Virtual Platforms: Hypervisor-level Support for Increased Consolidation

Priyanka Tembey
Ada Gavrilovska
Karsten Schwan

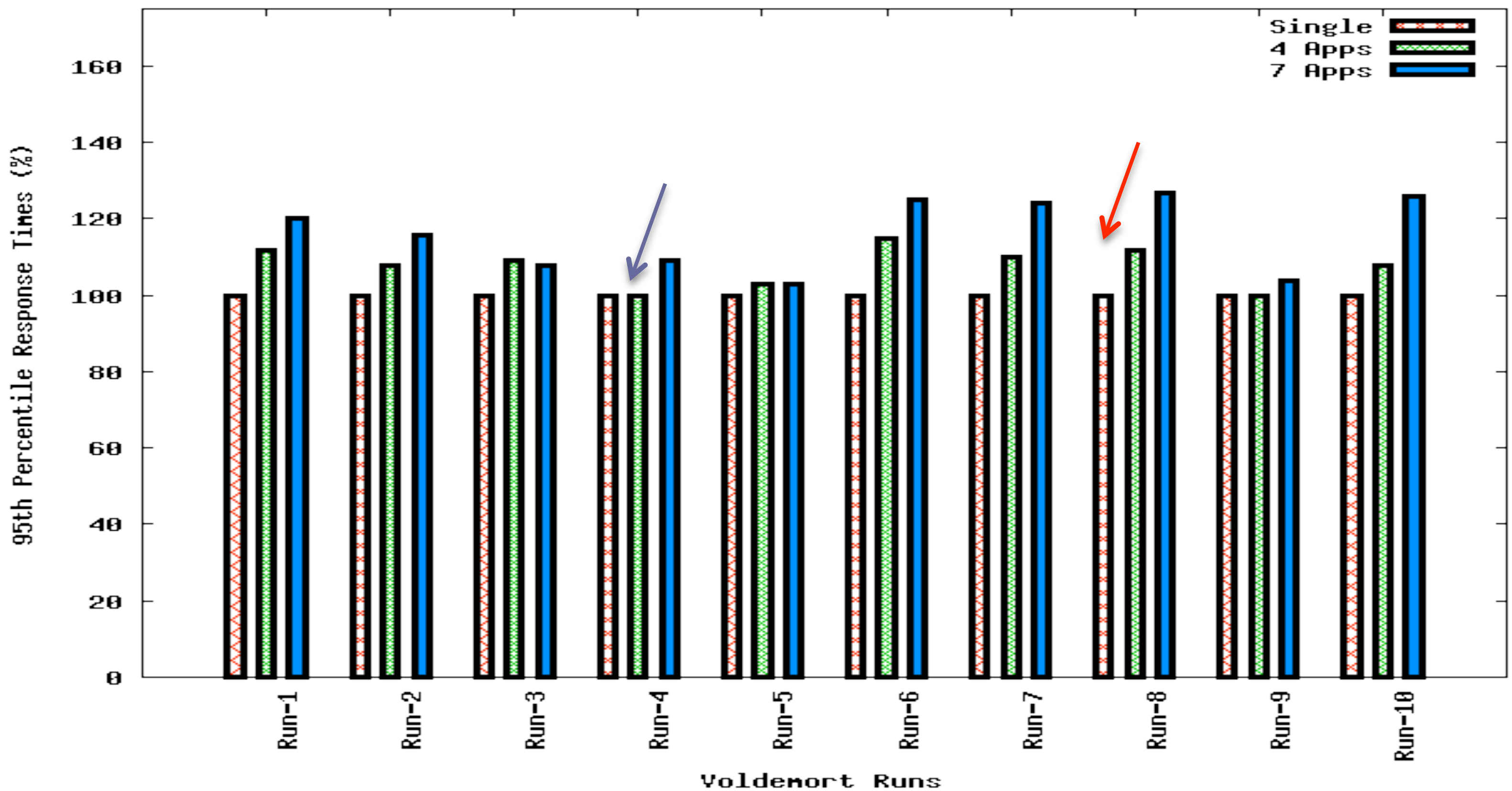
http://www.istc-cc.cmu.edu/

Intel Science & Technology Center for Cloud Computing

Application Consolidation

- Increasing core-counts + system virtualization => consolidation
- Hypervisors allocate resource (CPU, memory…) shares to application-VMs
  - Benefits server resource utilization
  - Limit and control sharing effects to maintain desired and predictable performance

Potential Range of Sharing Effects: Voldemort



- 95th percentile response times of Voldemort show unpredictable variation across runs. Worst-case degradation is 23% (Run-8).
- Some configurations/co-locations better suited. E.g., (Run-5)
- **What hardware/software methods can help make such resource allocations as in Run-5?**

Potential Range of Sharing Effects: Voldemort

- Experimental platform:
  - Hardware: 32 core Westmere Processor with 4 NUMA sockets (8 cores per socket), 32G RAM per NUMA node, 24MB LLC per socket
  - Software: Xen 4.1 + Dom0 running 2.6.32 kernel + Guest VMs running Linux 3.0.2

Potential Range of Sharing Effects: Voldemort

- Applications representative of “Cloud-mix”:
  - Voldemort server + YCSB workload client: Key-value store used at LinkedIn, supports replication and in-memory backend (Multi-VM application)
  - Phoenix Shared Memory MapReduce with Pthreads (HPCA’07)
- Experiment scenarios:
  - Single (Baseline)
  - 4-Apps: Voldemort + 2 Matrix-Mult + 1 WordCount
  - 7-Apps: Voldemort + 3 Matrix-Mult + 3 WordCount
- Methodology: Run applications choosing distinct startup order for each run (different colocations)

