IBM Watson Research Center Report

Research on Scalability and Performance of Operating Systems on Multicore Architectures

YAN CUI

Postdoctoral researcher at Columbia University

Outline

Scalability (Ph.D. Thesis)

❶ Background

❷ Operating System Scalability Research

- System Interface Scalability Analysis
- Simulation and Avoidance of Scalability Collapse
- Hardware Resource Contention Avoidance
- Scalability Bottlenecks Localization Method

❸ Summary

❹ Fast Auto-tuning Operating System Project

Performance (Postdoc at Columbia)

Outline

❶ Background

❷ Operating System Scalability Research

- System Interface Scalability Analysis
- Simulation and Avoidance of Scalability Collapse
- Hardware Resource Contention Avoidance
- Scalability Bottlenecks Localization Method

❸ Summary

❹ Fast Auto-tuning Operating System Project

Multicore Challenge

memory latency

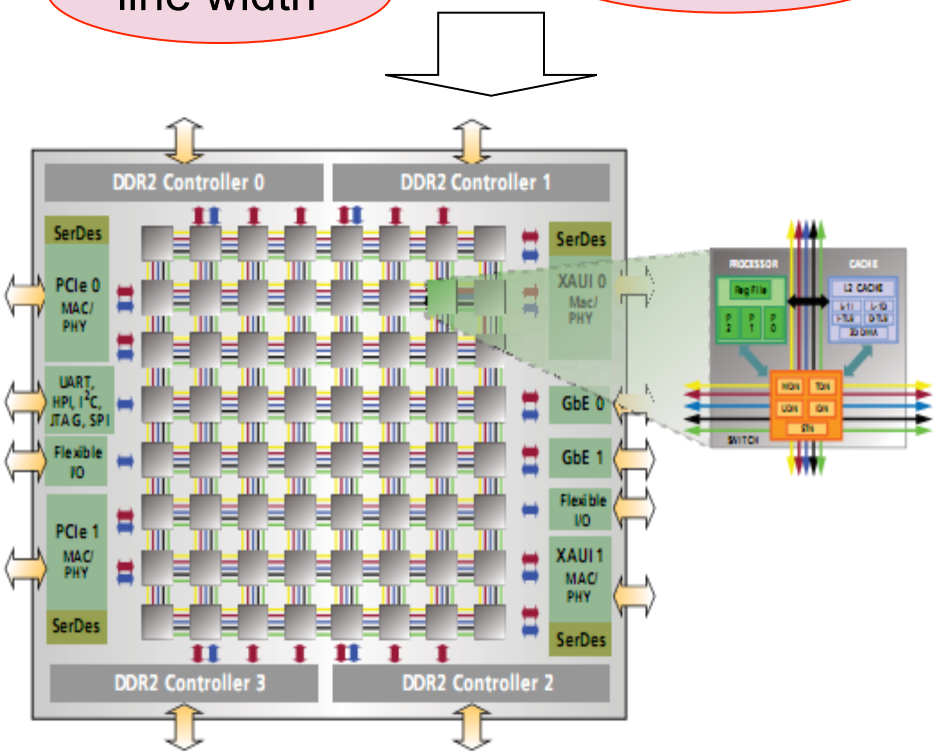
power

complexity

multicores

line width

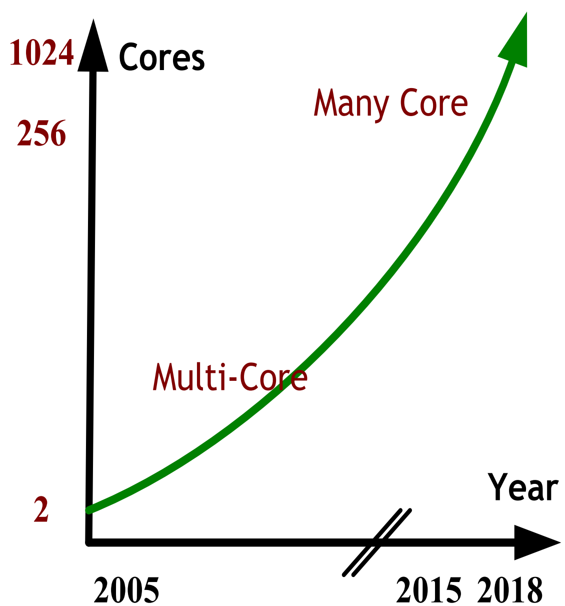
transistors

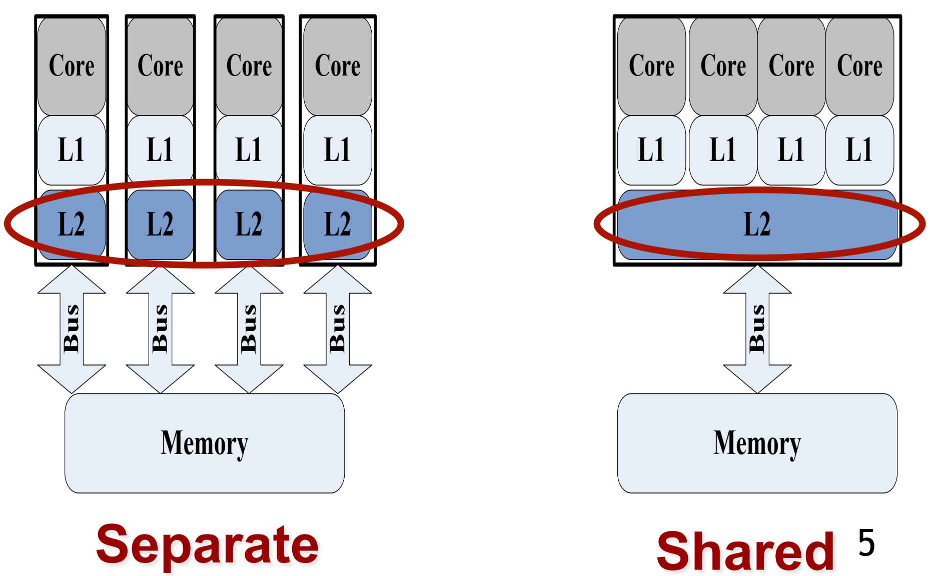


- Industry
  - Widely adopted
  - IBM/Intel/AMD/Sun/…
- Environments
  - Ubiquitous multicore
  - Server/PC/desktop/ embedded systems/…

Multicore Challenge

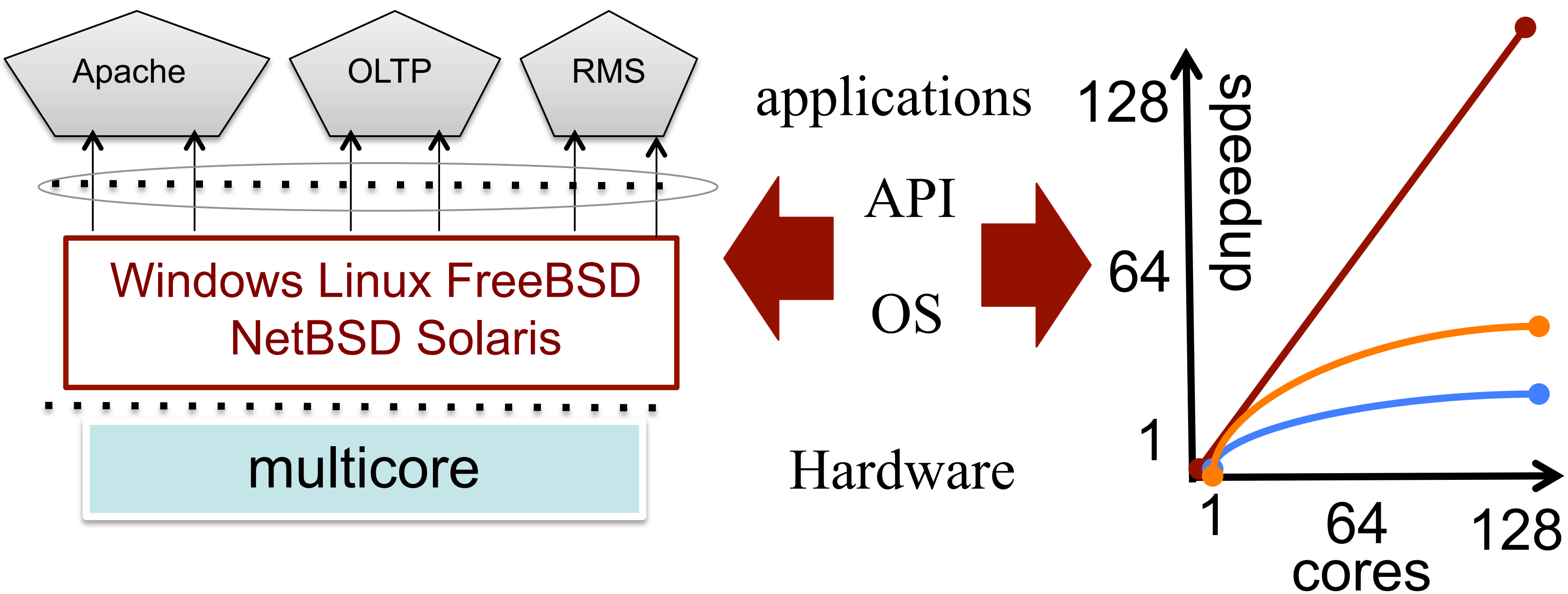
- Multicore v.s. SMP
  - Number of Cores can Become Larger
    - SMP: Low-end(2CPU), Middle(4~8CPU), High-end(>16CPU)
    - CMP: 4~8 cores multicore systems, **1000+ cores (<10year)**
      E.g. Intel's 80 cores chip & Tilera's 100 core chip
  - Hardware Resource is Shared(e.g., LLC)





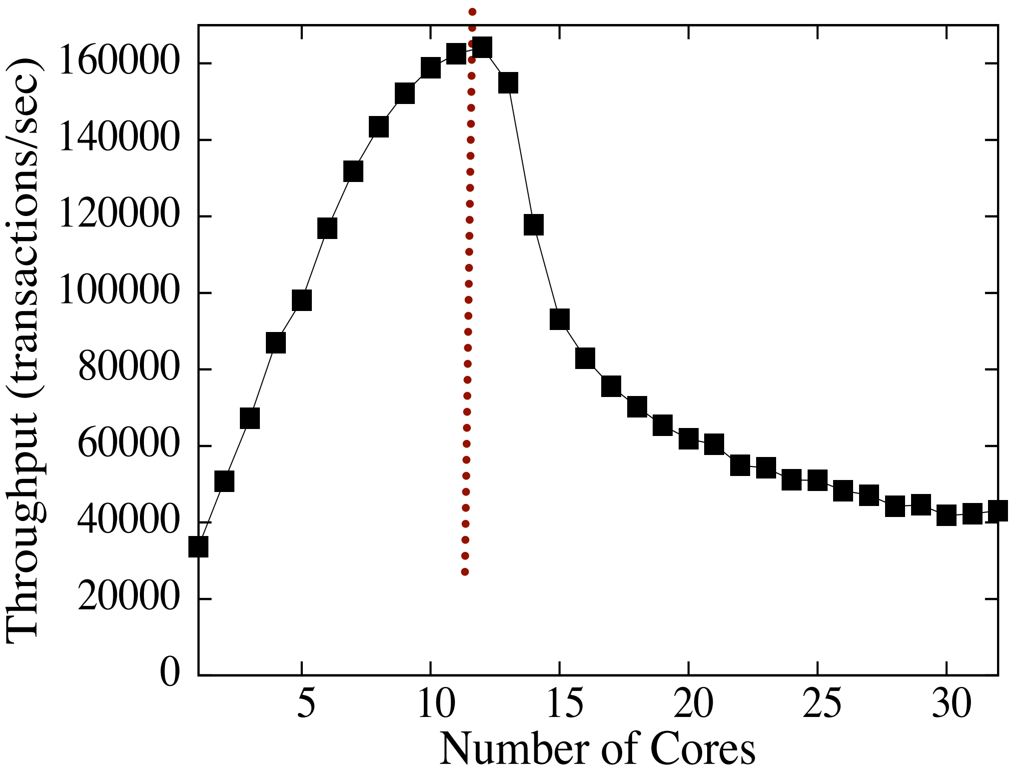
Multicore Challenge

- Scalability of Operating Systems
  - Whether the performance can increase with #cores
    - Speedup model with fixed time (Gustafson S=N.(1-P)+P)
  - Focus on the operating system layer
    - Designed for small scale SMP   ignore multicore characteristics (large number of cores, resource sharing)



Scalability Bottleneck

- Spinlock contention in kernel



OS: Linux

Platform: AMD 32 cores

Benchmark: File Server

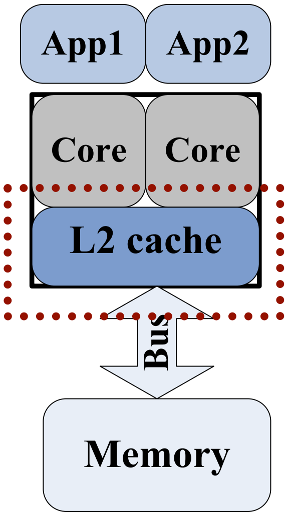
File System: tmpfs

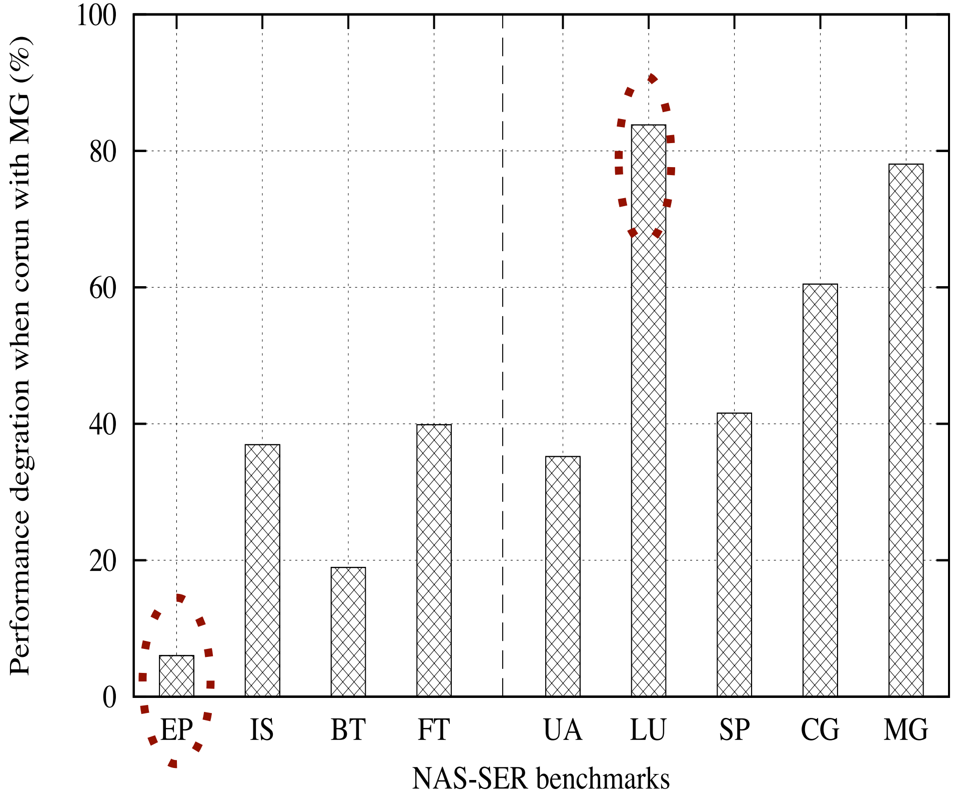
Bottlenecks: spinlock contention for file descriptor table and statistics in memory file system

Throughput decreases because of kernel lock contention

Scalability Bottleneck

- Shared Hardware Resource Contention





OS: Linux

Platform: Intel 8core

Benchmark: NAS

Bottleneck: Hardware resource contention

Hardware resource contention (e.g., LLC) increases execution time 6%~84%

Scalability Bottleneck

- Sharing is the root cause of operating system scalability problems [OSDI'08]
  - Reflection of sharing

    sharing of kernel data ➔ lock contention

    sharing of hardware cache ➔ cache contention

    sharing of address bus ➔bus contention
  - The pros and cons of sharing

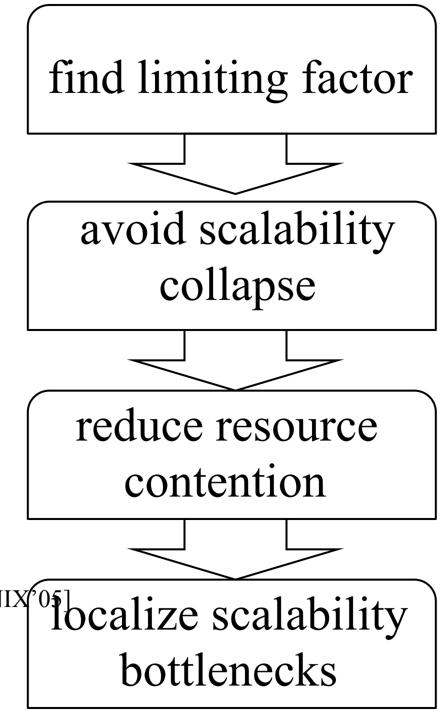
    Pros: communication fast(low contention)

    Cons: introduce extra overhead, significantly degrades scalability (high contention)

[1].Silas Boyd-Wickizer, et al. Corey: An Operating System for Many Cores. In Proceedings of OSDI 2008.

Research Contents

- How to analyze scalability bottlenecks
  - Acquire the speedup limiting factors of system interfaces by kernel code analysis
- How to avoid scalability collapse caused by lock [OSDI'10]
  - Propose a discrete event based lock simulator (LockSim)
  - Propose a requester-based scalable lock protocol
  - Propose a lock-contention-aware scheduler
- How to reduce hardware resource contention [ASPLOS'10, USENIX'05]
  - Propose a resource contention aware scheduler
- How to localize scalability bottlenecks [EuroSys'10]
  - Propose a scalability value based bottleneck detection methods



[1].Slias Boyd-Wicizer, et al, "An Analysis of Linux Scalability to Many Cores". In OSDI 2010.

[2].Sergey Zhuravlev, et al, "Addressing Shared Resource Contention in Multicore Processors via Scheduling", in ASPLOS 2010.

[3].Alexandra Fedorova, et al, "Performance of Multithreaded Chip Multiprocessors for Operating System Design", in USENIX ATC 2005.

[4].Aleksey Pesterev, "Locating Cache Performance Bottlenecks Using Data Profiling", in EuroSys 2010.

