X-ARCC: Extreme-Scale Adaptive Resource-Centric Computing

Steven Hofmeyr
LBNL

John Kubiatowicz
UC Berkeley

The OSR Principal Investigators (PI) Meeting
May 23rd 2016

X-ARCC Project

- Goals
  - Discover and demonstrate useful mechanisms for exascale OS
  - Experimental research, not engineering effort (no production code)
- Collaboration between LBNL and UCB SwarmLab
  - Converging trends between HPC, Cloud, Mobile & Swarm
  - Energy is key limitation
  - Massive parallelism in dynamic, unpredictable environments
- Continuation of Tessellation OS project
  - Collaboration between LBNL and UCB Parlab
  - That was focused on single node multicore

Exascale Systems will be Dynamic

- Changing hardware resources: loss of nodes, addition of new nodes, DVFS, etc
- New asynchronous, massively parallel programming models
- Applications can change on the fly, e.g. visualization to steer simulation

Address with Adaptive Resource-Centric Computing (ARCC):

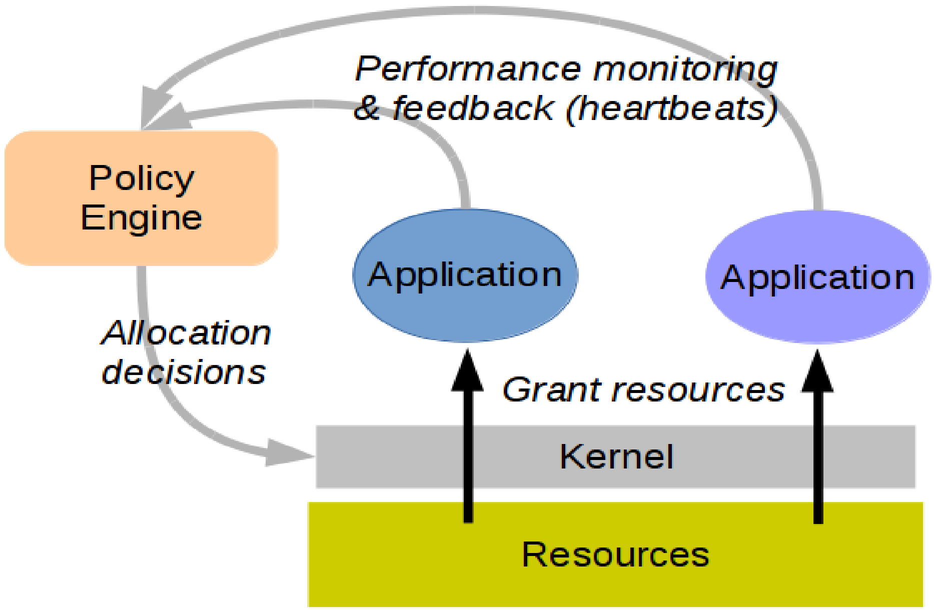
Change resource allocations dynamically according to current application behavior & system state to maximize performance & utilization for all applications

BERKELEY LAB

SWARMLAB UC BERKELEY

ARCC Feedback Control Loop

Mechanisms for dynamically allocating resources to multiple competing apps or app components based on performance requirements



