Dynamic Virtual Machine Scheduling in Clouds for Architectural Shared Resources

Jeongseob Ahn, Changdae Kim, **Jaeung Han**,
Young-ri Choi†, and Jaehyuk Huh

KAIST and †UNIST

KAIST UNIST

Challenges for Cloud Computing

- Virtual machines (VM) share physical resources
  - Improve overall utilization of limited resources
  - Resource contention: potential performance degradation
- Multi-cores has enabled the sharing of architectural resources
  - Shared last level caches (LLCs) and memory controllers
- Contention on such architectural resources
  - Major reason for performance variance
  - VM performance affected by co-runners

KAIST UNIST

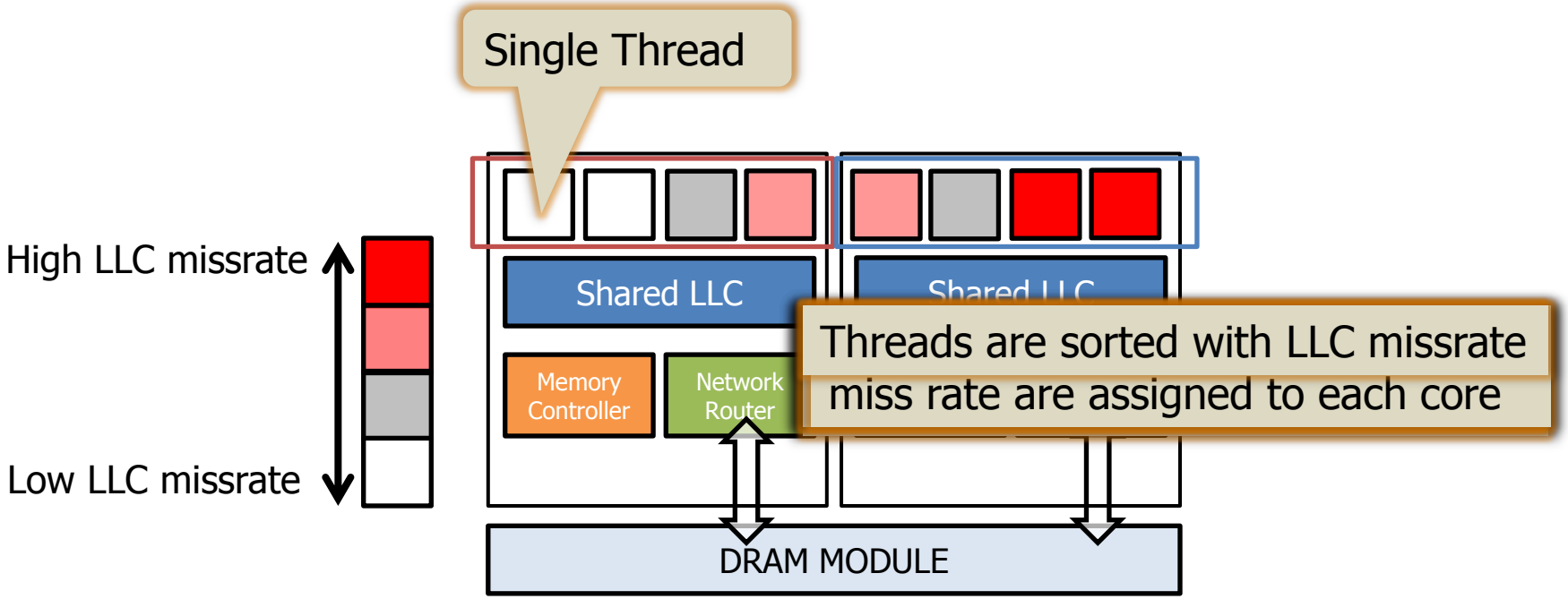
Shared Cache Contention

- Shared cache contention
  - When a core generates excessive cache misses and evicts the cached data from the other cores, contention occurs

- Way to solve cache contention problem
  - Partitioning caches
    - Qureshi et al. [MICRO'06]
    - Suh et al. [HPCA'02]
  - Scheduling threads carefully in multiple LLC environment
    - Merkel et al. [EuroSys'10]
    - Zhuravlev et al [ASPLOS'10]

