iBench: Quantifying Interference in Datacenter Applications

Christina Delimitrou and Christos Kozyrakis

Stanford University

IISWC – September 23th 2013

Executive Summary

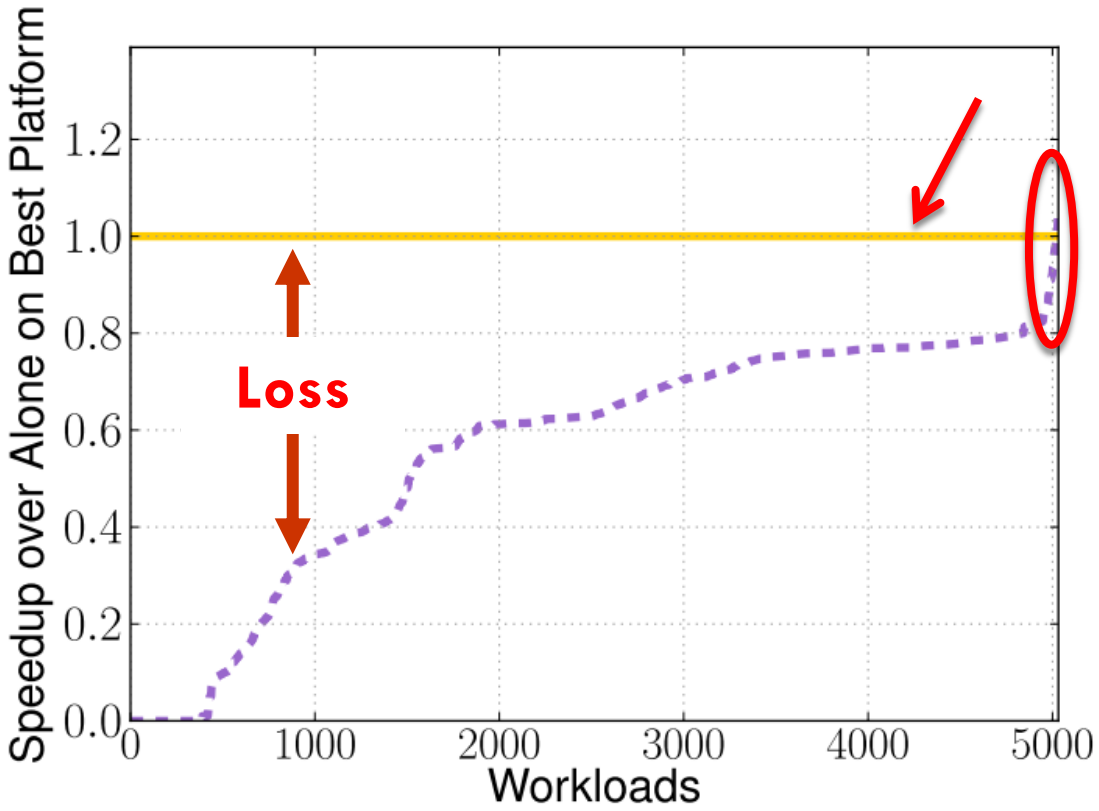
- Problem: Increasing utilization causes interference between co-scheduled apps
  - Managing/Reducing interference → critical to preserve QoS
  - Difficult to quantify → can appear in many shared resources
  - Relevant both in datacenters and traditional CMPs
- Previous work:
  - Interference characterization: BubbleUp, Cuanta, etc. → cache/memory only
  - Long-term modeling: ECHO, load prediction, etc. → training takes time, does not capture all resources
- iBench is an open-source benchmark suite that:
  - Helps quantify the interference caused and tolerated by a workload
  - Captures many different shared resources (CPU, cache, memory, net, storage, etc.)
  - Fast: Quantifying interference sensitivity takes a few msec-sec
  - Applicable in several DC and CMP studies (scheduling, provisioning, etc.)

Outline

- Motivation
- iBench Workloads
- Validation
- Use Cases

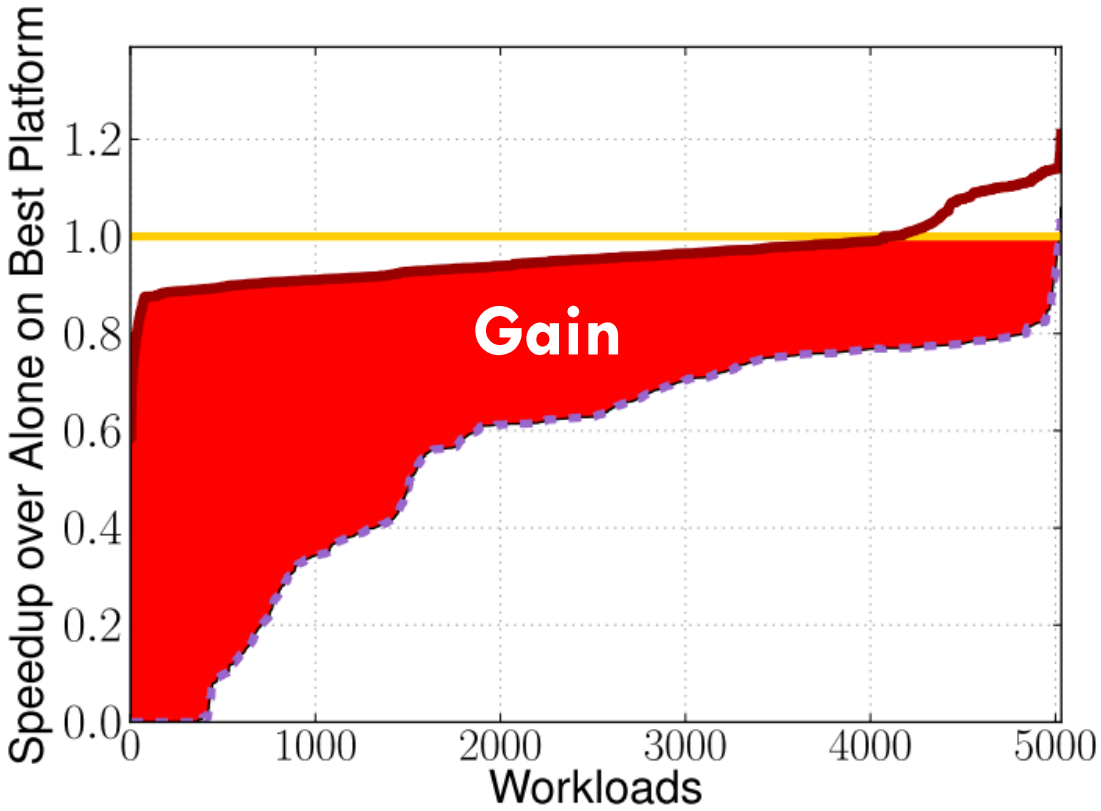
Motivation

- Interference is the penalty of resource efficiency
  - Co-scheduled workloads contend in shared resources
  - Interference can span the core, cache/memory, net, storage