Outline

❶ Background

❷ Operating System Scalability Research

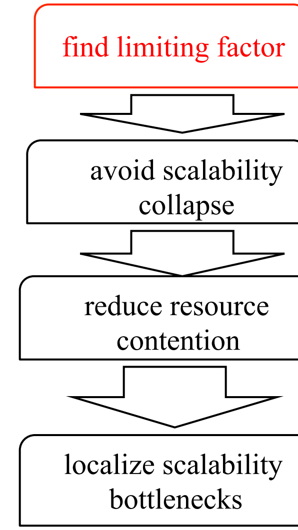
- System Interface Scalability Analysis
- Simulation and Avoidance of Scalability Collapse
- Hardware Resource Contention Avoidance
- Scalability Bottlenecks Localization Method

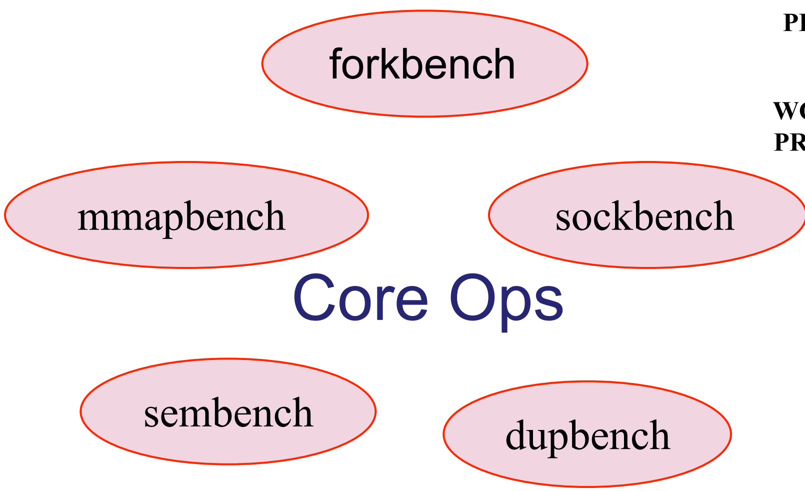
❸ Summary

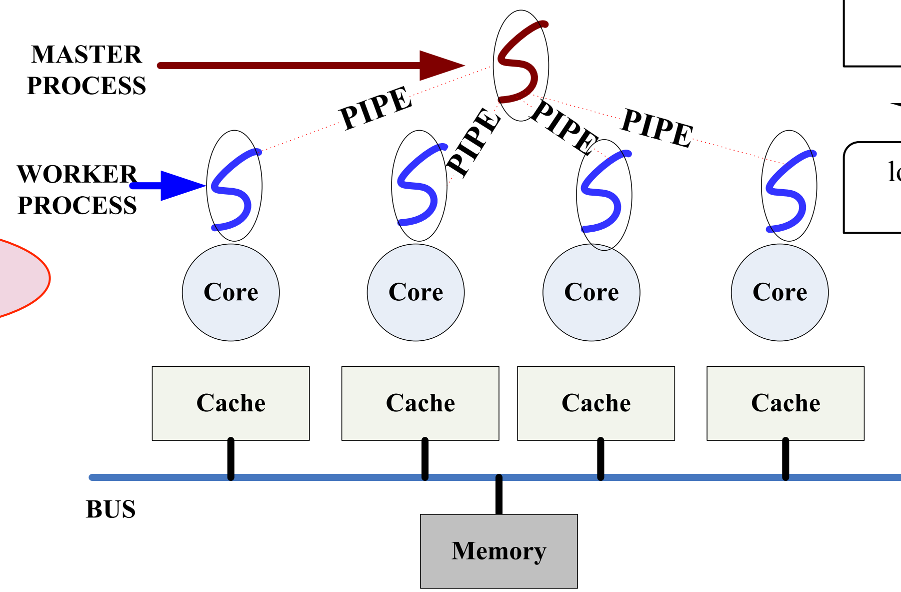
❹ Fast Auto-tuning Operating System Project

Analyze System Service Interface

- Design micro-benchmark suite
  - 5 important and core interfaces managed by a unified framework

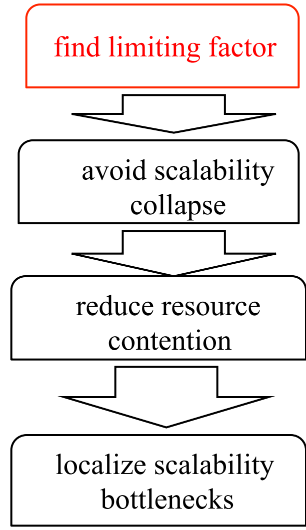




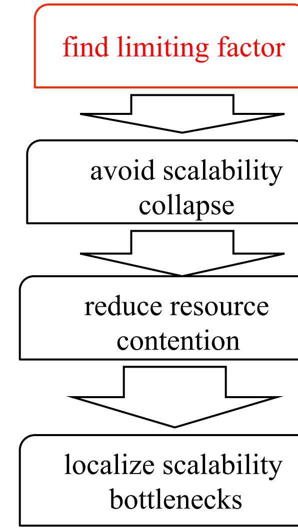


Analyze System Interface: Method

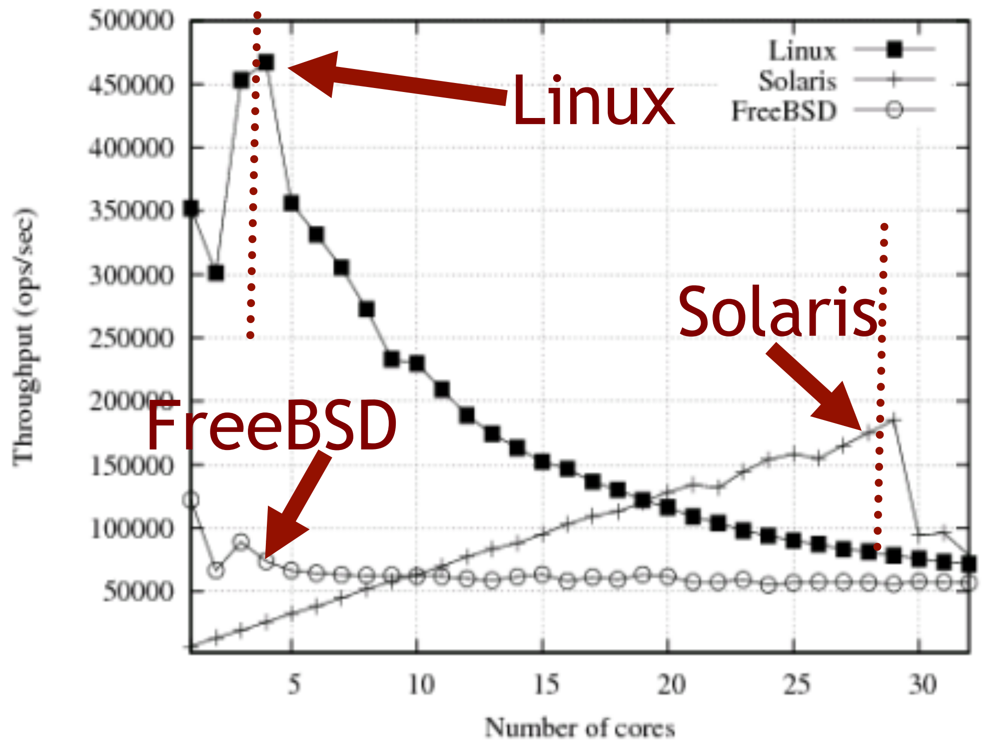
- Operating Systems (POSIX compatible)
  - Linux 2.6.26.8
  - OpenSolaris 2008.11
  - FreeBSD 8.0-CURRENT
- Hardware Platform
  - AMD NUMA 8*4 = 32
- Profiling tools (function execution time, lock usage)
  - Linux: Oprofile /proc/lock_stat
  - Solaris: Dtrace lockstat
  - FreeBSD: lock profiling
- Binding Interface (avoid effect of scheduling)
  - Linux sched_setaffinity()
  - Solaris pset_bind()
  - FreeBSD cpuset_setaffinity()



Design Benchmark Test: mmapbench

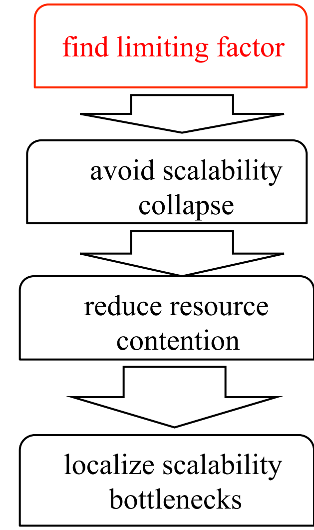


- mmapbench
  - Every process repeatedly do mmap() in 500M bytes of the same file, touches each page and releases the mapping

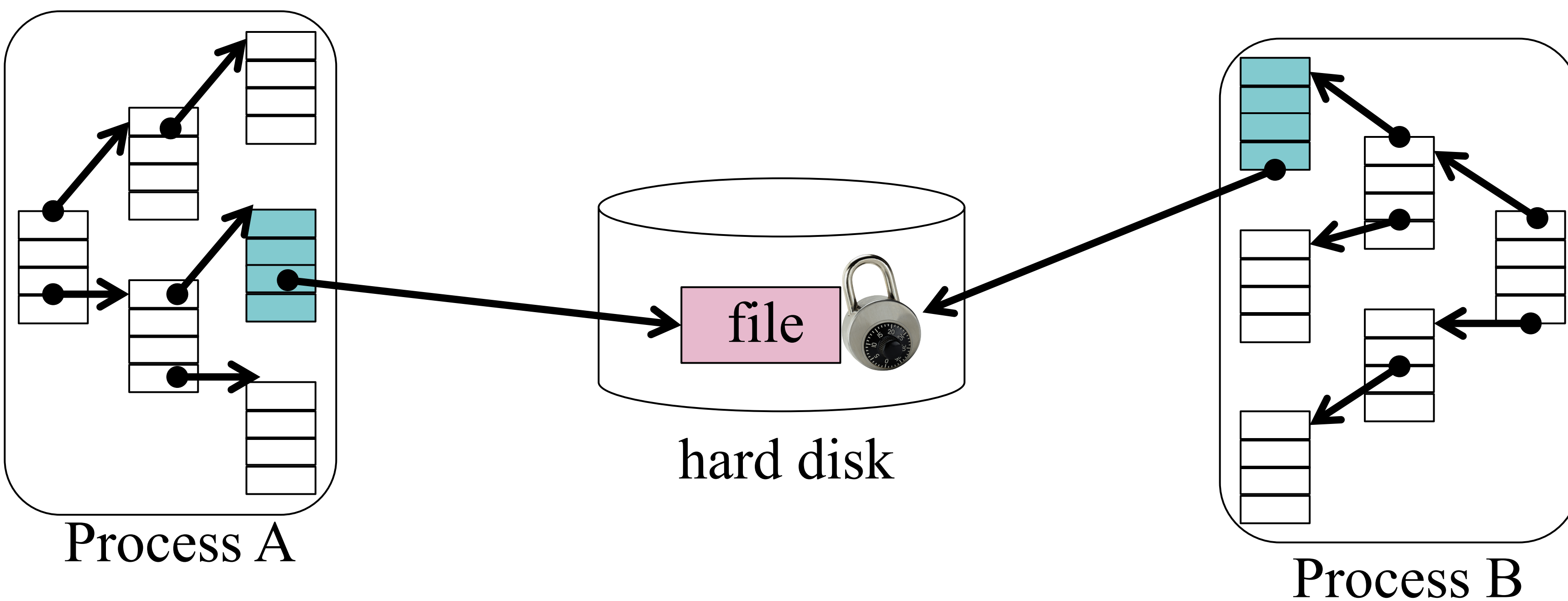


All Operating System have Scalability Problems

Design Benchmark Test: mmapbench

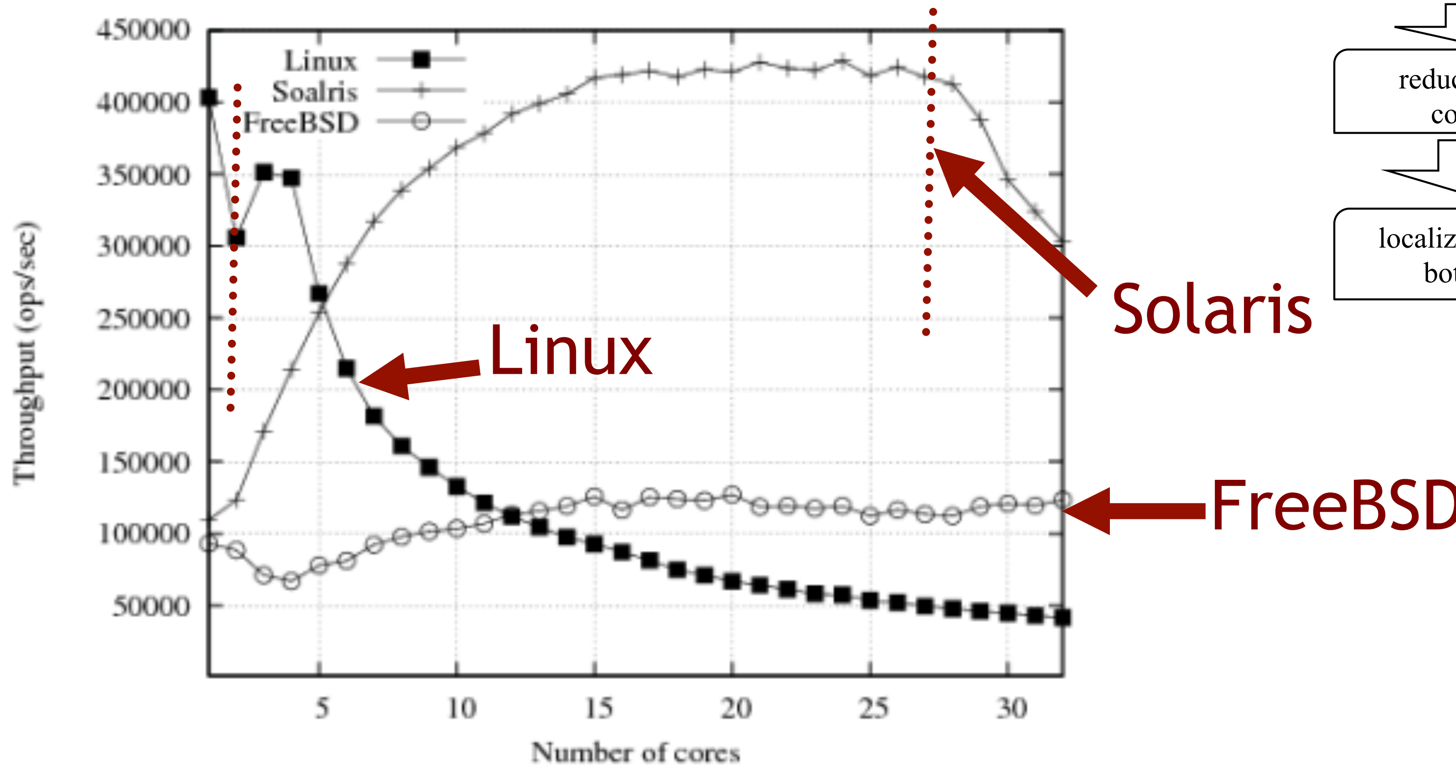


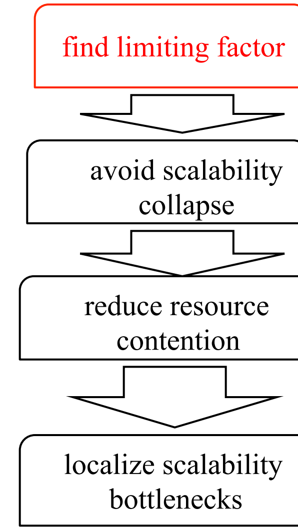
- Bottleneck
  - Lock protecting the mapping file
  - Different timing to acquire locks
    - Linux: mapping
    - FreeBSD: update statistics on vnode
    - Solaris: set mapping policy to vnode of mapped file



Design Benchmark Test: sockbench

- sockbench
  - Each process repeatedly calls socket() and close()

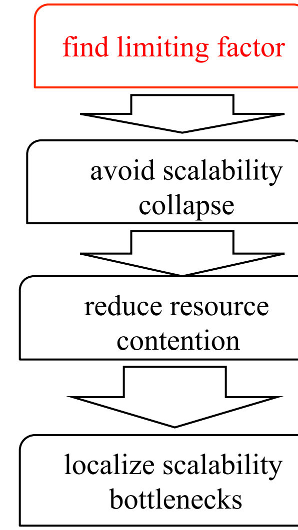


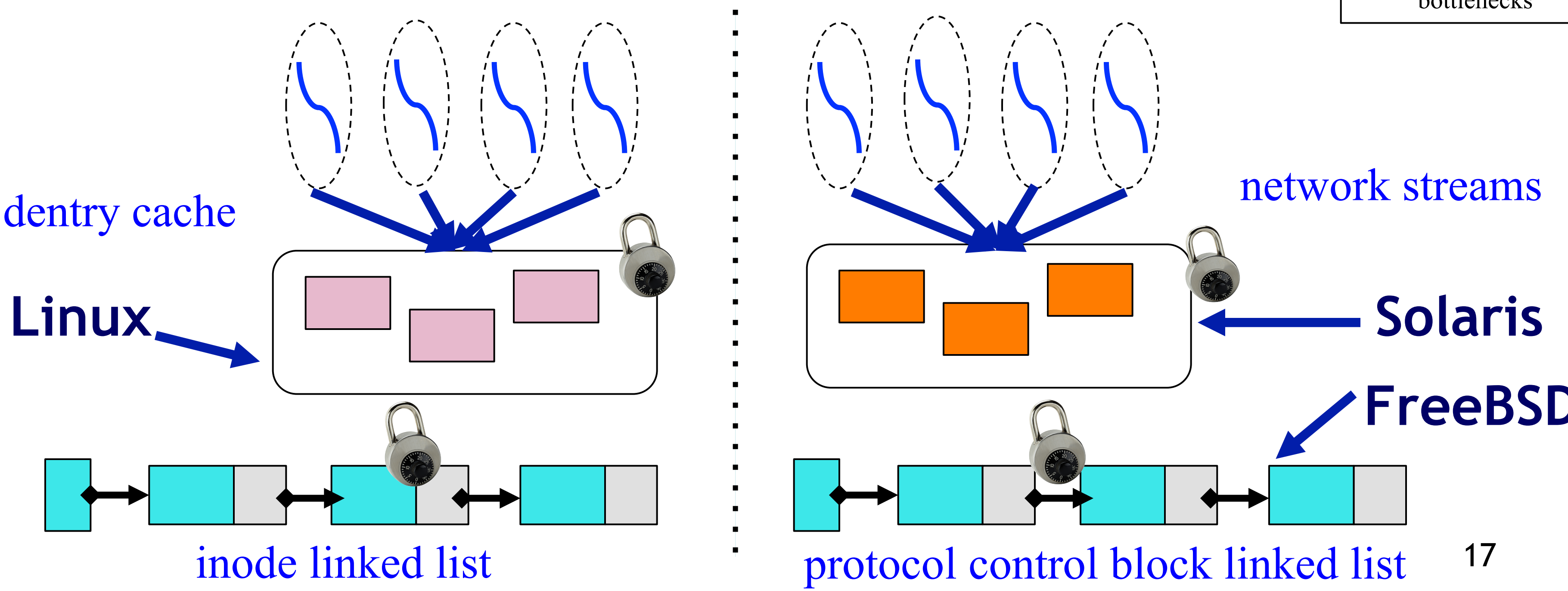


All systems have scalability Problems

Design Benchmark Test: sockbench

- Bottleneck
  - dentry cache lock and inode linked list lock (linux)
  - network stack contention
    - Solaris reference count update for same type of protocol
    - FreeBSD protocol control block linked list maintaining