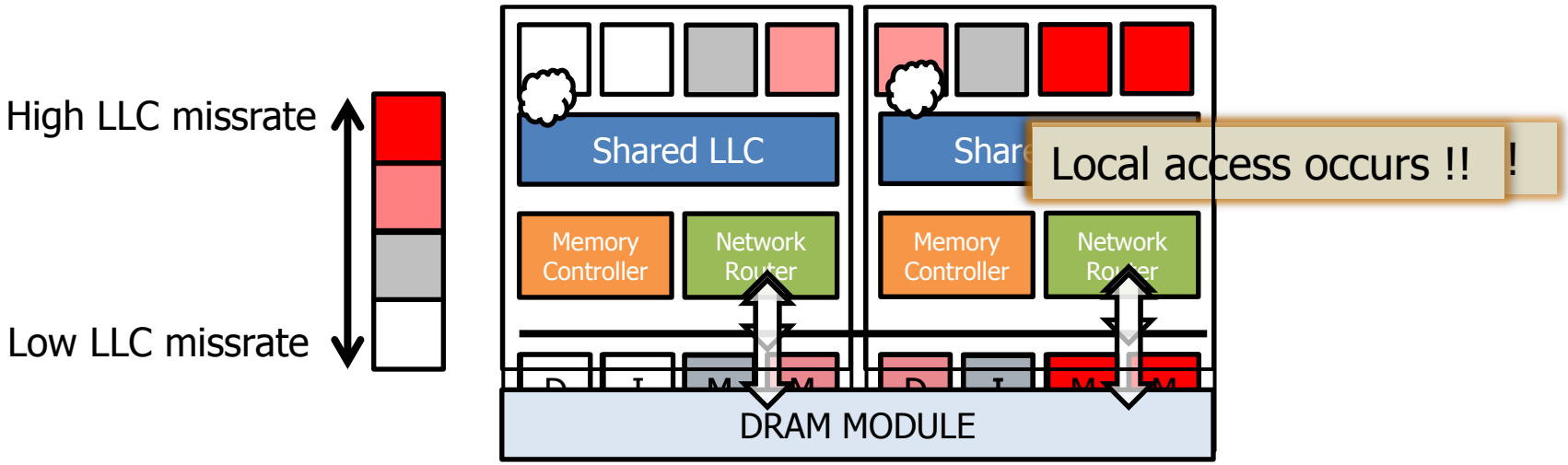
Shared Cache Scheduling Opportunities

- Scheduling threads to mitigate interferences in shared caches
  - Zhuravlev et al. [ASPLOS'10]
  - Evenly distribute threads according to LLC miss rate



Cache Contention + NUMA Affinity

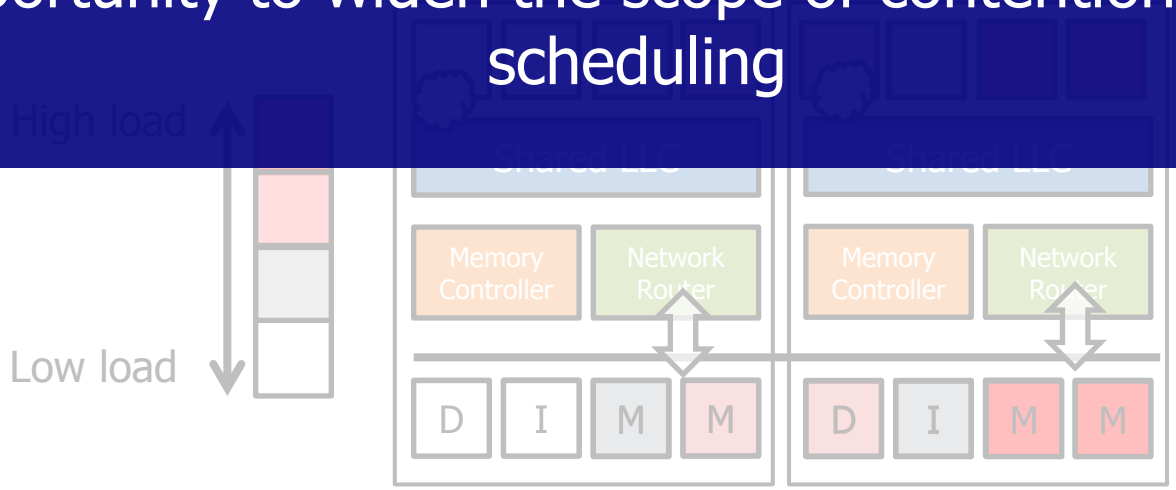
- NUMA affinity complicates such cache aware scheduling
  - Blagodurov et al. [USENIX ATC'11]
    - Investigated the impact of NUMA on cache-aware scheduling to reduce negative interferences among threads



Cache Contention + NUMA Affinity

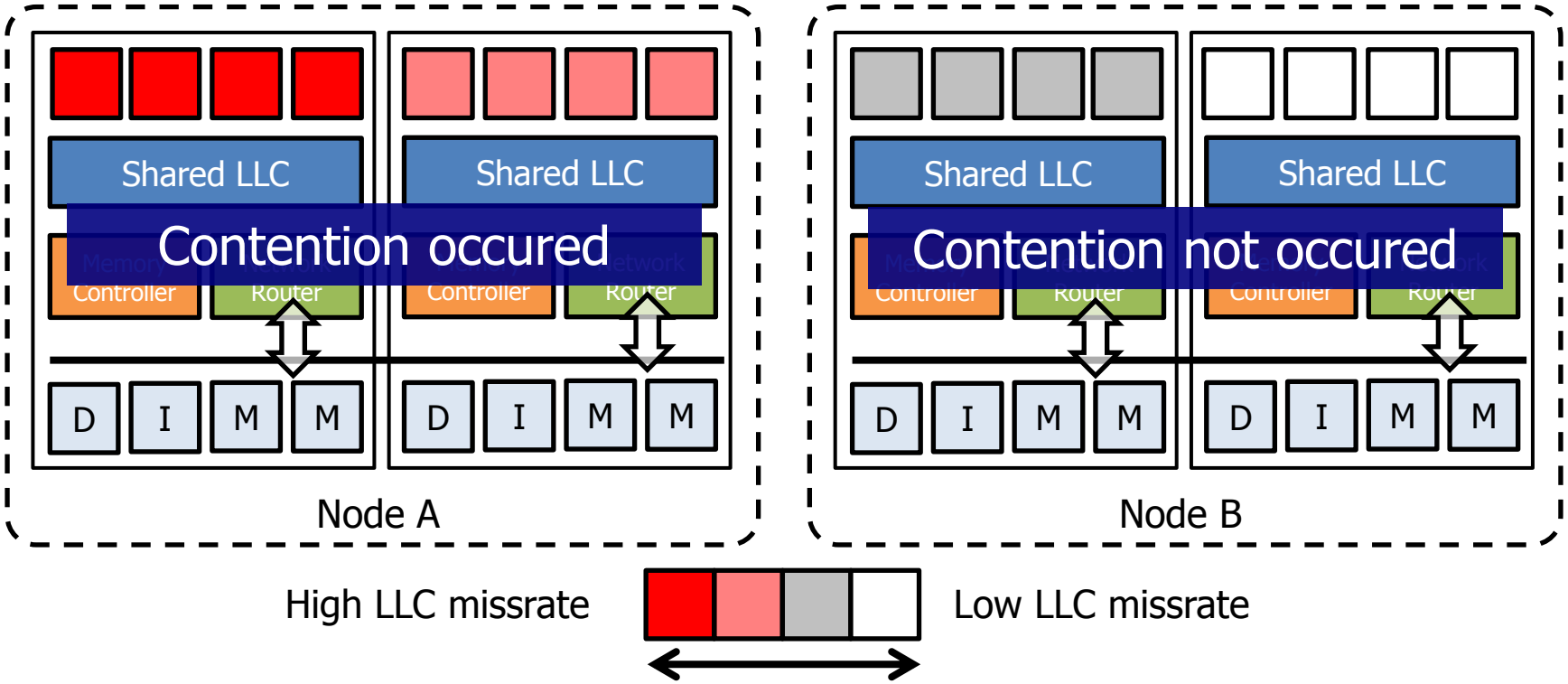
- NUMA affinity complicates such cache aware partitioning
  - Blagodurov et al. [USENIX ATC'10]
    - Investigated the impact of NUMA on cache-aware scheduling to reduce negative interferences among threads

