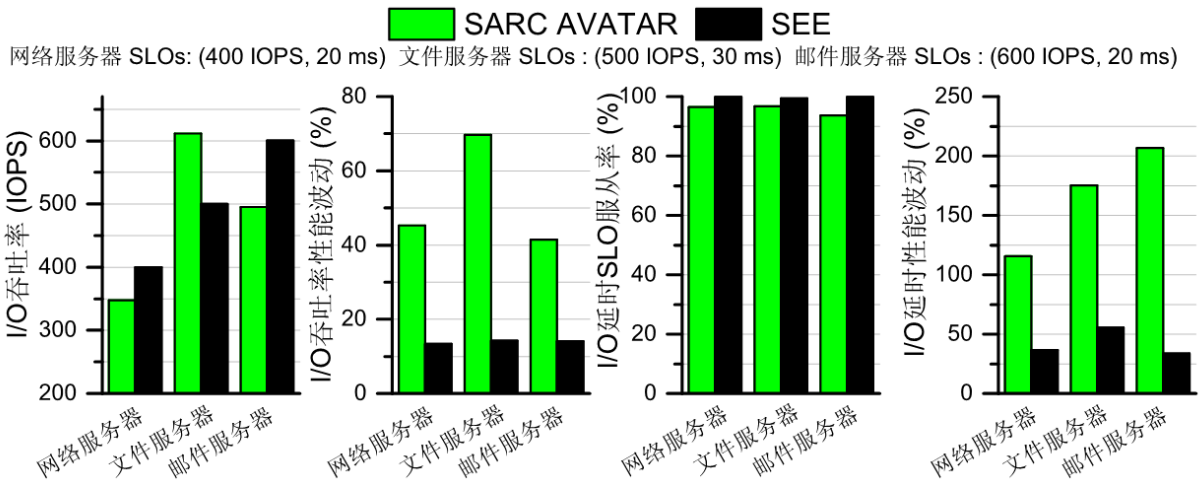



只要存储资源足够（即最大资源利用率< 100%），在SASLO的控制下并行虚拟机数量的增加几乎不会对SLO服从率、性能波动和均值误差产生影响。

I/O吞吐率SLO保障对I/O延时性能的影响:




3台虚拟机在SASLO的SLO执行引擎(SEE)控制下秒级I/O延时的SLO服从率分别为99.92%、99.42% 和99.83%；而在SARC AVATAR系统调度下为96.42%、96.75% 和93.58%。且SEE控制下各虚拟机秒级I/O延时性能波动 均 远低于SARC AVATAR。

# 本讲小结

- 为何需要SLO精确保障
- 如何实现SLO精确保障
- 交叉运用控制论经典方法进行创新
  - 提出面向虚拟机的SLO精确保障和基于SLO约束性能优化的解决方案，以实现多性能指标的精确保障。