Limiting Factors Summary

	Linux	Solaris	FreeBSD
forkbench	Create and delete VMA contends on lock protecting the file	Page faults contends on read-write lock protecting the mapping file	Page faults contends on mutex lock protecting the mapping file
mmapbench	Create and delete VMA contends on lock protecting the file	Set memory policy contends on read-write lock protecting the file	Update vnode info contends on mutex lock protecting the file
dupbench	Perfect scalability	Closing file descriptor contends on adaptive lock of hash table	Witness overhead increases with the number of cores
sembench	Read lock contention of the global semaphore	Perfect scalability	Perfect scalability
sockbench	Spinlock contention of global dentry cache and inode linked list	Create and delete network streams contends on reference count of stack	read- write lock contention of protocol control block

find limiting factor
↓
avoid scalability collapse
↓
reduce resource contention
↓
localize scalability bottlenecks

1. Lock contention in operating system is an important scalability limiting factor
2. The contention degree of the lock can be so serious that the speedup can decrease with the number of cores (scalability collapse)

Summary

- Contribution
  - Understand the scalability limiting factors of system service interfaces by micro-benchmark analysis
- Publications
  - IEEE CLUSTER, IEICE Transactions
    - **Yan Cui** , Yu Chen, Yuanchun Shi, "Experience on Comparison of Operating System Scalability on the Multicore Architecture" , in **CLUSTER** 2011.
    - **Yan Cui**, Yu Chen, Yuanchun Shi, "Comparing Operating System Scalability by Microbenchmarking" , in IEICE Transactions on Information and Systems (**IEICE Transactions**)

Outline

❶ Background

❷ Operating System Scalability Research

- System Interface Scalability Analysis
- Simulation and Avoidance of Scalability Collapse
- Hardware Resource Contention Avoidance
- Scalability Bottlenecks Localization Method

❸ Summary

❹ Fast Auto-tuning Operating System Project

Simulation of Scalability Collapse

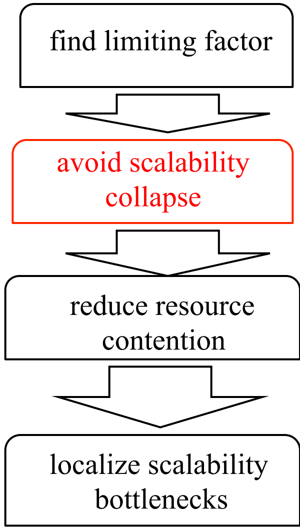
- How to model and simulate heavy spinlock contention (scalability collapse)
- Related Work
  - Mean value analysis model lock and cache misses[OSR'78, TON'98]
  - Exploit the characteristics of stable behavior in spinlock contention to simulate[SIGCOMM'93]
  - Approximate Mean value analysis to model cache miss[TR'98]
  - probability model [ISCA'10]

  Ignore the characteristics of ticket spinlock

  cannot reproduce scalability collapse

  - Markov chain based model[OLS'12]

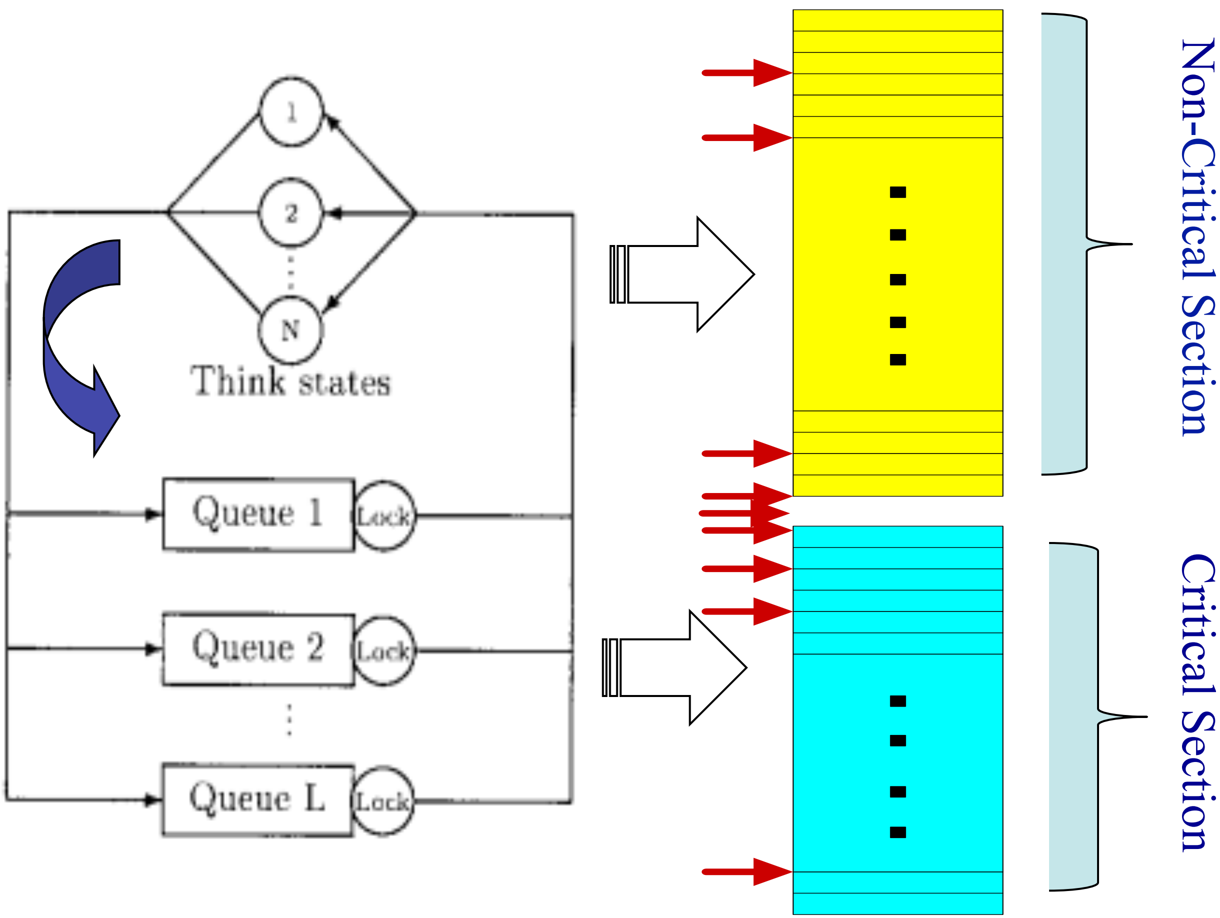
  Complex   Ignore cache misses in critical and non-critical sections

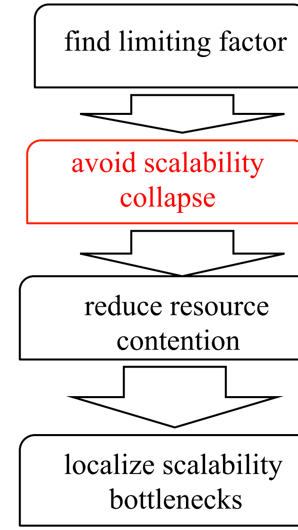


[1].D.Gillbert. "Modeling Spin Locks with Queuing Networks". In OSR 1978.
[2].M.Bjorkman and P.Gunningberg, "Performance Modeling of Multiprocessor Implementation of Protocols", in TON 1998.
[3].M.Bjorkman and P.Gunningberg, "Locking Effects in Multiprocessor Implementation of Protocols", in SIGCOMM 1993
[4].D.L.Eager, D.J.Sorin and M.K.Vernon, "Analysis Modeling of Burstiness and Synchronization Using Approximate MVA", in TR 1998
[5].Stijn Eyerman and Lieven Eeckhout, "Modeling Critical Sections in Amdal's Law and its implications for Multicore Design", In ISCA 2010
[6].Slias Boyd-Wickizer, et.al, "Non-Scalable Locks are Dangeous", in OLS 2012

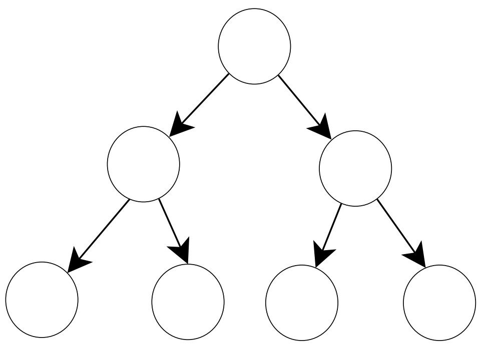
The Model

- LockSim: Discrete event simulation based simulator
    - Based on queuing theory model

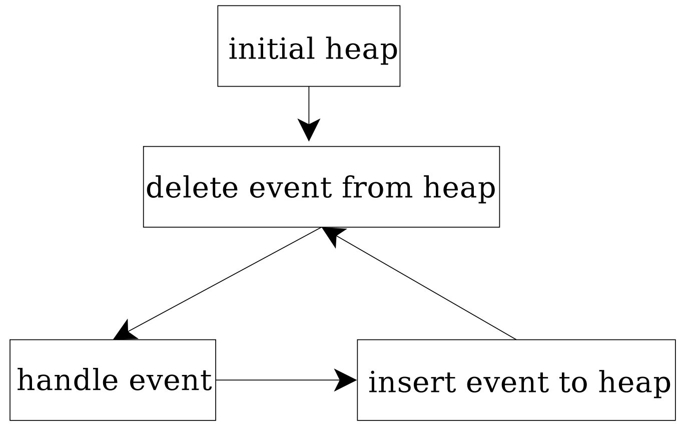


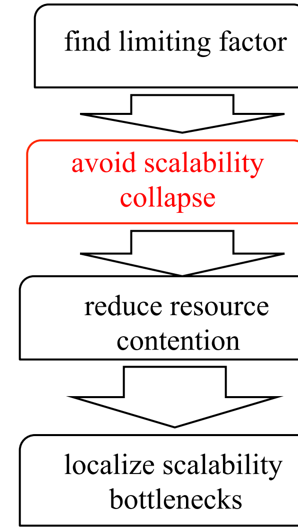


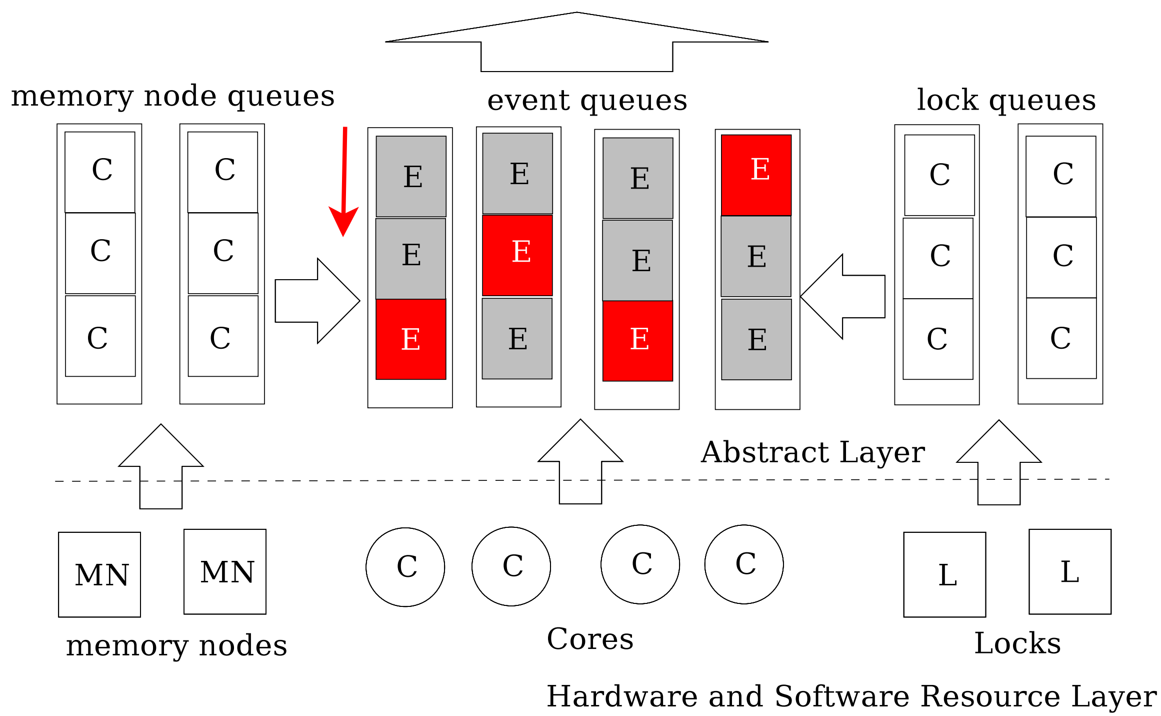
Data Structures and Operations



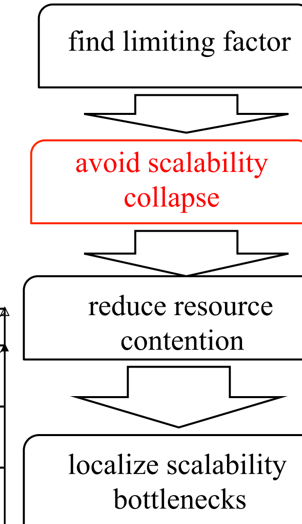
Heap based on event time stamp



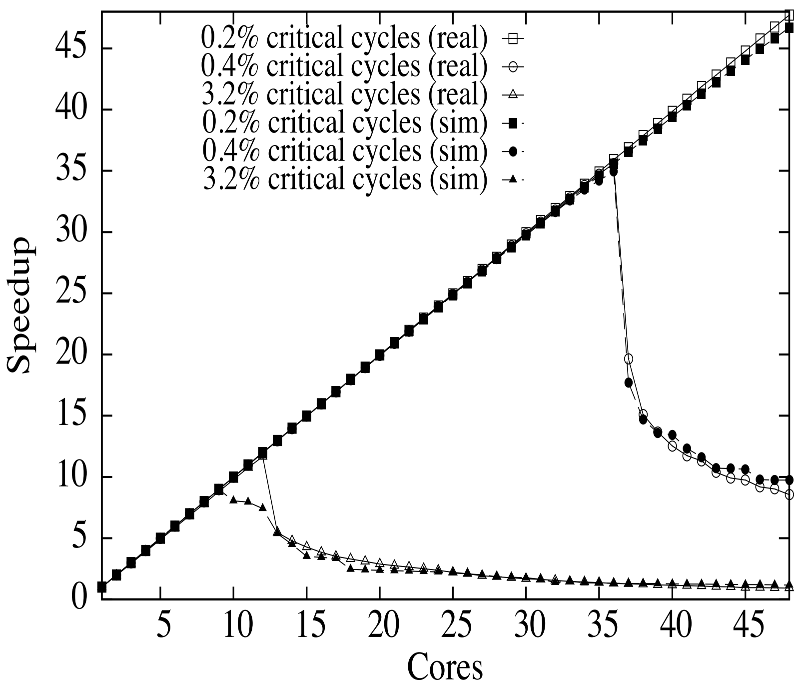


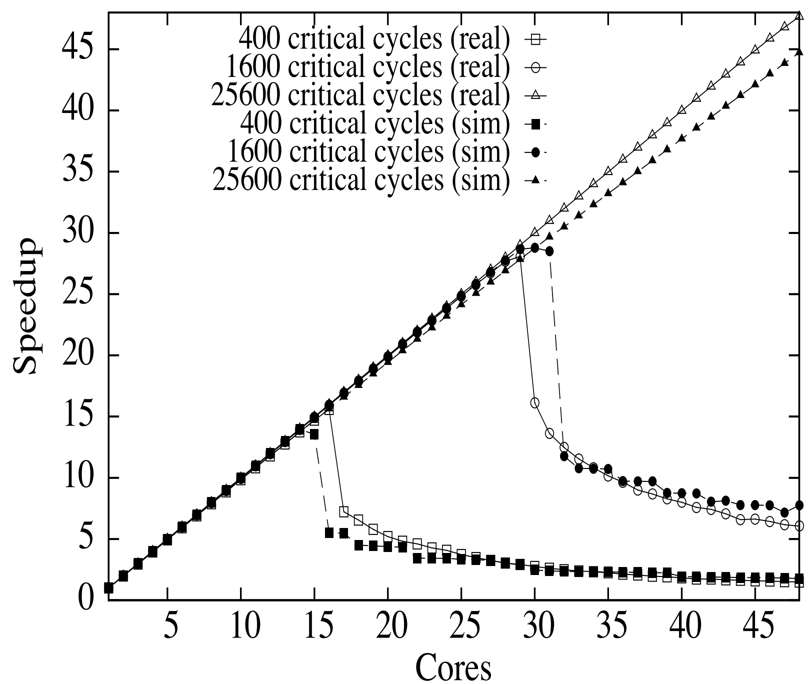


Experiments and Evaluation



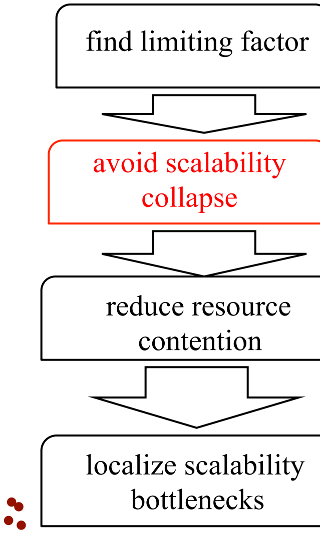
- Scalability Collapse Reproduction
  - Match closely with real world measurements

