On Horizontal Decomposition of the Operating System

Gang Lu

Beijing Academy of Frontier Science and Technology

BPOE-7, ASPLOS 2016
Atlanta, GA, USA

Contact: lugang@mail.bafst.com

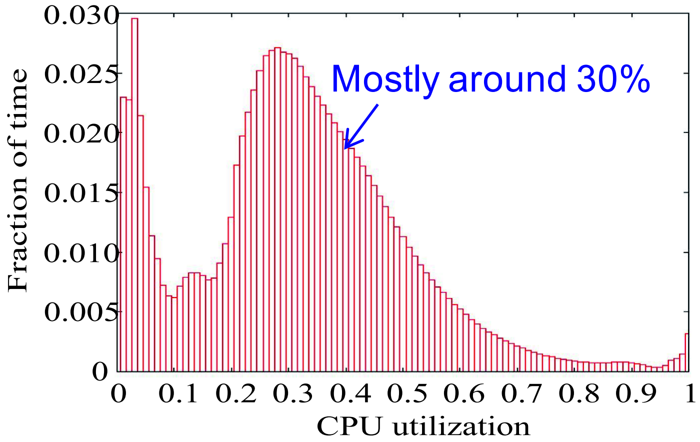
Technical Report: https://arxiv.org/abs/1604.01378

Content

- Background & motivation
- Related work
- Design and implementation
  - A new OS model—horizontal OS model
  - A new OS abstraction—subOS
  - The first prototype—RainForest
- Evaluation
- Conclusion

Low resource utilizations

- **Average utilization < 40%[1]**
  - **Google: < 40%**
  - **Amazon: < 30%**
  - **VMWare: < 30%**
  - **Mozilla: < 10%**
  - **Others: < 10%**

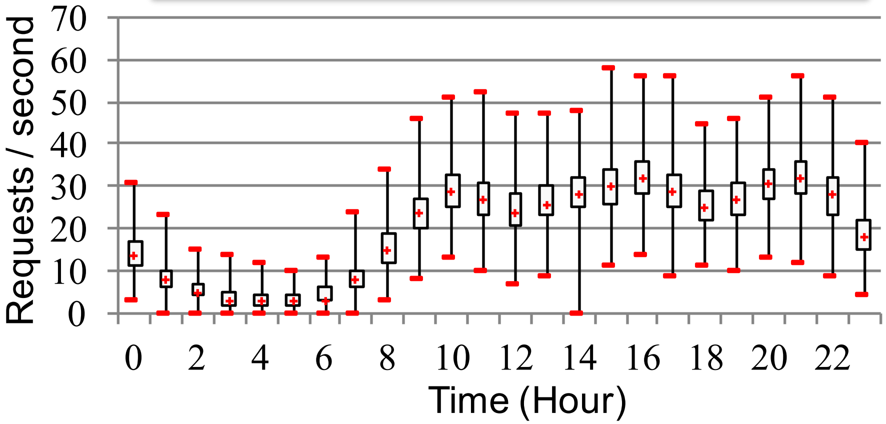


Application specific resource requirements[2]

Online service
+
Offline analytics

- Search engine
- Social networks
- E-commerce
- Multi-media
- Bioinformatics

Workload fluctuation (Resource overprovision)



[1] Warehouse-Scale Computing: Entering the Teenage Decade. Luiz Andre Barroso, Google. ISCA'11
[2] Bigdatabench: A big data benchmark suite from internet services. Lei Wang, etc.. HPCA'14.

Workload consolidation?

- **Consolidate workloads: to simultaneously run on the same machine**
  - **Severe interference**

Degradation > 20%

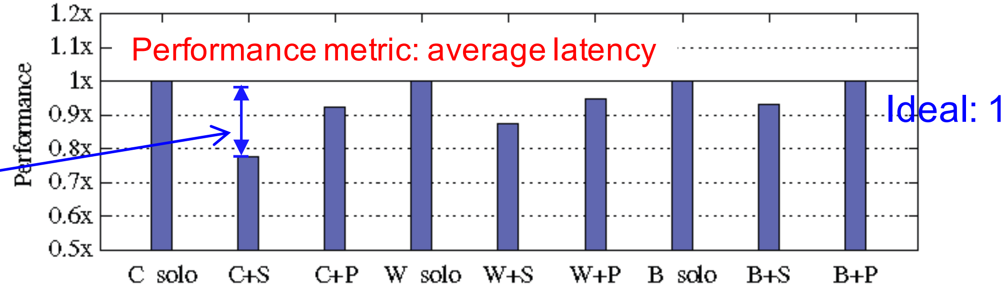
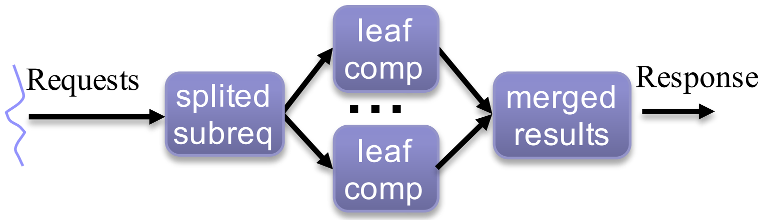


Fig 1. Performance interference scenarios of consolidating different applications in Google data centers，*solo* denotes running alone [1]

  - **Large scale online services are more interference sensitive**



Only all leaf components satisfy QoS, the root (merged responss) can satisfy QoS

$P(root<1s)=P(leaf<1s)^N$

P: percentage of requests satisfying QoS

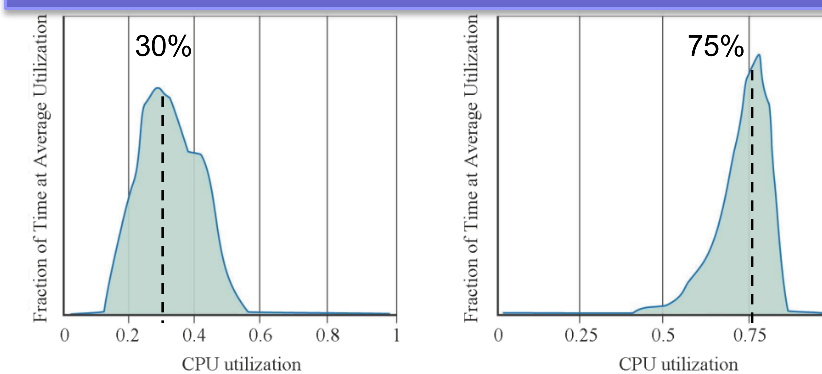
If we take tail latency as the performance metric, for a 100-leaf distribution, to make

$P(root < 1s)=0.99$，

We must guarantee：

$P(leaf < 1s) >0.999899$

Google deploys online services and offline batches in different clusters[2]



Online services

Offline batches

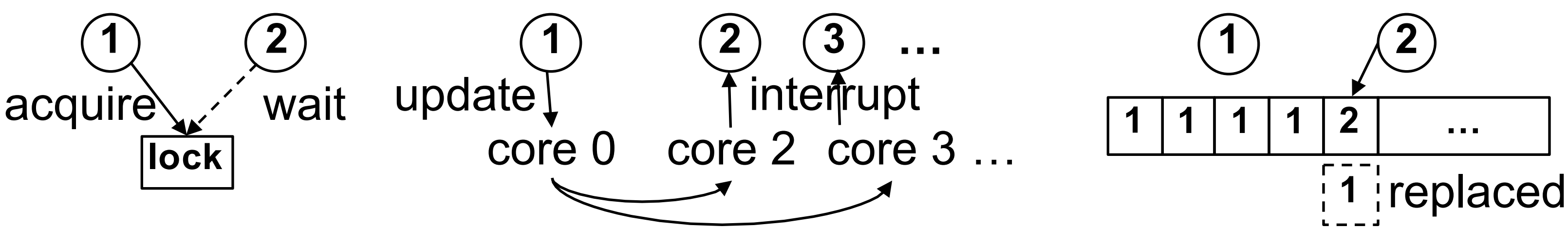
Fig 2. Resource utilization distributions of Google's two clusters[2]

[1] The impact of memory subsystem resource sharing on datacenter applications. Lingjia Tang, etc.. ISCA'11.
[2] The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition. Google, Inc. . 2013

Causes of interference

- **Interference points——shared states**

Structures protected by locks	Synchronization among replicas	Shared software buffers
big kernel lock，vm_commited_as，vfsmount…	TLB shootdown	file system buffer inode cache，dentry cache…
contend lock => stalls	forced sync => interruption	cache eviction=> reload

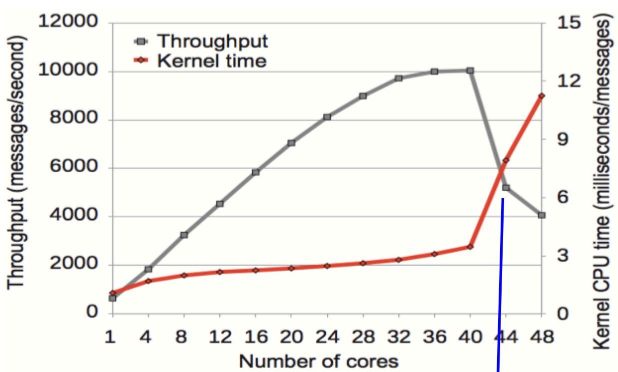


Globally shared states → soft-level interference; sync cost → (with increasing cores) → scalability problem

Globally shared states → (cache consistency) → sync cost in architecture level → scalability problem

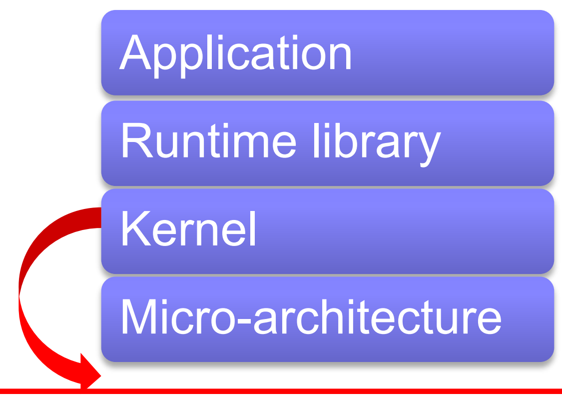
Increasing cores challenge scalability

- **Manycore processors become a trend**
  - **Intel released 18-core processor**
    - **4 Sockets constructs a 72-core server**
  - **The low resource utilization problem deteriorates**
    - **More resource waste for running a single workload**
    - **Performance bottleneck with increasing cores**



Scalability evaluation of Email server Exim running on Linux[1]

Performance degrades dramatically with increasing execution time of kernel



Many shared structures in the kernel (also cause interference)

[1] An analysis of linux scalability to many cores. S. Boyd-Wickizer, etc.. OSDI'10.