# 第三部分

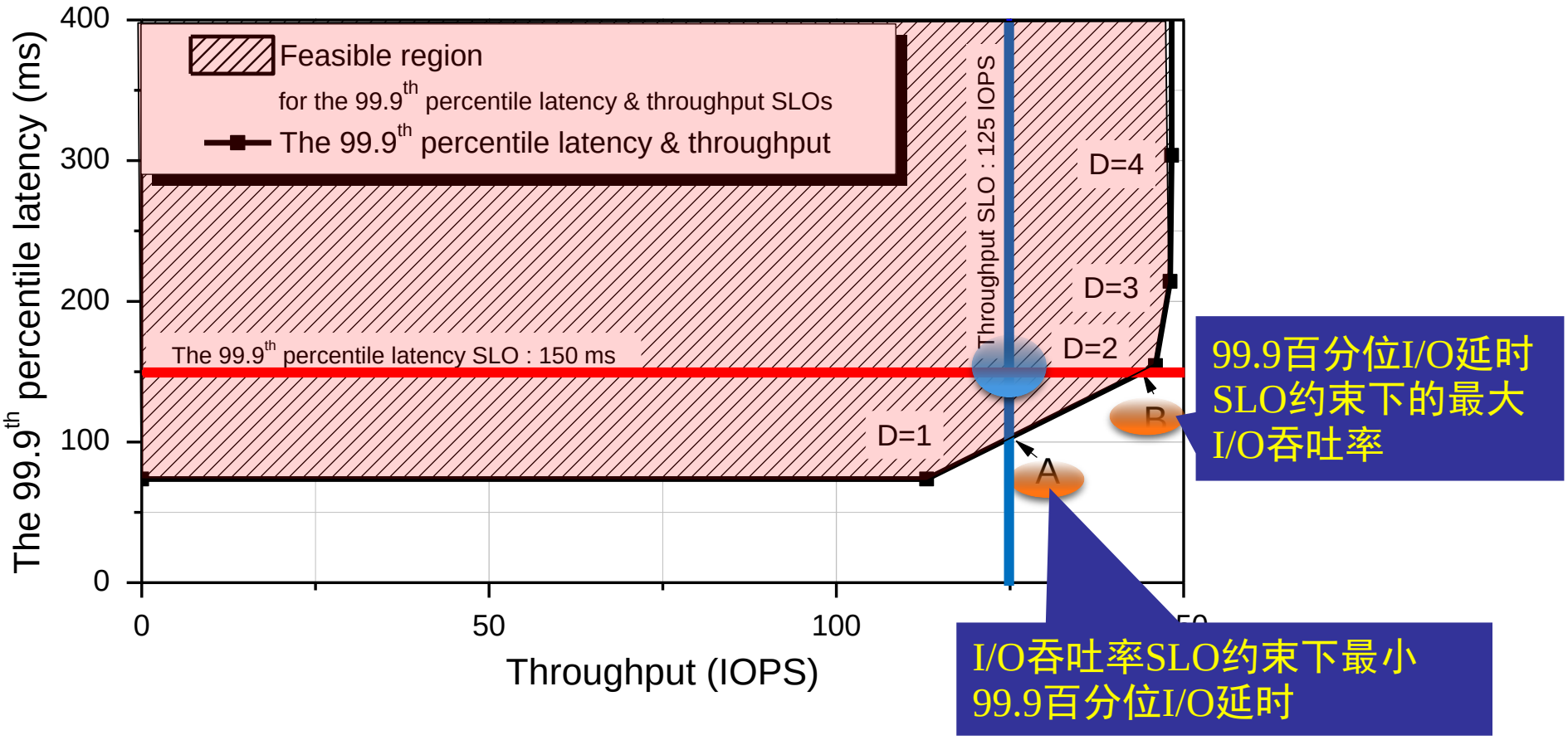
# 两维SLO约束的I/O优化

PSLO: enforcing the Xth percentile latency and throughput SLOs for consolidated VM storage[C]//Proceedings of the Eleventh European Conference on Computer Systems. 2016