However, cloud systems with virtualization open a new opportunity to widen the scope of contention-aware scheduling



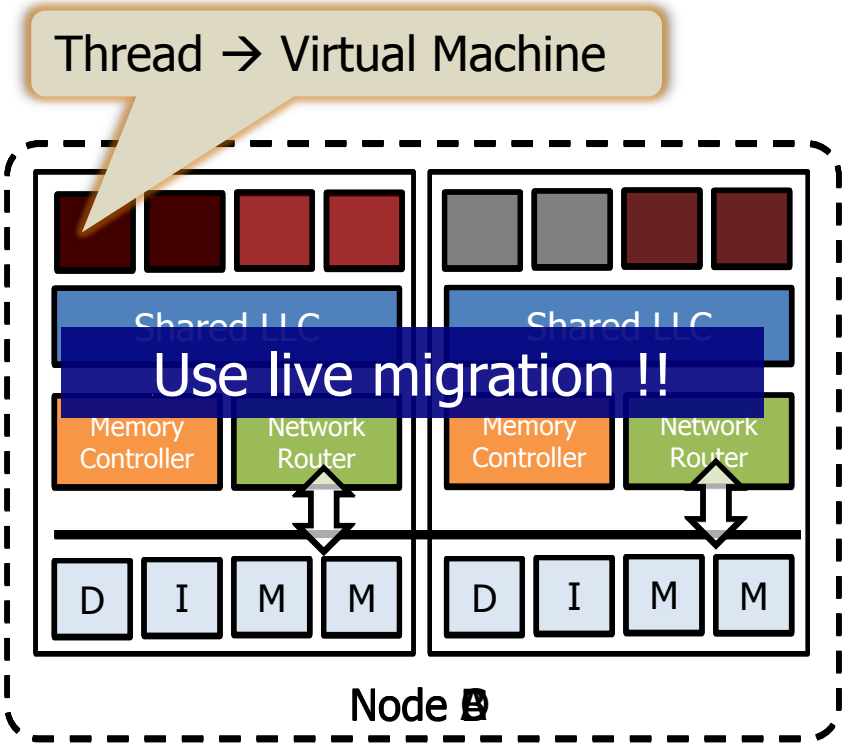
Scheduling Opportunities in Clouds

- Intra-system scheduling limits the opportunity to search the best groups of threads
  - Each machines are scheduled efficiently, but it is inefficient in global view



Scheduling Opportunities in Clouds

- In virtualized cloud system, we have an increased chance to find a better grouping of VMs

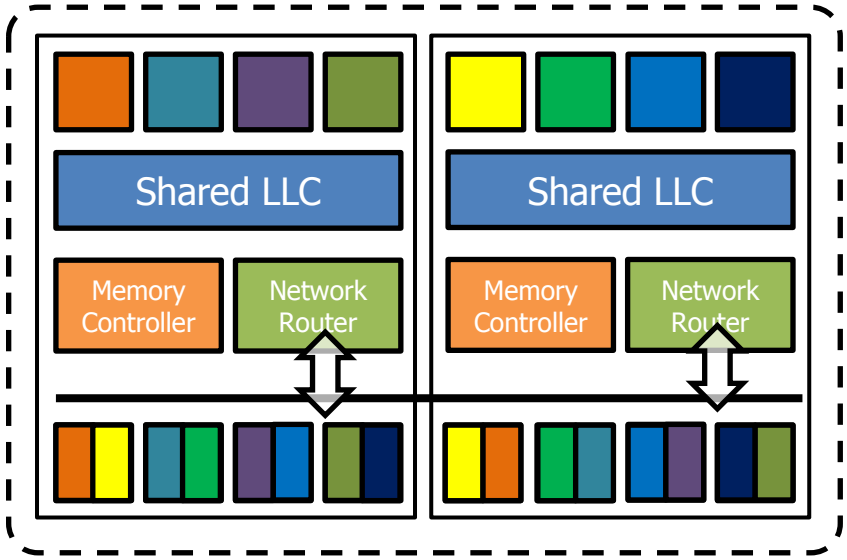


Performance Implication in Clouds

- Compare six different VM mapping policies for cache & memory
  - 2 version of cache sharing aspect
    - Best case (B) : map VMs to cores to minimize the sum of LLC misses from all the sockets in the clouds system
    - Worst case (W) : map VM with highest difference between the largest and smallest per-socket LLC misses in the clouds system
  - 3 version of NUMA affinity
    - Best allocation (B) : all VM memory pages are allocated in their local sockets
    - Worst allocation (W) : memory pages of all VMs are allocated in their remote sockets
    - Interleaved allocation (I) : assigns the memory pages of a VM to be always in both sockets in an interleaved way