Isolation & Scalability of Linux

- **Take Linux as an example**

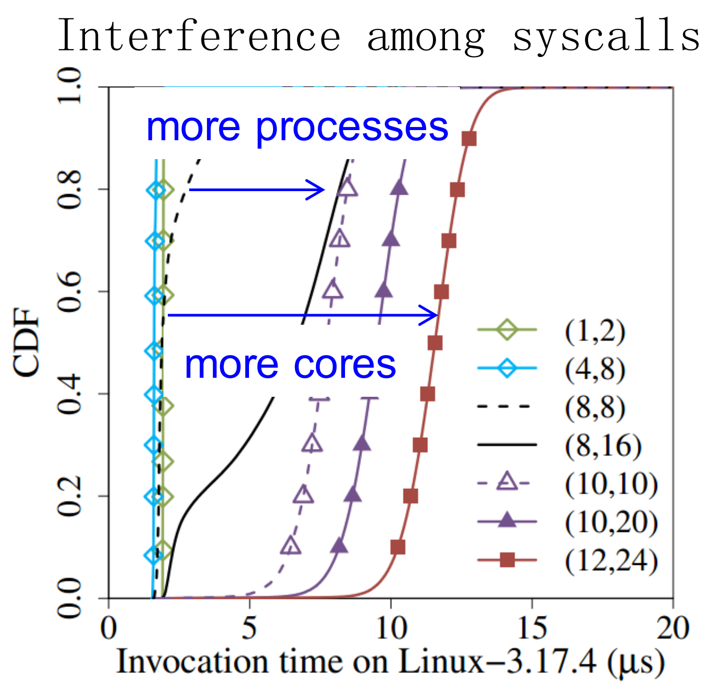


Fig 1. Latency distribution of syscall mmap,(x, y) denotes y processes run on x cores （*Left is better*）

Causes：updates of the global variable *vm_commited_as*

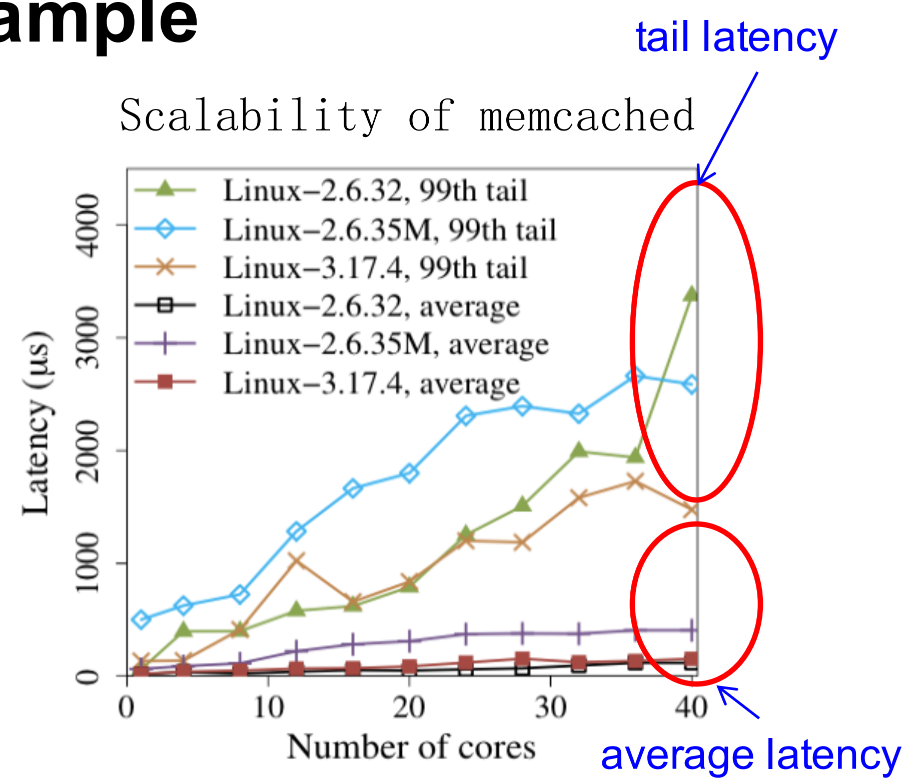


Fig 2. Tail latency performance of memcached with increasing cores （Smaller is better）

Causes：lock-protected structures, like inode list, Dcache lists

Motivation

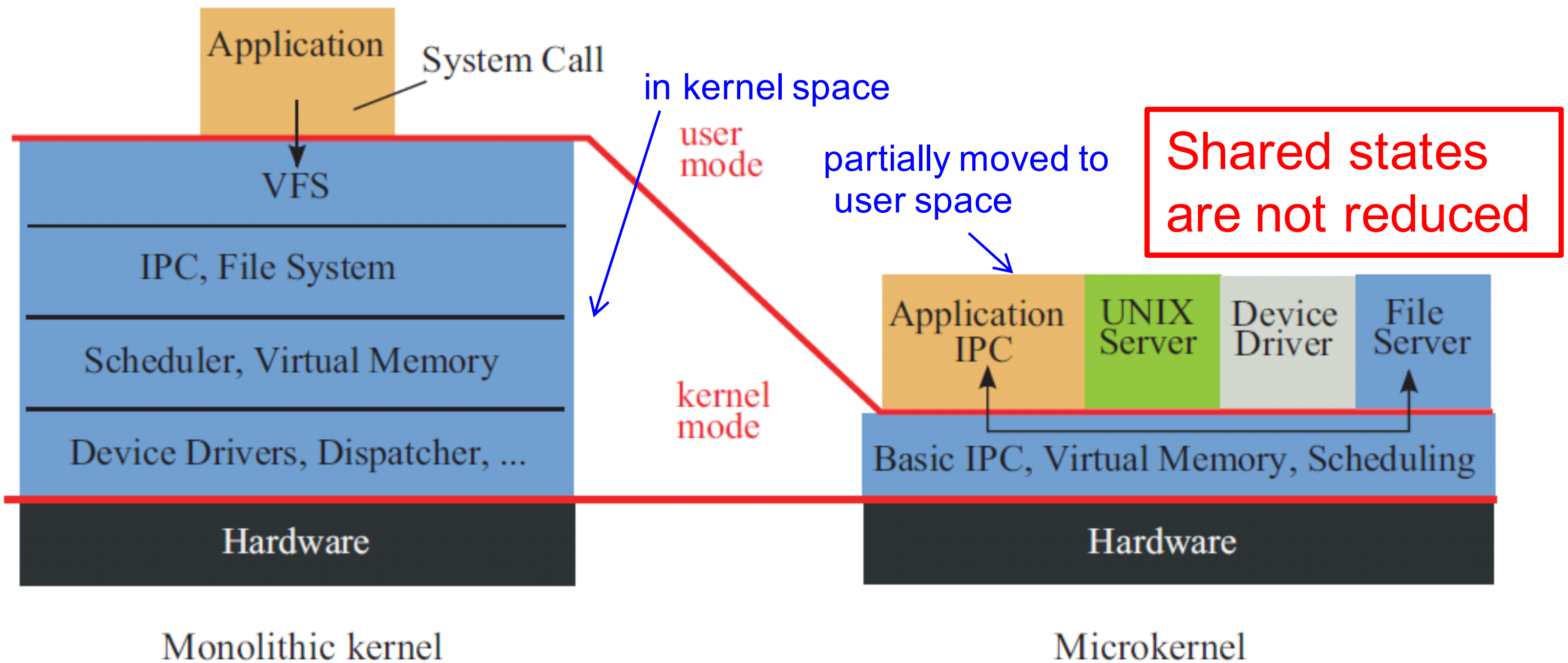
- Improve performance isolation from the OS level
  - Construct high-isolation OS structures and prototype
    - Improve isolation for consolidated workloads
    - Improve tail latency performance
- Improve the scalability from the OS level
  - Construct high-scalability OS structures and prototype
    - Improve OS scalability for manycore platforms

Content

- Background & motivation
- Related work
- Design and implementation
  - A new OS model—horizontal OS model
  - A new OS abstraction—subOS
  - The first prototype—RainForest
- Evaluation
- Conclusion

Monolithic kernel & micro kernel

- Many globally shared data structures
  - Monolithic： mostly in kernel space
  - Micro： partially in kernel space， partially user space



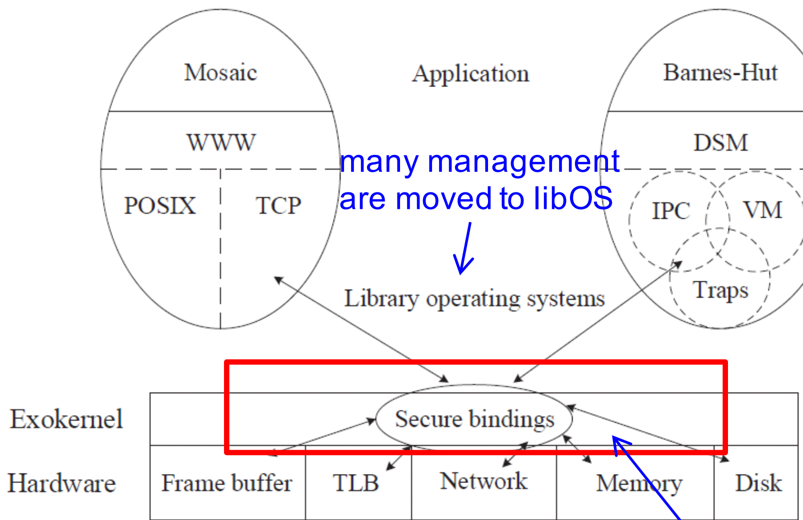
Linux[1]， Unix， Windows

Mach[2]， L4

[1] Linux kernel website. https://www.kernel.org/.
[2] On micro-kernel construction. Liedtke, Jochen. ACM, 1995.