# 研究现状

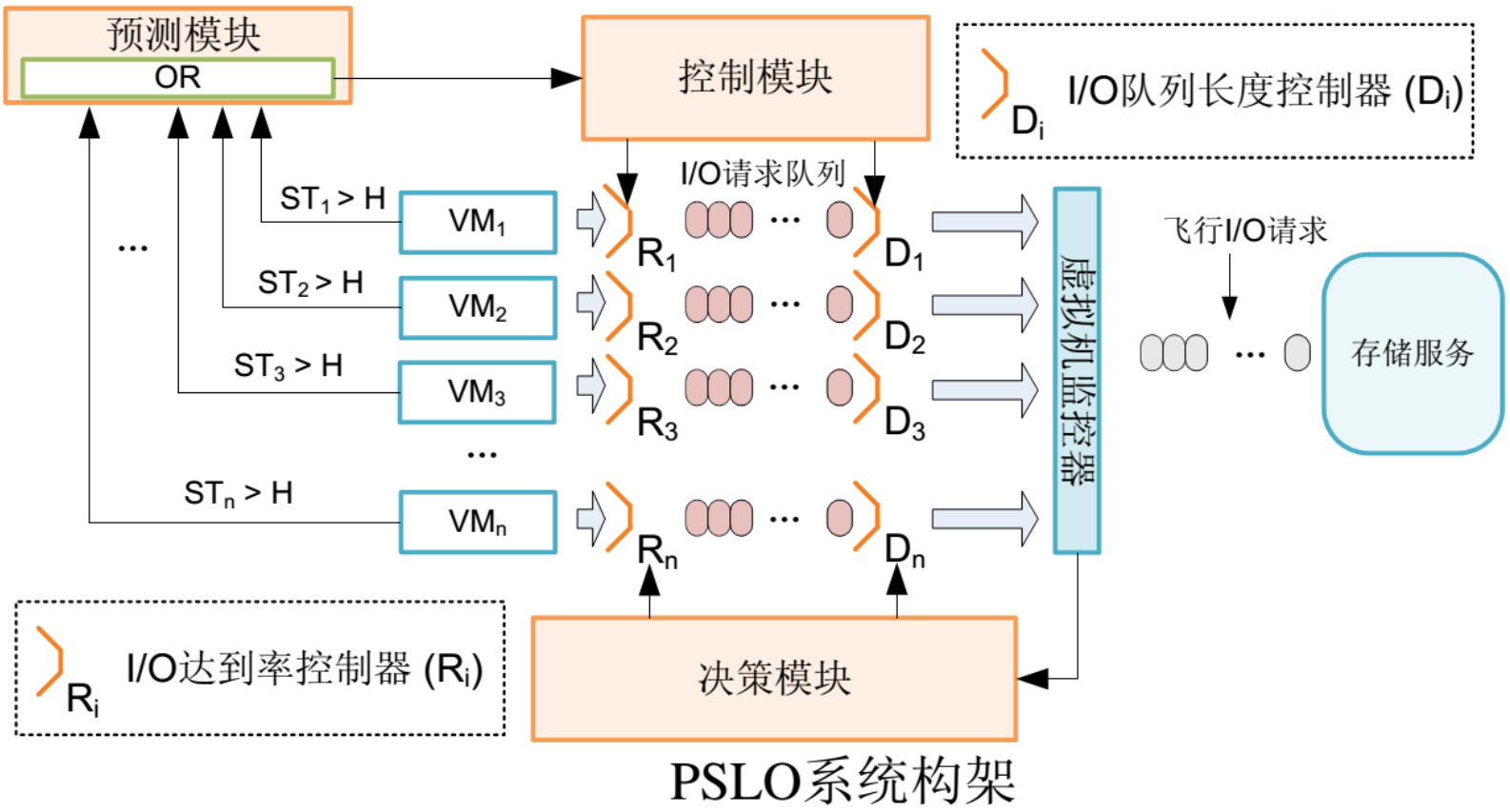
- 性能保障和优化并不孤立，而是相辅相成的概念，即保障是优化的重要前提，而优化是基于保障约束的资源利用最大化。
- 现有的存储性能保障和优化的相关研究主要面向I/O 吞吐率、I/O 带宽和I/O 延时这三类性能指标，未能充分考虑不同性能指标SLO保障的隔离，以及性能优化行为与SLO保障行为的隔离。
- 所以，现有方案难以有效实现多性能指标SLO精确保障，想进一步挖掘现有资源优化性能自然就难上加难。