Consolidation: Performance Effects

- Hypervisors limited in ability to provide performance isolation
  - Application performance depends on resources beyond CPU, memory, includes shared resources as memory bandwidth, I/O that are *not easily partitioned* using current hardware support
  - Application resource requirements are *elastic*
- Consolidation =>
  - arbitrary *interference* in shared resource shares which may have detrimental performance implication
  - interference effects are *hardware- and workload-specific* -- different application-resource share mappings may experience entirely different interference effects

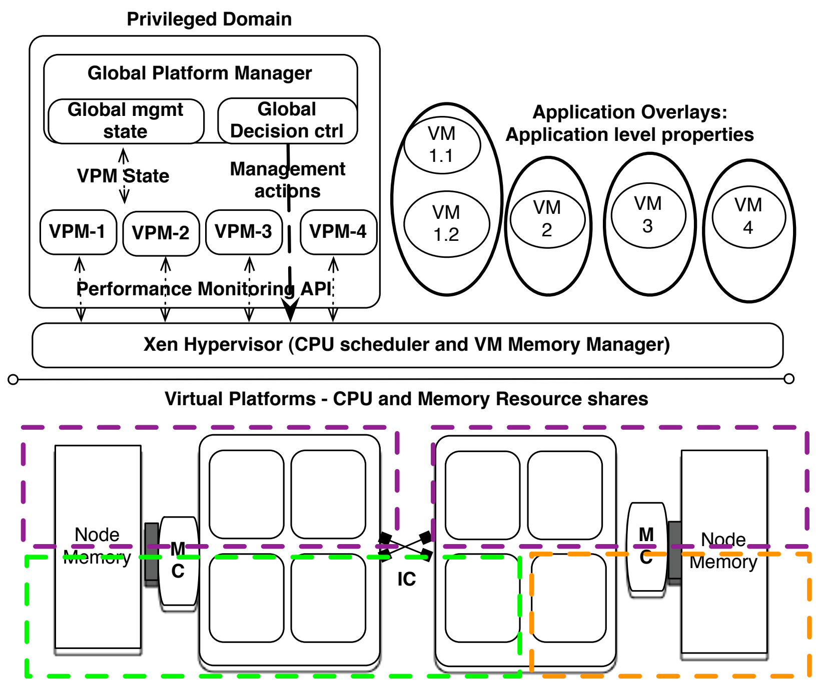
Virtual Platforms

- System-level methods to manage application resources keeping isolation as a first class resource management principal
  - Create, allocate and maintain **Virtual Platforms (VPs) as hypervisor-level resource commitments**
  - **Virtual Platform ==** resources allocated to all VMs representing an application/tenant
  - **Online interference models** for shared resource points (Caches, MC, IC)

- Implementation in Xen Hypervisor evaluated with enterprise/cloud application mixes
  - Less performance variation, improved performance predictability

Virtual Platform (VP) Architecture

- Per-application VP monitor
  - VP monitors track application VMs' usage of the shared resources using black-box techniques (hardware performance counters)
  - How ***intensively*** an application uses CPU, caches and memory bandwidth (MC, IC)?
  - How ***sensitive*** an application is to interference at these shared resource types?



Modeling application resource use intensity

- Why measure how intensively an application uses shared resources?
  - Measure of its "contentiousness" at shared resource points in system
- Approximate resource share use
  - CPU: CPU utilization
  - LLC: Using L2 and L3 miss counters

  (L2-L3)/L2misses per 1000 instructions
  - Memory Bandwidth: L3miss/1000 instructions