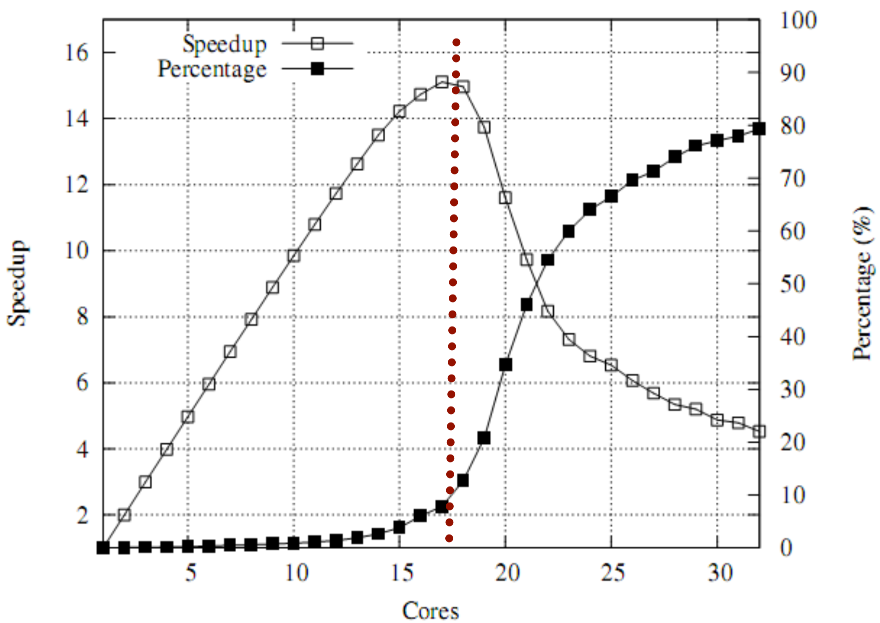


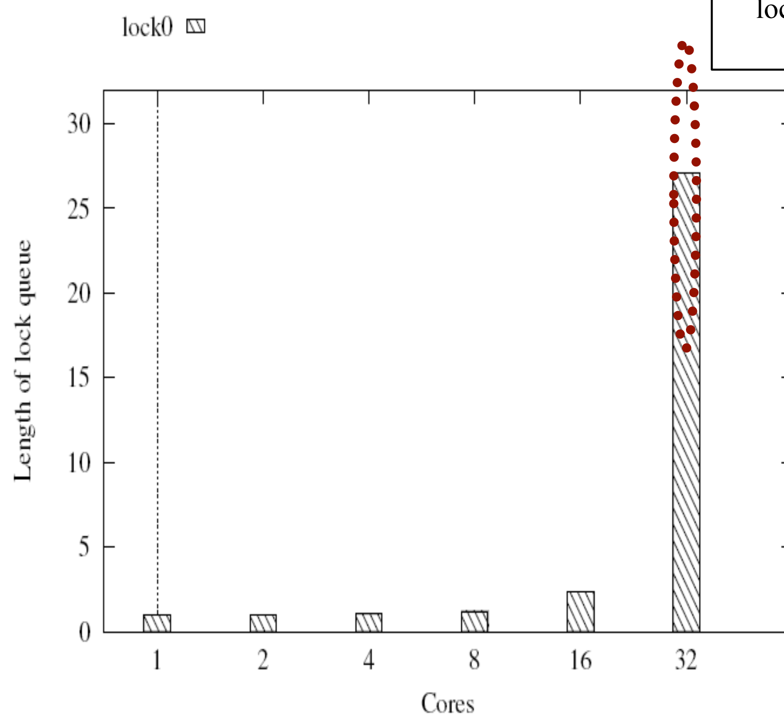


Experiments and Evaluation

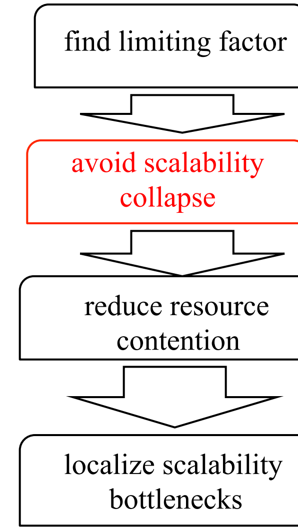
- Strong relativity between scalability collapse and average lock waiting time percentage and the number of cores waiting for a lock







Scalability Collapse Avoidance



- Related Work
  - Fine-grained lock: split critical sections

    Time consuming hard error prone[OSR'09]
  - Mutex lock: waiting locks by sleeping

    Large context switch overhead (10000 cycles+)
  - Adaptive lock: waiting by adaptively sleeping and spinning

    Decide when to sleep or spin by heuristic rules

    hard to achieve full performance potential[LKML.org'09]
  - MCS lock[TOCS'91]: lock requesters wait on local variable

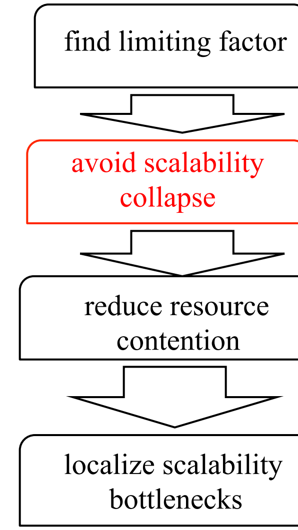
    Repeated spinning on the local variable affect the scalability

[1].D.Wentzlaff and A.Agarwal, "Factored Operating Systems (fos): the case for a scalable operating system for multicores", in OSR 2009.

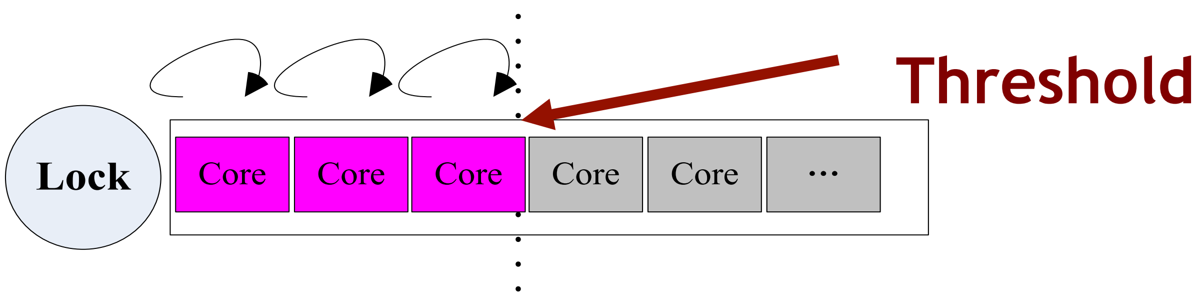
[2]." Adaptive Spinning Mutexes", http://lkml.org/lkml/2009/1/14/393.

[3].J.Mellor-Crummey and M.L.Scott, "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors", in TOCS 1991

Requester Based Scalable Lock

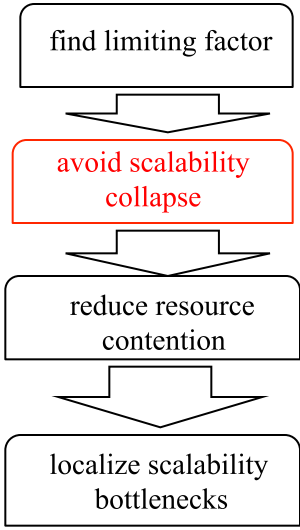


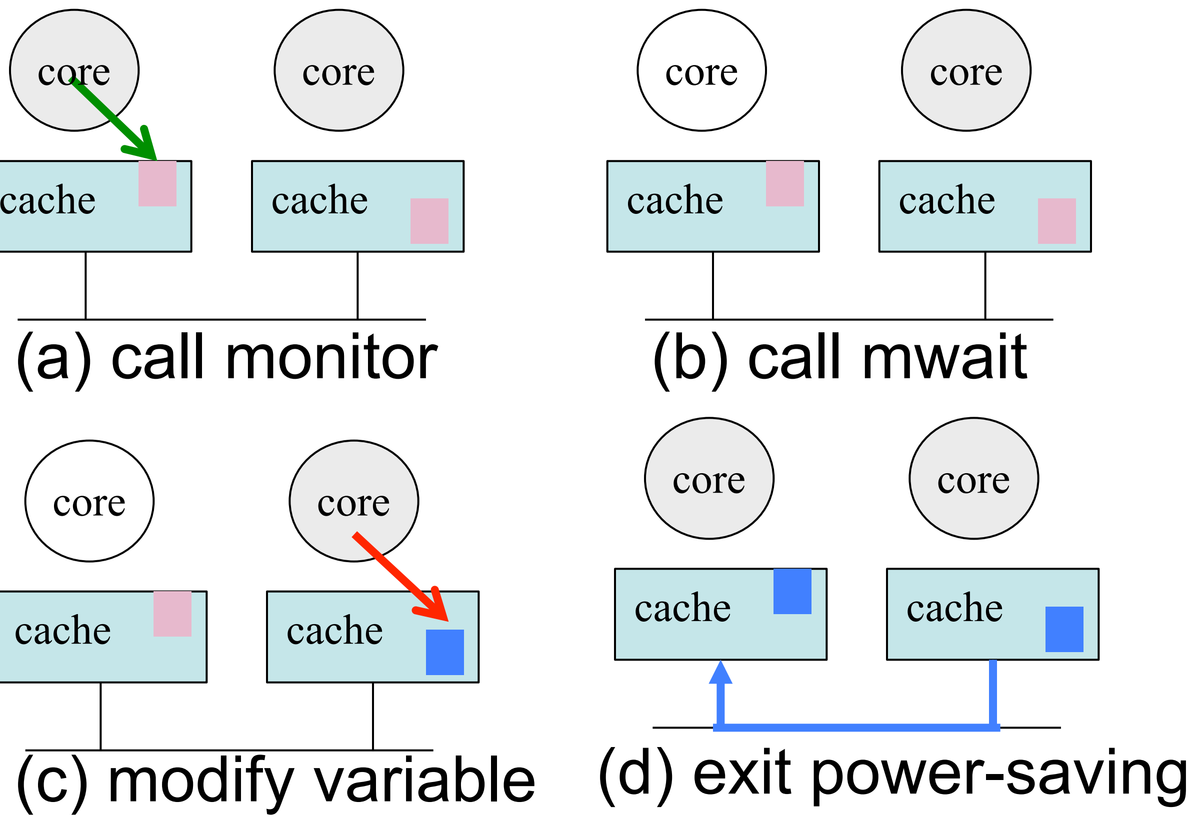
- Insights: the number of requesters for a lock increases significantly when scalability collapses
  - observed in LockSim verified in real systems
- Basic Method
  - Monitor the number of lock requesters. If larger than the threshold, enter power-saving state (monitor/mwait), or else wait by spinning
  - Check whether to exit power-saving state by looking at local variables
  - scalable, energy efficient and can be used with any type of spin locks



Scalability Optimizations

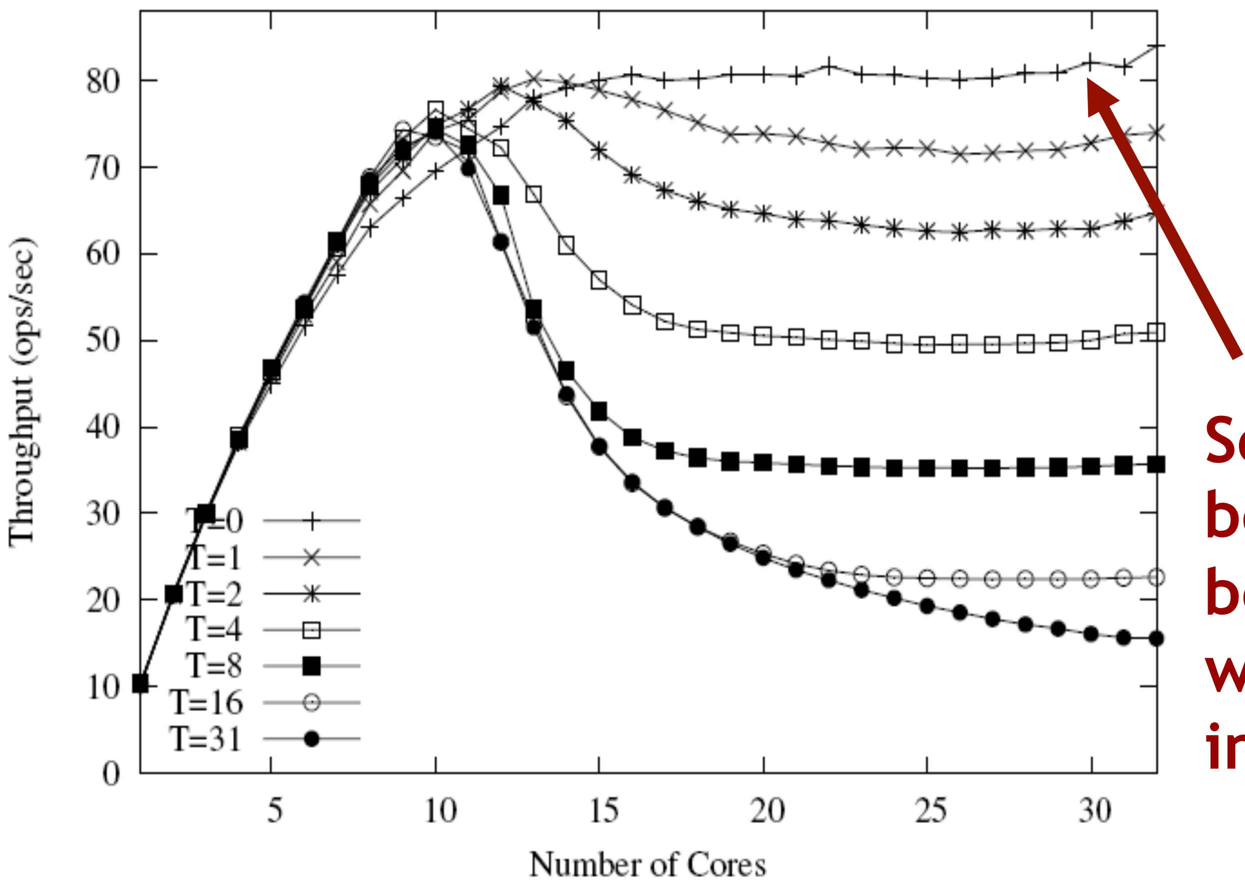
1. light-weight lock waiting mechanism (monitor/mwait)
2. per-CPU waiting queue
3. Fast prediction of the number of lock requesters
4. light-weight mechanism of waiting lock requesters
5. False sharing avoidance

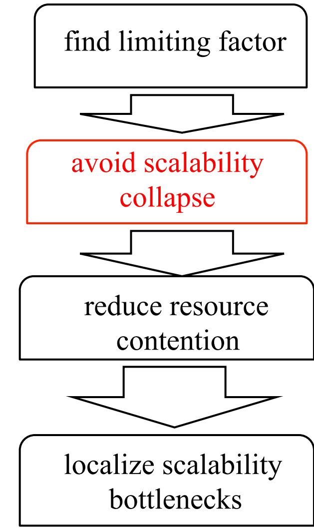




Threshold Selection

- threshold 0->1->2->4->8->16->31



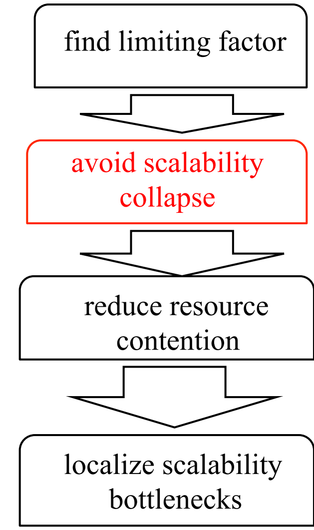


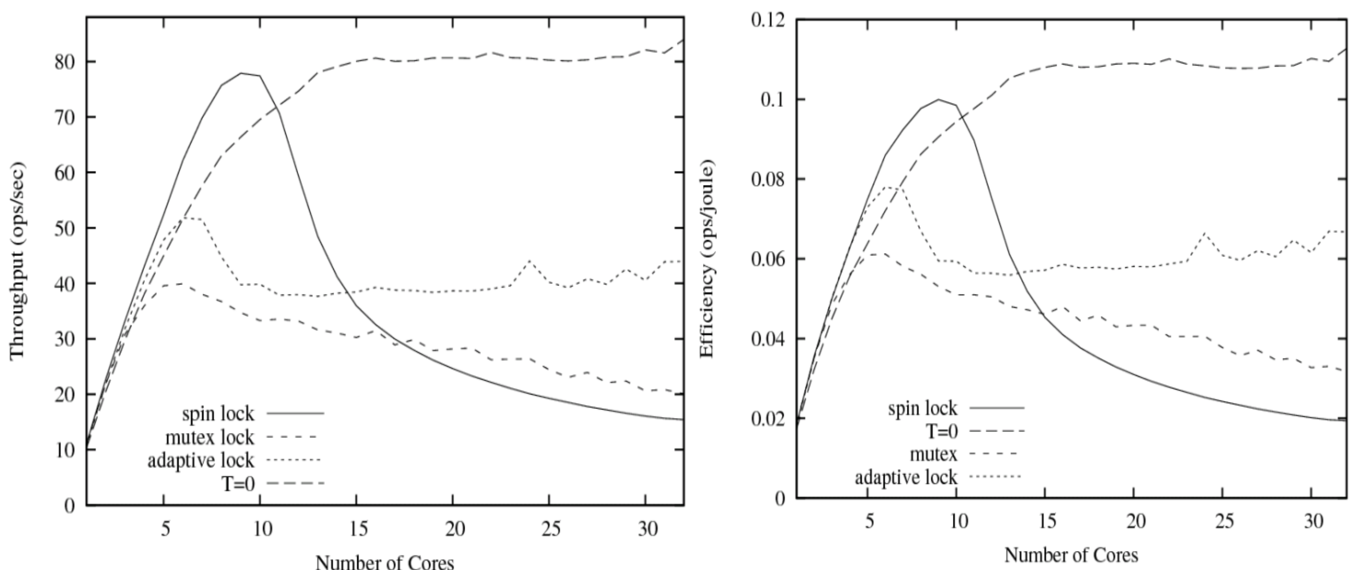
Setting threshold to be 0 achieves the best scalability (light weight power-saving instruction)

parallel find

Experiments and Evaluation

- kernel: 2.6.29.4 2.6.32
- platforms: AMD NUMA 32、Intel NUMA 40
- benchmarks: mmapbench、sockbench、parallel find、kernbench、parallel postmark
- Power-meter: 380801





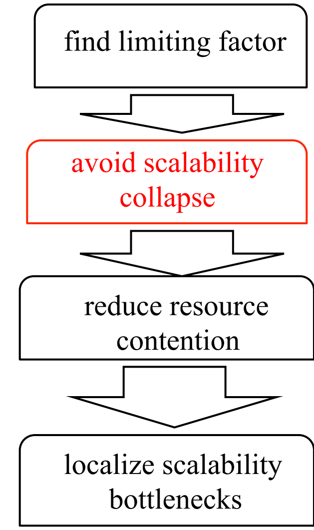
Compared with spin lock, mutex and adaptive lock