# 研究动机




通过控制I/O并行度优化既定百分位I/O延时或者I/O吞吐率。

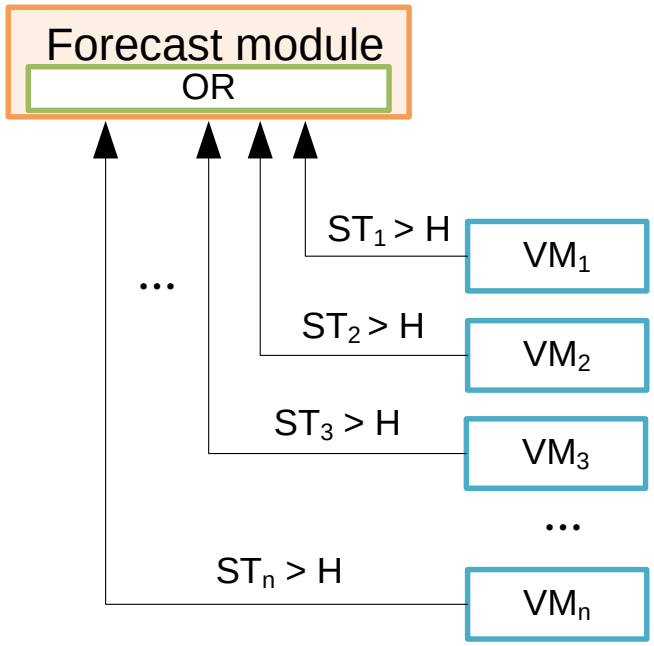
# 总体设计




PSLO系统构架

PSLO由三个关键功能模块、预测模块、控制模块和决策模块组成。该系统可以根据用户的具体需要设定每个虚拟机必须满足的百分位尾延时SLO和吞吐率SLO。然后PSLO可以根据不同的SLO策略进行这两类SLO的保障和性能优化。

# 预测模块




**n个并行虚拟机发出SLO违例预测的概率：**

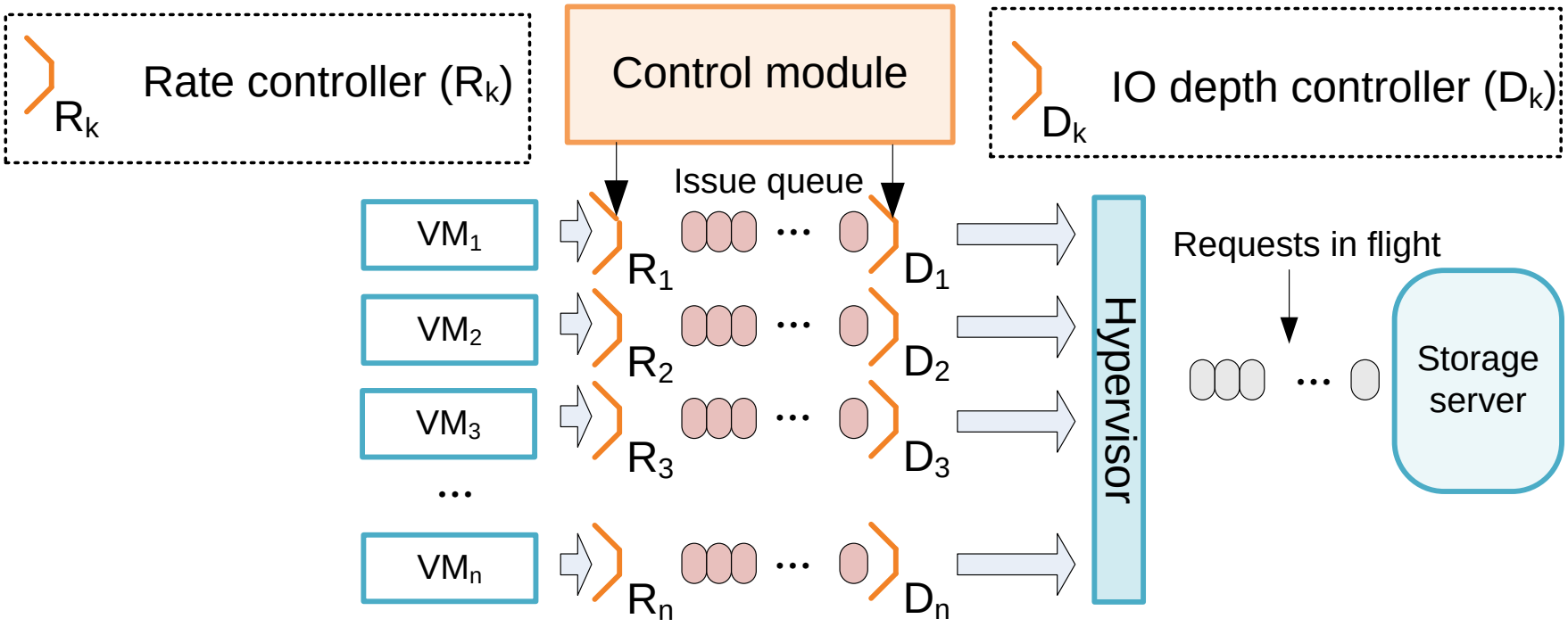
$$\zeta = 1 - \prod_{k=1}^{n} (1 - p_k)$$