Exokernel & VMMs

- Shared data structures still exist
  - Exokernel： reduced kernel functions
  - VMM： centralized resource management, an exokernel-like structure



many management are moved to libOS

Fig 1. Architecture of exokernel[1]

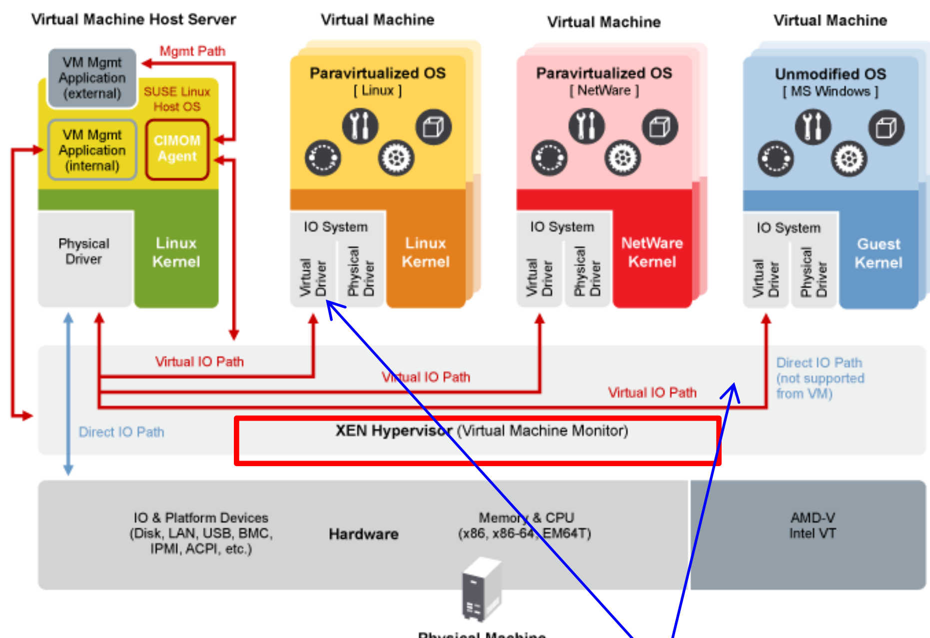


Fig 2. Architecture of Xen[2]

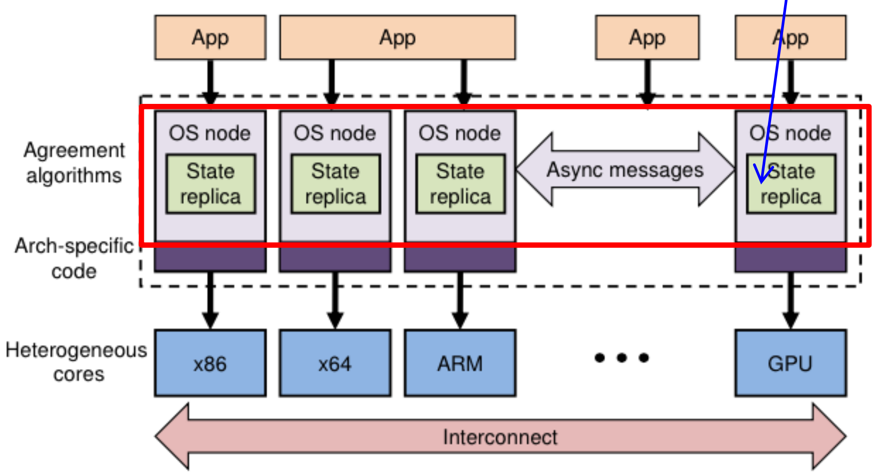
central access control still exists

hypervisor & virtualized resources; central access control

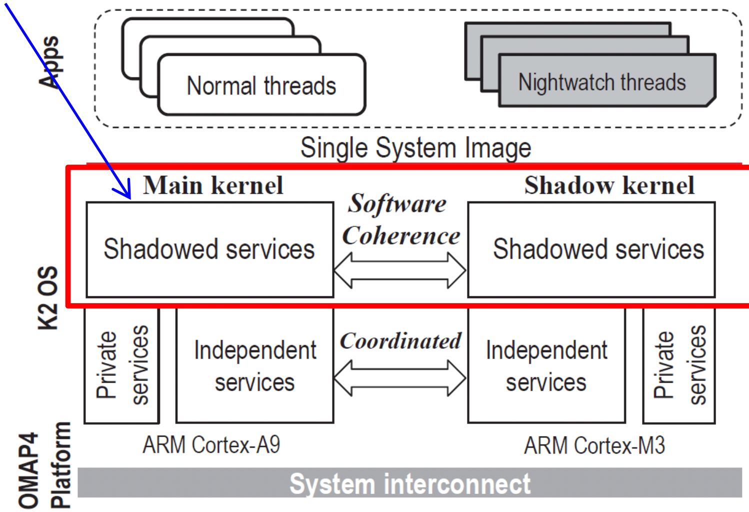
[1] Exokernel: An operating system architecture for application-level resource management. D. R. Engler, et.al.. ACM, 1995.
[2] Xen and the art of virtualization. P. Barham, et al..SIGOPS OSR, 2003.

Multi-kernel structures

- **Globally shared data structures**
  - **Need to maintain global consistency**
    - **Distributed message communication (one-phase, two-phase commitment)**
    - **Distributed shared memory**



Architecture of Barrelfish [1]



Architecture of K2[2]

States are still globally shared

[1] The multikernel: a new os architecture for scalable multi- core systems. A. Baumann, et al.. SOSP'09.
[2] K2: A mobile operating system for heterogeneous coherence domains. F. X. Lin, et al.. ASPLOS'14.

Summary of existing OS models

- **Make balance between "sharing" and "isolation"**
  - **First sharing, later isolation**
    - **Make kernel in charge of both resource provision and management**
    - **Have to share states globally**
    - **Construct OS abstractions upon shared states in kernel space**

We cannot get rid of maintaining consistency of globally shared states

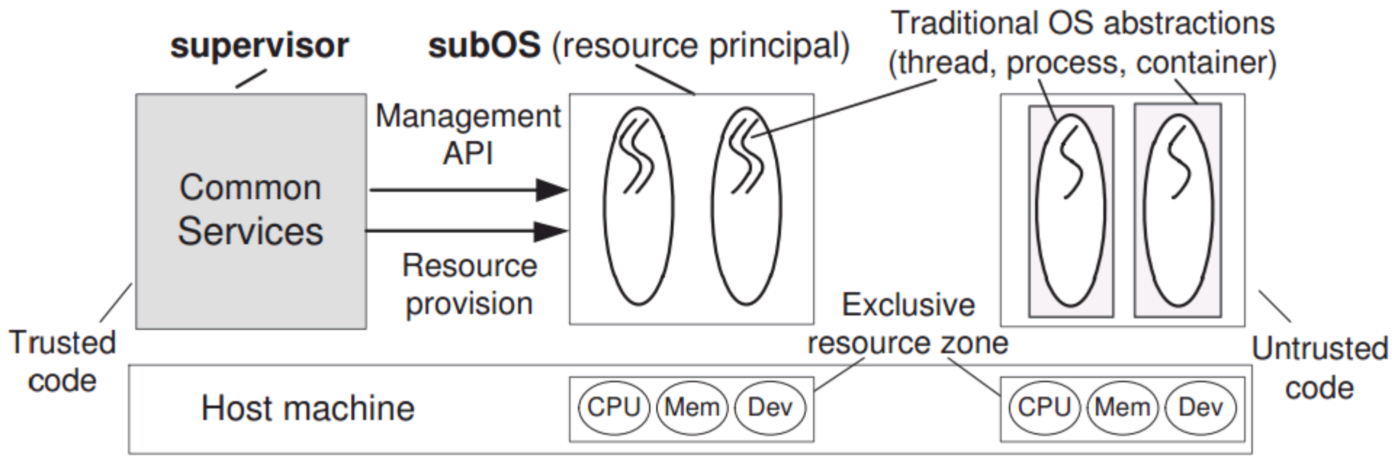
First isolation, later sharing?

Content

- Background & motivation
- Related work
- Design and implementation
  - A new OS model—horizontal OS model
  - A new OS abstraction—subOS
  - The first prototype—RainForest
- Evaluation
- Conclusion

Our method: horizontal OS model

- **A new OS model——First isolation, later sharing**
  - **Design principles**
    - **Horizontally decomposed OS functions** — resource management
    - **Isolation and elasticity of OS instances** — isolation and elasticity
    - **Confined and on-demand state sharing** — isolation and sharing



Architecture of the horizontal OS model

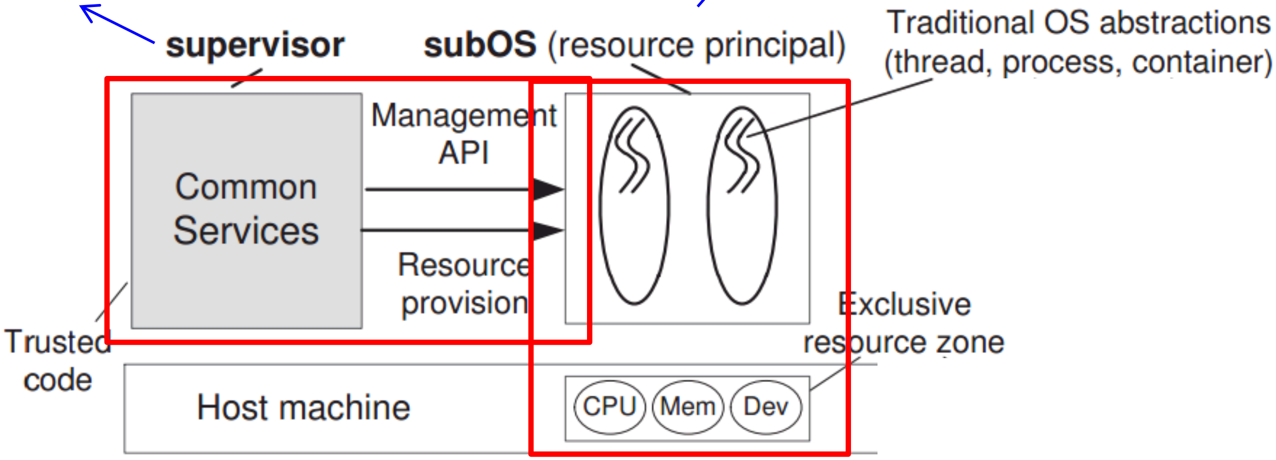
Horizontal OS model

- **A new OS model**
  - **Design principles**
    - **Horizontally decomposed OS functions**
      - **"resource provisioning" & "resource management"**

resource management → isolation and elasticity → isolation and sharing

Discover, monitor, provision physical resources: resource pool, allocation/reclaim