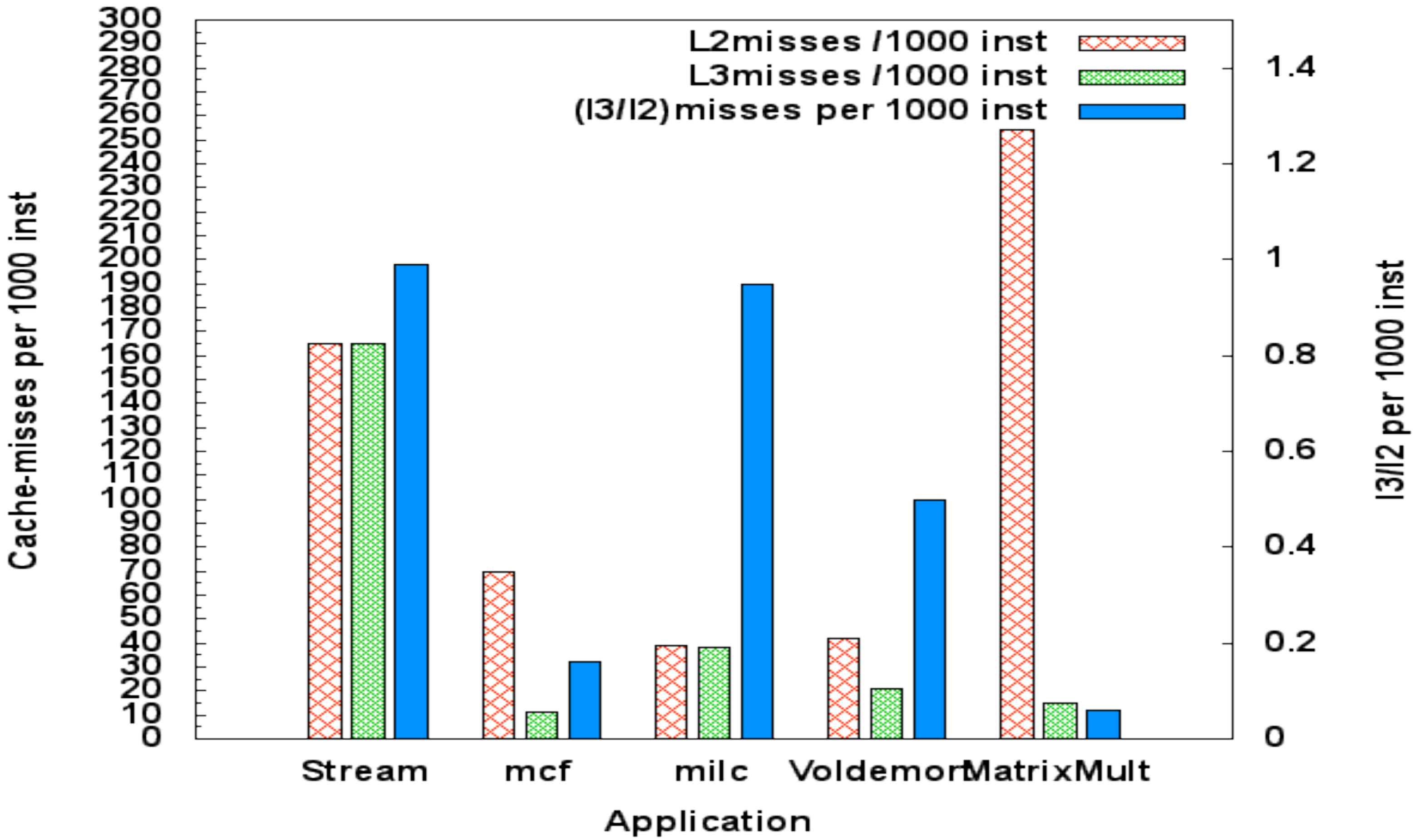
Modeling application resource sensitivity

- Why measure sensitivity?
  - Measure of "hurt" caused to application due to contention at particular resource type
  - E.g., A streaming application may be cache-intensive, not cache-sensitive
- Measuring sensitivity to contention at memory subsystem (Memory Factor (MF)):
  - L3/L2 per 1000 instructions: Fraction of L2misses served by memory
  - Higher MF: Higher sensitivity to memory latency and contention

Application mix characterization

- Memory intensity classes (L3misses/1000inst)
  - <2 (Pugs)
  - >2, <15 (Terriers)
  - >15 (Bulldogs)
- Memory Factor sensitivity classes (L3/L2 per 1000inst)
  - <0.25
  - >0.25 , <0.6
  - >0.6