**虚拟机$VM_k$侦测未来I/O延时违例的概率：**

$$p_k = \Pr(ST_k \geq H)$$

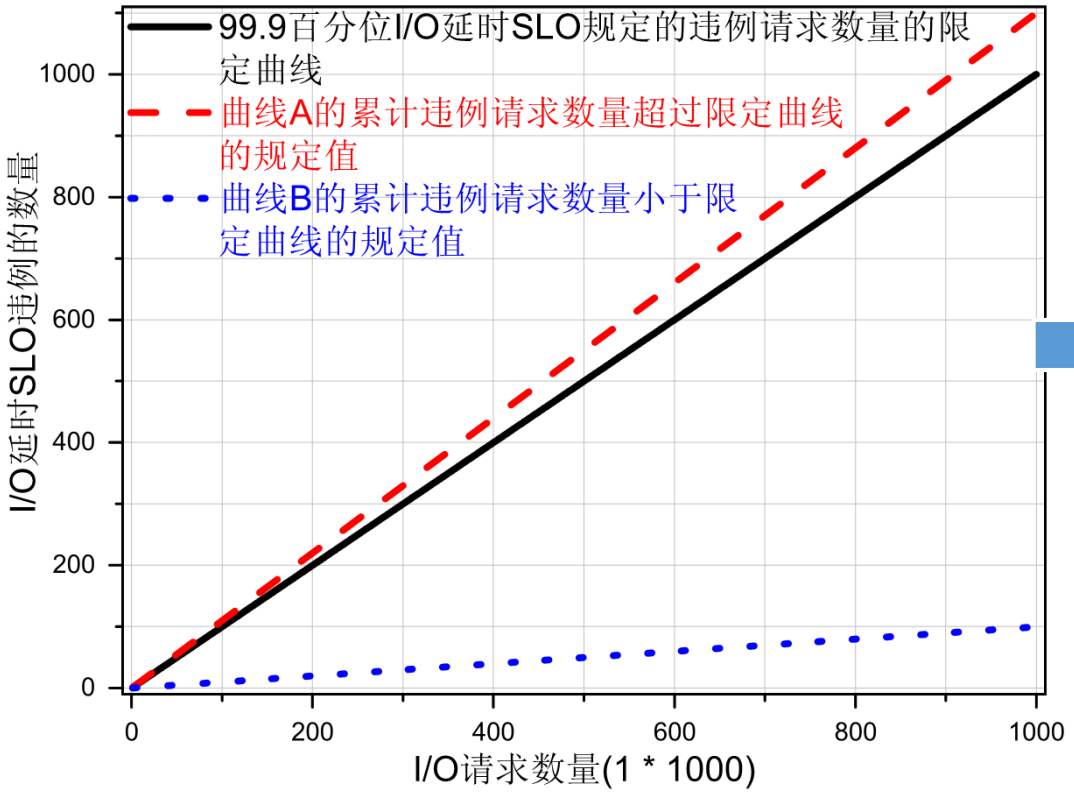
预测模块充分利用共享存储的虚拟机有I/O延时违例的时间局部性这一特征进行预测。

# 控制模块




控制模块通过动态调整IO并行度（Di）和虚拟机的IO到达率（Ri）对虚拟机级的既定百分位延时和I/O吞吐率进行控制。

# 决策模块




- 曲线A位于边界线之上, 导致99.9百分位延时SLO的违例。
- 曲线B 远在边界线之下, 这意味着很大的I/O吞吐率优化空间.
- 最理想的情况是曲线和边界线重合，这样I/O吞吐率将被最大化。