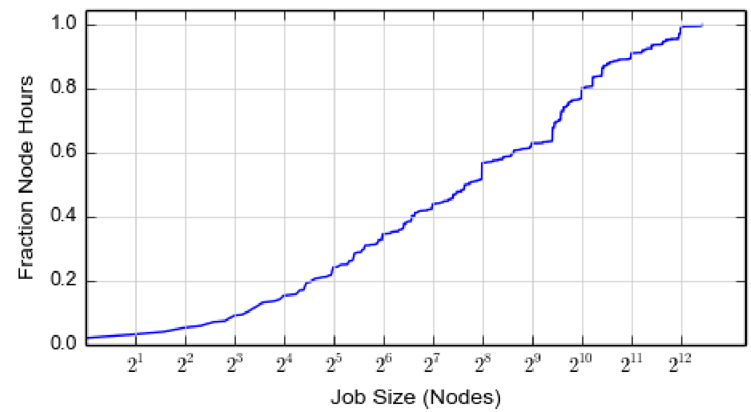
Exascale Apps will be Complex

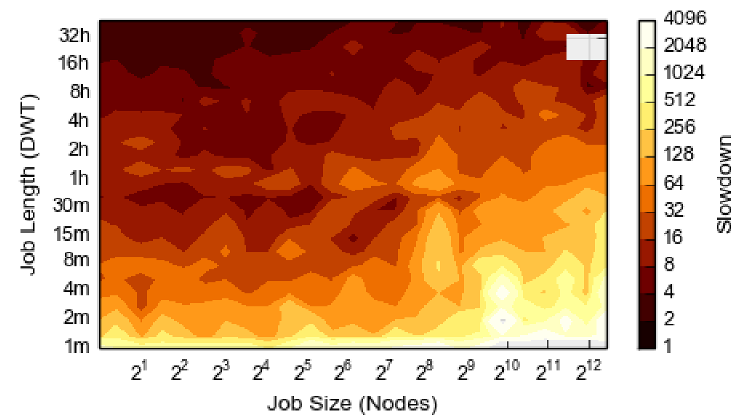
- Multiple components, each with different resource requirements, different scheduling, etc
- In-situ & in-transit analytics and visualization
- Complex pipelines, e.g. genome assembly
- Modern languages, JITs, DSLs (big data, machine learning)
- Node-local services, e.g scalable checkpoint/restart

ARCC: support multiple independent apps/app components per node, i.e. share nodes

Node Sharing Simulations

- Sharing instead of batch scheduling?
  - Simulations of batch vs timeshare
  - Real job data from Edison over 620 days
  - Utilization ~90%
  - 50% core-hours used by jobs > 100 nodes, 20% used by jobs > 1000 nodes
- Measuring QoS/fairness
  - Slowdown = turnaround / DWT
  - Batch scheduling: longer-running, smaller jobs have lower slowdown
  - Timeshare: constant slowdown