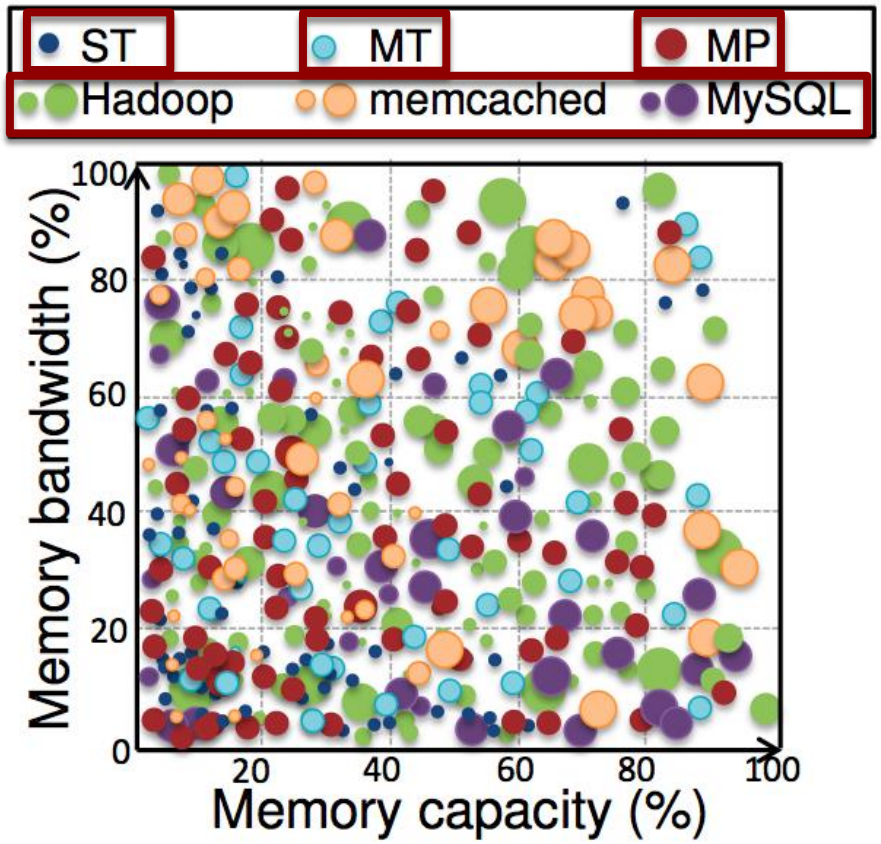
Motivation

- Interference is the penalty of resource efficiency
  - Co-scheduled workloads contend in shared resources
  - Interference can span the core, cache/memory, net, storage



Motivation

- Exhaustive characterization of interference sensitivity against all possible co-scheduled workloads → infeasible



Motivation

- Instead profile against a set of carefully-designed benchmarks
  - Common reference point for all applications

- Requirements for interference benchmark suite:
  - Consistent behavior → predictable resource pressure
  - Tunable pressure in the corresponding resource
  - Span multiple shared resources (one per benchmark)
  - Not-overlapping behavior across benchmarks

Outline

- Motivation
- iBench Workloads
- Validation
- Use Cases

iBench Overview

- iBench consists of 15 benchmarks
  - Each targets a different system resource
- First design principle: benchmark intensity is a tunable parameter
- Second design principle: benchmark impact increases almost proportionately with intensity
- Third design principle: each benchmark only (mostly) stresses its target resource (no overlapping effects)

iBench Workloads

- Memory capacity/bandwidth [1-2]
- Cache:
  - L1 i-cache/d-cache [3-4]
  - *L2 capacity/bandwidth [3'-4']*
  - LLC capacity/bandwidth [5-6]
- CPU:
  - Integer [7]
  - Floating Point [8]
  - Prefetchers [9]
  - TLBs [10]
  - *Vector [11]*
  - Interconnection network [12]
- Network bandwidth [13]
- Storage capacity/bandwidth [14-15]

Memory Capacity

- Progressively increase memory footprint (low memory bandwidth usage)
- Random (or strided) access pattern (using a low-overhead random generator function)
- Uses single static assignment (SSA) to increase ILP in memory accesses
- Fraction of time in idle state depends on intensity levels → decreases as intensity increases

```
// for intensity level x
while (coverage < x%) {
    // SSA: to increase ILP
    access[0]  += data[r] << 1;
    access[1]  += data[r] << 1;
    ...
    access[30] += data[r] << 1;
    access[31] += data[r] << 1;
    // idle for tx = f(x)
    wait(tx);
}
```

Memory Bandwidth

- Progressively increases used memory bandwidth (low memory capacity usage)
- Serial (streaming) memory access pattern
- Accesses happen in a small fraction of the address space (> LLC)
- Fraction of time in idle state depends on intensity levels → decreases as intensity increases

```
// for intensity level x
for (int cnt = 0; cnt < access_cnt; cnt++) {
    access[cnt] = data[cnt]*data[cnt+4];
    // idle for tx = f(x)
    wait(tx);
}
```

Processor benchmarks

- CPU (Int/FP/vector):
  - Progressively increase CPU utilization → launch instructions at increasing rates
    - For integer, floating point or vector (of applicable) operations
- Caches:
  - L1 i/d-cache: sweep through increasing fractions of the L1 capacity
  - L2/L3 capacity: random accesses that occupy increasing fractions of the capacity of the cache (adapt to specific structure, number of ways, etc. to guarantee proportionality of benchmark effect with intensity)
  - L2/L3 bandwidth: streaming accesses that require increasing fractions of the cache bandwidth

I/O benchmarks

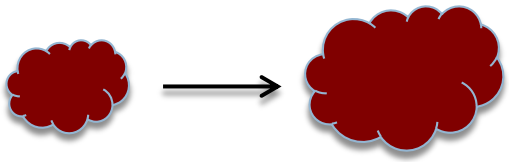
- Network bandwidth:
  - Only relevant for the characterization of workloads with network activity (e.g., MapReduce, memcached)
  - Launches network requests of increasing sizes and at increasing rates until saturating the link
  - The fanout to receiving hosts is a tunable parameter
- Storage bandwidth:
  - Streaming/serial disk accesses across the system's hard drives (only cover subsets of the address space to limit capacity usage)
  - Accesses increase as the intensity of the benchmark increases → until reaching the sustained disk bandwidth of the system

Outline

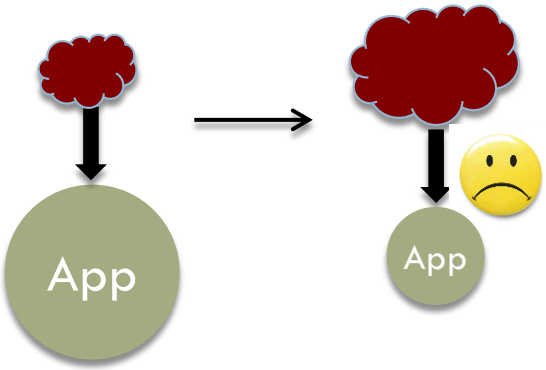
- Motivation
- iBench Workloads
- Validation
- Use Cases

Validation

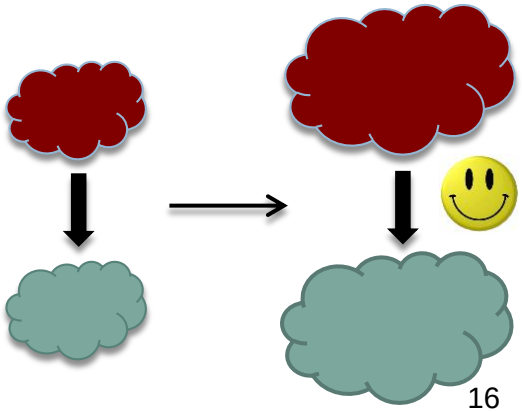
1. Individual iBench workloads behavior: create progressively more pressure in a resource