控制模块通过动态调整IO并行度（Di）和虚拟机的IO到达率（Ri）对虚拟机级的既定百分位延时和I/O吞吐率进行控制。

# 决策模块

实际的I/O延时违例数和上限的差值

$$e(k,t) = G(X_k,t) - M_k(t) \quad (6)$$

$$E(k,t) = \frac{e(k,t)}{D_k(t)} \quad (7)$$

使 e(k,t) 独立于$D_k$，因为它与负载I/O特征和 延时SLO有关

$$A(k,t) = Max(\sigma * \frac{e(k,t)}{G(X_k,t)}, 0) \quad (10)$$

$$P(t) = Min_{1 \le k \le n} A(k,t) + 1 \quad (11)$$

P(t) 是一个比例因子，它可以决定下一个时段所有聚合虚拟机I/O吞吐率目标提升的程度

如果百分位延时SLO能够得到满足，则 P(t) > 1

如果新更新的吞吐率目标是可行的, …

$$Th_k^G(t+1) = \begin{cases} P(t) * Th_k^G(t) & if\ \forall j, Th(j,t) \ge Th_k^G(t), \\ Max(\eta * Th_k^G(t), Th_k^{SLO}) & otherwise. \end{cases} \quad (12)$$

$$R_k(t+1) = Max(\frac{Th_k^G(t)}{Th_k(t)} * R_k(t), Th_k^G(t)) \quad (13)$$

如果百分位延时SLO不能得到满足， E(j,t) < 0

$$D_k(t+1) = \begin{cases} 1 & if\ \exists j, E(j,t) < 0, \\ Max(D_k(t)-1, 1) & else\ if\ Th_k(t) > Th_k^G(t), \\ D_k(t)+1 & else\ if\ E(k,t) \ge U, \\ D_k(t) & otherwise. \end{cases} \quad (14)$$

Th(k, i): 第i 时段$VM_k$ 的实际吞吐率

$Th^G_k(t+1)$:第t+1 时段$VM_k$ 的吞吐率目标

# 两维SLO约束的I/O优化

## 性能测试

测试平台介绍：

主机: 2台 PowerLeader PR2760T servers
*) 2 Intel Xeon E5620 quad-core 处理器,
*) 12GB 内存,
*) 40Gbps Mellanox MT26428 ConnectX VPI Infiniband网卡。

存储: 2类存储子系统
*) 16-disk (7200RPM, 250GB) RAID 0 磁盘阵列
*) Fusion-io ioScale 2, 825GB Multi Level Cell (MLC) SSD

虚拟化软件: Xen 4.2

测试目标:

*) 评测PSLO在任意百分位延时SLO约束下优化吞吐率和保障吞吐率分配的公平性的能力。
*) 评测PSLO在吞吐率SLO约束下优化任意百分位延时的能力。

# 性能测试

## 测试场景

MSN (VM数量: 25 VMs, I/O延时 SLO: 100ms，90% 服从率, I/O吞吐率 SLO: 3750 IOPS)

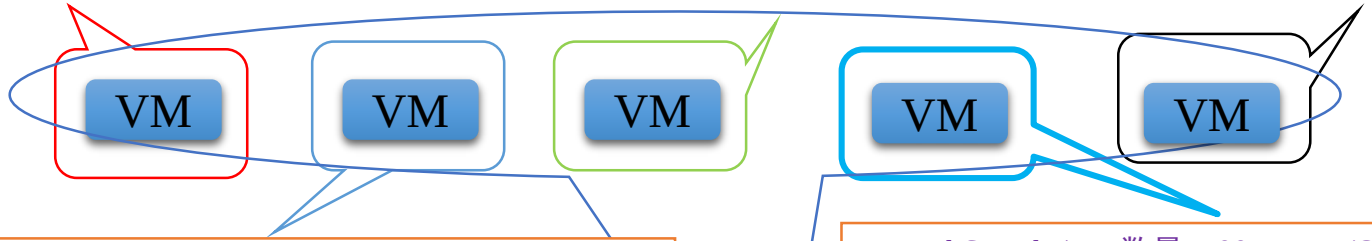
VM-level (I/O吞吐率SLO: 150 IOPS
99.6th 百分位I/O延时SLO: 100ms)