Interleaved memory allocation

- Interleaved memory allocation
  - Remove NUMA affinity
  - Constant memory access latency regardless of the location of VM

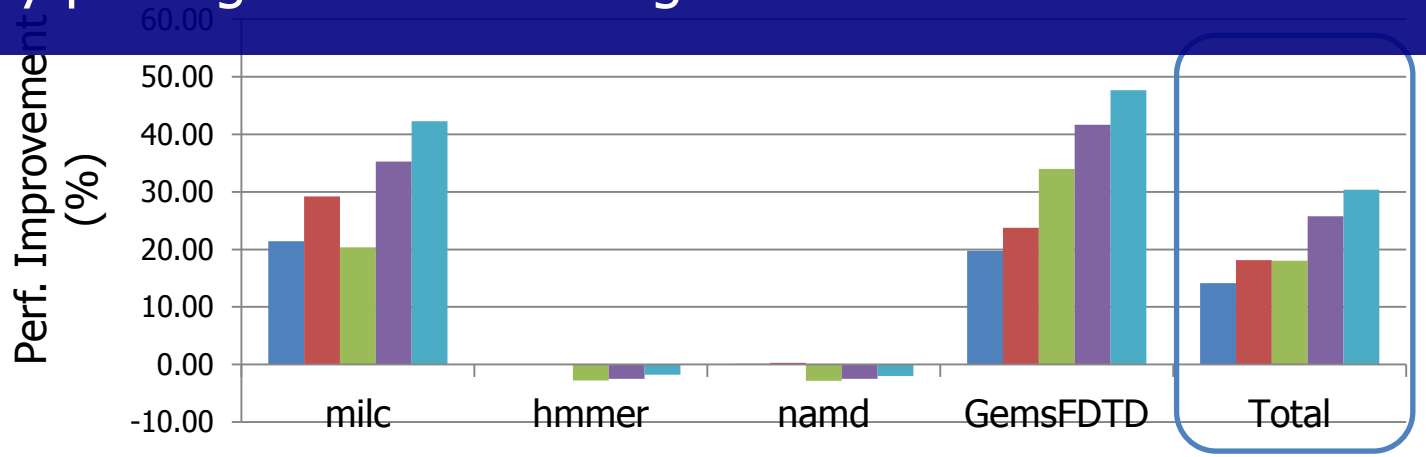


Access latency : NUMA best < NUMA interleaving < NUMA worst

Performance Implication in Clouds

- Normalized with cache worst & NUMA worst case
  - W-I : Cache worst & NUMA interleaved
  - W-B : Cache worst & NUMA best
  - B-W : Cache best & NUMA worst
  - B-I : Cache best & NUMA interleaved
  - B-B : Cache best & NUMA best

There is significant potential for performance improvements by placing VMs considering architectural shared resources

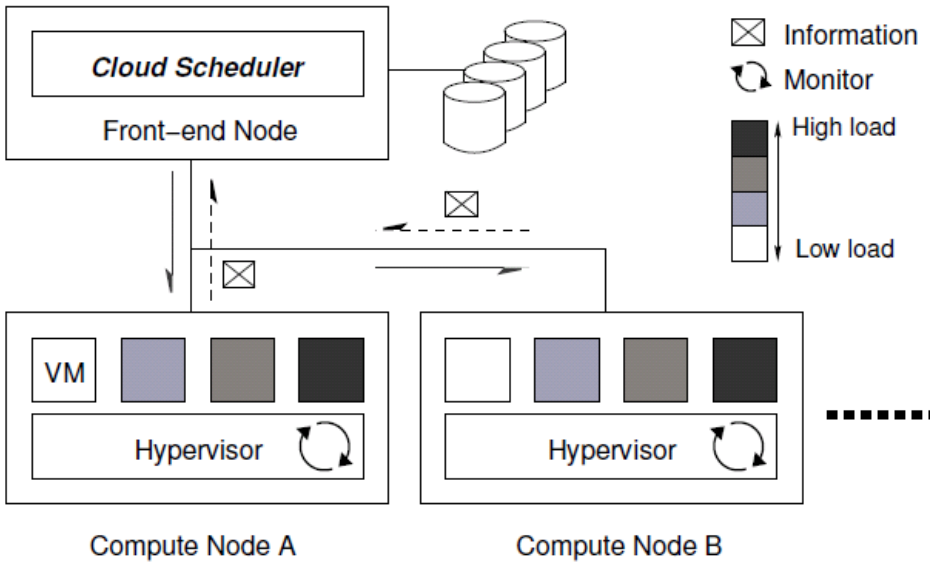


KAIST UNIST

Is Static Placement Possible?