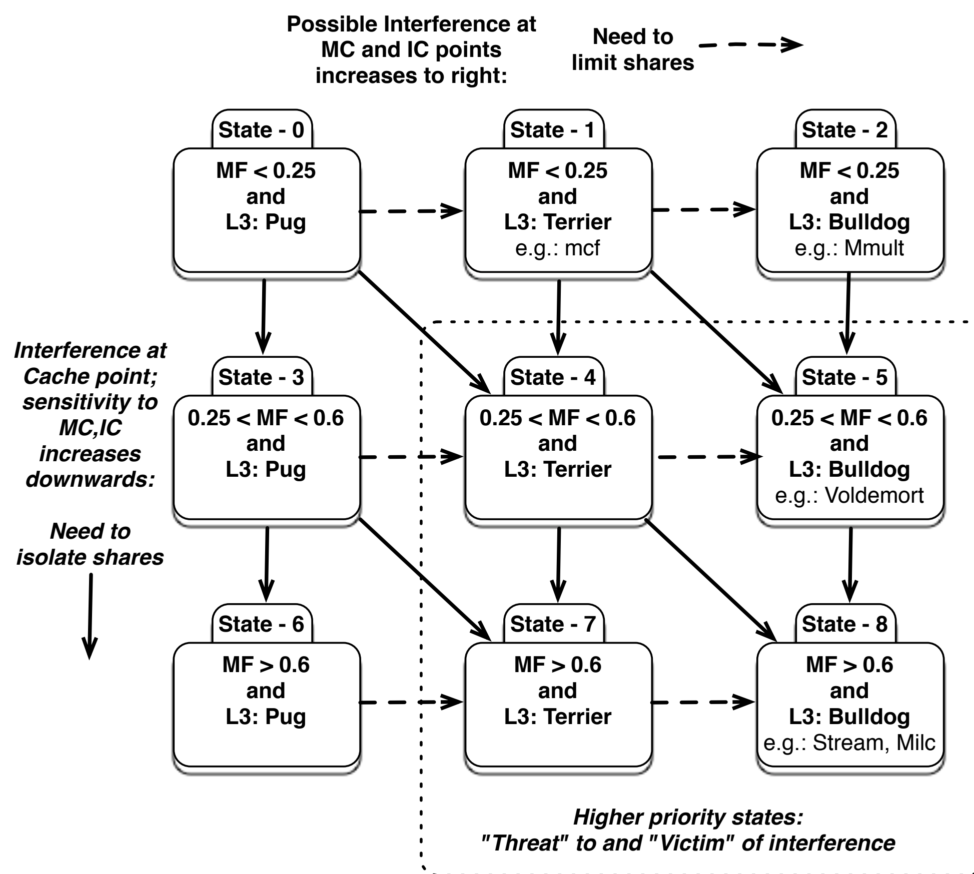
Application mix characterization



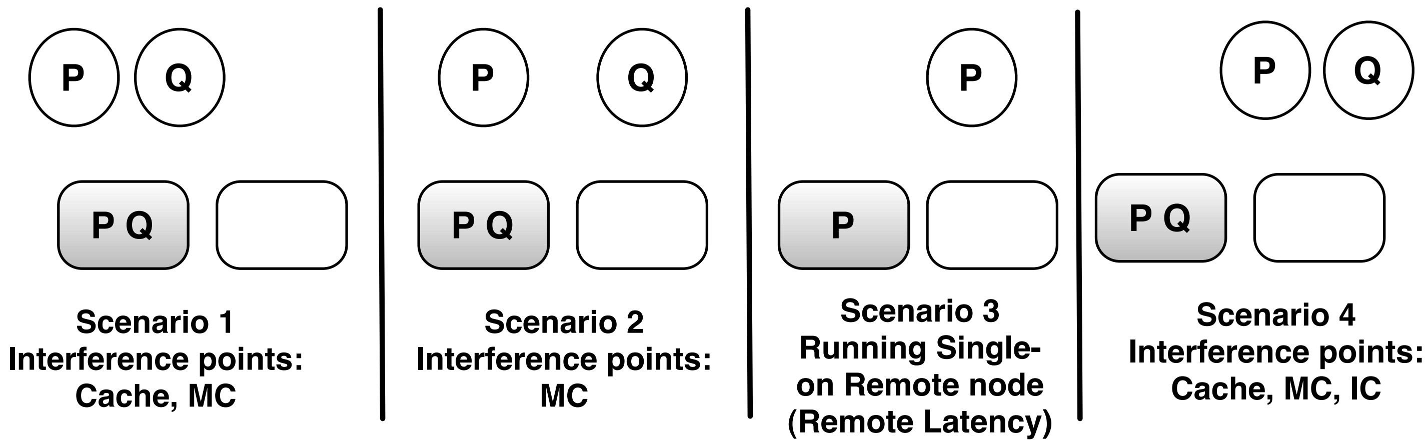
Memory contention/latency sensitive: Stream, Milc, Voldemort
Cache-sensitive: Mcf, MatrixMult, **Memory-intensive:** Stream,Milc

State-Transition model of interference

- VP monitor keeps track of "MF sensitivity and Memory Intensity states" periodically
  - A transition to higher MF sensitivity state: Possible LLC interference,
    - VP monitor alerts Global Platform Manager
  - Transition down: LLC interference
  - Transition right: Higher memory intensity
  - Special case (State-8): Monitored by Global Platform Manager