TPC-E (VM数量: 100 VMs, I/O延时 SLO: 200ms， 90% 服从率, I/O吞吐率 SLO: 100000 IOPS)

VM-level (I/O吞吐率: 100 IOPS
99.9th百分位I/O延时SLO: 200ms)

File copy (VM数量: 100 VMs, 无I/O延时 SLO, I/O吞吐率 SLO: 15000 IOPS)

VM-level (I/O吞吐率SLO: 150 IOPS
无既定百分位I/O延时SLO)




Exchange (VM数量: 15 VMs, I/O延时 SLO: 100ms，90% 服从率, I/O吞吐率 SLO: 2250 IOPS)

VM-level (I/O吞吐率 SLO: 150 IOPS
99.3th -ile百分位I/O延时SLO: 100ms)

WebSearch (VM数量: 100 VMs, I/O延时 SLO: 200ms，90% 服从率, I/O吞吐率 SLO: 10000 IOPS)

VM-level (I/O吞吐率SLO: 100 IOPS
99.9th百分位I/O延时Latency SLO of 200ms)

共享存储构架

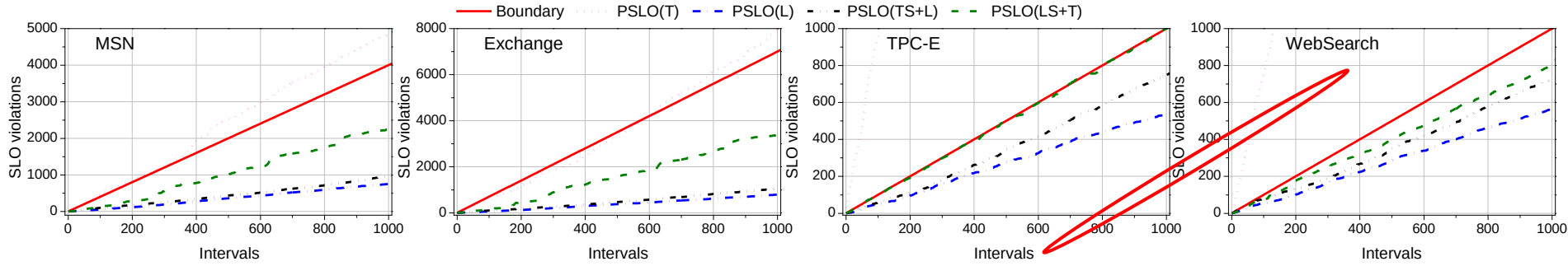
# 性能测试

SLO策略

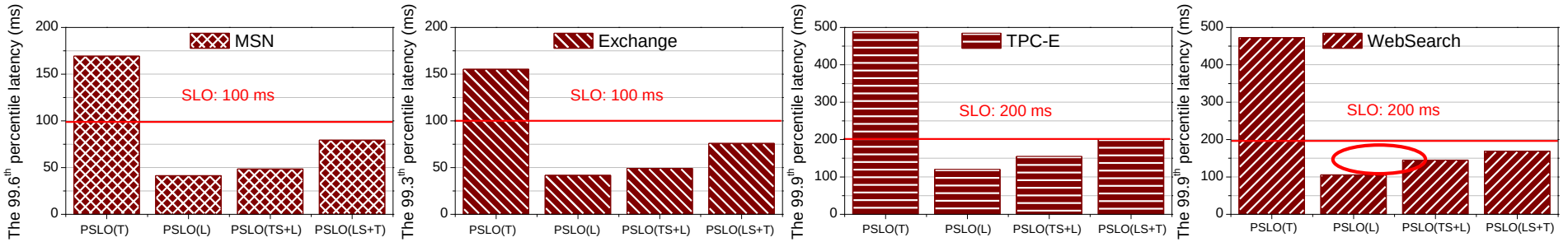
| 策略 | 保障吞吐率SLO | 保障既定百分位延时SLO | 吞吐率分配公平性 | 吞吐率优化 | 延时优化 |
|---|---|---|---|---|---|
| PSLO(TS+L) | Y | Y | Y | N | Y |
| PSLO(LS+T) | Y | Y | Y | Y | N |
| PSLO(L) | N | Y | N | N | Y |
| PSLO(T) | Y | N | Y | N | N |
| No PSLO | N | N | N | N | N |