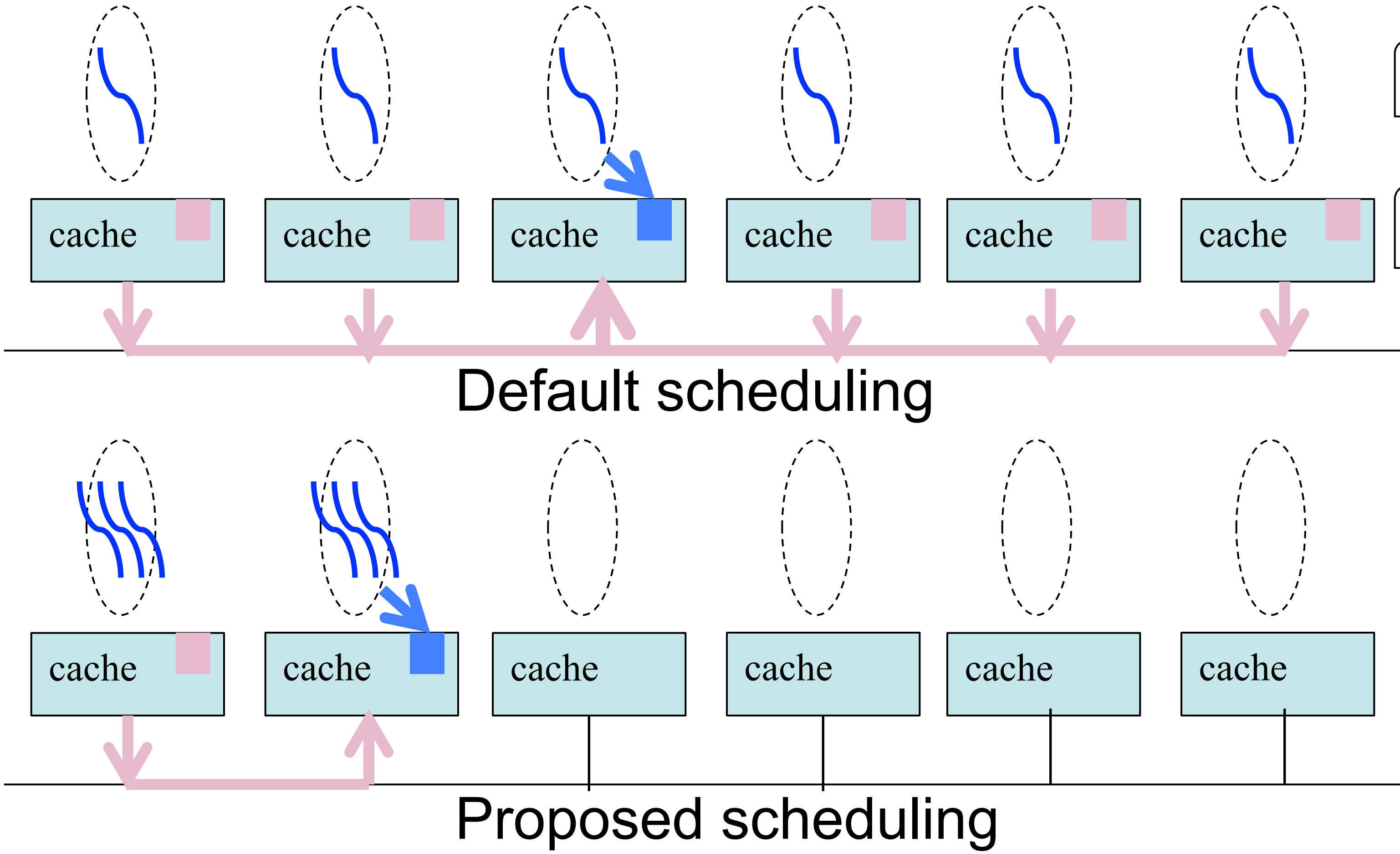
parallel find

Requester-based scalable lock performs better than other locks in scalability and energy efficiency

Lock Contention Aware Scheduler

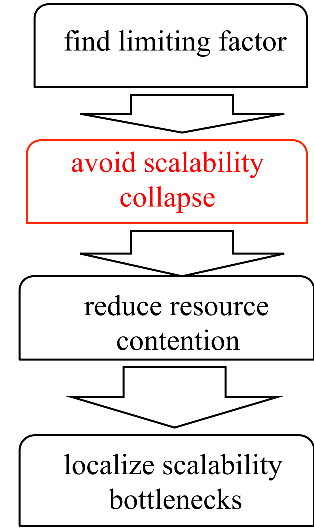
- Insight: the average lock waiting time percentage increases significantly when scalability collapses

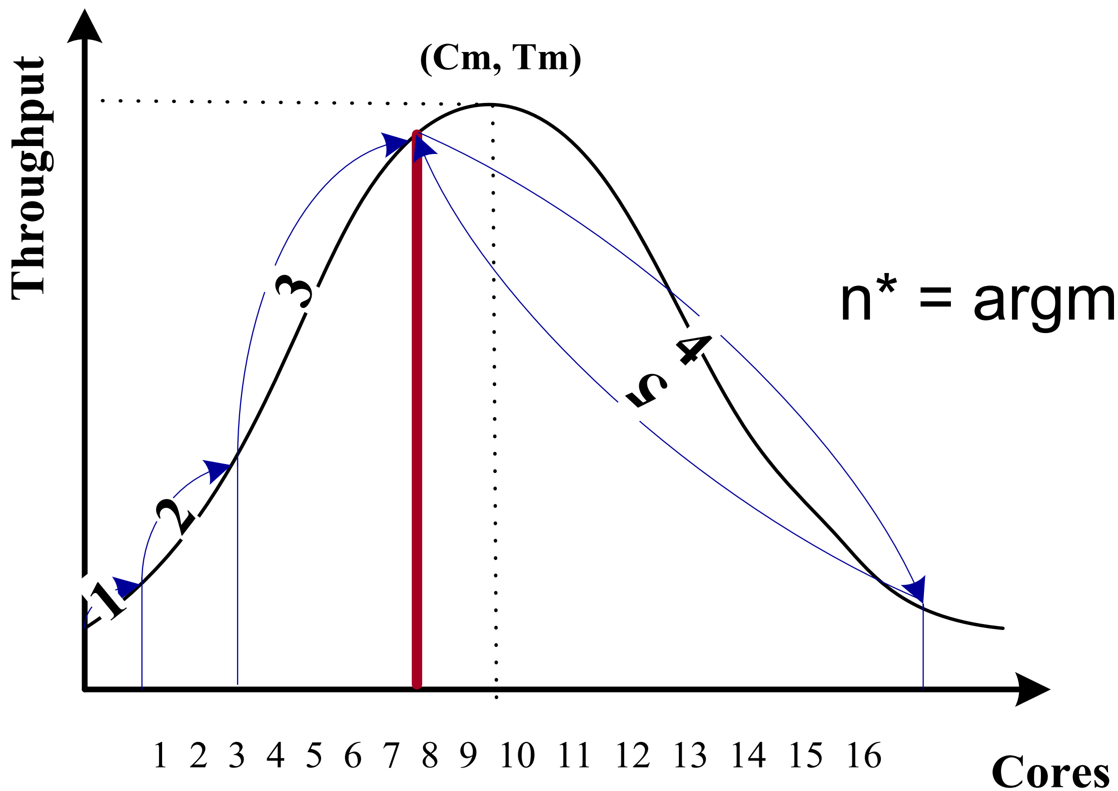




Key Issue

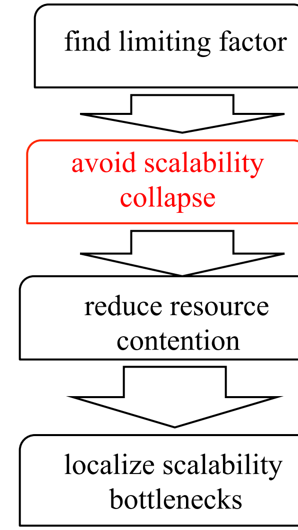
- How to decide the number of cores for lock intensive tasks
  - p(n) is acquired by using a lock wrapper
  - Avoid online measurement error by voting and migration state machine



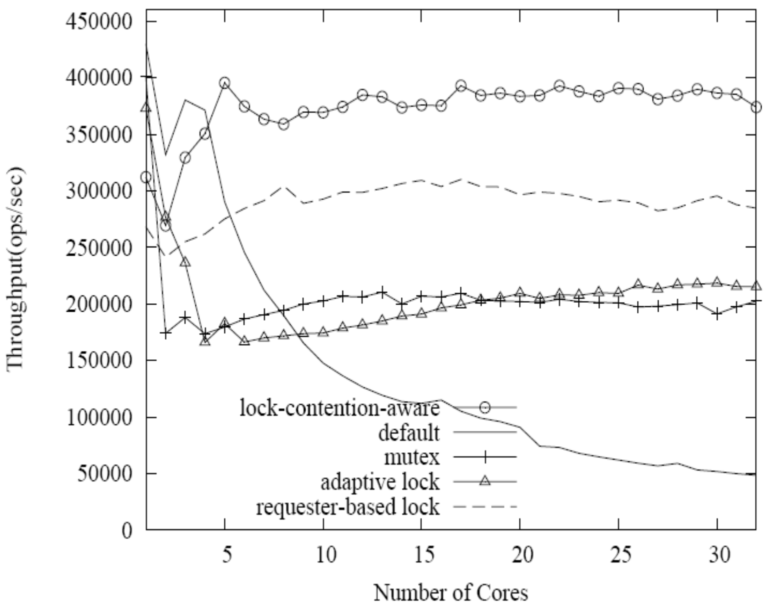


$$n^* = \text{argmax}\ \{n.(1-p(n))\}$$

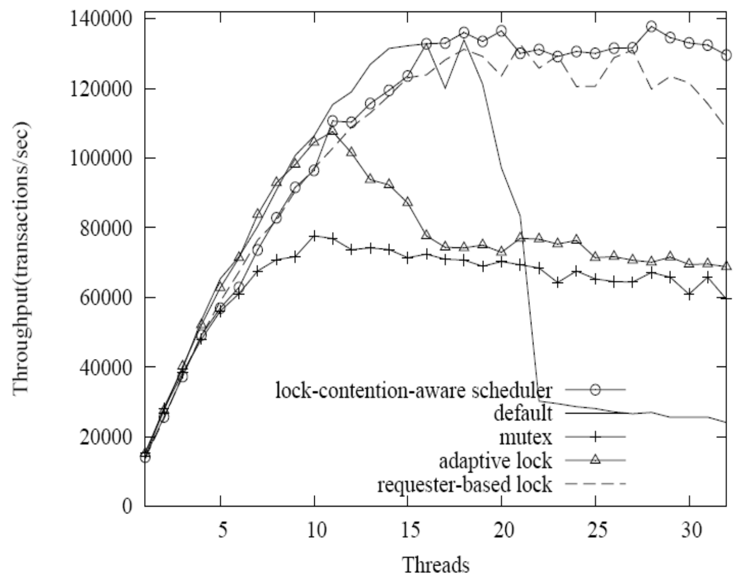
Experiments and Evaluation



Compared with requester-based lock, spinlock, mutex and adaptive lock



sockbench



parallel postmark

Lock-contention aware scheduler performs better than all the other locks. The results of energy efficiency are similar to the scalability results

Summary

- Contributions
  - Simulation and Avoidance of Scalability Collapse
    - Discrete Event Simulation based Simulator Accurate and Fast
    - Requester based scalable lock special instructions and local variable based spinning
    - Lock contention aware scheduler model based searching
- Publications and Patents
  - IEEE transactions on Computers, ACM Transactions on Architecture and Code Optimization, Concurrency and Computation: Practice and Experience, MASCOTS, ICPADS one Chinese patent
    - **Yan Cui**, Yingxin Wang, Yu Chen, Yuanchun Shi, "Lock Contention Aware Scheduler: A Scalable and Energy Efficient Method for Addressing Scalability Collapse on Multicore Systems", in ACM Transactions on Architecture and Code Optimization (**TACO**)
    - **Yan Cui,** Yingxin Wang, Yu Chen, Yuanchun Shi, "Requester-Based Lock: A Scalable and Energy Efficient Locking Scheme on Multicore Systems", in IEEE Transactions on Computers(**TC**).
    - **Yan Cui**, Yu Chen, Yuanchun Shi, "Towards Scalability Collapse Behavior", in Concurrency and Computation: Practice and Experience(**CCPE)**
    - **Yan Cui**, Yingxin Wang, Yu Chen, Yuanchun Shi, etc, "Reducing Scalability Collapse via Requester-Based Locking on Multicore Systems", in **MASCOTS** 2012
    - **Yan Cui**, Weida Zhang, etc, "A Scheduling Method for Avoiding Kernel Lock Thrashing on Multicores", in **ICPADS** 2010
    - **Yan Cui**, Weiyi Wu, etc, "A Discrete Event Simulation Model for Understanding Kernel Lock Thrashing on Multicores" in **ICPADS** 2010
    - 秦岭, 陈渝，**崔岩**, 吴谨, 实现自适应锁的方法以及多核处理器系统, 申请号:201110394780, 公开号:CN102566979

Outline

❶ Background

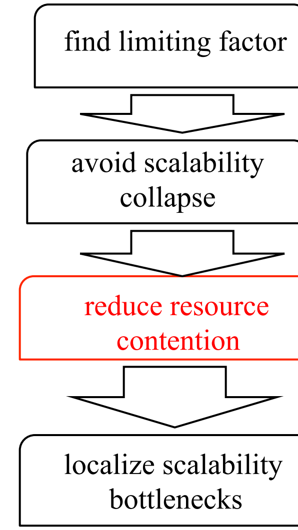
❷ Operating System Scalability Research

- System Interface Scalability Analysis
- Simulation and Avoidance of Scalability Collapse
- Hardware Resource Contention Avoidance
- Scalability Bottlenecks Localization Method

❸ Summary

❹ Fast Auto-turning Operating System Project

Hardware Resource Contention Avoidance



- **Related Work**
  - Hardware partition: Extended LRU[MICRO'06][HPCA'02]

  Need special hardware support, coarse-grain partition

  - Software partition: page coloring[HPCA'08][WIOSCA'07]

  Large recoloring overhead if behavior changes frequently

  - scheduling based: no special hardware requirement, low overhead[ASPLOS'10, USENIX'05, IPDPS'05]

1. No accurate and stable metric to measure how many resource are used by tasks
2. Only focus on context switching or load balancing modifications
3. No practical implementations

[1].M.K.Qureshi and Y.N.Patt, "Utility-Based Cache Partitioning: A low-overhead, high performance, runtime mechanism to partition shared caches", in MICRO 2006

[2].G.E.Suh, S.Devadas and L. Rudolph, "A New Memory Monitoring Scheme for Memory Aware Scheduling and Partitioning", in HPCA 2002

[3].Jiang Lin, et. al, "Gaining Insights into Multicore Cache Partitioning: Bridging the gap Between Simulation and Real Systems" in HPCA 2008

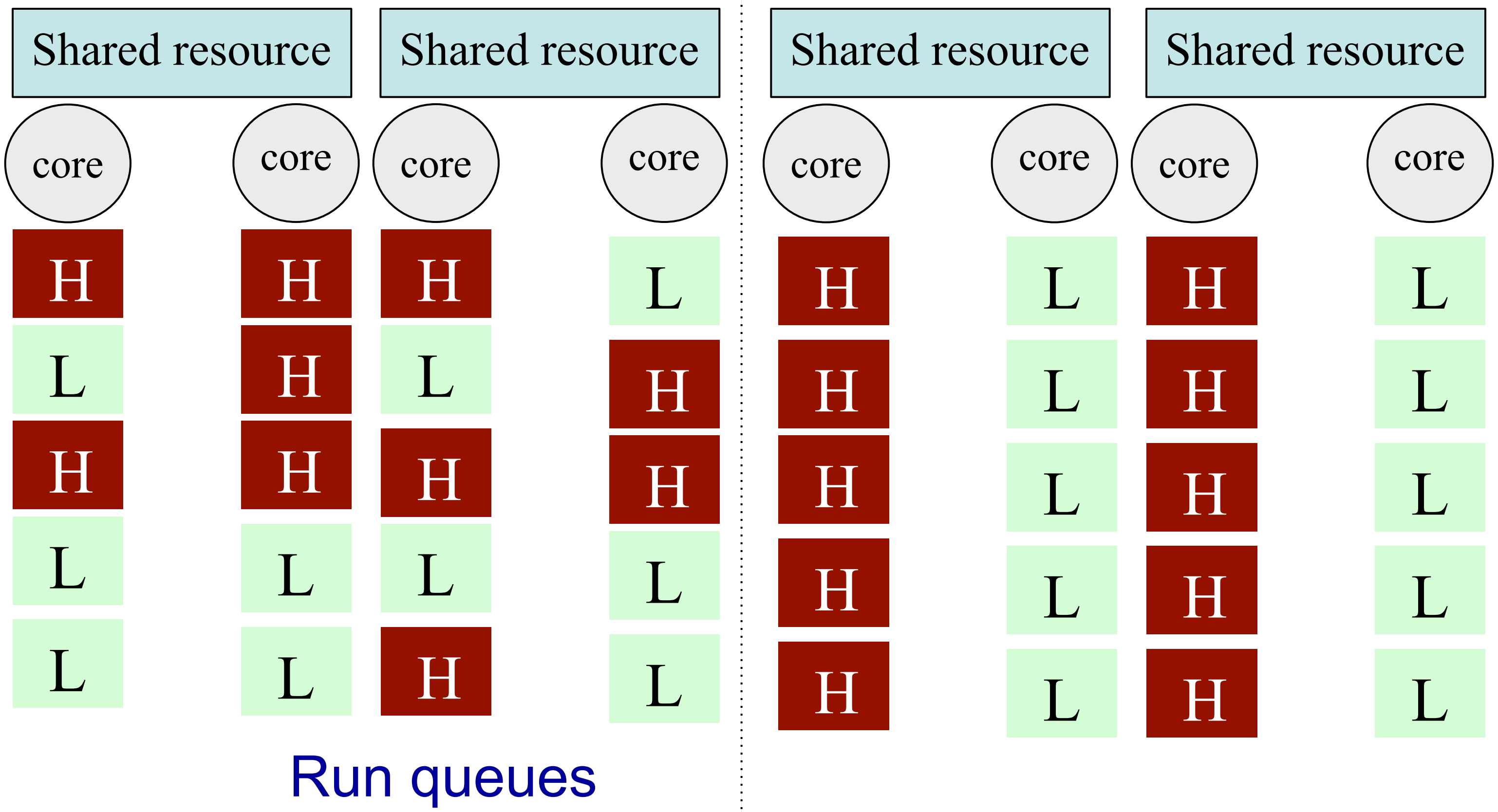
[4].D.Tam et al, "Managing Shared L2 Cache on Multicore Systems in Software", in WIOSCA 2007