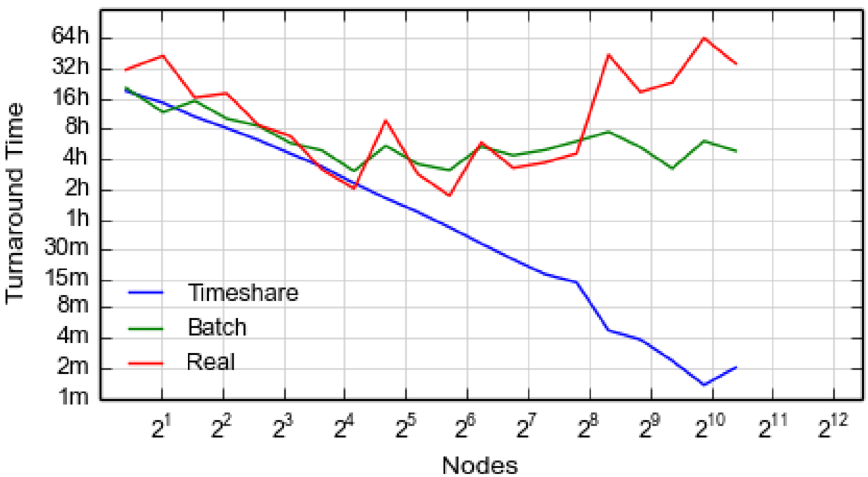
BERKELEY LAB

SWARMLAB UC BERKELEY

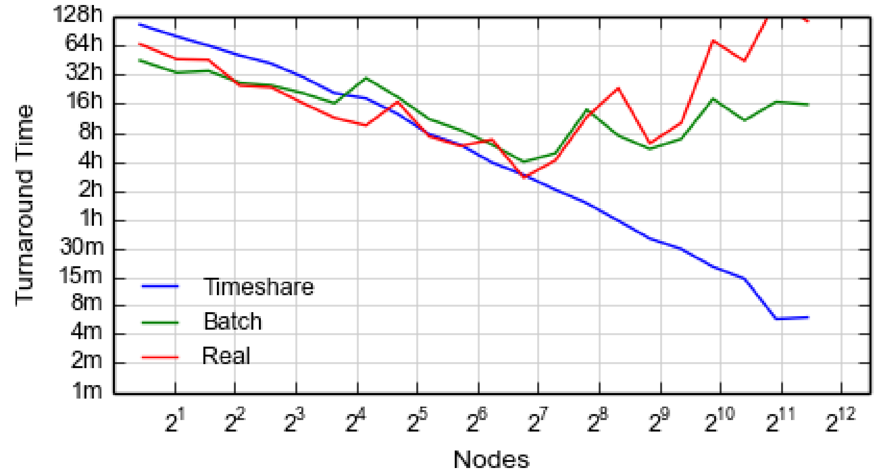
Scaling Implications

For scalable apps, what concurrency is best on a busy system to minimize turnaround?

HipMer



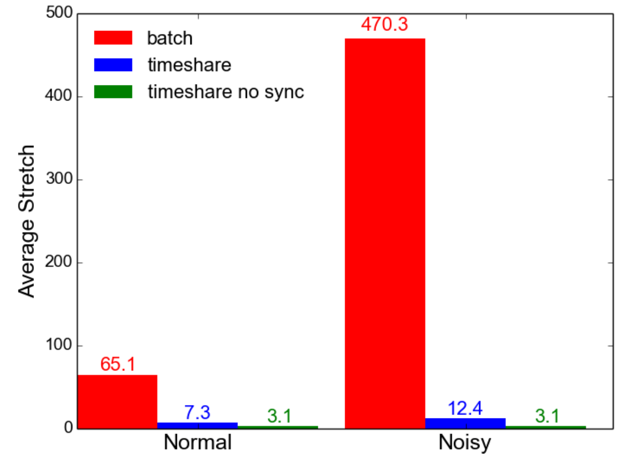
GTC

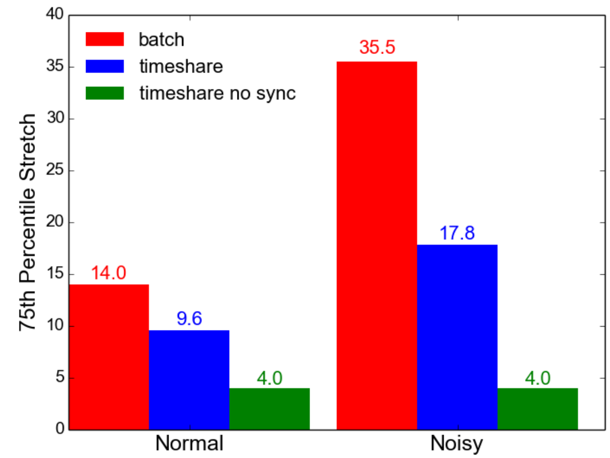


- Batch: turnaround doesn't scale due to bias in slowdown
- Timeshare: turnaround scales (as expected)

Impact of Noise

- Simple noise model
  - Each minute, 0.001 prob. of each node running at (½, 1) speed
  - More benign than turbo-boost?
  - Big increase in the long-tail of batch scheduled jobs
- Noise and prog. model
  - Relax assumption about BSP, e.g. async tasking
  - Noise-tolerant
  - Even if async prog models are less efficient, overall system utilization & turnaround could still be better





BERKELEY LAB

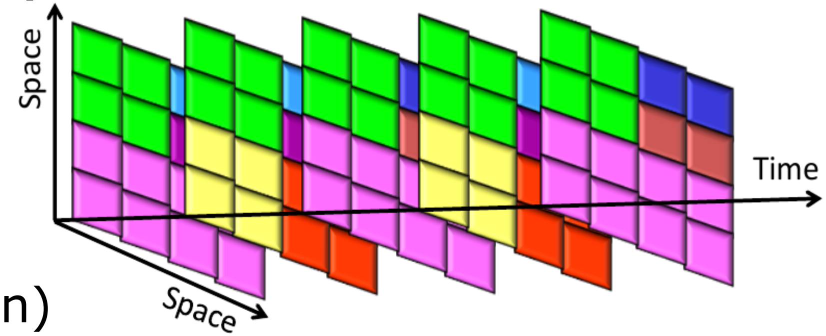
SWARMLAB UC BERKELEY

Node Sharing with X-ARCC

- Node sharing AND performance predictability
  - Cells and two-level scheduling
- Each app runs in a **cell**:
  - Guaranteed resources & enforced performance isolation
  - "Bare metal" control over own resources
- System services:
  - Services provide QoS guaranteed access to shared hardware resources
  - Services run in cells and can use other services
- Communication between cells via secure channels