2. Impact of iBench workloads to other applications: cause progressively higher performance degradation

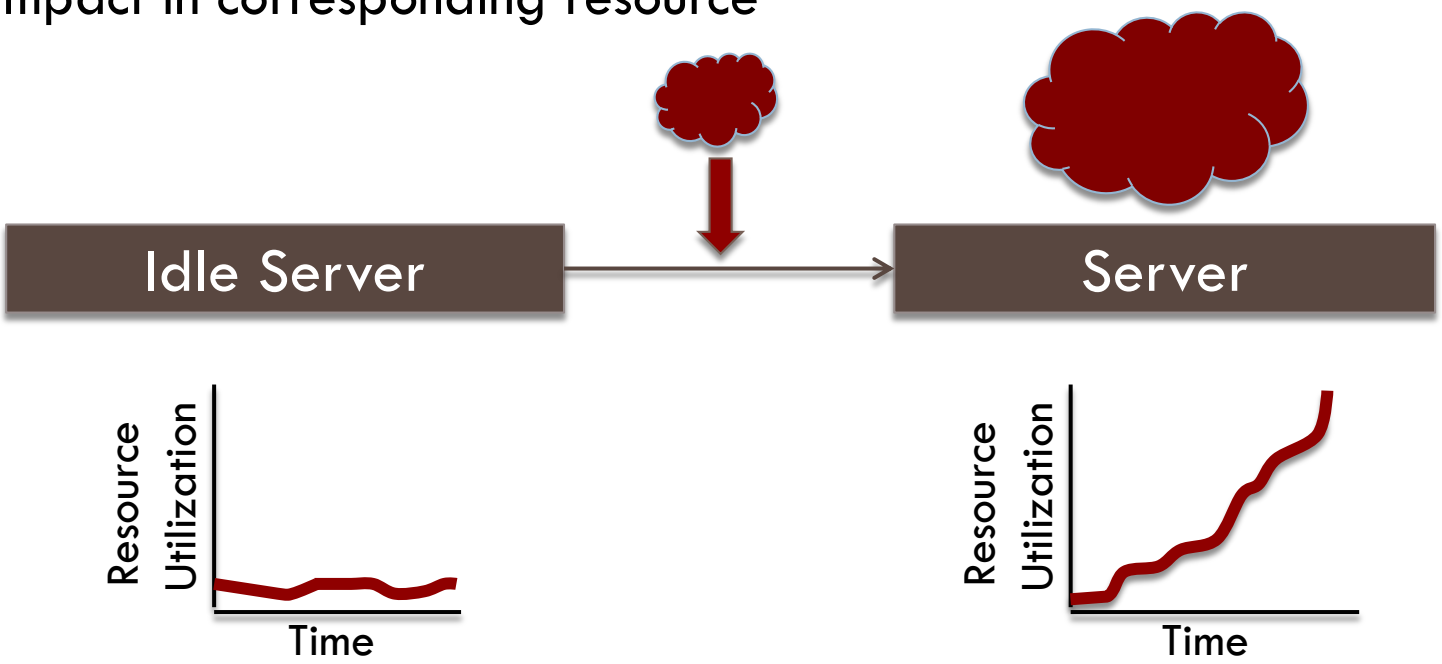


3. Impact of iBench workloads on each other: the pressure of different workloads should not overlap



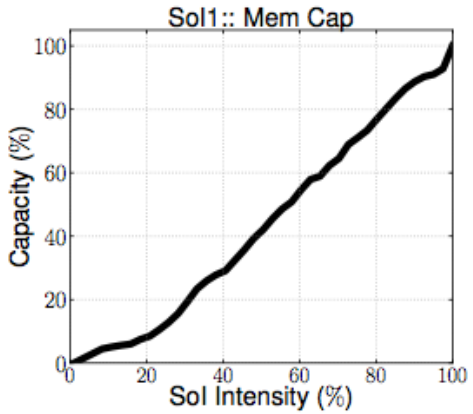
Validation: Individual benchmarks

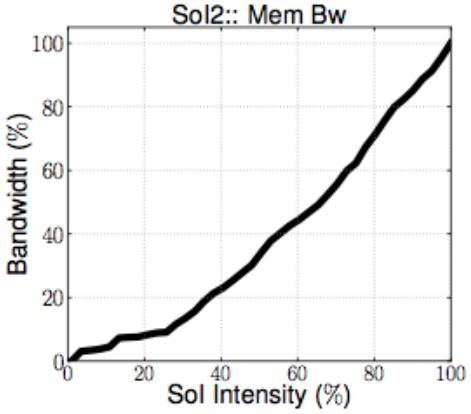
- Increasing intensity of each benchmark → proportionately increasing impact in corresponding resource

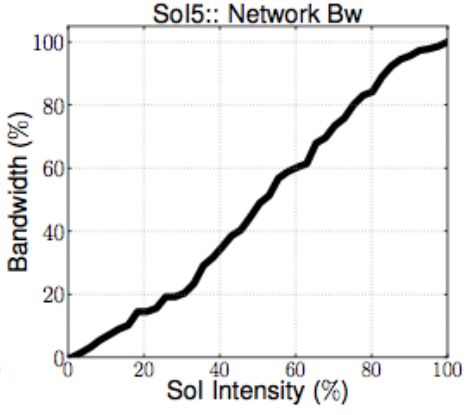


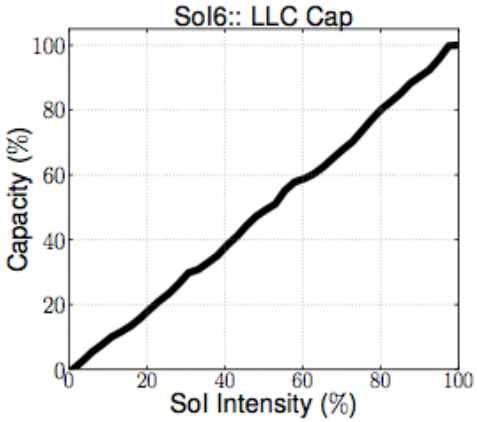
Validation: Individual benchmarks

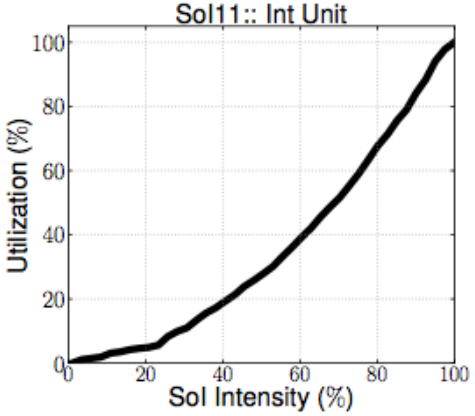
- Increasing intensity of each benchmark → proportionately increasing impact in corresponding resource