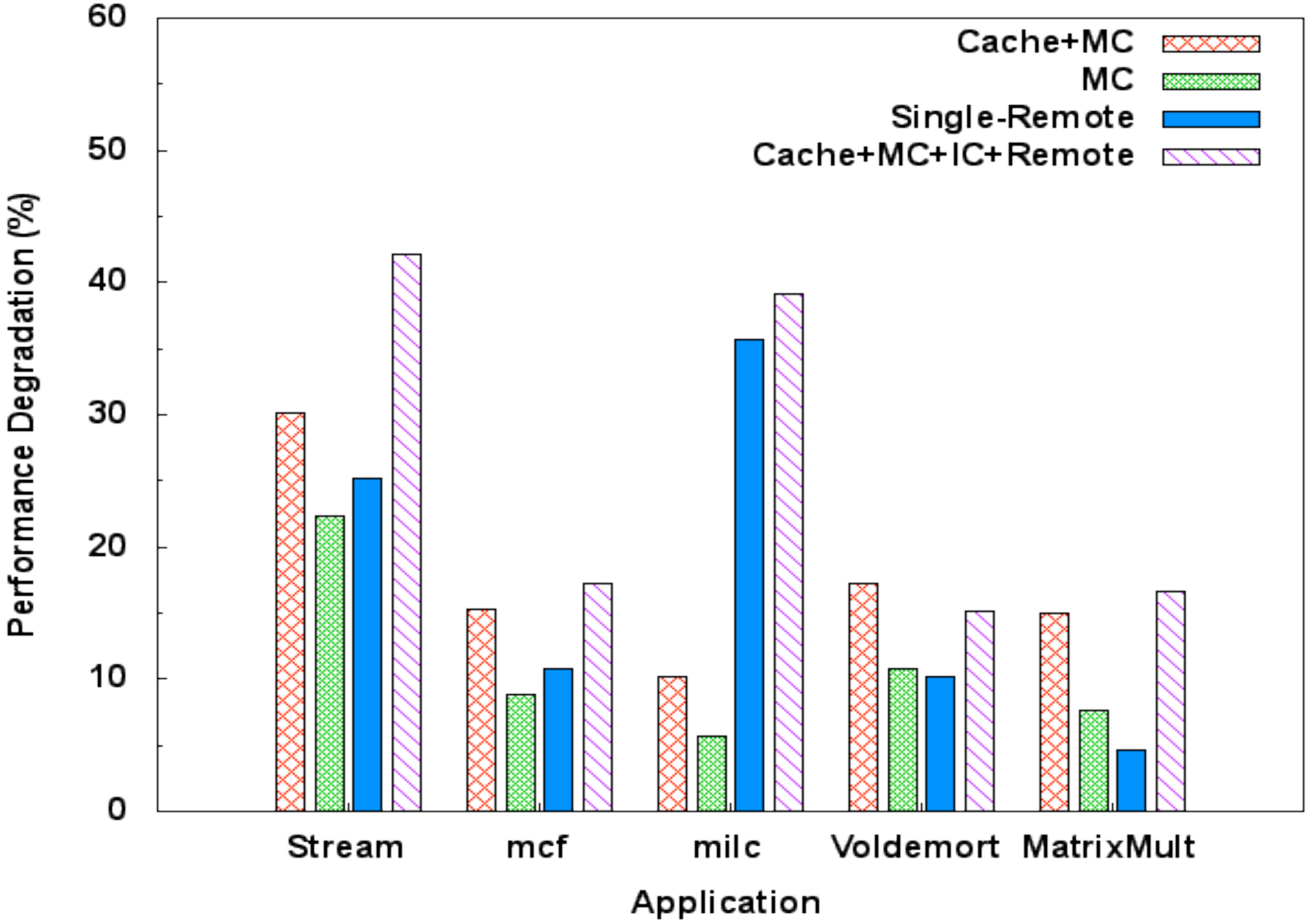


Validating Interference Model

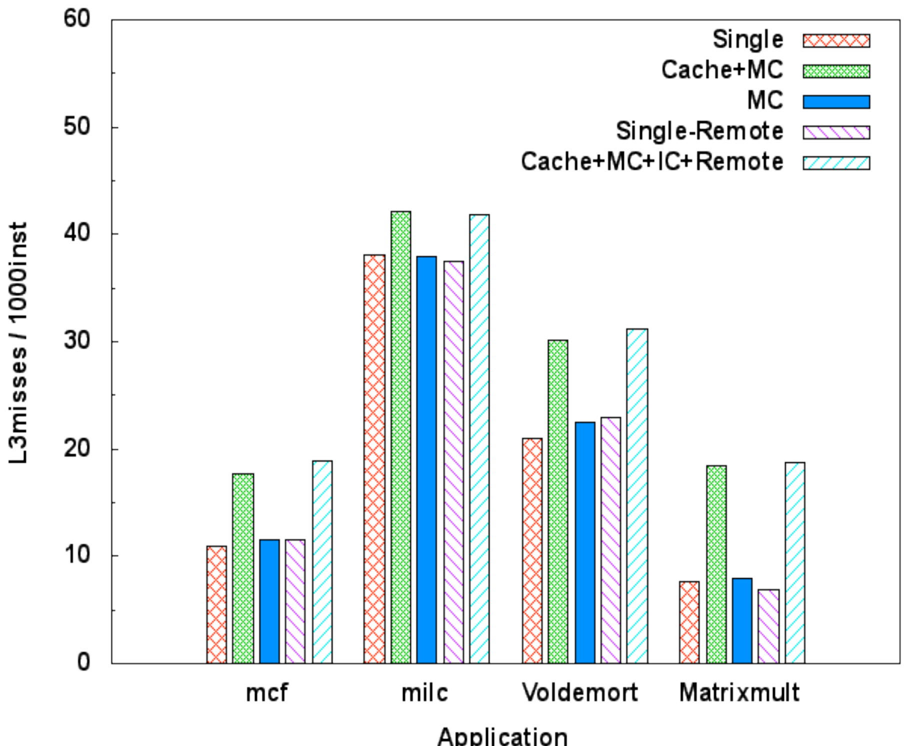


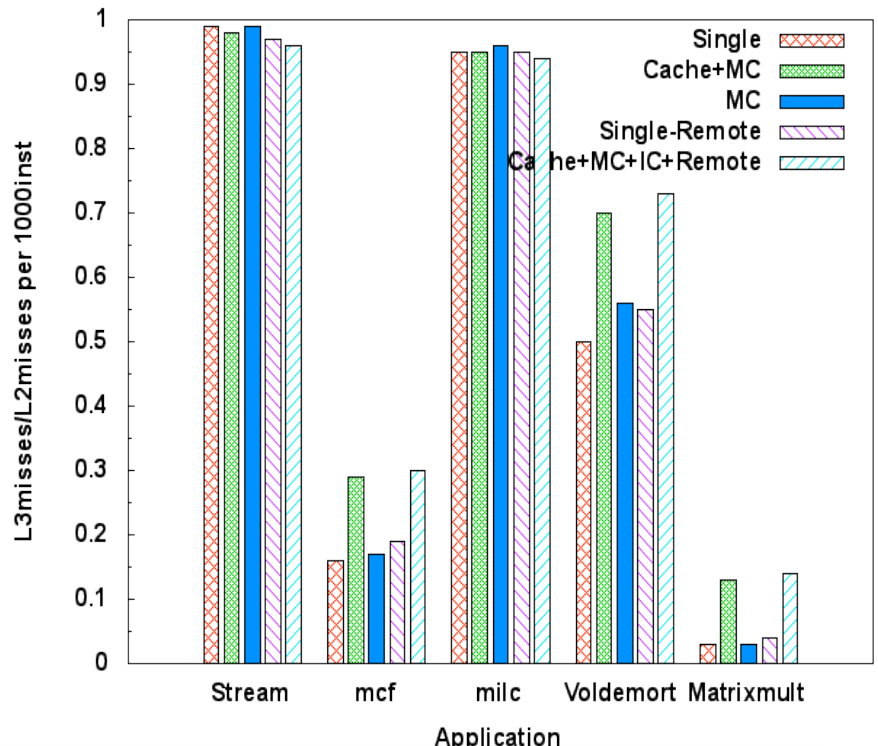
- Need to validate that the state-transition model can detect potential interference at shared resource points
- Experiment with different application colocation scenarios choosing different interference points
  - Isolating each interference point