Two-Level Scheduling

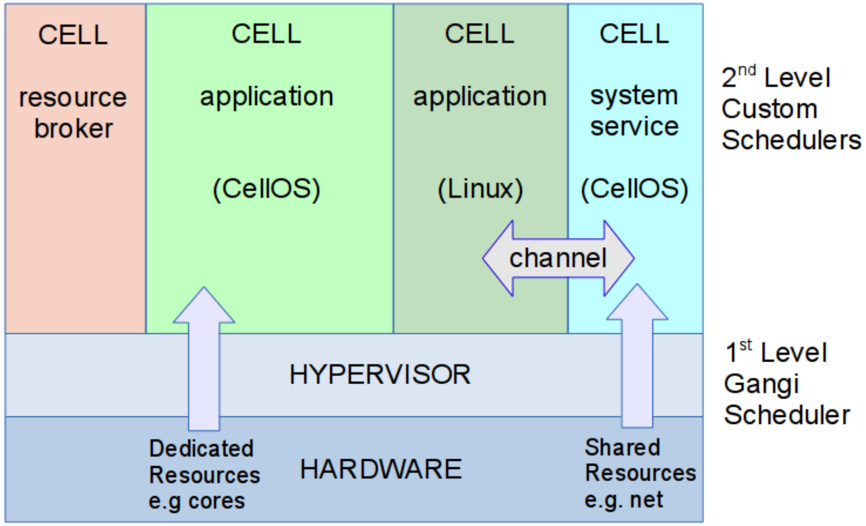
- Separate allocation of resources *to* cells (1st level) from management of resources *within* cells (2nd level)
- First Level (traditional OS role)
  - Manage conflicting resource demands of multiple apps
  - Space-time partitioning with gang-scheduling (predictability & flexibility of resource allocation)



- Second-level (runtimes role)
  - Manage resources for single app or set of cooperating apps
  - Customization through user-level scheduling & memory management
  - Minimize OS & other interference to make runtime design & implementation simpler & performance modeling possible

Implementing X-ARCC

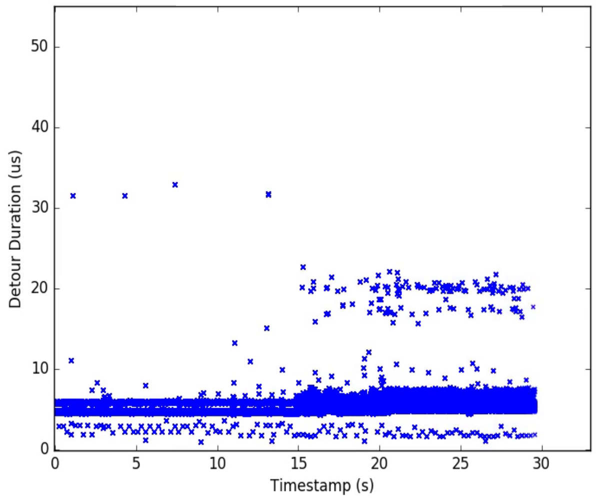
- Use virtualization (Xen)
  - Supports both bare-metal runtimes & full virtual machines
- First level (hypervisor):
  - **Gangi** scheduler for cells
  - Multiple scheduling policies: gang, best-effort, EDF, dedicated, event-driven
- Second level (VM):
  - Developed **CellOS**, based on Xen Mini-OS
  - Customizable scheduling
  - Simple memory management (no virtual memory)
  - Services include networking, file system, block, log & gui
  - But: *unikernels* are becoming popular – use instead of CellOS