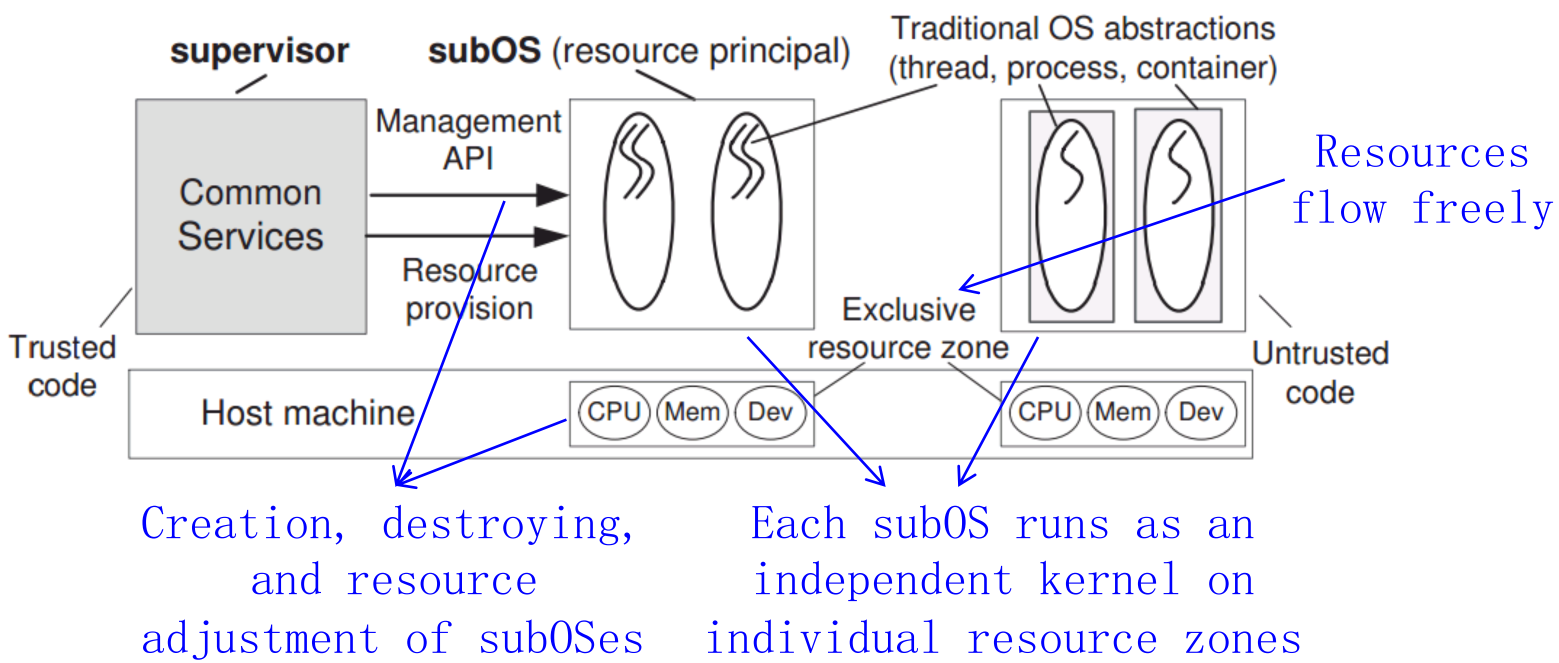
Directly drive physical resources, provide upper-layer abstractions



Horizontal OS model

- **A new OS model**
  - **Design principles**
    - **Horizontally decomposed OS functions**
    - **<u>Isolation and elasticity of OS instances</u>**

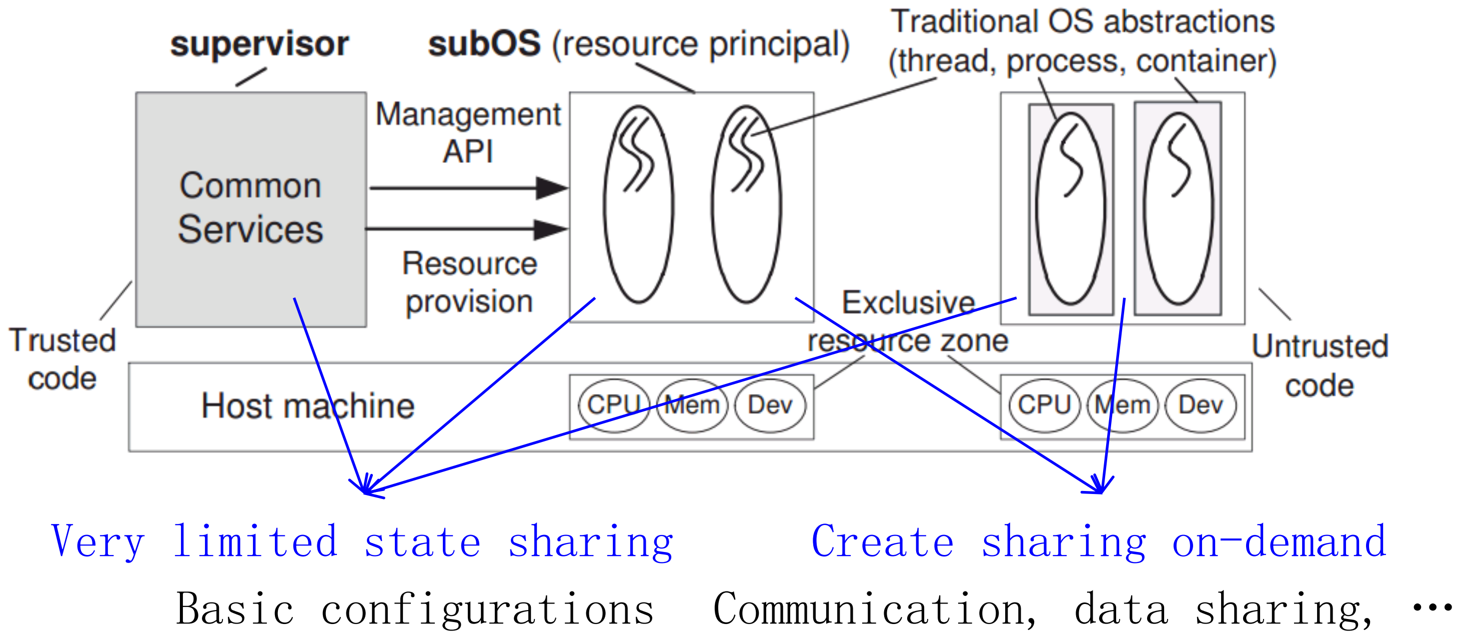
resource management → isolation and elasticity → isolation and sharing



Horizontal OS model

- **A new OS model**
  - **Design principles**
    - **Horizontally decomposed OS functions**
    - **Isolation and elasticity of OS instances**
    - **Confined and on-demand state sharing**

resource management → isolation and elasticity → isolation and sharing



Content

- Background & motivation
- Related work
- Design and implementation
  - A new OS model—horizontal OS model
  - A new OS abstraction—subOS
  - The first prototype—RainForest
- Evaluation
- Conclusion

OS abstractions

- **Traditional OS abstractions**
  - **Process, thread, container, virtual machine**

Process (e.g. Linux)	subOS
creating and destroying of a process, resource allocation and reclaim	dynamically create/destroy a subOS, resource adjustment
inter-process communication(IPC)	construct communication channels among subOS based on shared memory
parent-child relationship (fork)	self-propagation (fork a child subOS)
resource accounting provided by kernel（*proc* file system）	coarse-grained accounting in supervisor, fined-grained accounting in each subOS
kernel provides sys calls and services	supervisor provides APIs
obtain hardware topology and resource information through system calls	get the information through each subOS

A new OS abstraction

- Relationship between subOS and others
  - subOS can run processes, containers, even other OS structures, like VMM, micro-kernel, etc.
  - subOSes can be run in VMMs

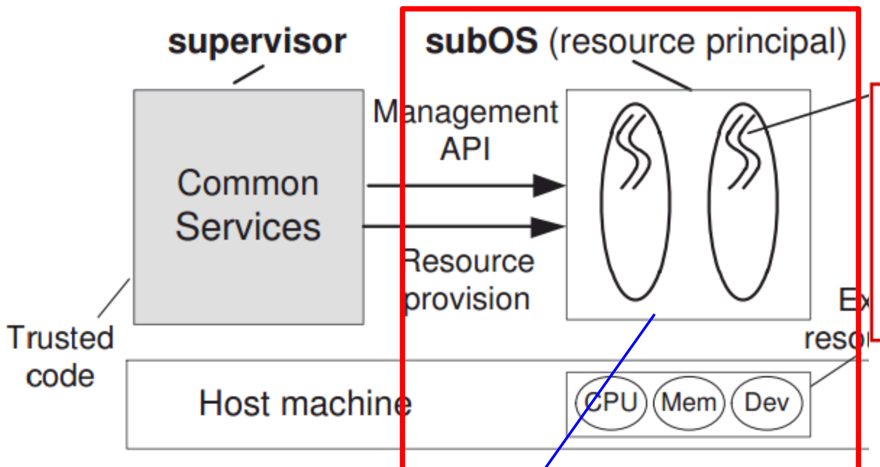


Fig 1. The horizontal OS model

A process (or libOS), a full-featured kernel, container, virtual machine, etc.