Performance degradation due to interference



Performance degradation explained by state-transition model

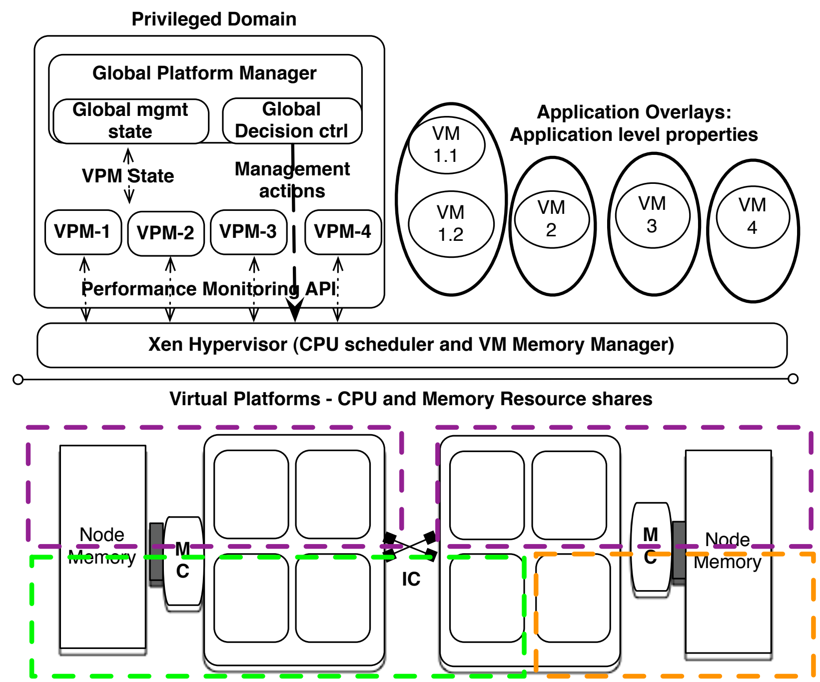




- **Mcf: (Cache)** State-1 to State-5 with increased MF sensitivity and Memory intensity
- **Milc: (Memory)** State-8 application, not much variation at VP-monitor level, needs to be managed at global level
- **Voldemort: (Cache + Memory)** State-5 to State-8

Virtual Platform (VP) Architecture

- Global Platform Manager
  - Creates and allocates initial CPU, Memory resource shares
  - Creates VP monitor per Virtual Platform
  - Topology-awareness of VP to Platform resources (colocated VPs knowledge w.r.t LLC, MC, IC)
  - Invoked by VP monitor to *mitigate interference*
  - Uses software methods to improve isolation



Mitigating interference

- Mitigate interference: maintain resource shares by reducing congestion at interference point
- Global Platform Manager uses:
  - CPU caps: Indirect control of Memory bandwidth use for highly memory-intensive applications
  - VCPU migration amongst NUMA nodes
  - VM memory ballooning across NUMA nodes
- Use current application state knowledge to choose "better" mitigation action
  - E.g., Use VCPU migration + Memory ballooning for MF>0.6