[5].S.Zhuravlev, et al, "Addressing Shared Resource Contention in Multicore Processors via Scheduling", in ASPLOS 2010

[6].Alexandra Fedorova, et al, "Performance of Multithreaded Chip Multiprocessors and Implications for Operating System Design", in USENIX 2005

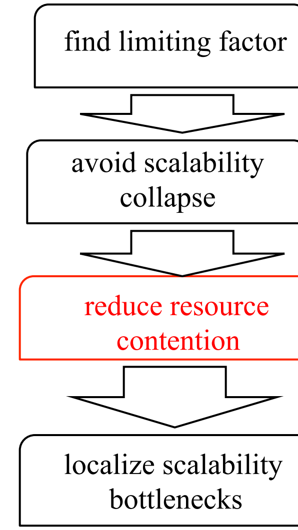
Hardware Resource Contention Avoidance

- Basic method



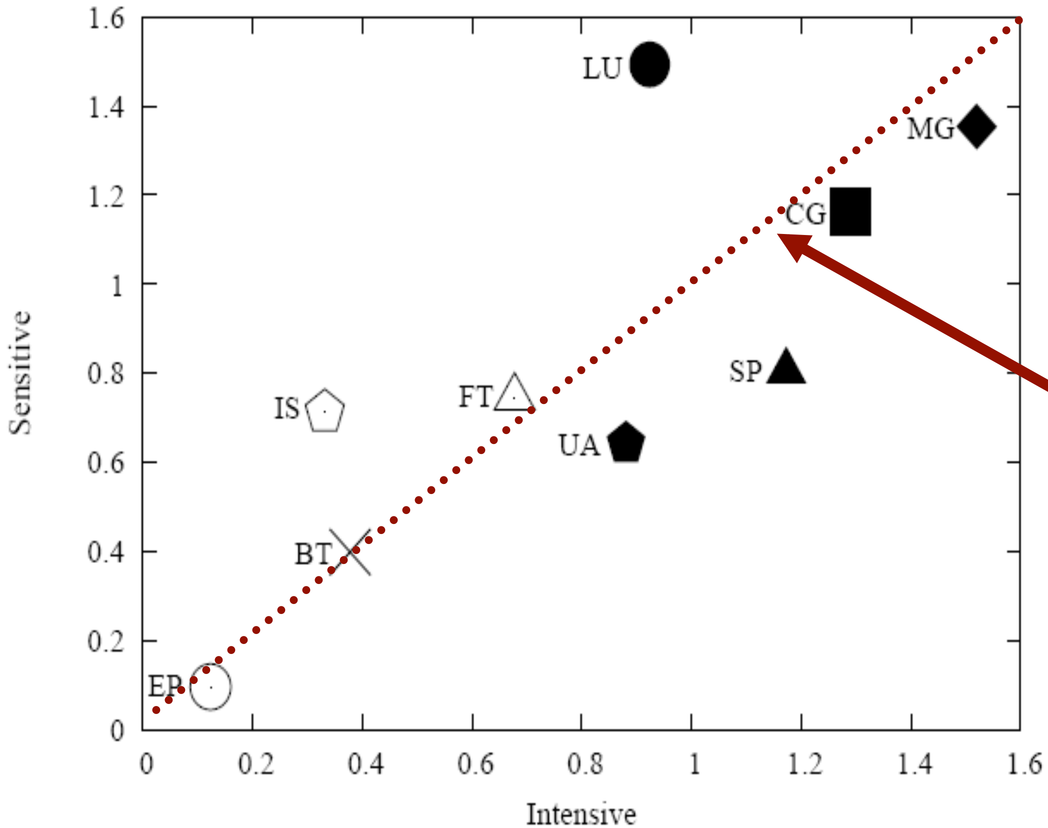
Default scheduling

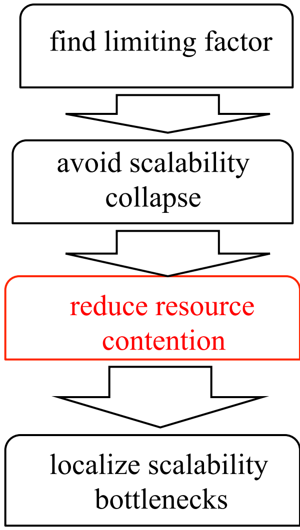
Proposed scheduling



Key Issue

- How to decide the resource usage of each task
  - performance degradation matrix Di,j is performance degradation of j when co-runing with i
  - calculate the intensity (row) and sensitivity(column)

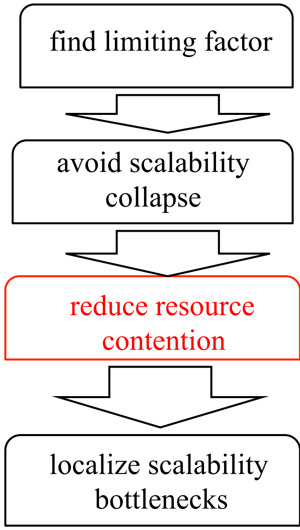




Strong correlation between intensity and sensitivity

Key Issue

- How to decide the resource usage of each task
  - intensity, but cannot acquire online
  - heuristic metric acquired by performance counters
  - accuracy: coefficient stability: length of column vector

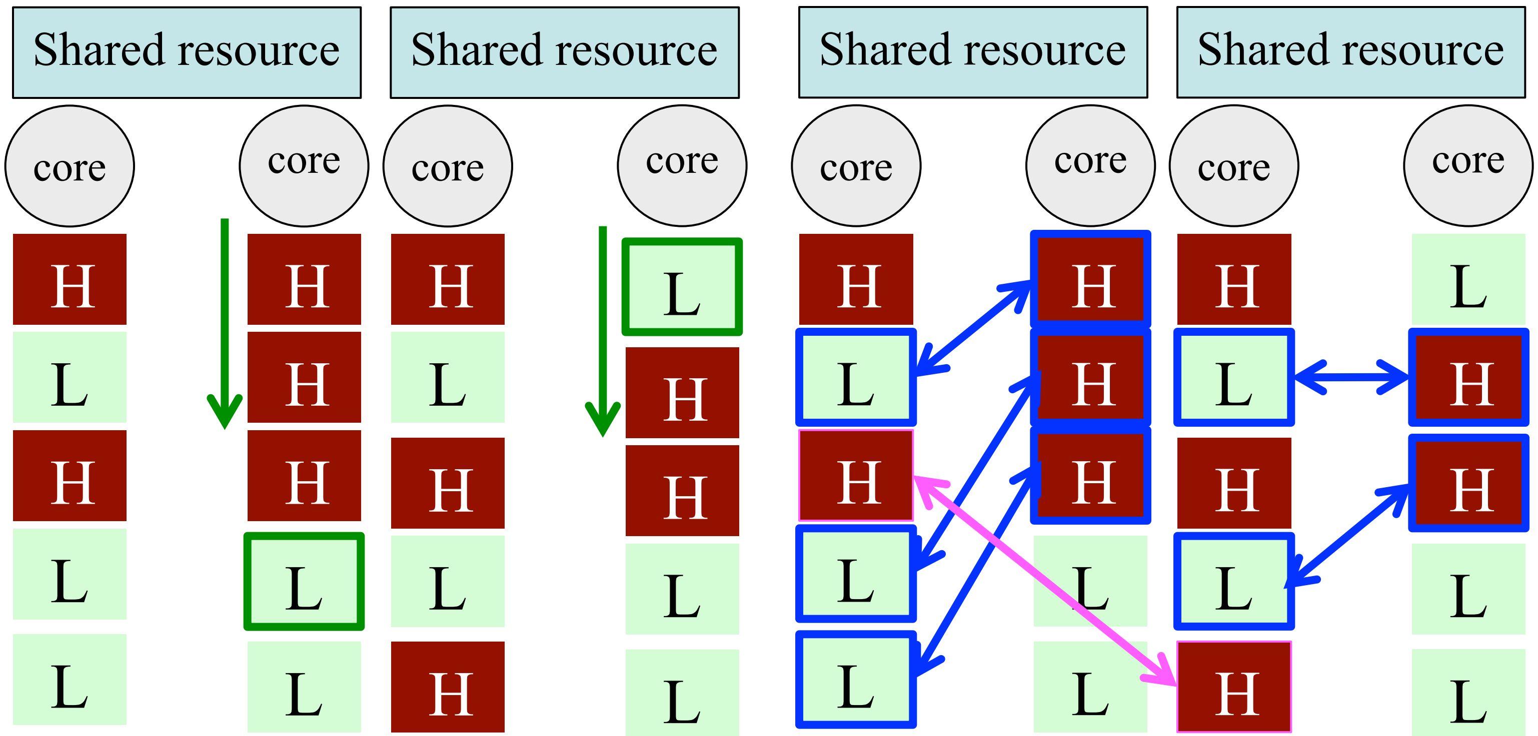


Metric	Coefficient	Stability	Ratio
$\frac{\#LLC\ cache\ misses}{\#inctructions}$	0.69	277.20	2.49×10^{-3}
$\frac{\#LLC\ cache\ misses}{\#cycles}$	0.81	549.60	1.47×10^{-3}
$\frac{\#stall\ cycles}{\#cycles}$	0.81	0.54	1.5
$\frac{\#bus\ transactions}{\#cycles}$	**0.99**	293.48	3.37×10^{-3}
$\frac{\#LLC\ access}{\#instructions}$	0.82	**0.0099**	82.83

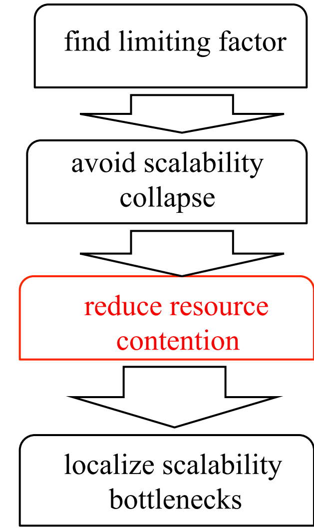
Select the number of LLC accesses per instruction

Key Issue

- How to schedule according resource usage
  - context switching: slave core select suitable task
  - load balancing: task exchanges between master-slave cores and master and master cores

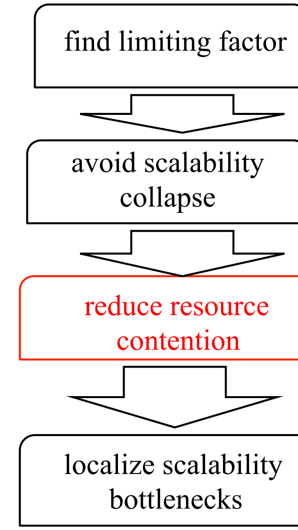


Run queues

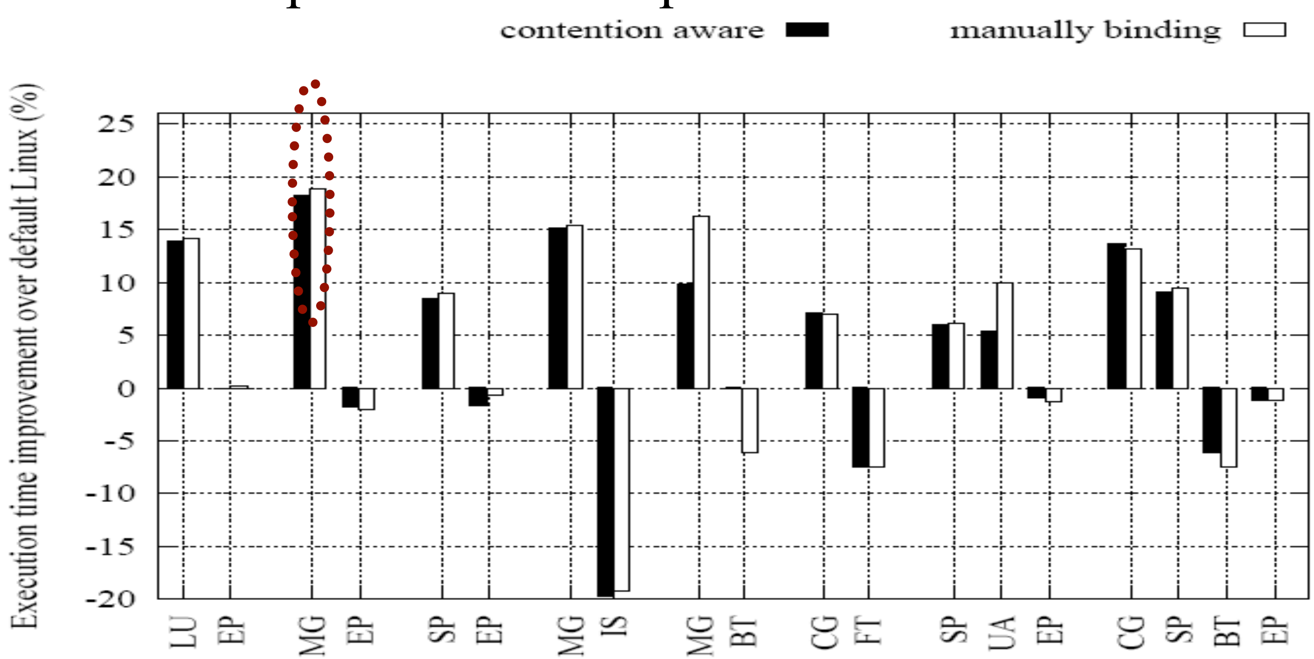


Experiments and Evaluation

- Kernel: 2.6.21.7 2.6.32
- Schedulers: CFS、RSDL、O(1)
- Platform: Intel 8 cores
- Benchmarks: 8 workloads from NAS-SER



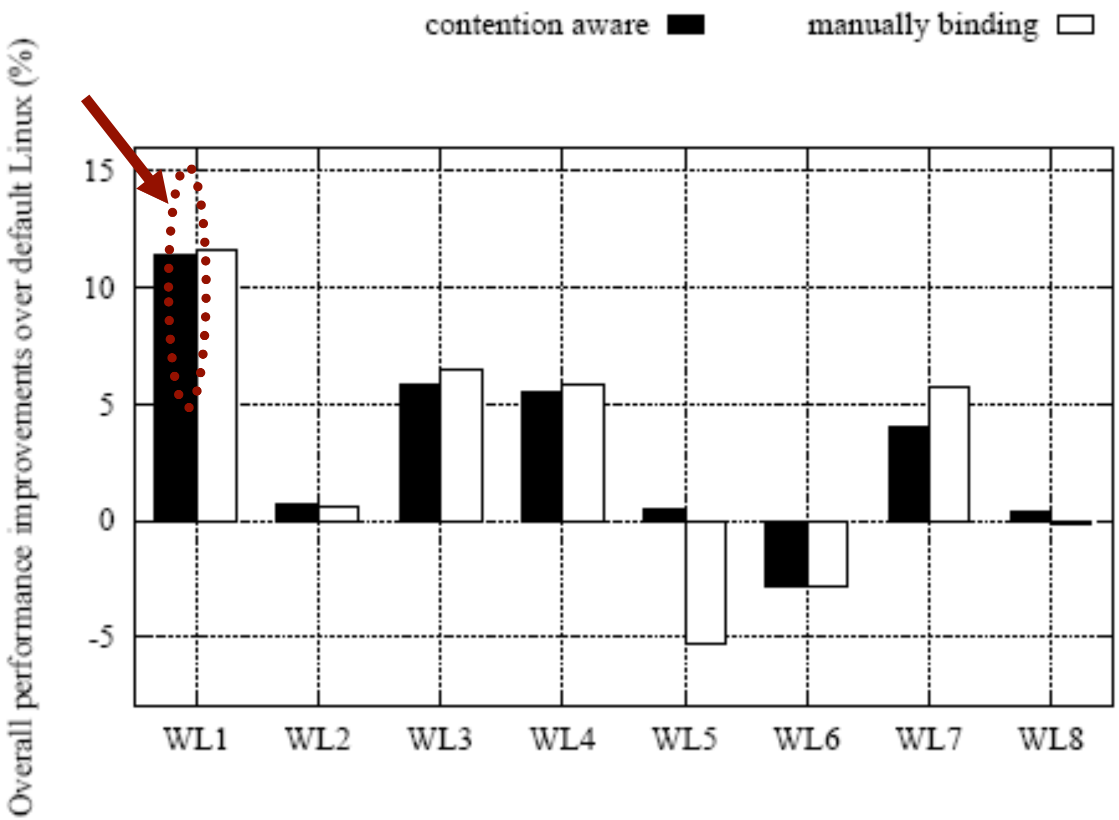
- Execution time reduction
  - 20% performance improvements

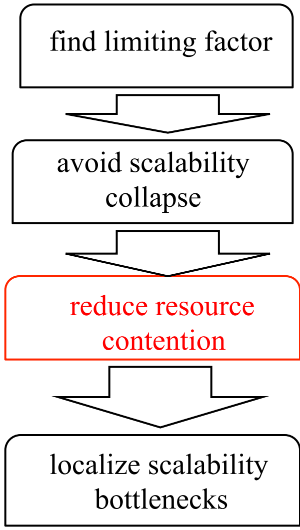


Hardware resource contention aware CFS scheduler

Experiments and Evaluation

- Scalability Improvements
  - 11.39% performance 12.85% scalability





Summary

- Contributions
  - propose a resource contention aware scheduler
    - Select the heuristic metric in the systematic way
    - Improved context switching and load balancing mechanisms
    - Real system integration
- Publications and Patents
  - Oxford Computer Journal and two Chinese patents
    - **Yan Cui**, Yingxin Wang, Yu Chen, Yuanchun Shi, "Mitigating Resource Contention on Multicore Systems via Scheduling", in Oxford Computer Journal(**CJ**)
    - 刘仪阳, 陈渝, 谭玺, **崔岩**, 一种线程调度方法、线程调度装置及多核处理系统, 申请号: 201110362773, 公开号:CN102495762
    - 刘仪阳,张知缴, 方帆, 陈渝, **崔岩**, 一种内存分配方法、装置及系统, 申请号: 201210176906.X