- Maybe possible in single system
  - Execute same application mostly
- Impossible in clouds environment
  - VM behaviors cannot be predicted
  - Dynamic scheduling & placement is needed

Memory-Aware Cloud Scheduling



- Computing node
  - Check LLC misses with PMC, and send LLC miss and NUMA affinity information to front-end node
- Front-end node
  - Based on VM status information from all computing nodes, it makes global scheduling decisions

KAIST UNIST

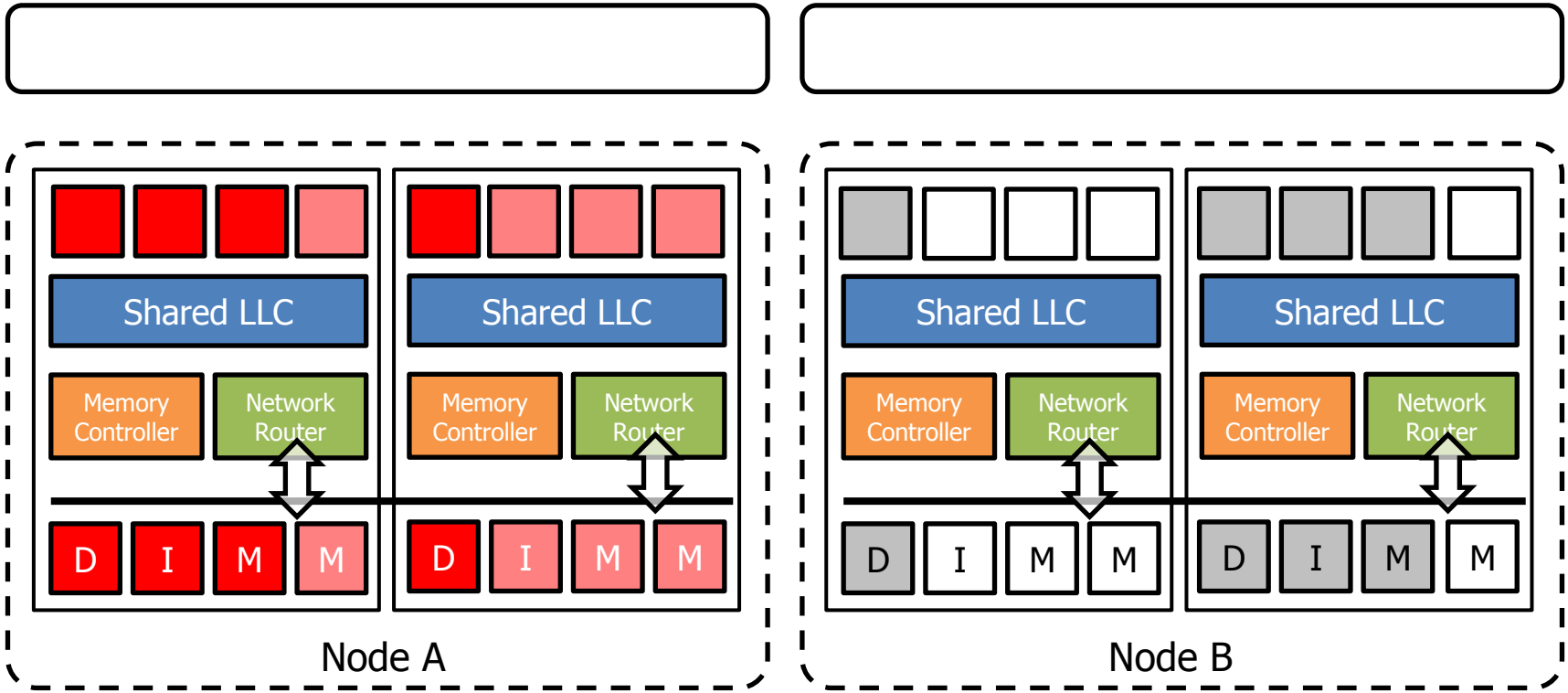
Two Scheduling policies

- Cache-aware scheduler
  - Only considers the contentions on shared caches
  - Composed of two phases
    - Local phase : VMs are rescheduled on intra-core
    - Global phase : VMs are migrated between inter-cores
- NUMA-aware scheduler
  - Extends the cache aware scheduler for NUMA affinity of VMs
  - Only exists global phase
    - To keep NUMA affinity