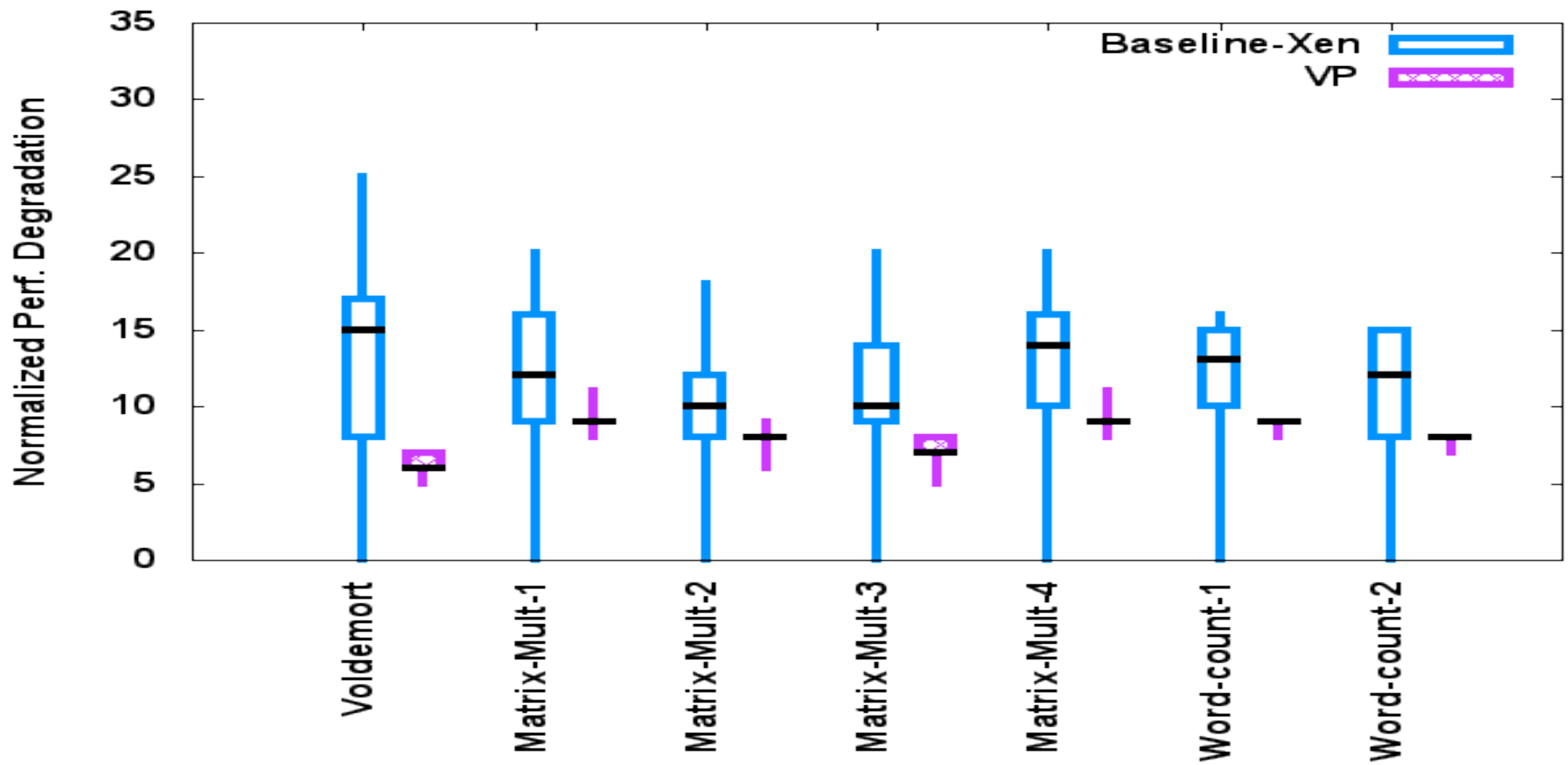
Evaluation

- Application mixes
  - Stream-SPEC (3 Stream + 2 Milc + 2 Mcf + 2 Lbm)
  - Voldemort-MapReduce (1 Voldemort + 3 Matrix-Mult + 3 Wordcount)
  - StreamingServer-MapReduce (1 StreamingServer + 3 MatrixMult + 3 Wordcount)
- 4-VCPU VMs + 4G Memory, no CPU sharing, prefetching disabled
- Why disabled Prefetching?
  - No software control
  - Hard to quantify use of memory bandwidth per application using performance counters

Experiment Methodology

- Baseline-Xen: Each application-mix executed in different startup order to remove colocation bias
  - Observe best performance and normalize other performance values to this case
- VP-Xen: Similar runs with VP-enabled Xen
  - Performance normalized to best performance in Baseline-Xen case

Voldemort-Mapreduce



- Voldemort has higher MF sensitivity than Matrix-Mult and Wordcount
- Voldemort worst-case times improved almost 3 times when contention caused by Matrix-Mult is mitigated using VP methods
- ~8% overhead in moving applications to "good" configuration (as of the baseline)