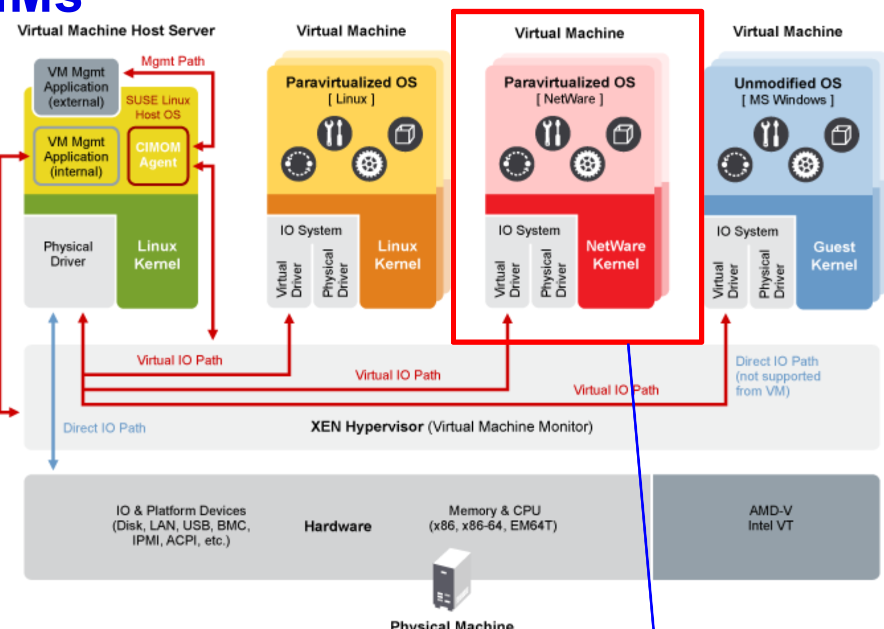


Fig 2. The VMM architecture

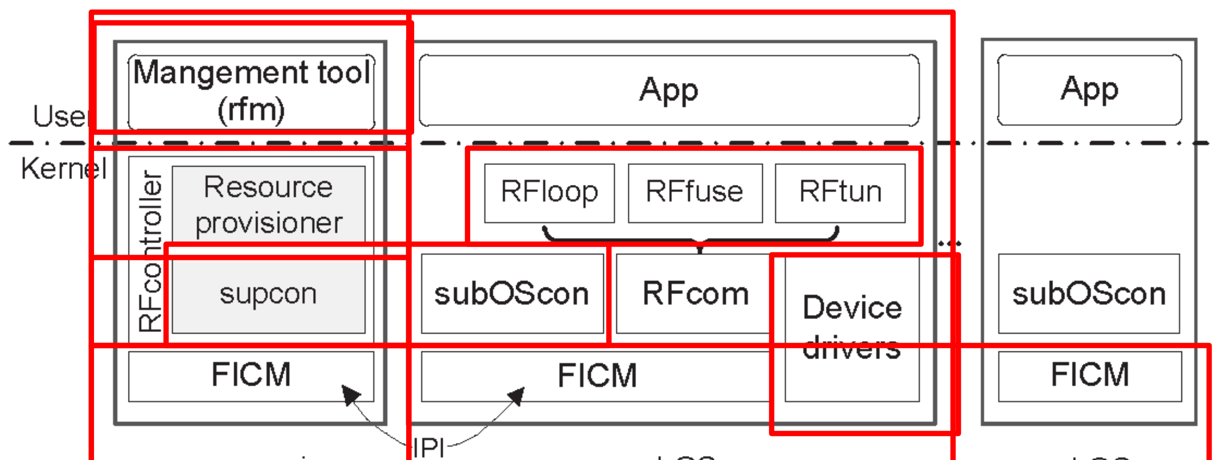
A horizontal OS with subOSes

Content

- Background & motivation
- Related work
- Design and implementation
  - A new OS model—horizontal OS model
  - A new OS abstraction—subOS
  - The first prototype—RainForest
- Evaluation
- Conclusion

A prototype: RainForest

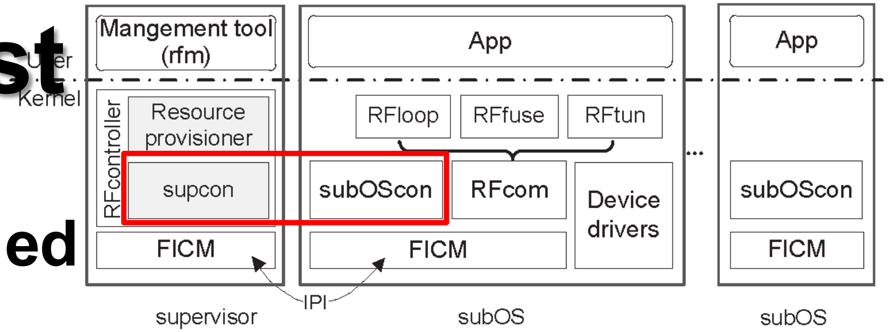
- Based on the horizontal OS model
- Works on homogeneous SMP platforms
  - X86, a single Root Complex, global cache consistency
- Architecture of RainForest
  - supervisor is based on Linux, booted with the hardware
  - subOS adopts independent monolithic kernel, booted by supervisor



System functions of managing a subOS need to be re-built because of structural change!

IPI: inter-processor interrupt

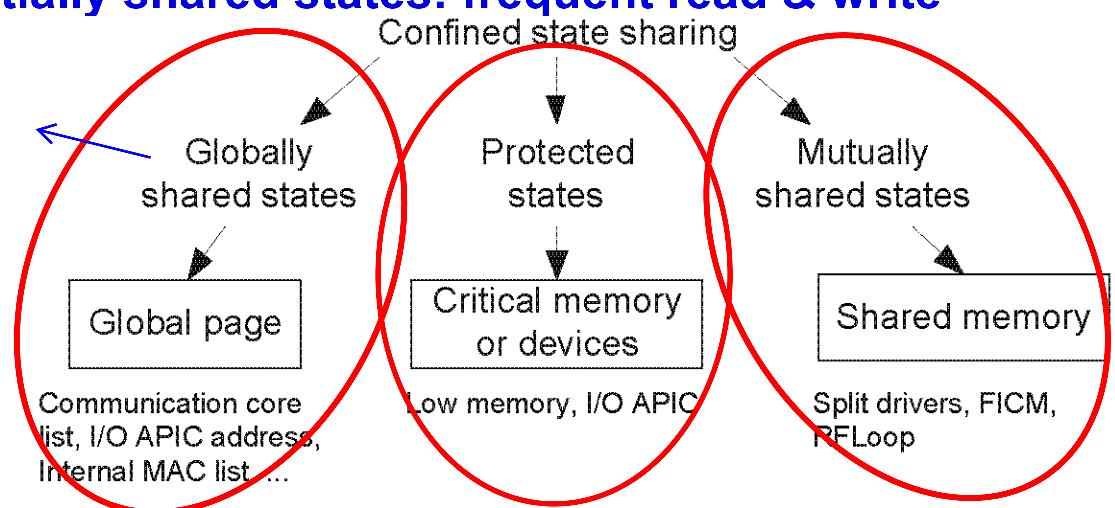
Design of RainForest



- **Shared states are managed using different policies**
  - **Globally shared states: more read, less write**
  - **States protected by supervisor: R/W infrequently**
  - **Partially shared states: frequent read & write**

Reduce global shared states and participation of supervisor

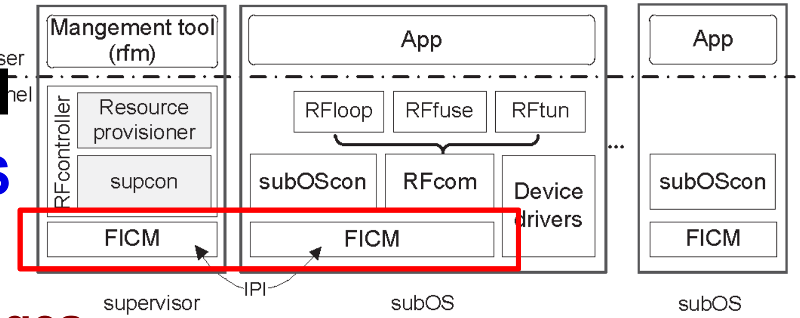
Traditional method

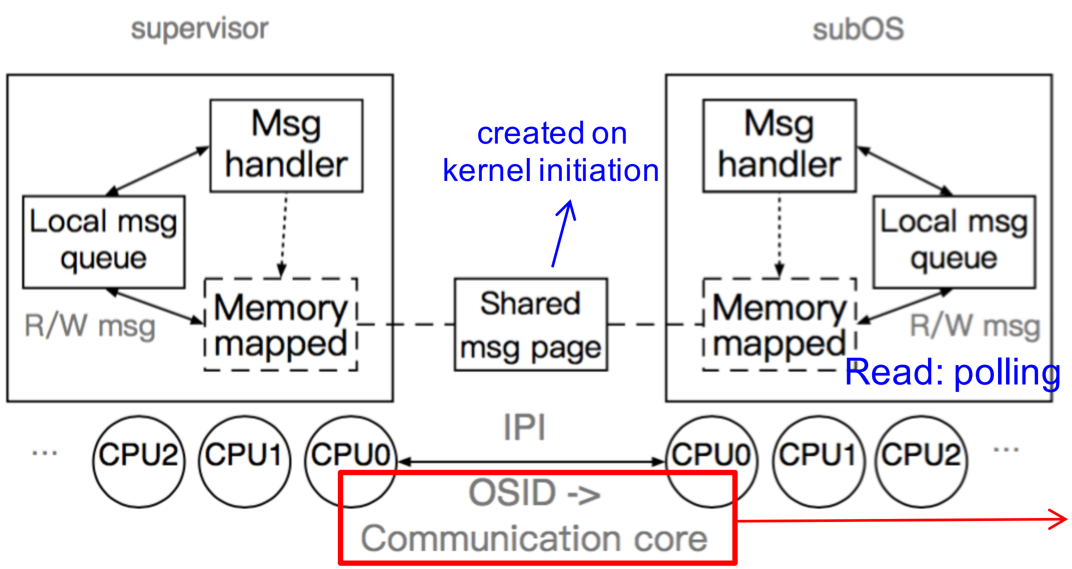


lock protection	supervisor authorization	self-defined shared states in shared memory
globally shared	globally shared	small sharing scope
more reads	R/W infrequently	frequent R/W