Cache-Aware Scheduler

- Local phase
  - Unit of scheduling is VM, not thread

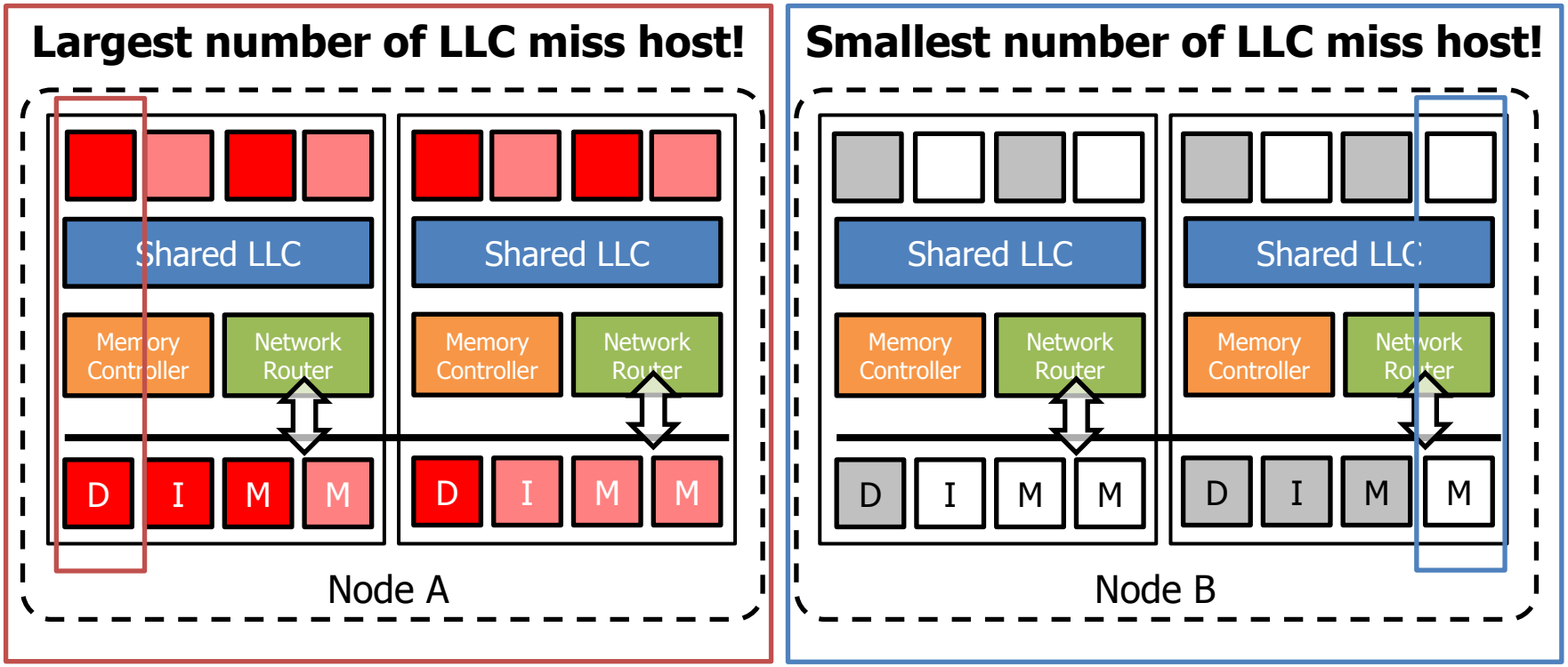
VMs i | 5.VMs h | 1.VMs are sorted with LLC missrate in each system | ach core | urces



Cache-Aware Scheduler

- Global Phase
  - Do not aware NUMA affinity

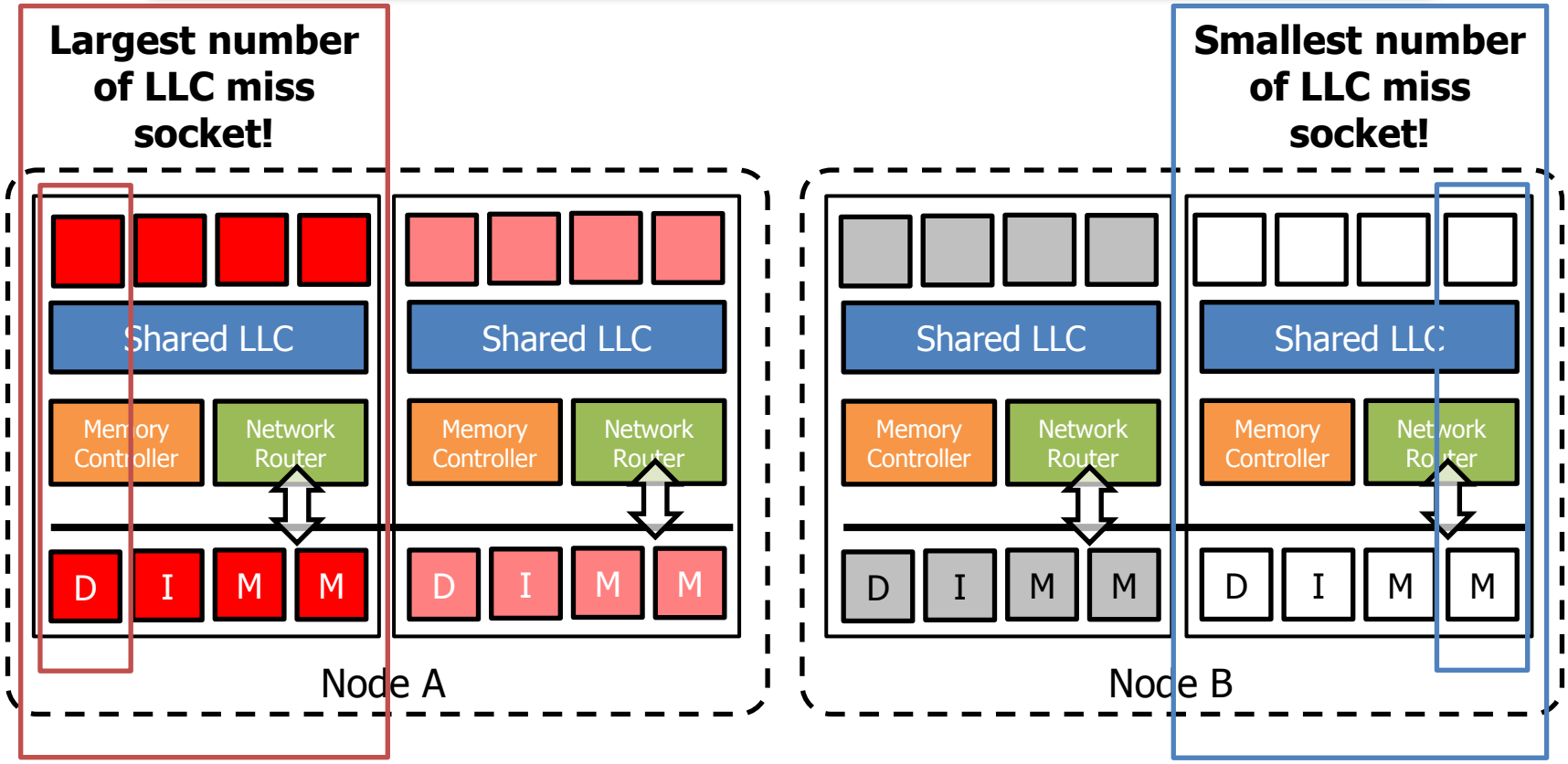
1. 3. If LLC difference is larger than *threshold,* swap them



NUMA-Aware Scheduler

- Similar with global phase of cache-aware scheduler, but unit of migration is socket, not host.