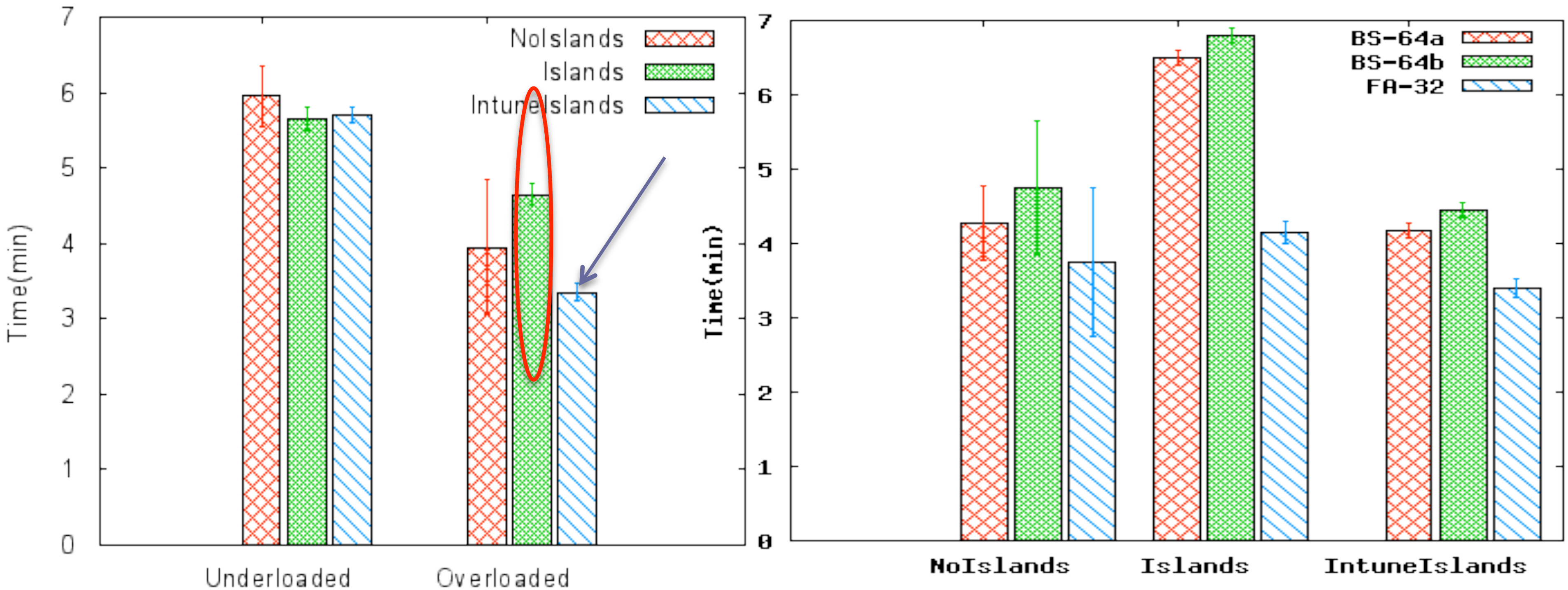
Platform Manager Revisited

- Current implementation – centralized platform manager and per-VP (per-application) platform monitor
- However, need for scale, and thermal and power constraints, lead to platform designs with
  - increase in corecounts, complex memory hierarchies…
  - tile based design (e.g., SCC)
  - special(ized) engines (e.g., asymmetric or heterogeneous cores and accelerators), including software-based specialization (e.g., run RT-scheduler on subset of cores)

Islands of Resources

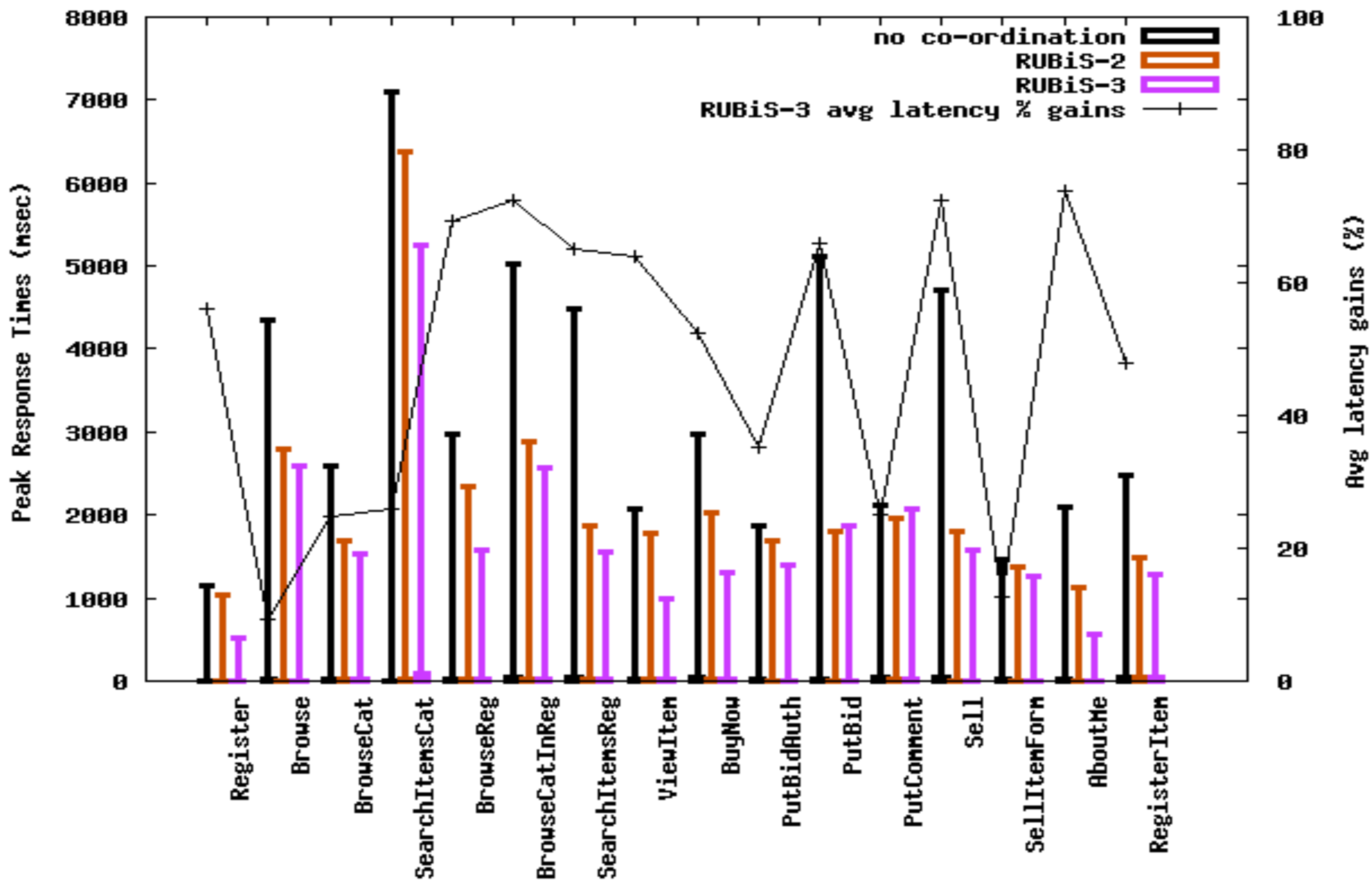
- Resource Islands – resource sets under control of independent managers
  - E.g., multiple/different CPU schedulers for sets of cores; different runtime/scheduler for graphics or communication cores
- Applications – Virtual Platforms – are overlayed across islands
- Coordination mechanisms to advise island managers to adjust resource allocations or, if possible, to trade resources across islands

Need for island coordination



Islands must coordinate to trade resources to meet elastic application requirements.

Pipelined Web application: RUBiS



Thank you. Questions?