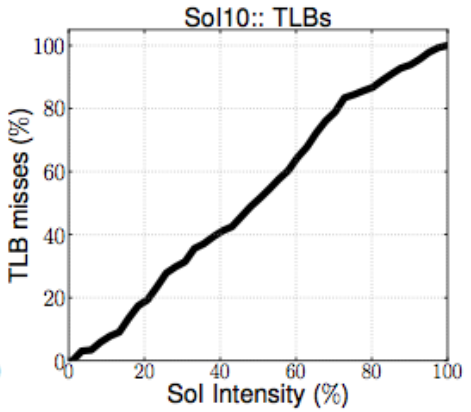






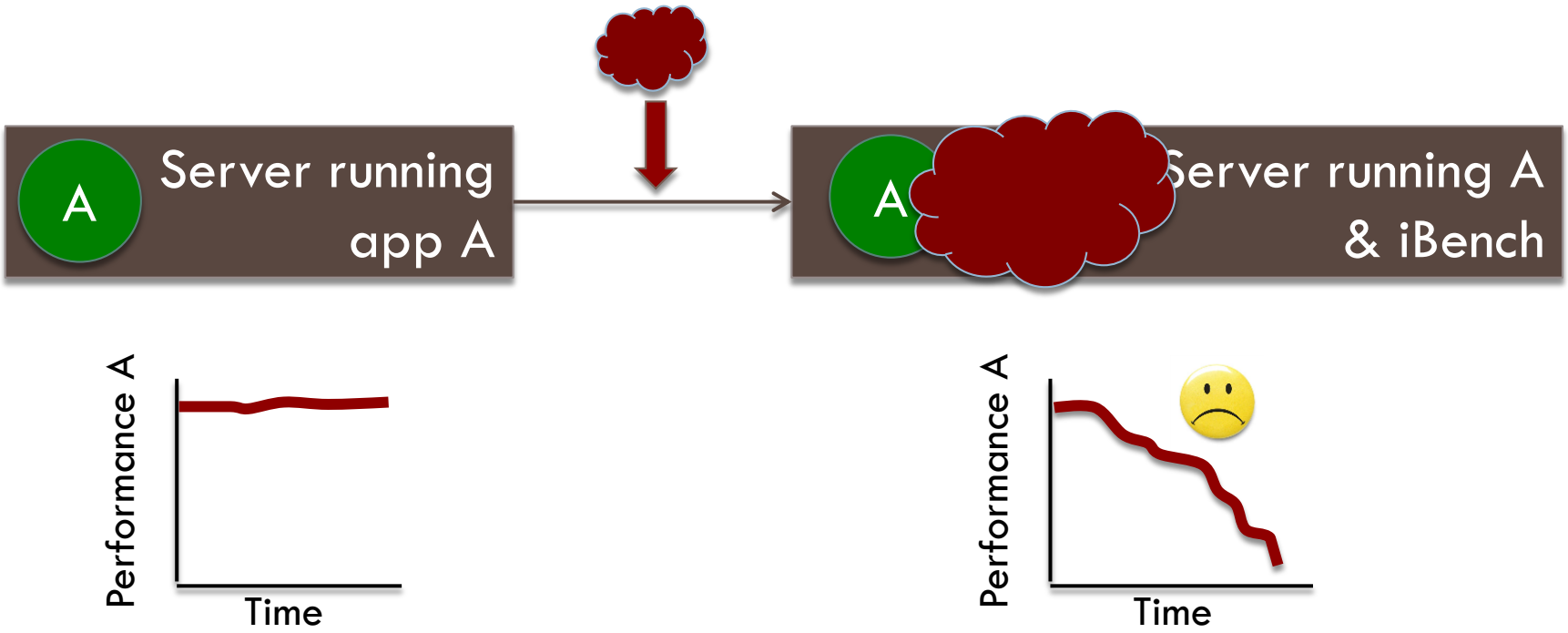






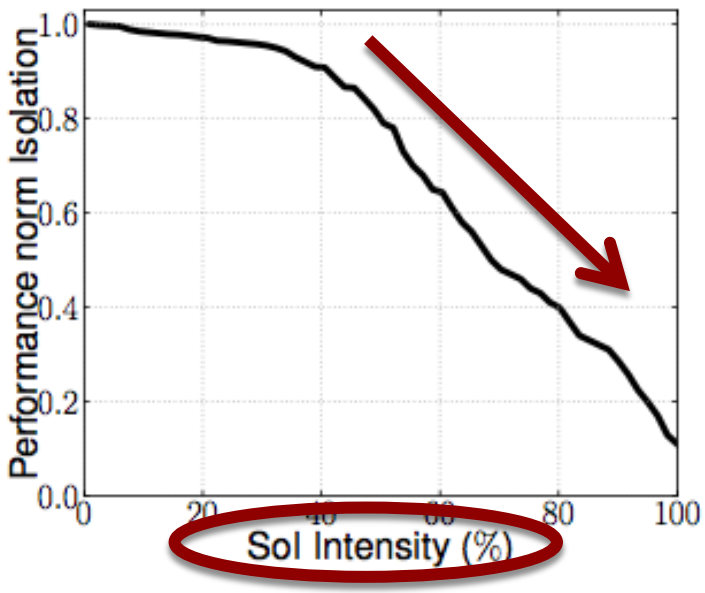
Validation: Impact on Performance

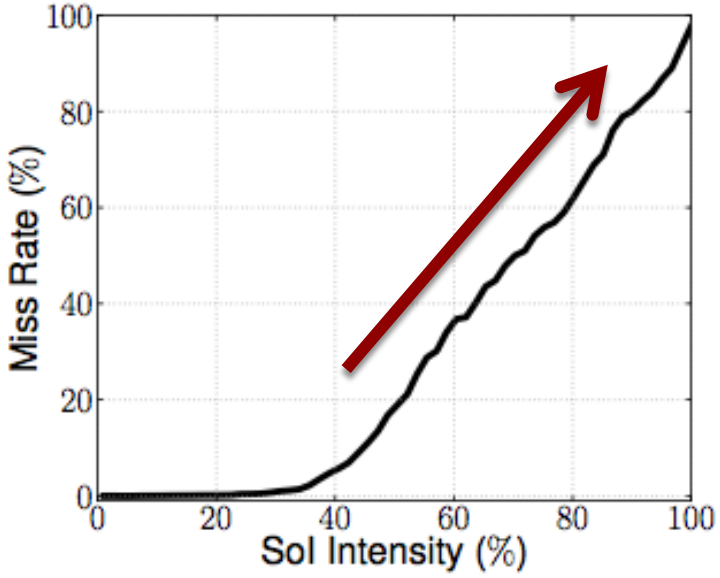
- Inject a benchmark in an active workload → tune up intensity → record increasing degradation in performance