Design of RainForest

- **Communication based on IPI**
  - **FICM: a basic inter-core/subOS communication channel**
    - **Designed for short, fast messages**





Working mechanism of FICM

Globally shared states

A global page:
lock1 to protect
OSID-Communication core list

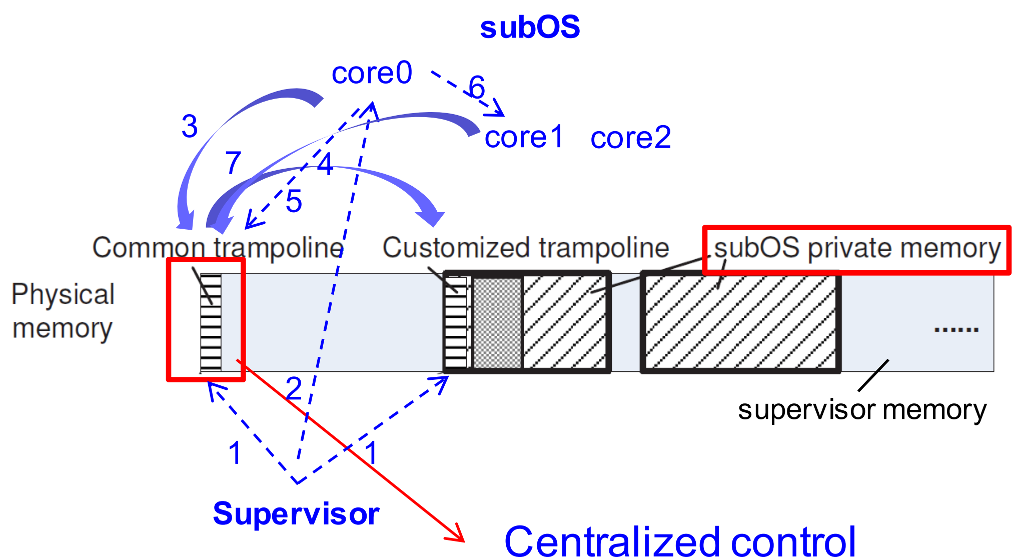
lock2 to protect
OSDI-VIF MAC list
……

Basis of the elasticity of subOS

OSID is the ID of a subOS

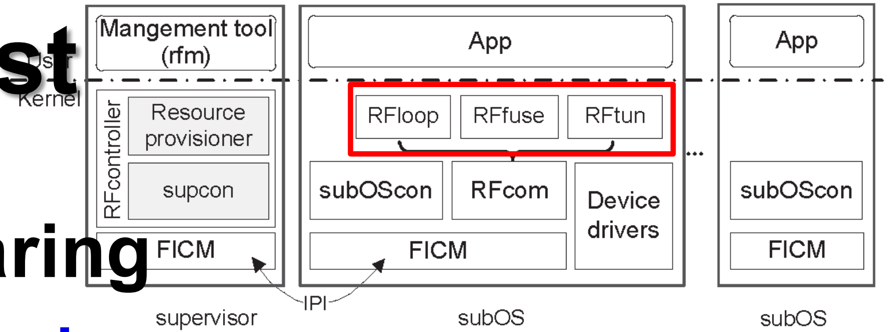
Design of RainForest

- Boot procedure of a subOS
  - supervisor assists to RESET the first core(BSP)
  - Then other cores are booted by subOS itself



Protected by supervisor:
Low memory
I/O APIC pin
......

Design of RainForest



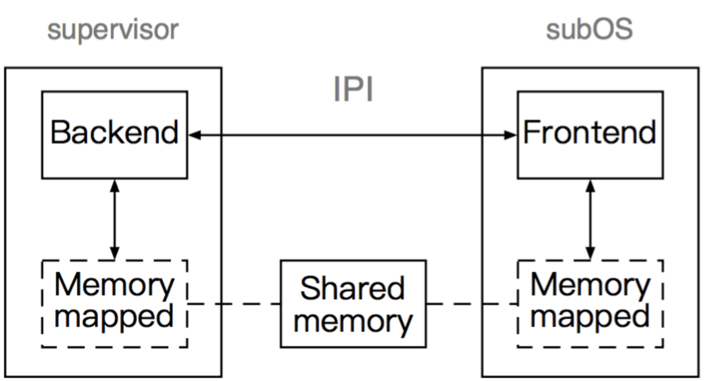
- **On-demand state sharing**
  - **Based on IPI and shared memory**
    - **subOSes communicate using RFloop**
    - **other split device drivers**

Split driver in RainForest

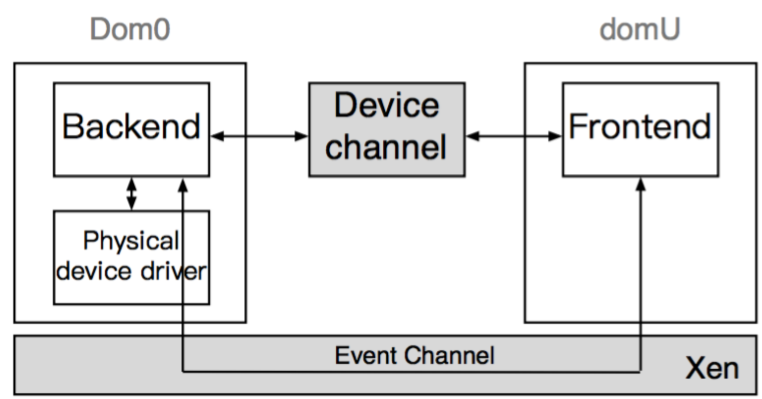
1. Combination of IPI and polling
2. Self-controlled memory mapping
3. Direct memory translation (VA-PA)

Split driver in Xen

1. Globally shared event channels
2. VMM managed memory mappings
3. Two-phase memory translation



Frontend backend structure (RainForest)



Frontend backend structure (Xen)

Implementation of RainForest

- A full-featured OS
  - supervisor and subOS are based on Linux-2.6.32
  - Most functions implemented as modules, easy to install or remove
  - Runs stable in many platforms
    - **Intel Xeon E5620, E5645, E5-2620, E5-2640, and E7-8870**

The refactoring efforts based on Linux-2.6.32

Component	Number of Lines
The primary booted OS instance	994
The RFcontroller module	1837
A subOS	1969
FICM / RFcom / subOScon	1438 / 1522 / 1331
RFloop / RFtun / RFfuse	752 / 2339 / 980
others (rfm)	4789

Core parts: not much modified

Including insertions and modifications

Content

- Background & motivation
- Related work
- Design and implementation
  - A new OS model—horizontal OS model
  - A new OS abstraction—subOS
  - The first prototype—RainForest
- Evaluation
- Conclusion

Evaluation and analysis

Testbed and benchmarks

Fig 1. Version information of different systems

Systems	Kernel Ver.	Software Ver.	Execution entity
Linux-2.6.32	2.6.32	-	Processes
Linux-3.17.4	3.17.4	-	Processes
Linux-2.6.35M	2.6.35Modified	-	Processes
LXC-host	2.6.32	0.7.5	-
LXC-containers	2.6.32	-	Containers
Xen-VMM	-	4.0.0	-
Xen-domain 0/U	2.6.32	-	VM
RainForest-Supervisor	2.6.32	1.0	-
RainForest-subOSes	2.6.32	1.0	subOS

Fig 2. Information of two different servers

	Server-A	Server-B
CPU type	Intel Xeon E5645	Intel Xeon E7-8870
Number of cores	6 cores@2.4GHz	10 cores@2.4GHz
Number of threads	12	20
CPU sockets	2	4
L1 DCache	32KB, 8-way associative, 64 byte/line	
L1 ICache	32KB, 4-way associative, 64 byte/line	
L2 Cache	256 KB, 8-way associative, 64 byte/line	
L3 Cache	12 MB	30 MB
	16-way associative, 64 byte/line	
DRAM capacity	32 GB, DDR3	1 TB, DDR3
Network interface cards	8 Intel igb ethernet 1000Mb/s	
	2 Broadcom ethernet 1000Mb/s	
Hard disk drives	8 Seagate 1TB 7200RPM, 64MB cache	

Fig 3. Benchmarks and main configurations

Benchmarks	Ver.	Main configurations	Sources
Will-It-Scale	1.0	Infinite loops	Mbench
SPEC CPU	2006	-	Mbench
PARSEC	3.0	Large datasets	Mbench
cachebench	1.0	-	Mbench
netperf	2.6.0	-	Mbench
iozone	3.420	-	Mbench
memcached	1.2.2	Keep full load	mosbench & Mbench
Search(nutch)	1.1	Request rate: Xen(240req/s) Others(300req/s)	BigDataBench & Mbench
Spark	1.0	TPC-DS, BigDataBench	BigDataBench & Mbench
Hadoop	1.0.2	BigDataBench	BigDataBench & Mbench
Metis	1.0	1~5G records	mosbench

Features evaluated:

- subOS performance
- tail latency performance
- performance isolation
- elasticity
- scalability
- performance of individual components

Evaluation: performance isolation

（Smaller is better）

	0	1	2	3	4	5	6	7	8	9	10
0	1	3	6	30	22	0	0	0	1	0	0
1	2	8	7	49	36	2	1	2	1	0	0
2	0	0	0	7	4	0	0	0	0	0	0
3	4	16	23	50	47	1	1	1	2	0	0
4	1	11	20	50	47	0	1	1	1	0	0
5	2	7	10	36	32	33	15	0	1	0	0
6	7	15	7	51	48	25	0	0	5	0	1
7	4	9	11	40	36	31	3	32	2	0	0
8	0	0	0	0	0	0	0	0	50	0	0
9	1	14	3	0	0	0	0	1	90	0	0
10	0	0	0	2	0	0	0	0	72	1	1