Outline

❶ Background

❷ Operating System Scalability Research

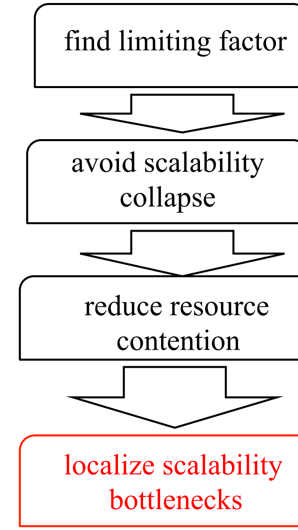
- System Interface Scalability Analysis
- Simulation and Avoidance of Scalability Collapse
- Hardware Resource Contention Avoidance
- Scalability Bottlenecks Localization Method

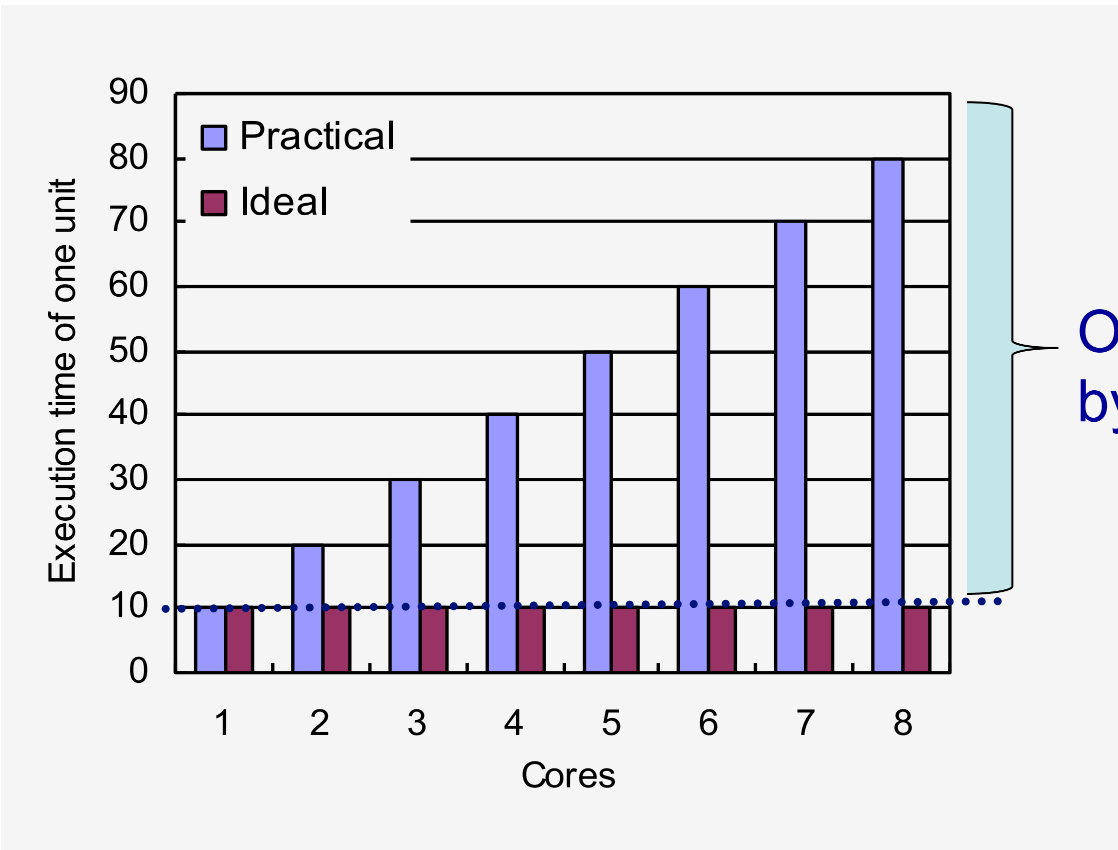
❸ Summary

❹ Fast Auto-tuning Operating System Project

Function Scalability Value

- Insight: If scalability bottlenecks exist, the execution time per unit work on multicores should be longer than that on single core

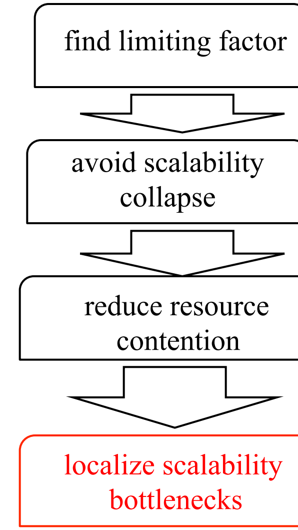


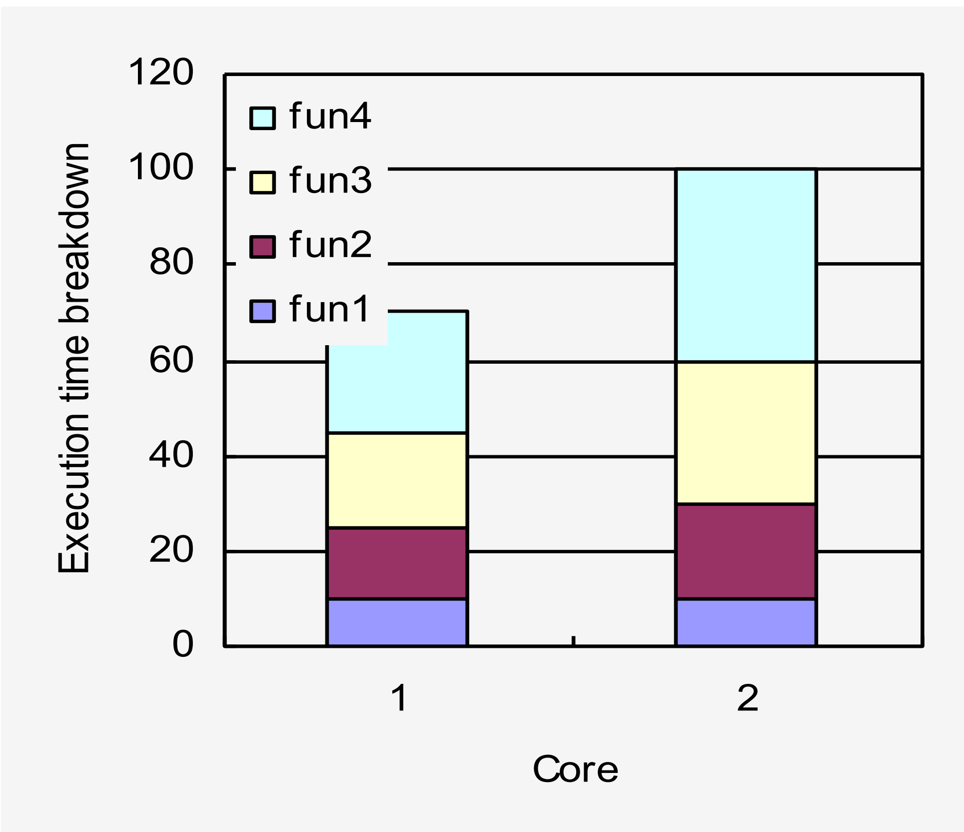


Overhead caused by bottlenecks

Function Scalability Value

- Breakdown the execution time of per unit work to each function defined as execution time difference on multicore and single core
- Functions at whole software stack are included (OS, library, apps)





fun(4) = 40-25=15

fun(3) = 30-20=10

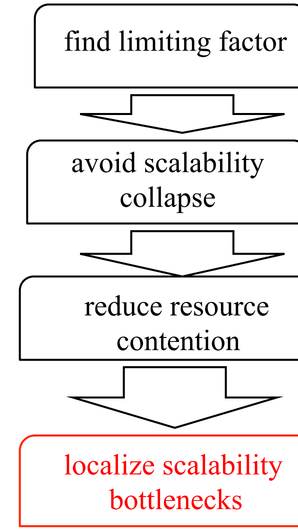
fun(2) = 20-15 =5

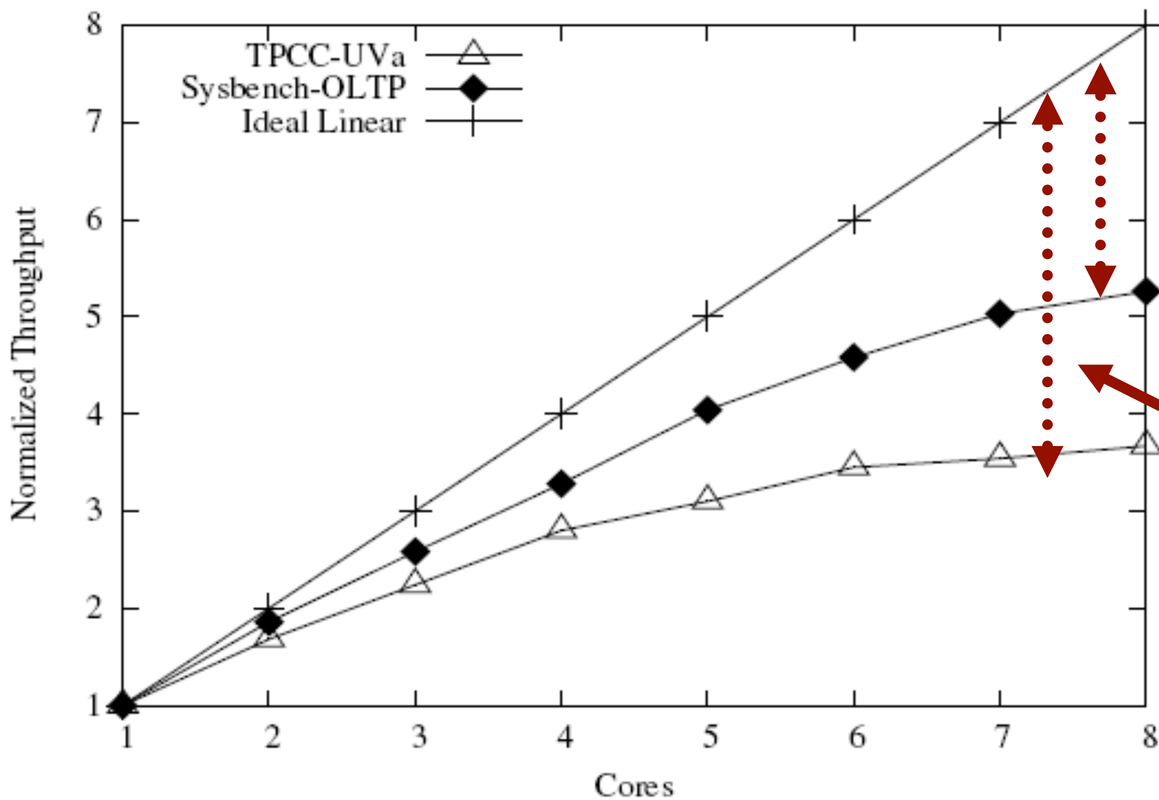
fun(1) = 10-10=0

Analyze function 4 first for scalability bottleneck

Experiments and Evaluation

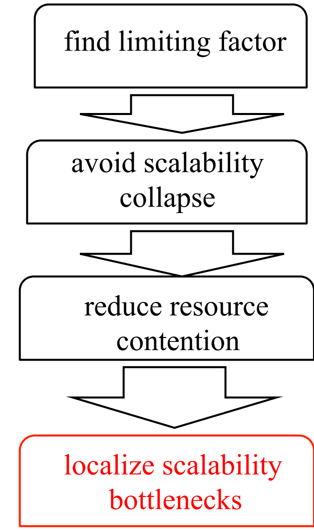
- Kernel: 2.6.25
- Platform: Intel 8 core
- Benchmark: OLTP TPCC-UVa (PostgreSQL) and Sysbench-OLTP (MySQL)
- Scalability





There are scalability bottlenecks

Scalability Bottleneck Localization Method



- Analyze using function scalability value

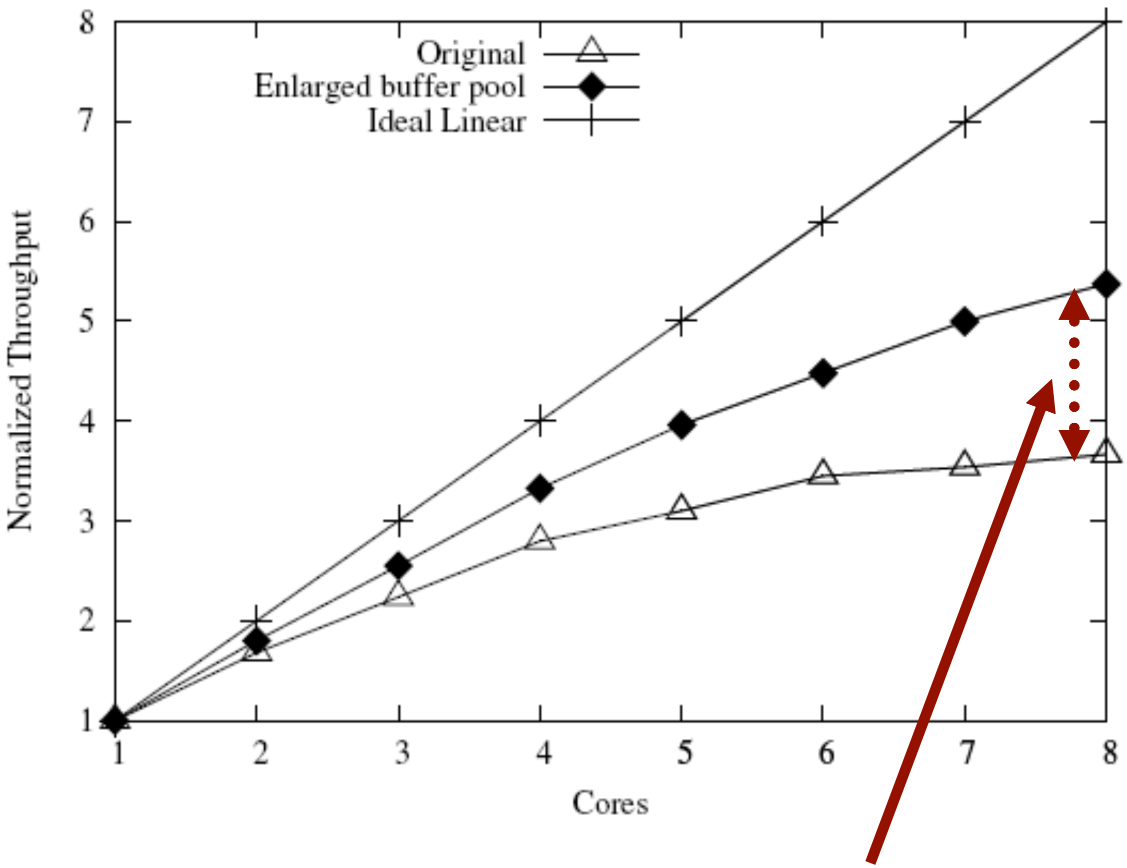
function	location	T_s	T_m	*scalability value*	weight
copy_user_generic_string	kernel	123.20	433.69	310.49	10.77%
ipc_lock	kernel	0.21	238.14	237.93	8.26%
task_rq_lock	kernel	2.13	217.82	215.69	7.49%
hrtick_set	kernel	3.39	161.75	158.36	5.50%
LWLockAcquire	database	113.62	270.40	156.78	5.43%
hash_search	database	82.09	178.82	96.73	3.25%
find_busiest_group	kernel	0.003	88.75	88.75	3.08%
XLogInsert	database	62.27	149.56	87.29	3.03%
schedule	kernel	13.20	92.35	79.15	2.75%
LWLockRelease	database	82.78	161.05	78.27	2.71%

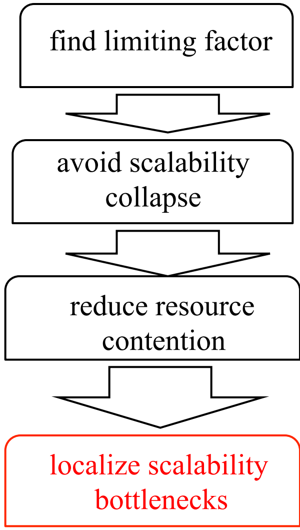
TPCC-UVa

1. copy_user_generic_string(): copy data from kernel to user
2. Indicate contention at database buffer pool
3. Increases the buffer pool until this function does not exist

Scalability Bottleneck Localization Method

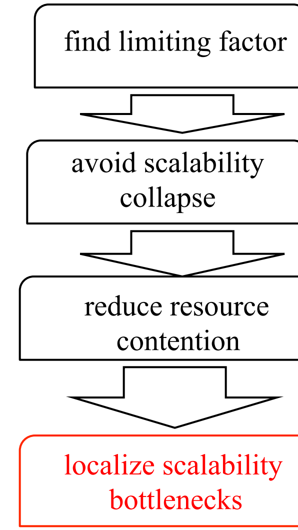
- Exploit Scalability Value





46.52% improvements in scalability

Scalability Bottleneck Localization Method



- Exploit function scalability value

function	location	T_s	T_m	*scalability value*	weight
LWLockAcquire	database	104.93	225.81	120.88	11.14%
XLogInsert	database	52.76	141.16	88.40	8.15%
task_rq_lock	kernel	3.65	64.60	60.95	5.62%
ipc_lock	kernel	0.40	51.35	50.95	4.70%
LWLockRelease	database	74.35	123.99	49.64	4.58%
hash_search	database	91.32	137.83	46.51	4.29%
hrtick_set	kernel	5.93	49.86	43.93	4.05%
HeapTupleSatisfiesSnapshot	database	69.27	105.15	35.88	3.31%
__copy_user_nocache	kernel	6.32	38.81	32.49	3.00%
memset	library	21.55	46.72	25.17	2.32%

TPCC-UVa

1. Other bottlenecks: database locks scheduling overhead System V IPC lock 2% improvements in scalability
2. Operating system is not the largest bottleneck now hard to iterate further(needs redesign the architecture)

TPCC-Uva: 49%, Sysbench-OLTP: 15.27% totally remove the scalability limiting factors in operating system

Summary

- Contribution
  - Propose a function scalability value based bottleneck localization method
    - 49% improvements in scalability

- Publications
  - IEEE ISPASS
    - **Yan Cui** , Yu Chen, Yuanchun Shi, "Scaling OLTP Applications on Commodity Multicore Systems" , in ISPASS 2010.