Validation: Impact on Performance

- mcf from SPECCPU2006 (memory intensive) + LLC capacity

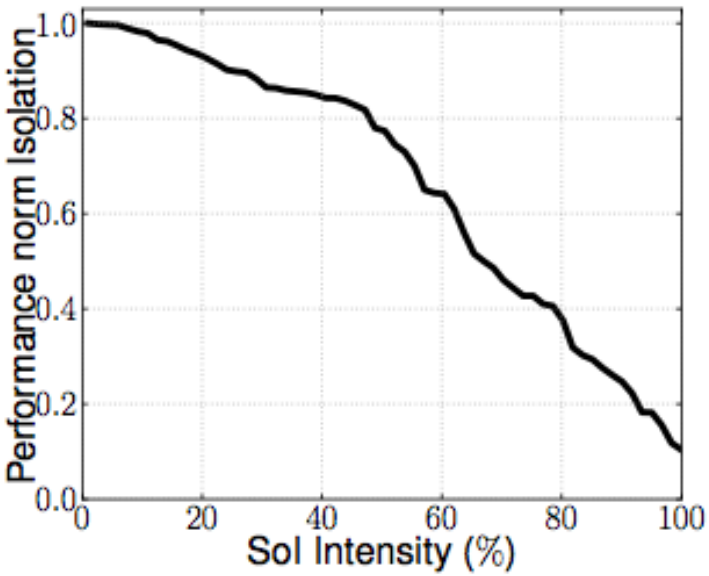


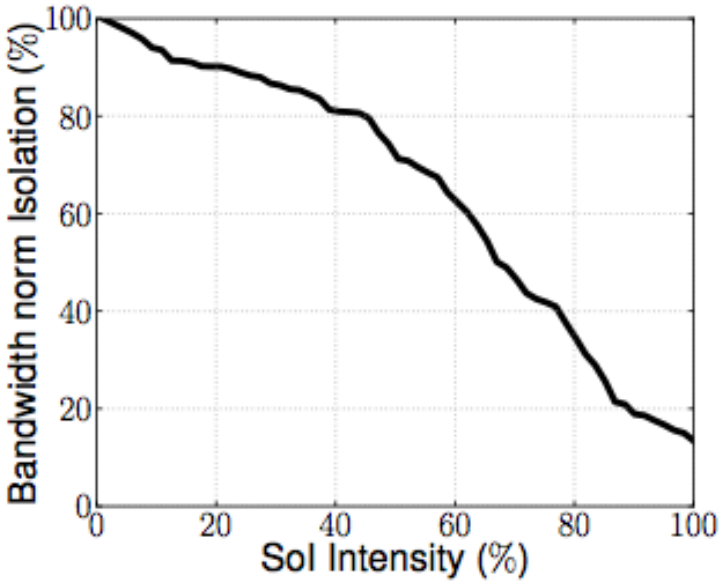


- Performance degrades as intensity of LLC capacity benchmark increases

Validation: Impact on Performance

- memcached (memory + network intensive) + network bandwidth

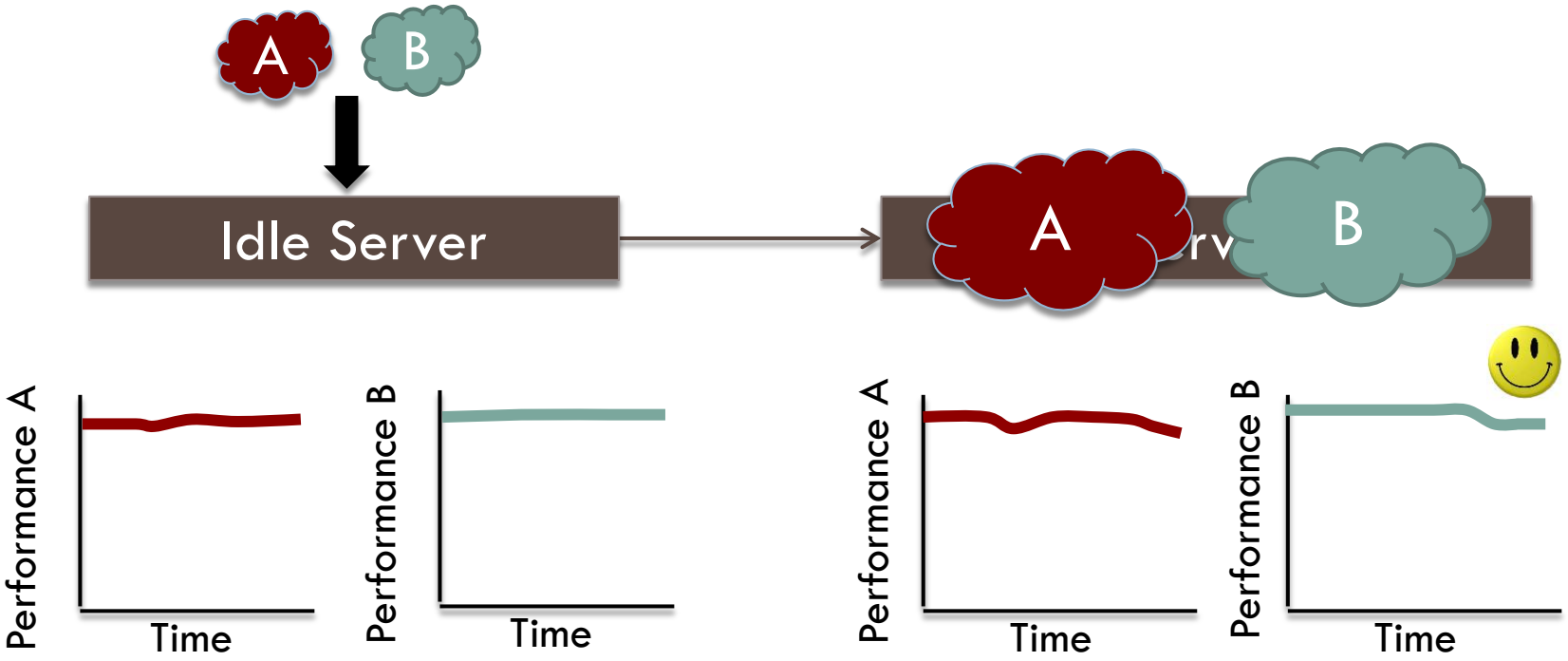




- QPS drops as intensity of network bw benchmark increases

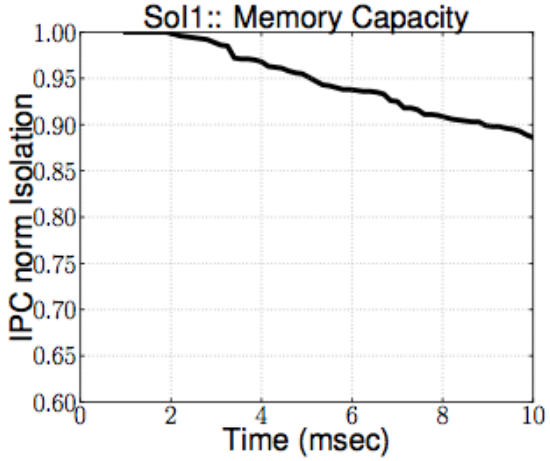
Validation: Cross-benchmark Impact

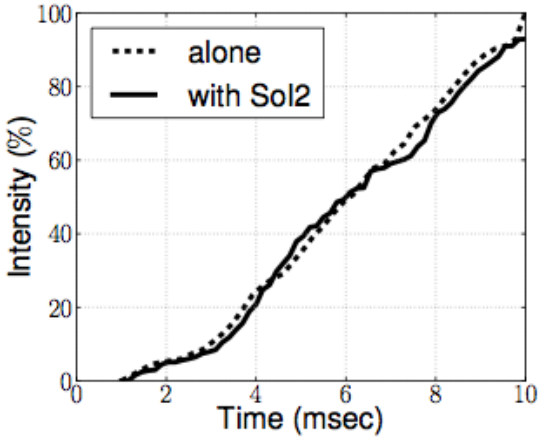
- Co-schedule two iBench workloads on the same machine → tune up intensity → minimal impact on each other