(a) Linux-3.17.4

	0	1	2	3	4	5	6	7	8	9	10
0	1	3	6	30	23	1	1	1	0	0	0
1	2	8	8	52	36	1	1	1	1	1	1
2	0	0	0	7	4	0	0	0	2	0	0
3	4	13	24	50	47	1	1	1	1	0	0
4	2	9	21	51	47	1	1	0	0	1	0
5	7	24	25	44	42	0	0	0	3	3	2
6	6	16	18	51	49	0	0	8	3	3	2
7	6	13	11	45	42	1	7	0	2	1	2
8	0	0	0	0	0	0	0	0	0	0	0
9	2	8	1	10	2	0	2	0	18	0	8
10	0	0	0	1	0	0	0	0	0	0	0

(b) LXC

	0	1	2	3	4	5	6	7	8	9	10
0	1	5	10	36	26	1	1	1	0	0	1
1	3	9	11	50	35	0	0	0	0	0	0
2	0	0	0	5	3	0	0	0	0	0	0
3	2	9	15	49	43	0	1	1	0	0	0
4	2	4	10	41	35	0	0	0	0	0	0
5	0	0	0	5	0	0	0	4	0	0	0
6	0	0	0	27	30	0	0	5	0	0	0
7	13	0	3	45	19	12	5	12	0	0	12
8	0	0	0	0	0	0	0	0	65	0	0
9	0	0	0	0	0	0	0	72	89	0	0
10	11	10	5	2	2	1	0	61	56	2	1

(c) Xen

	0	1	2	3	4	5	6	7	8	9	10
0	1	4	6	21	13	0	1	0	1	0	1
1	2	6	1	30	17	1	1	1	1	1	1
2	0	0	0	1	0	0	0	0	0	0	0
3	0	0	4	26	18	1	1	1	0	0	0
4	0	2	7	28	21	1	1	1	1	0	0
5	4	7	7	14	24	0	15	0	3	1	1
6	1	7	5	25	21	2	0	1	2	2	2
7	2	7	9	27	24	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	2	2	0	0	0	0	0	0	0	0	1

(d) RainForest

Heatmap of the performance interference (degradation percentage of performance) of two workloads in a single server. x-axis denotes the interferers, y-axis denotes the victims.

Workload: *0-11 denotes different micro benchmarks covering CPU, cache, filesystem, and network*

Server: ***12 cores，32G RAM***

OS instances: ***2***

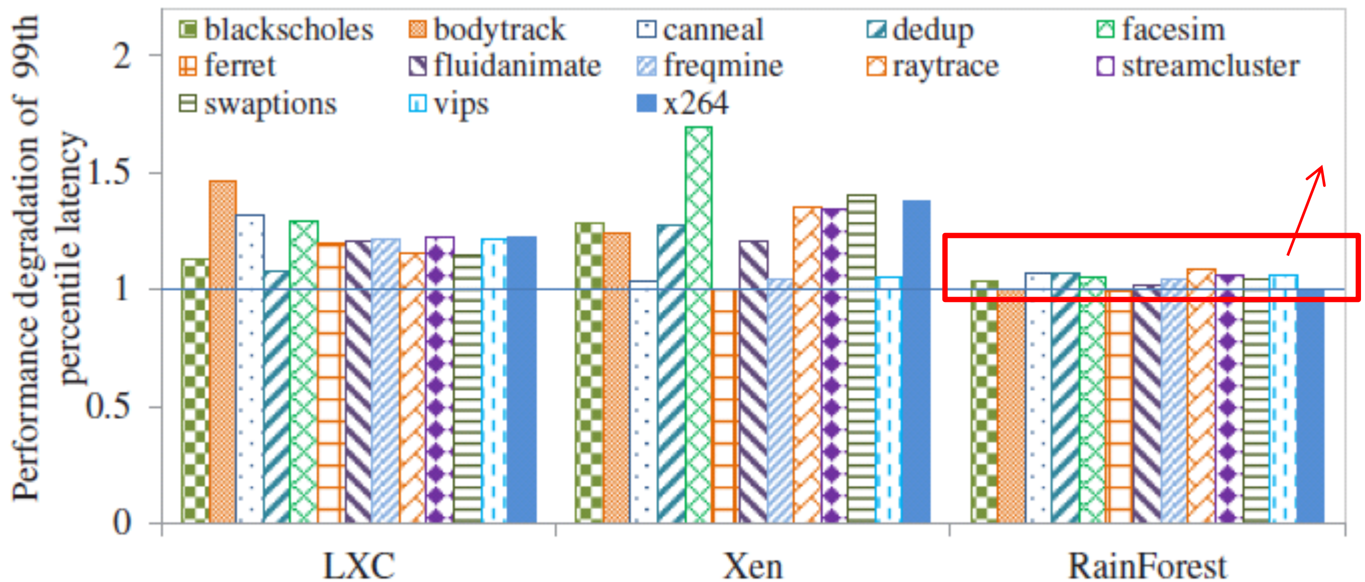
Every OS instance runs a workload on

6 cores,16G RAM

On RainForest, interference is lower

Evaluation: performance isolation

- **Online services interfered by offline batches**



Degradation of tail latency performance of Search on RainForest is less than 5%

Performance(99th percentile latency) degradation of Search when interfered by Parsec workloads, comparing to the performance with no interferer

Workload: *Search + Parsec*
Server: ***12 cores, 32G RAM***
OS instances: ***2***
Each OS instance runs a Search backend *or* a Parsec workload on *6 cores,16G RAM*

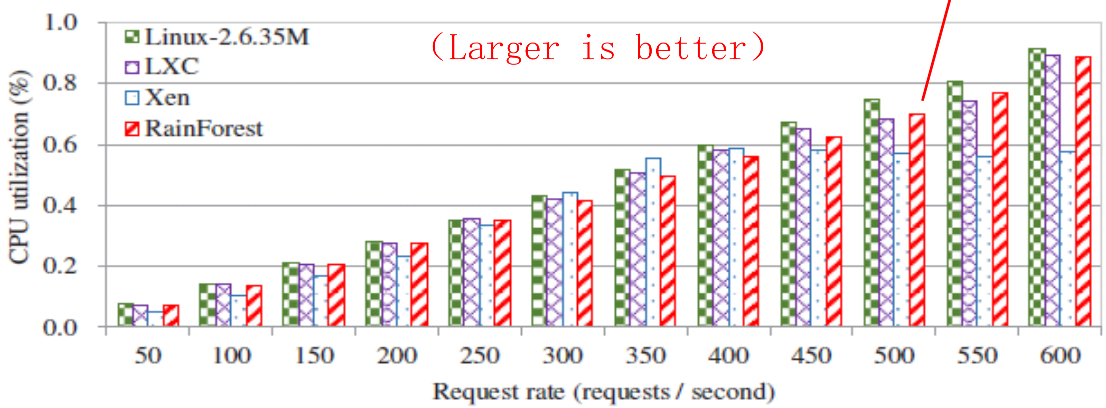
Evaluation: Tail latency

- **Tail latency performance of latency critical workloads**

(a) 99th percentile latency

Workload: ***Search***
Server: ***12 cores, 32G RAM***
OS instances: ***2***
Each OS instance runs a backend server of Search on *6 cores,16G RAM*

RainForest



(b) CPU utilization

Maximum throughputs of Linux, LXC, Xen, and RainForest are 400, 350, 350, and 500 (*QoS: 200ms*)

CPU utilizations: 59.8%, 58.0%, 55.7%, and 69.7% (*QoS: 200ms*)

Evaluation: cost of elasticity

- **Cost of resource adjustment**

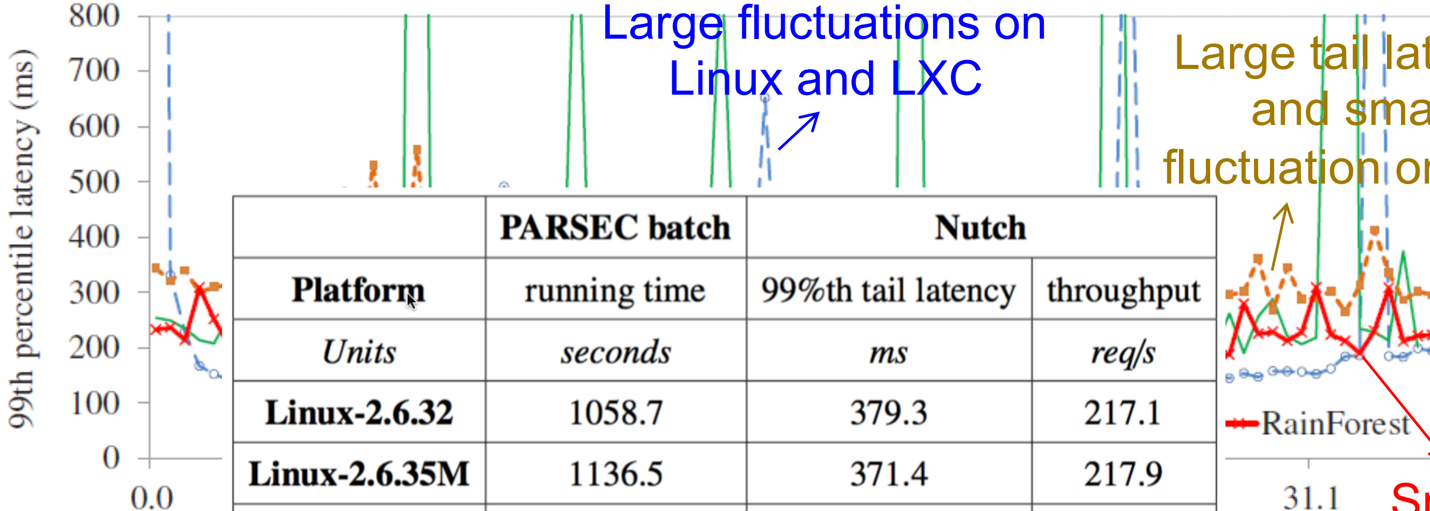
Table 1. Cost of resource adjustment in each system (in seconds).