# 性能测试

## 不同SLO约束下的性能优化



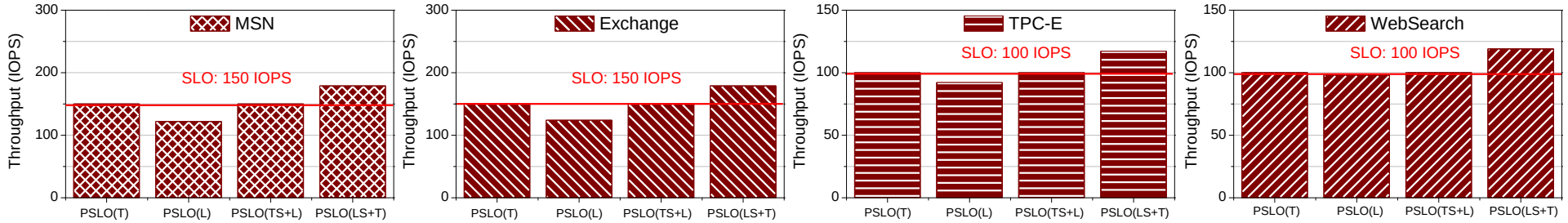

**The 99.6 th-ile** **The 99.3 th-ile** **The 99.9 th-ile** **The 99.9 th-ile**



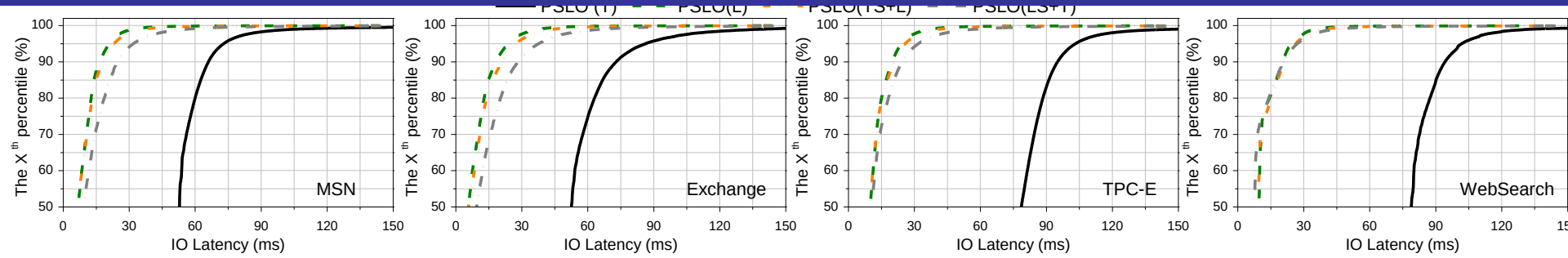

PSLO(LS+T)控制下, 既定百分位延时(200.4 ms)几乎和SLO 目标 (200 ms)一致。

# 性能测试

## 不同SLO约束下的性能优化




在PSLO(LS+T)控制下，四台虚拟机吞吐率分别相对于其吞吐率SLO提升19%, 19%, 18% 和 19%




PSLO(TS+L) 则能在准确保障吞吐率SLO的前提下优化既定百分位延时

# 本讲小结

- 提出聚合虚拟机环境中面向高存储资源利用率的尾延时SLO精确保障方案，该方案可在精确保障尾延时SLO的前提下优化I/O资源分配。
- 提出聚合虚拟机环境下既定百分位I/O延时和吞吐率SLO精确保障以及基于SLO的性能优化方案，该方案可以支持一台主机上运行的多个虚拟机具有完全不同百分位的I/O延时SLO和I/O吞吐率SLO，并实现基于上述两维SLO约束的I/O性能优化。

# 总结

- 数据中心虚拟化平台实践
- 虚拟化环境中存储系统面临什么问题
- 怎样进行服务质量精确保障
- 怎样在此基础上进行多维保障及优化