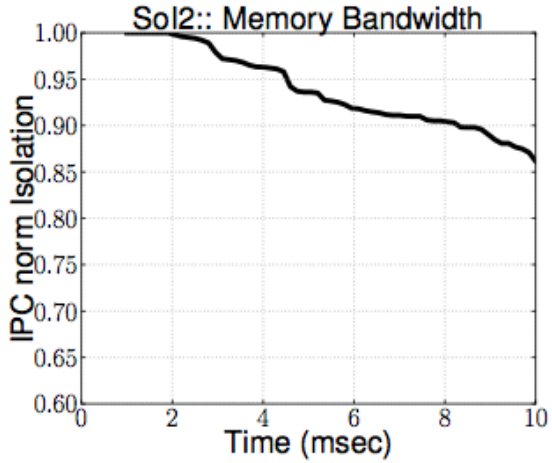


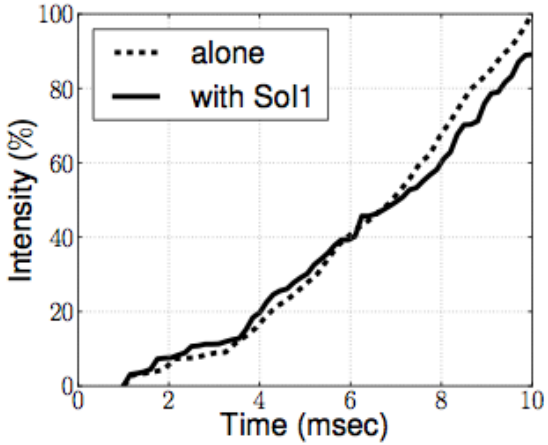
Validation: Cross-benchmark impact

- Co-schedule the memory capacity and memory bandwidth benchmarks









Outline

- Motivation
- iBench Workloads
- Validation
- Use Cases

Use Cases

- Interference-aware datacenter scheduling
- Datacenter server provisioning
- Resource-efficient application design
- Interference-aware heterogeneous CMP scheduling

Use Cases

- Interference-aware datacenter scheduling
- Datacenter server provisioning
- Resource-efficient application design
- Interference-aware heterogeneous CMP scheduling

Interference-aware DC Scheduling

- Cloud provider scenario:
  - Unknown workloads are submitted in the system
  - Cluster scheduler should determine which applications can be scheduled on the same machine
  - Scheduling decisions should be:
    - Fast → minimize scheduling overheads
    - QoS-aware → minimize cross-application interference
    - Resource-efficient → co-schedule as many applications as possible to increase utilization

- Objective: preserve per-application performance & increase utilization

DC Scheduling Steps

1. Applications are admitted to the system →
   - Profile against iBench workloads
   - Determine the contended resources they are sensitive to

2. Scheduler finds the servers that minimize the:

$$||i_t - i_c||_{L1}$$

3. If multiple, selects the least-loaded one (can add placement, platform configuration, etc. considerations)

Methodology

- Workloads:
  - Single-threaded: SPEC CPU2006
  - Multi-threaded: PARSEC, SPLASH-2, BioParallel, Minebench
  - Multiprogrammed: 4-app mixes of SPEC CPU2006 workloads
  - I/O-bound: Hadoop + data mining (Matlab)
  - Latency-critical: memcached

} 214 apps

- Systems:
  - 40 servers, 10 server configurations (Xeons, Atoms, etc.)
- Scenarios:
  - Cloud provider: 200 applications submitted with 1 sec inter-arrival times
  - Hadoop as the primary workload + batch best-effort apps
  - Memcached as the primary workload + batch best-effort apps

Methodology

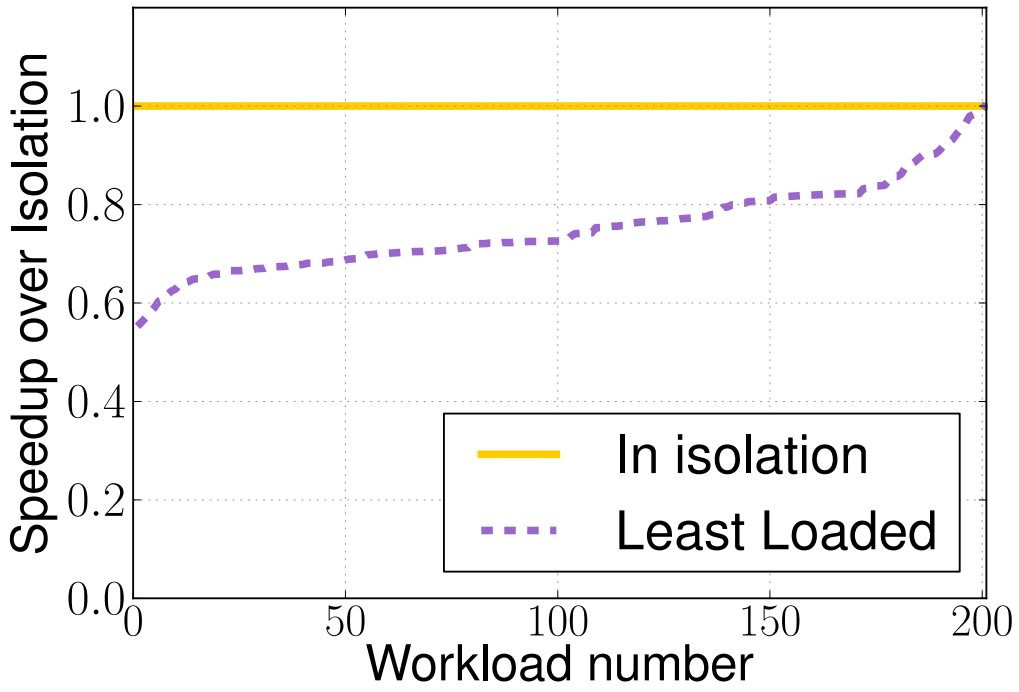
- Workloads:
  - Single-threaded: SPEC CPU2006
  - Multi-threaded: PARSEC, SPLASH-2, BioParallel, Minebench
  - Multiprogrammed: 4-app mixes of SPEC CPU2006 workloads
  - I/O-bound: Hadoop + data mining (Matlab)
  - Latency-critical: memcached

} 214 apps

- Systems:
  - 40 servers, 10 server configurations (Xeons, Atoms, etc.)
- Scenarios:
  - Cloud provider: 200 applications submitted with 1 sec inter-arrival times
  - Hadoop as the primary workload + batch best-effort apps
  - Memcached as the primary workload + batch best-effort apps

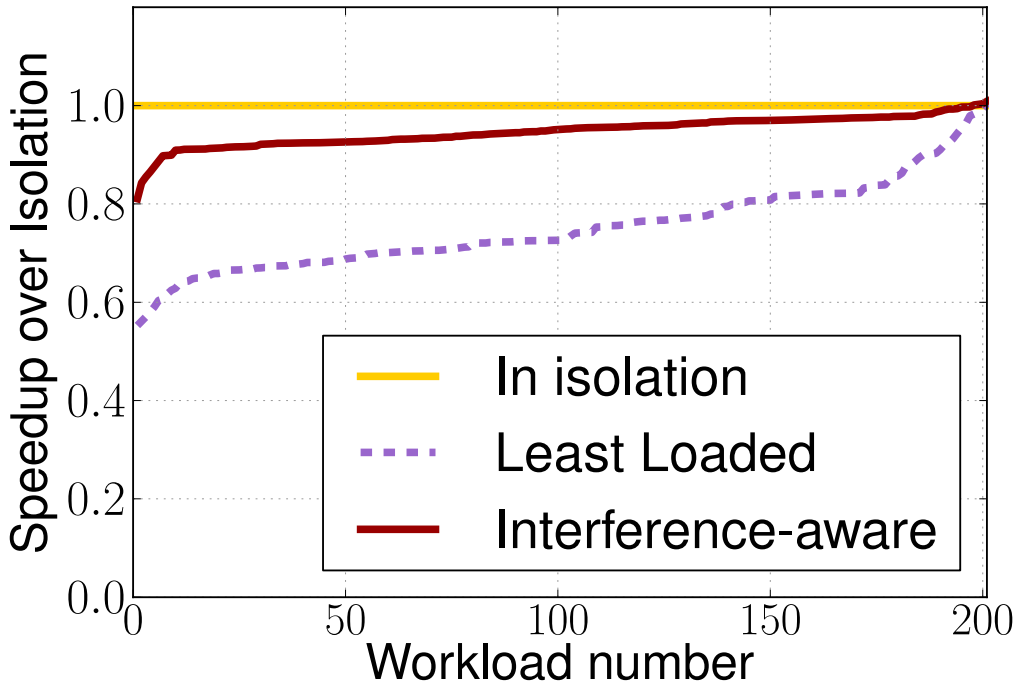
Cloud Provider: Performance

- Least-loaded (interference-oblivious scheduler) vs. interference-aware scheduling with iBench