Outline

❶ Background

❷ Operating System Scalability Research

- System Interface Scalability Analysis
- Simulation and Avoidance of Scalability Collapse
- Hardware Resource Contention Avoidance
- Scalability Bottlenecks Localization Method

❸ Summary

❹ Fast Auto-tuning Operating System Project

Summary

- Understand the key scalability limiting factors by analysis
  - micro-benchmark test suite
  - in-deep analysis and comparison of important system service interfaces
- Propose simulation and avoidance methods of scalability collapse
  - Discrete event simulation based simulator accurate and fast
  - Requester-based lock: select lock waiting methods based on #lock requesters
  - lock-contention-aware scheduler control contention by scheduling
- Propose a resource contention aware scheduling policy
  - a method to select heuristic metric: consider accuracy and stability
  - improved context switching and load balancing mechanisms for better resource usage
  - 13% speedup improvements near the optimal performance
- Propose a scalability value based bottleneck localization method
  - 49% speedup improvements

Outline

❶ Background

❷ Operating System Scalability Research

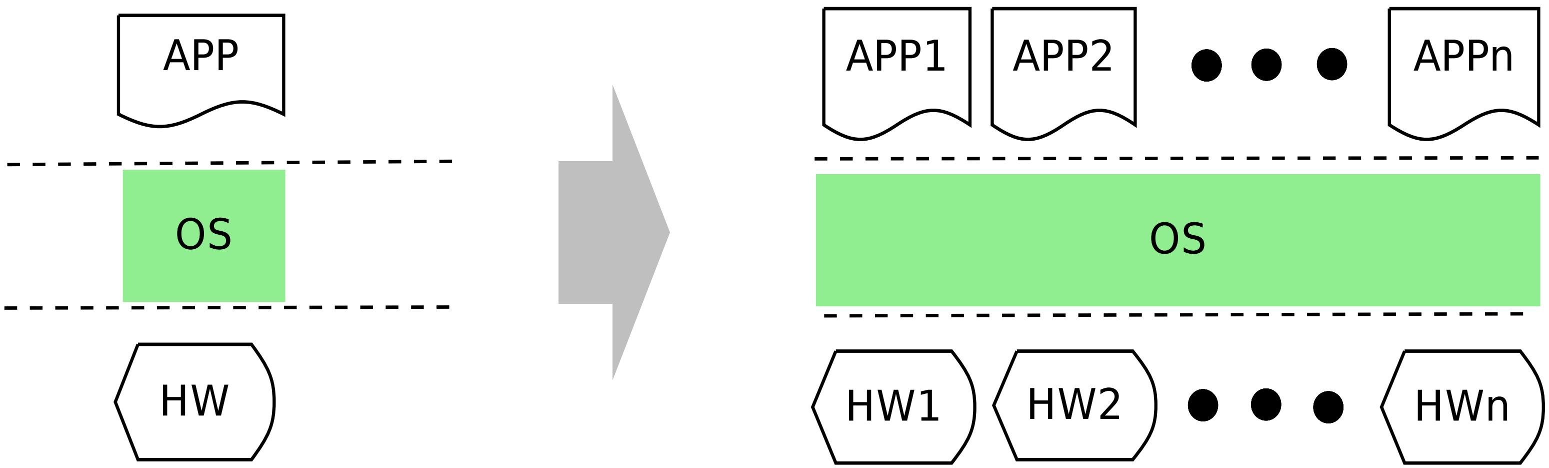
- System Interface Scalability Analysis
- Simulation and Avoidance of Scalability Collapse
- Hardware Resource Contention Avoidance
- Scalability Bottlenecks Localization Method

❸ Summary

❹ Fast Auto-tuning Operating System Project

Fast Auto-tuning Operating System

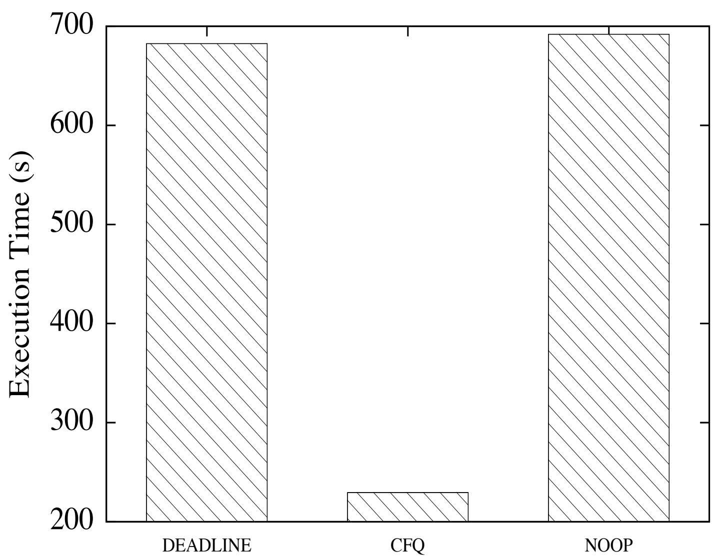
- Current operating system
  - Abstract hardware and provide API to application
  - code size increases rapidly with hardware and app types
  - reduce code complexity and improve flexibility (modules and parameters)



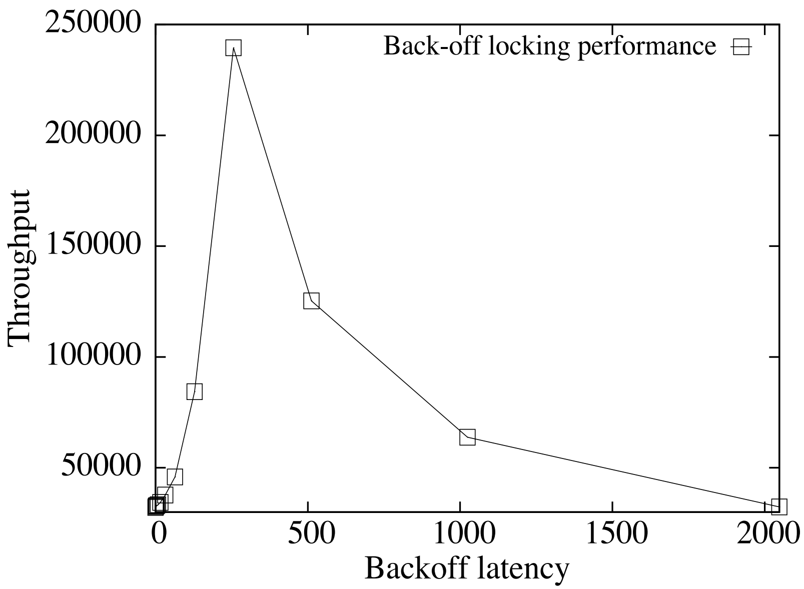
Software Stack

Fast Auto-tuning Operating System

- Knowledge Gap Between Developers and Users
  - Lots of configurable parameters in the kernel
  - For developer: tune parameters using a set of well-know benchmarks, but do not know user applications and requirements
  - For users: they know their applications, but they do not know how to tune kernel for their applications



Subsystem: Disk scheduling
Benchmark : Grep
Default: deadline



Subsystem: Synchronization
Benchmark : postmark
Default: 0

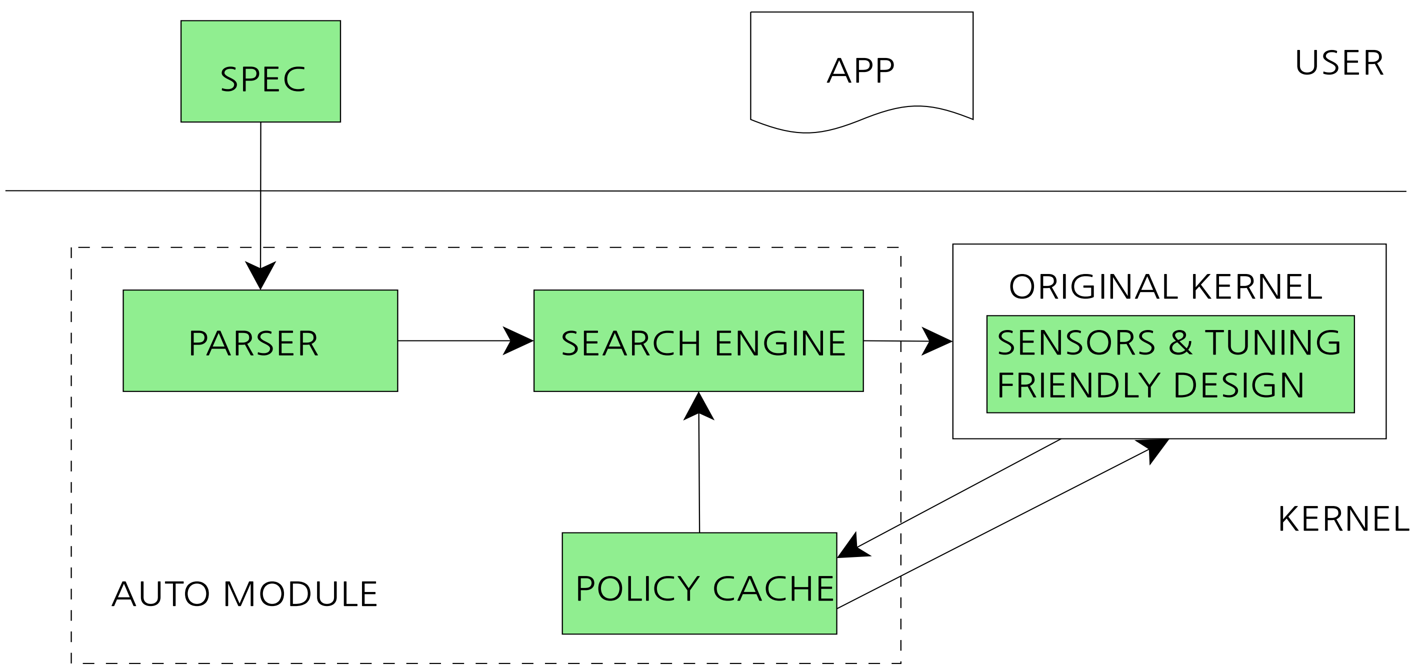
Fast Auto-tuning Operating System

- Software Architecture
  - kernel developer provides parameter specifications

<D:v1:spinlock:value:/proc/backoff_factor:int:0:0,32,2>

  - Auto-tuning kernel policies and parameters for users
  - Caching execution results in the policy cache

Insights: strong locality in policy reuse for some environment (build server)

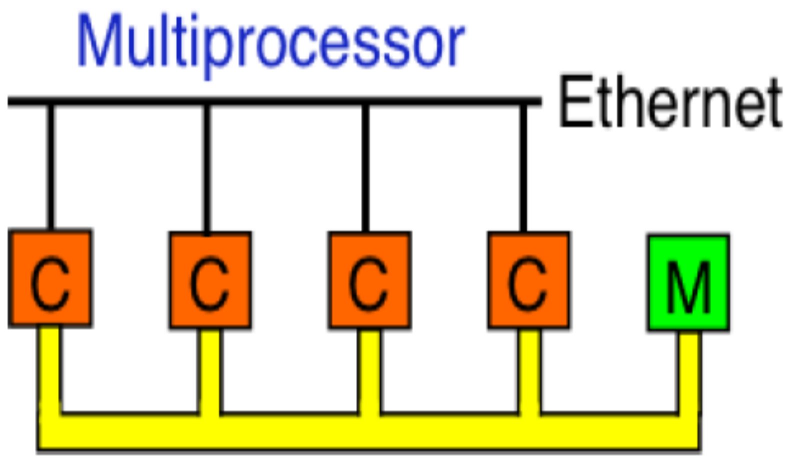


Thanks!

Any Questions?

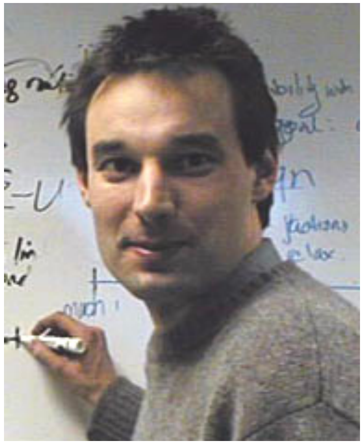
Backup Slices

- Differences between multicore and SMP
  - Parallel computing system
    - VU(1987)
    - Shared-memory cluster
    - 16 68030s (16MHZ)



Backup Slices

- Differences between Multicore and SMP
  - 1987: parallel computing does not solve a burning problem[APSys' 12]
    - There is a simpler way to improve performance
    - Clock rate doubles each 18 months
    - 16*16MHZ = 256MHZ
  - now: improve performance by parallel codes
    - Cannot buy a single core machine
    - parallel computing in new era
  - future: the number of cores increases exponentially & more shared hardware resources



[1].Frans Kaashoek. The Multicore Evolution and Operating Systems. In Keynote of APsys 2012.

Backup Slices

- Why throughput decreases because of lock contention?

Each thread 's core:

thread (void) {

while (1) {

some other code

spin_lock(&lock1);

critical section1

spin_unlock(&lock1);

some other code

…

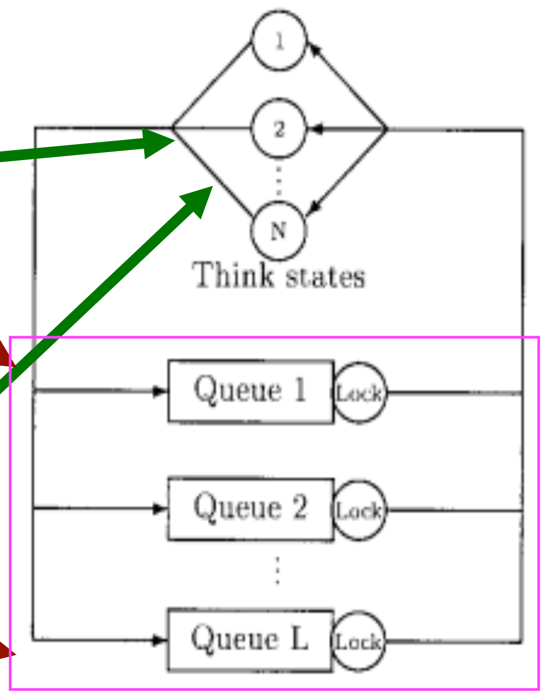
spin_lock(&lockL);

critical sectionL

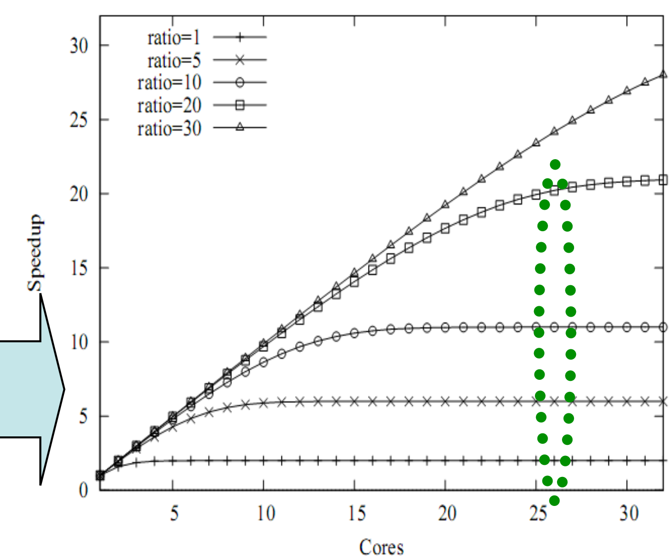
spin_unlock(&lockL);

}}

Delay center



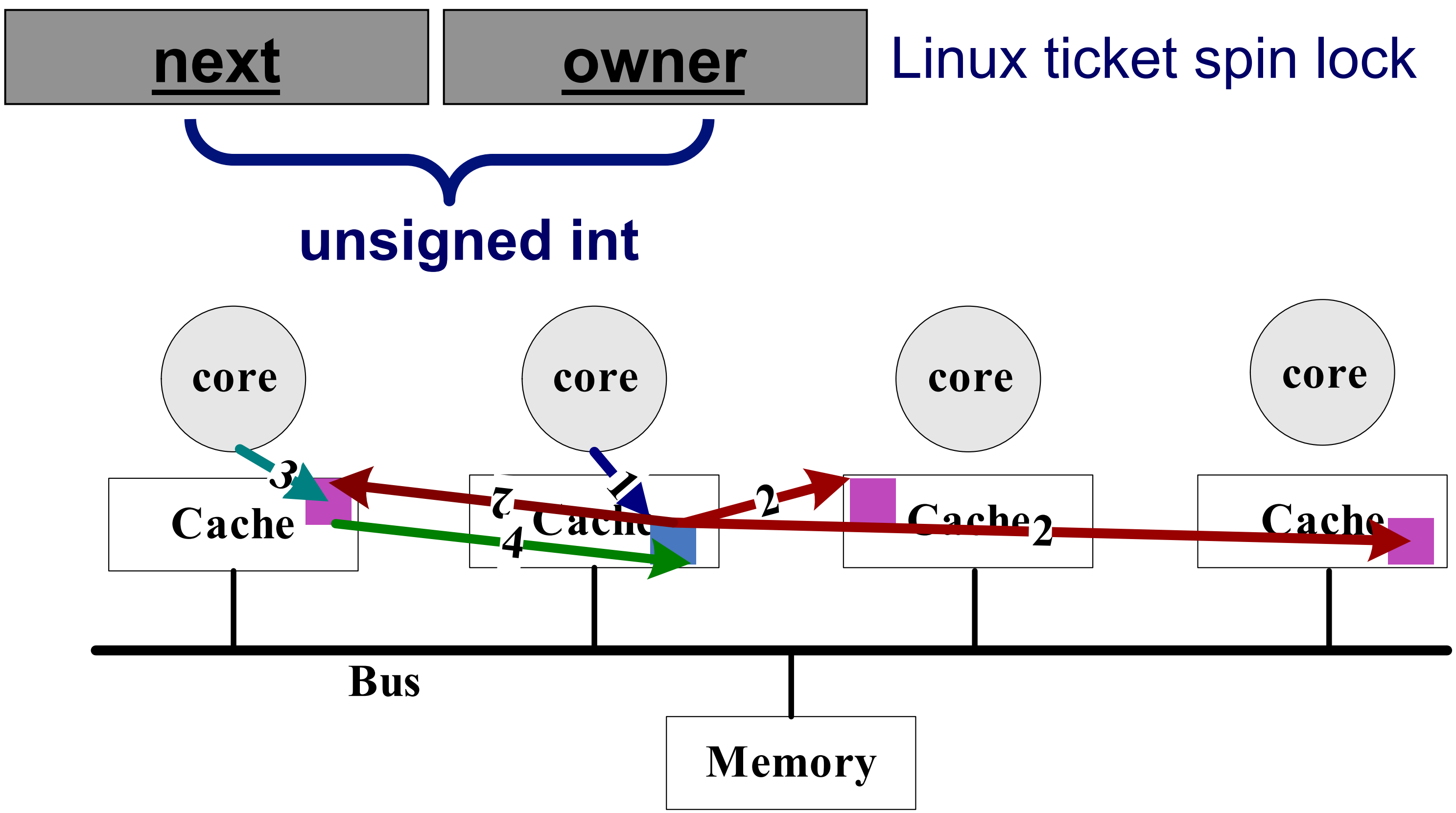
Queuing center



$$ratio = \frac{T_{NCS}}{T_{CS}}$$

Backup Slices

- Why throughput decreases because of lock contention?



Backup Slices

- probability of lock contention