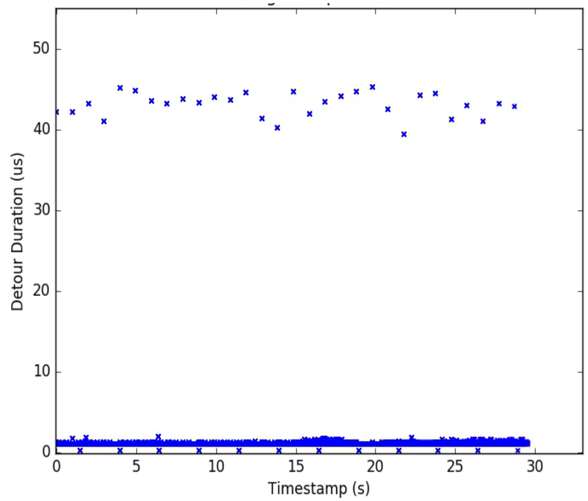
Reducing Noise

- Experiments
  - selfish detour on two socket machine
  - After 15s, kernel build on other socket
  - X-ARCC config: Xen+Gangi+unikernel
- Results
  - Without competing workload, Linux is more noisy than X-ARCC config
  - Using all Linux isolation features (cgroups, pinning, etc) still does not isolate competing workload
  - Competing workload no effect on X-ARCC config
- Found Kitten similar to X-ARCC

Linux Native



X-ARCC config



BERKELEY LAB

SWARMLAB UC BERKELEY

Monitoring Energy Usage in X-ARCC

- Need to treat energy as first class resource
  - Must accurately measure & attribute energy usage to cells
  - But energy measurements are coarse-grained, e.g. Intel RAPL counters are package level & wall metering is at node level
- XeMPower
  - Based on socket-level energy measurements with RAPL
  - Hardware performance counter models account for energy of simultaneously running cells
  - Estimators go from coarse-grained physical measurements to fine-grained energy attribution
- MARC
  - Generate models of power consumption of running applications

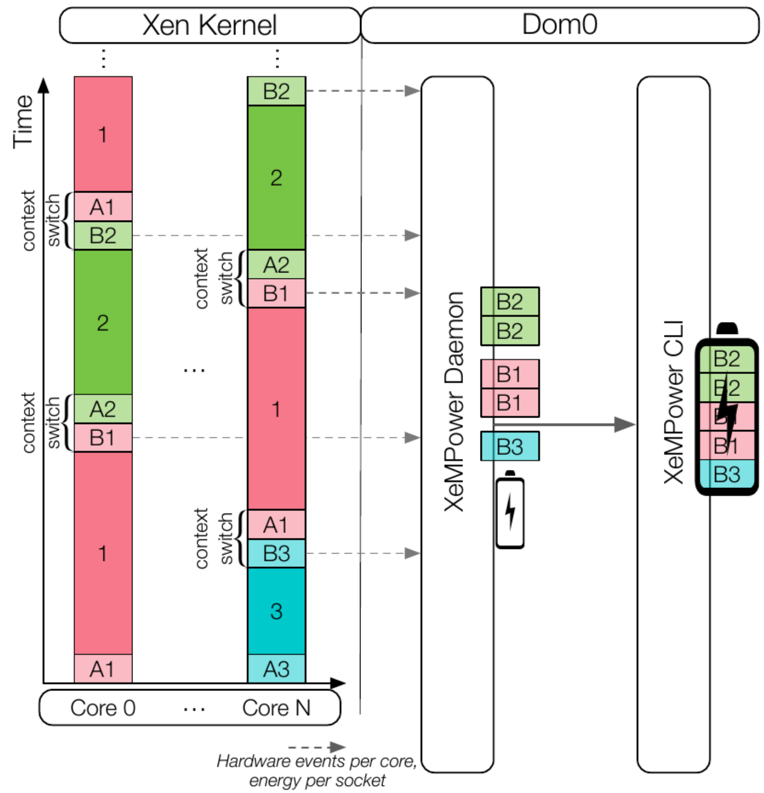
With M. Feroni, A. Damiani, A. Corna & M Santambrogio (Politecnico Milano)