Configuration	6 CPUs, 16G RAM		6 CPUs, 16G RAM		1 CPU		512M RAM	
Operations	create	destroy	create	destroy	online	offline	online	offline
LXC	2.1	~0	2.1	1	0.002	0.002	0.002	0.002
Xen	14.2	5.9	255.2	240.0	0.126	0.127	0.167	0.166
RainForest	6.1	~0	6.1	5.4	0.066	0.054	0.020	0.006

Evaluation: elasticity

Workload: *Search + Parsec*
Server: ***12 cores，32G RAM***
OS instances: ***2***
Each OS instance runs a Search backend or a Parsec workload initially on *6 cores,16G RAM*

- **Make cores adjust**



Large fluctuations on Linux and LXC

Large tail latency and small fluctuation on Xen

Small fluctuation on RainForest

Fig 1. ... arsec

	PARSEC batch	Nutch	
Platform	running time	99%th tail latency	throughput
Units	*seconds*	*ms*	*req/s*
Linux-2.6.32	1058.7	379.3	217.1
Linux-2.6.35M	1136.5	371.4	217.9
Linux-3.17.4	1054.4	410.8	219.1
LXC	1716.8	284.1	214.0
Xen	4731.0	305.4	209.4
RainForest	1520.0	230.2	214.4

Fig 3. Performance of Search and Parsec when co-located

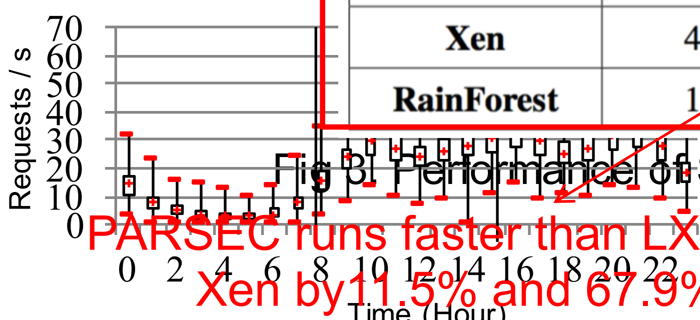


Fig 2. load v.s. time

PARSEC runs faster than LXC and Xen by 11.5% and 67.9%

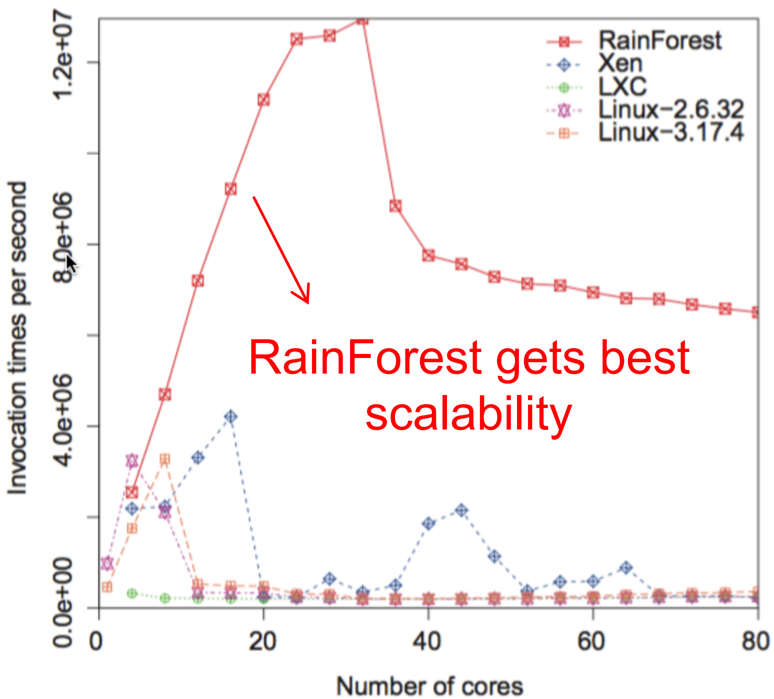
Smallest tail latency on RainForest

Similar throughput

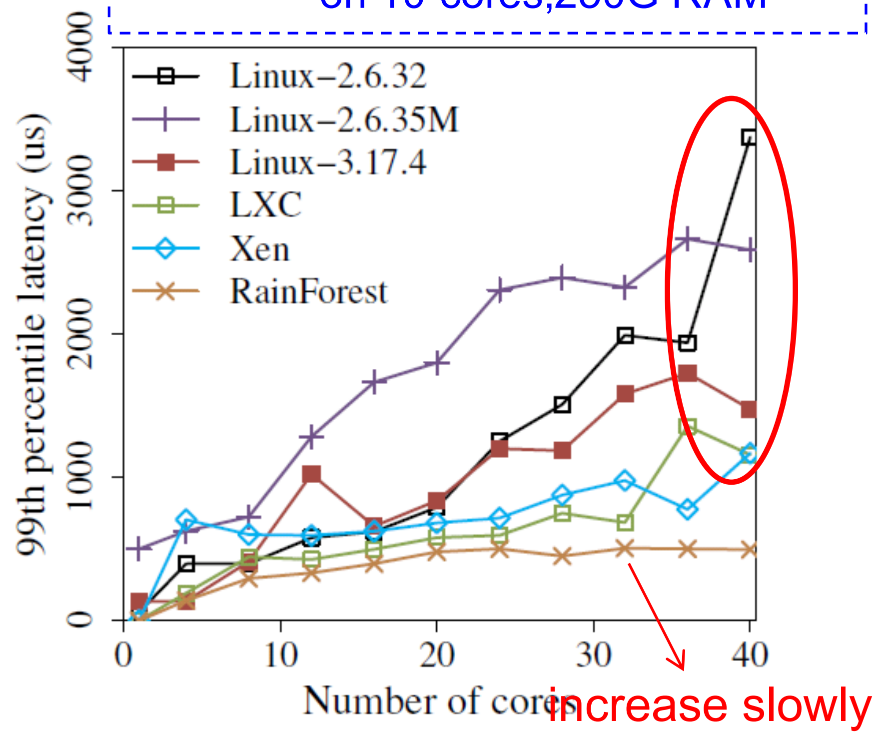
Evaluation: scalability

Workload: Wiil-It-Scale; memcached
Server: **40 cores**，**1T RAM**
OS instances: **4**
Each OS instance runs on 10 cores,250G RAM

- **Throughput & tail latency**



Throughput of system call *mmap* with increasing cores (larger is better)



Tail latency of *memcached* with increasing cores (smaller is better)

Content

- Background & motivation
- Related work
- Design and implementation
  - A new OS model—horizontal OS model
  - A new OS abstraction—subOS
  - The first prototype—RainForest
- Evaluation
- Conclusion

Conclusion

- Low resource utilization and emerging hardware calls for the structure change of the OS
  - The performance interference & scalability problems
- We propose a new horizontal OS model with a new OS abstraction
  - First isolation, later sharing
  - Three principles
- We build the prototype based on Linux and evaluation results show it surpasses Linux, Linux Containers, and Xen in performance isolation and scalability.
  - Source code will soon be available

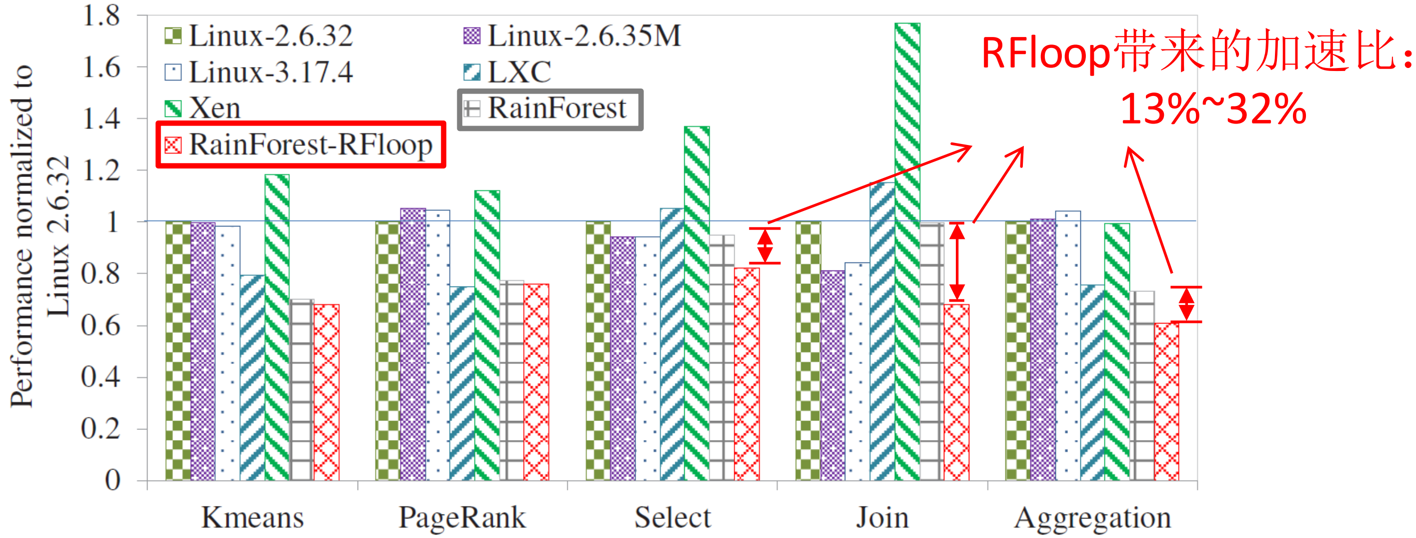
Any questions?

Backup

In-memory computing

测试程序：Spark典型负载
服务器资源：**40 核，1T RAM**
系统实例数：**4个**
每实例资源：**10核、250G RAM**

- Performance of Spark



Spark框架上各种典型负载的性能（越小越好）

RainForest相对于Linux、LXC 和Xen 的最大加速比分别达到1.64x、1.69x 和1.74x