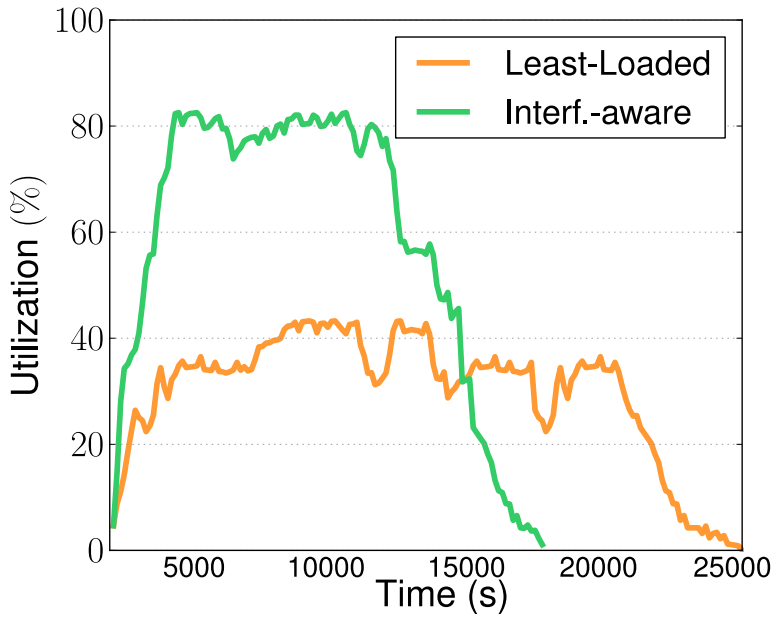
Cloud Provider: Performance

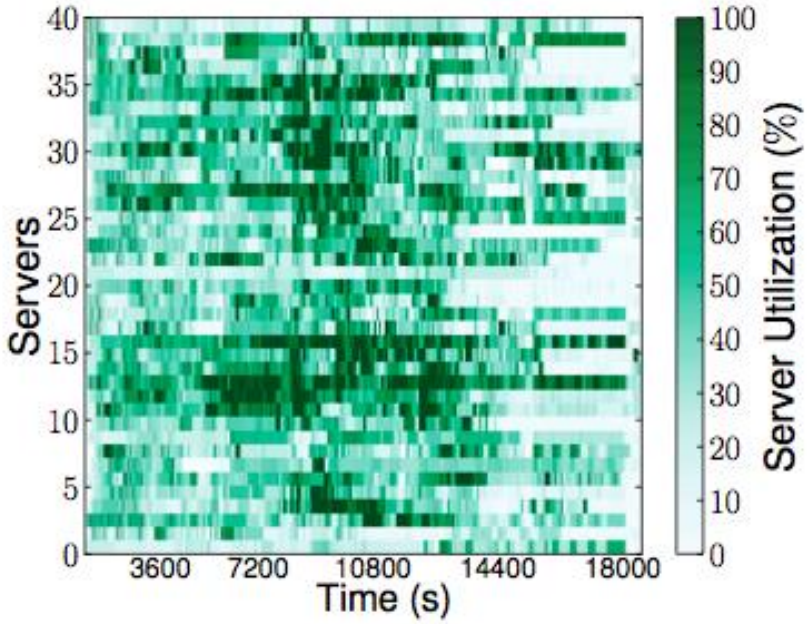
- Least-loaded (interference-oblivious scheduler) vs. interference-aware scheduling with iBench



- Performance improves by 16% on average (up to 28%).
- 60% of apps preserve their QoS – 5% with the least-loaded scheduler

Cloud Provider: Utilization





- Utilization improves by 38% compared to least-loaded
- The scenario completes 28% faster → higher resource-efficiency
- Individual servers operate at higher utilization without being oversubscribed

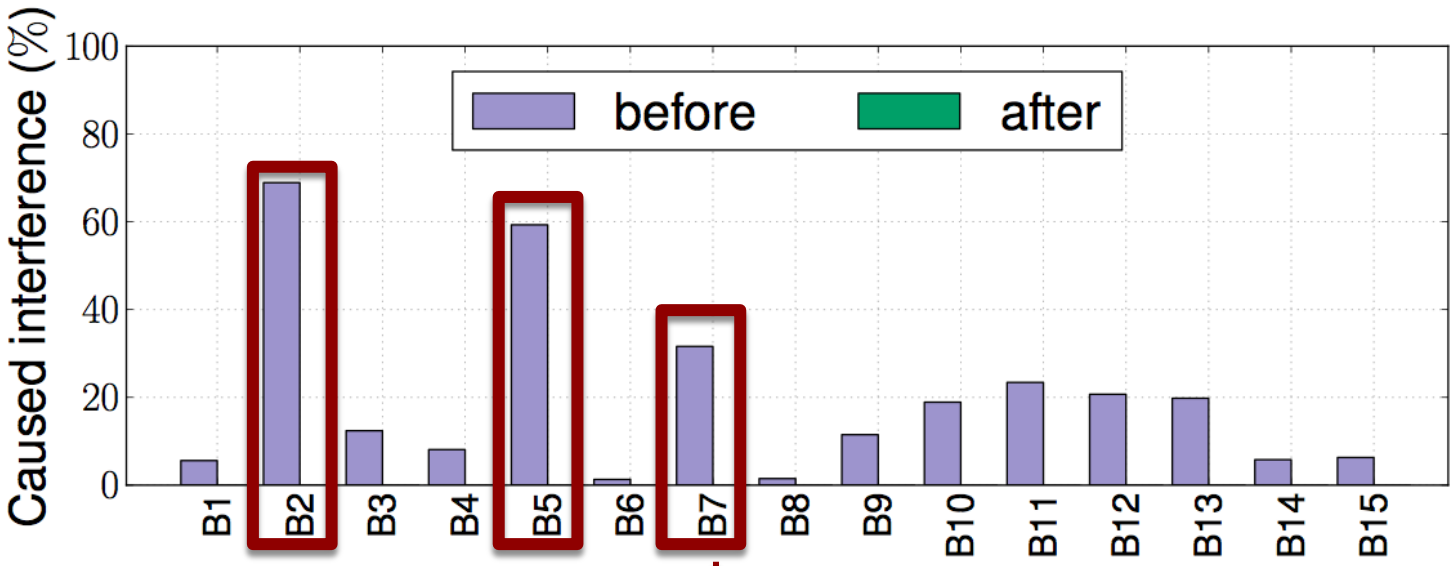
DC Server Provisioning

- Default server configuration not necessarily optimal for each DC workload (custom servers, Open Compute, etc.)
- Study the resources each workload stresses & the resources it is sensitive to using iBench → provision accordingly the machines that service that workload
- Offline characterization, but can also apply online to capture changes in application behavior