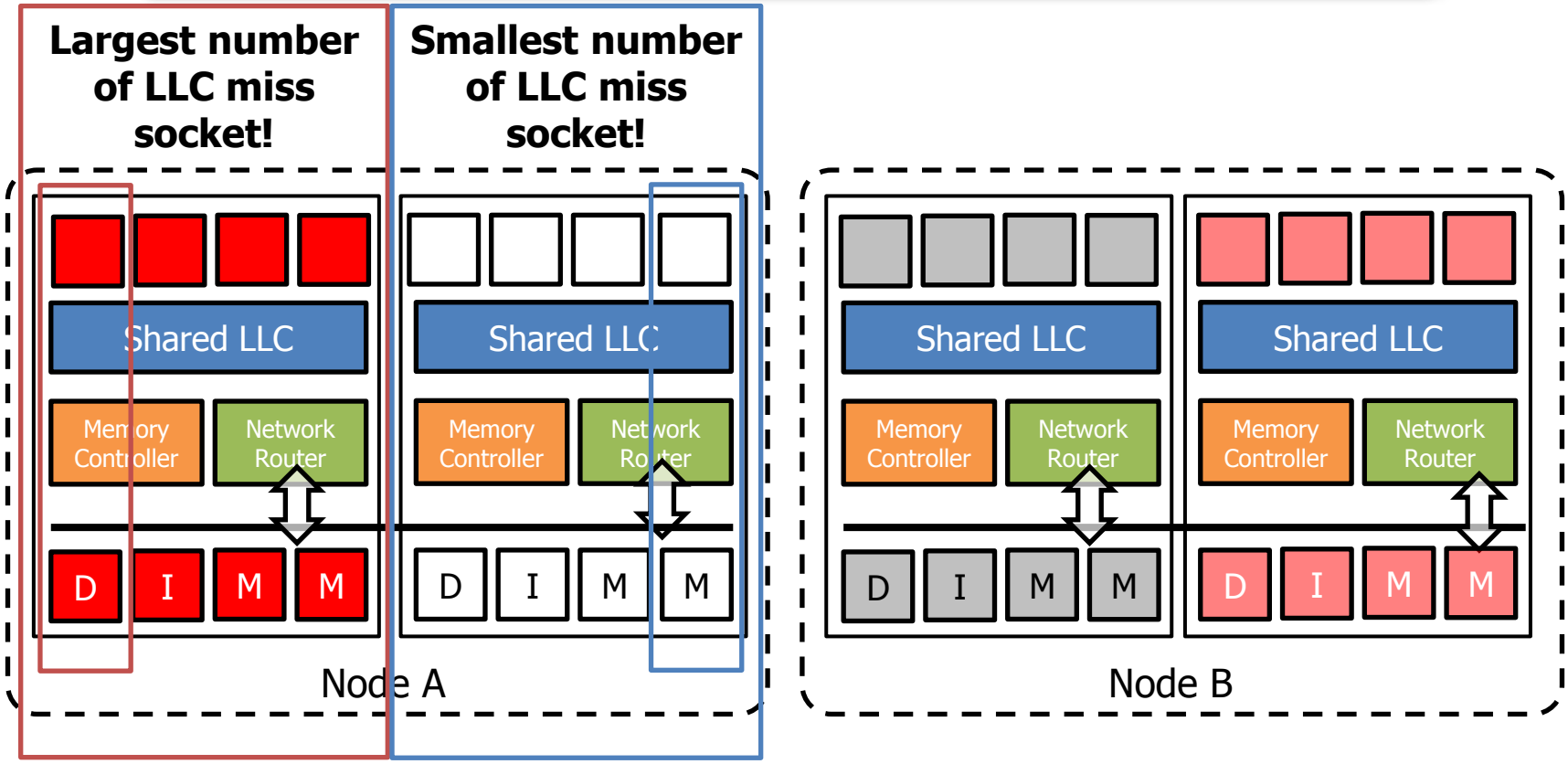
3. If LLC difference is larger than *threshold,* swap them



NUMA-Aware Scheduler

- Intra-host migration is also available.

3. If LLC difference is larger than *threshold,* relocate memory



Experimental Methodology

- Testbed
  - Front-end node (1)
    - Running proposed schedulers
    - Storage servers for VM images
  - Computing node (4)
    - Intel Xeon 8 cores on two chips
      - Each chip has 12MB LLC shared by 4 cores
    - Each VM employs a single core and 1GB memory
- On top of Xen hypervisor, each node runs 8 guest VMs
  - VMs use a Ubuntu distribution based on Linux kernel 2.6.18