DC Server Provisioning

- memcached instance:
  - 1000 clients
  - QoS target 40,000 QPS
  - latency constraint of 200usec
- Server: Xeon E5345 (4 cores, 8MB LLC, 16GB RAM), 1GB NIC
- Characterize the interference memcached puts on each resource captured by iBench

DC Server Provisioning



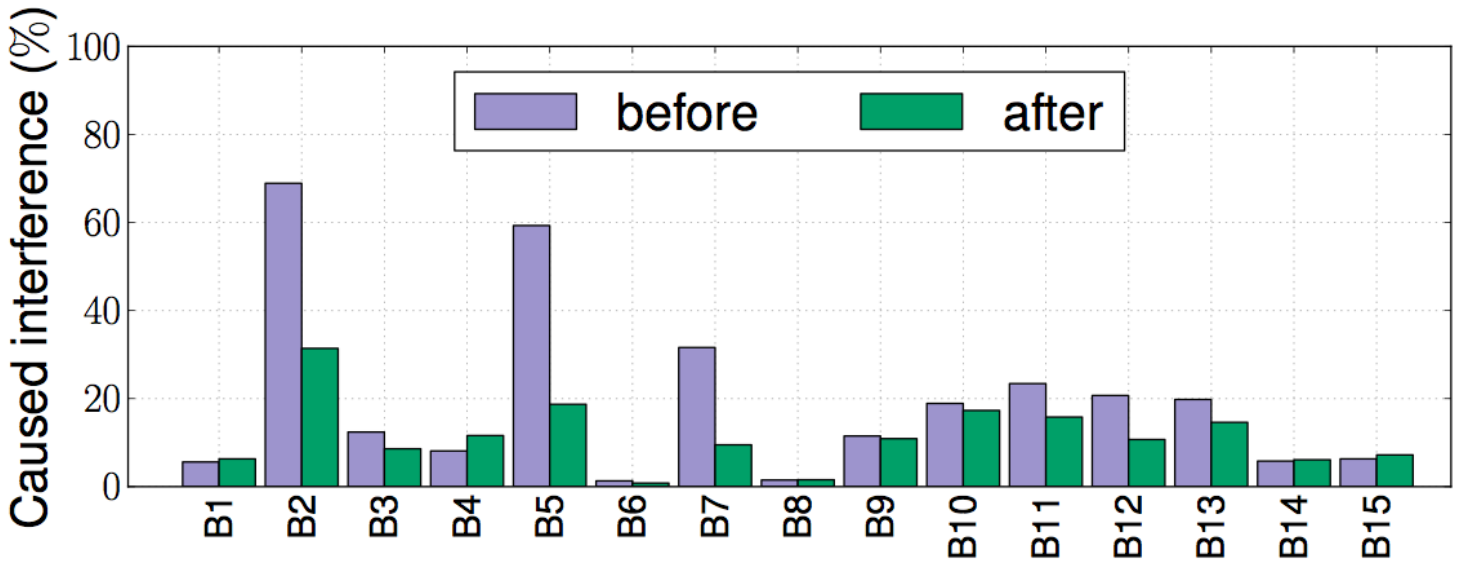
memory bw **LLC bw**

network bw

Switch to triple memory channel & 24GB RAM

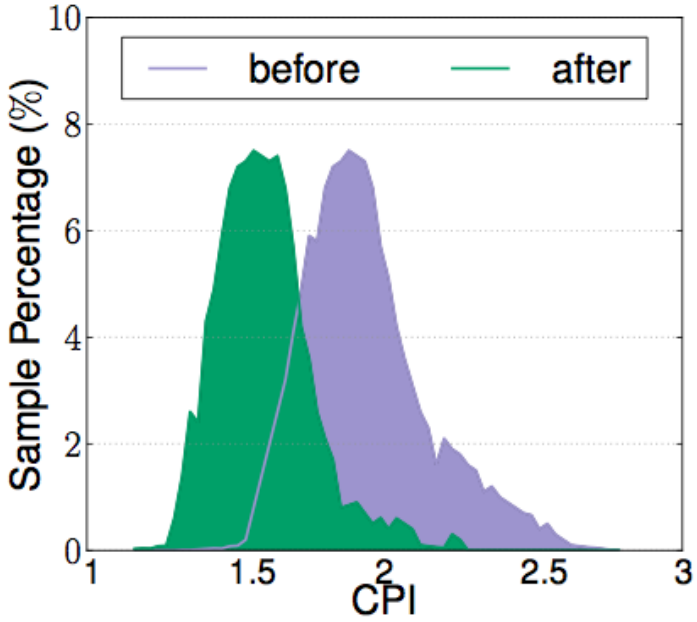
Switch to 10 GB NIC

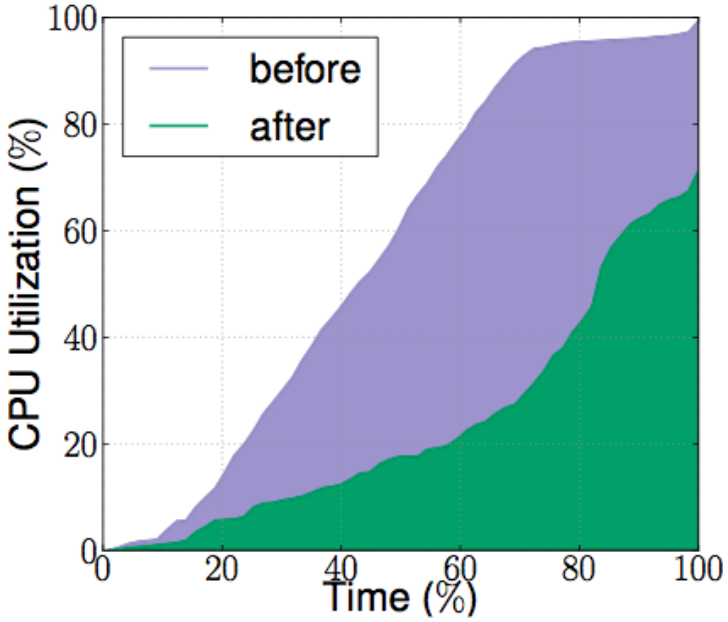
DC Server Provisioning



- Memory/cache contention is reduced
- Network contention is reduced
- Core contention starts becoming the bottleneck

DC Server Provisioning





- Change in interference profile reflects in performance & resource efficiency improvement
- IPC increases by 22% on average
- CPU throttling due to memory stalls reduces (utilization decreases by 41% on average)

Other Use Cases

- Resource-efficient application design
  - Reduce execution time by 35%
  - Reduce memory footprint by 44%

- Interference-aware heterogeneous CMP scheduling
  - Map app to specific core → minimize interference across co-scheduled workloads
  - Per-app performance improves by 36% compared to random app-to-core mapping
  - Memory stalls decrease by 18%
  - Network traffic decreases by 11%

Conclusions

- iBench is a set of benchmarks (contentious kernels) that put pressure on one of many shared resources
- It helps quantify the sensitivity workloads have to interference
- Each benchmark targets a specific resource → tunable intensity
- Applicable to both DC and conventional system studies

Questions??

Thank you

Questions: cdel@stanford.edu

Questions??

Thank you

Source code available soon at:
ibench.stanford.edu