Experimental Methodology

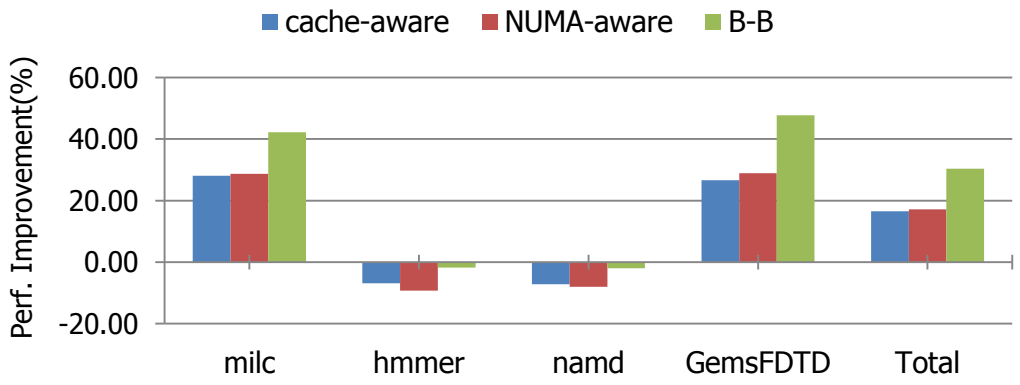
- Benchmark applications
  - Selected from SPECcpu 2006

	Memory bound		CPU-bound	
1	GemsFDTD	milc	hmmer	namd
2	omnetpp	lbm	gobmk	sjeng
	Memory bound			CPU-bound
3	cactusADM	gcc	soplex	namd
	Memory bound	CPU-bound		
4	libquantum	tonto	povray	sjeng
	Memory bound			
5	cactusADM	milc	omnetpp	soplex
	CPU bound			
6	gobmk	sjeng	namd	povray

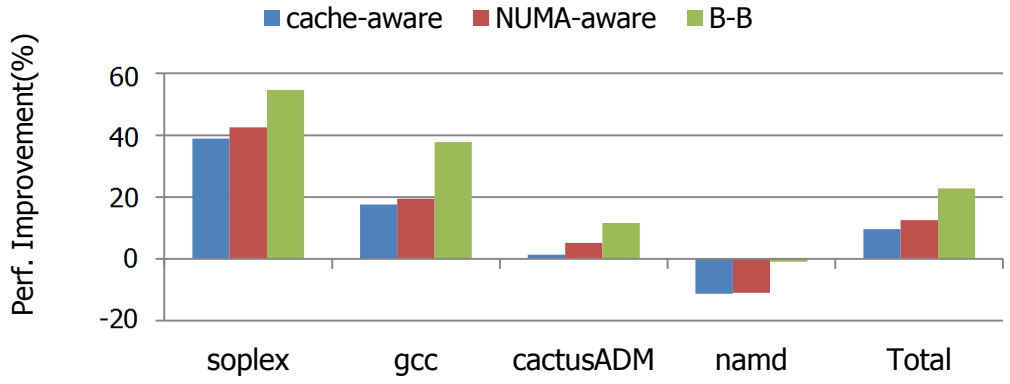
  - Each workload has 4 different applications
  - 8 instances of each application run on the 32VMs in our testbed

Performance Improvements

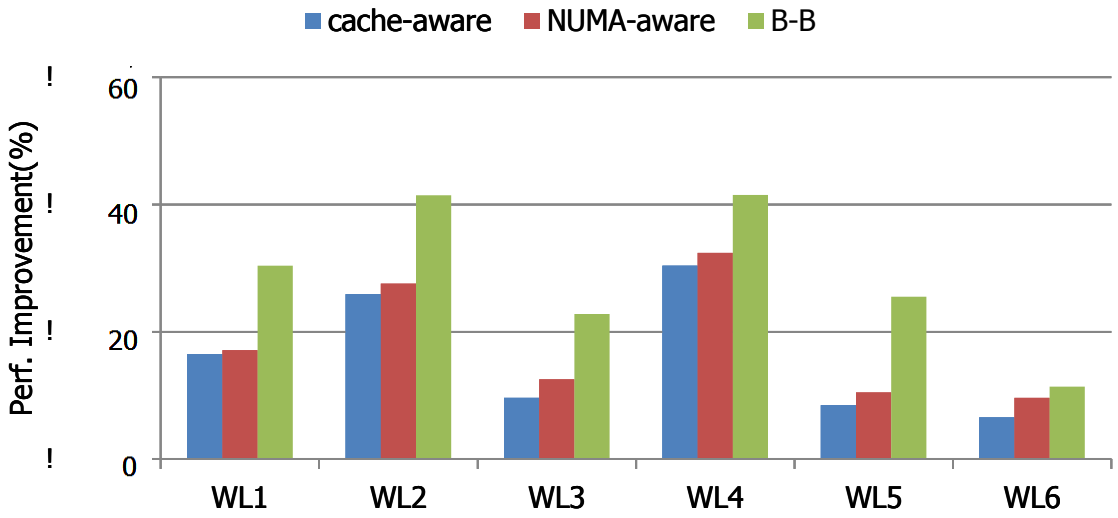
- WL1: 2 Memory bound + 2 CPU bound



- WL3: 3 Memory bound + 1 CPU bound



Performance Improvements



- Show similar trends except for WL6
  - WL6 consists of all CPU-bound applications

KAIST UNIST

Conclusions

- We proposed memory-aware cloud scheduling
  - Use live migration
- VM live migration can be used to mitigate architectural resource contentions
  - Cloud-level VM scheduler must consider such hidden contentions
- Future work
  - Extend our preliminary design of NUMA-aware scheduling
  - Investigate a systematic approach based on a cost-benefit analysis for VM migrations and contention reductions

KAIST UNIST

Thank you !

Dynamic Virtual Machine Scheduling in Clouds for Architectural Shared Resources

Jeongseob Ahn, Changdae Kim, Jaeung Han,
Young-ri Choi†, and Jaehyuk Huh

KAIST and †UNIST