BERKELEY LAB

SWARMLAB UC BERKELEY

XeMPower Implementation

- Hypervisor instrumentation
  - Track context switches in first-level scheduler
  - Record counters: cycles, LLC, branch, RAPL
- Service running in cell
  - Aggregate counters
  - Uses model of energy to split socket measurements & attribute to cells
- Monitoring overhead < 1%

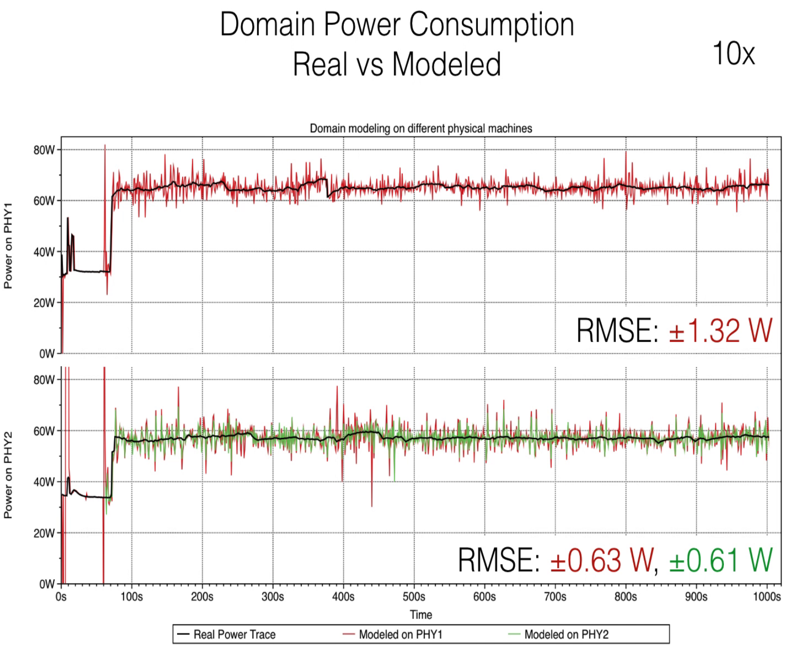


BERKELEY LAB

SWARM LAB UC BERKELEY

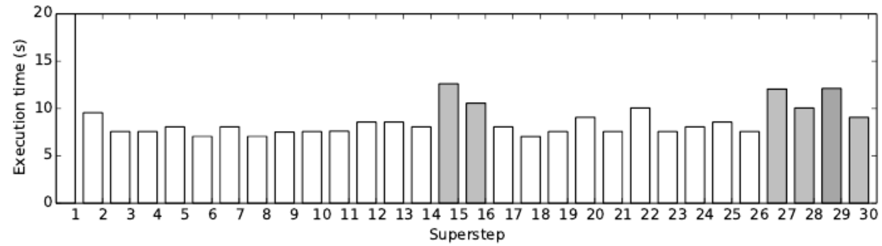
MARC

- **M**odeling and **A**nalysis of **R**esource **C**onsumption
  - Use traces from XeMPower
  - Model energy consumption of Xen domains with < 5% error
- Energy modeling
  - Trends accurately approxd by piecewise linear curves
  - Identify configurations
  - Approx energy consumption of config with linear fit
  - Build energy model on one physical machine and reuse for different machine with good accuracy

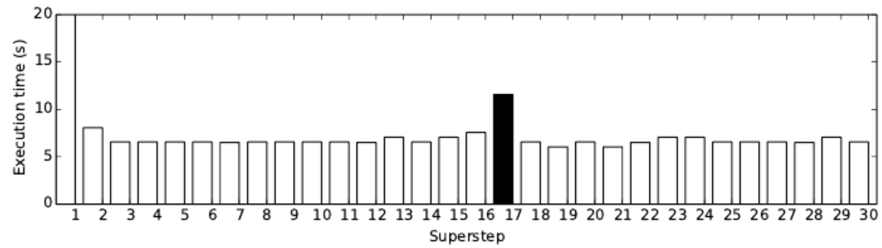


Scheduling Distributed Services

- Distributed services can be a problem
  - Independent decisions generate noise for distributed apps
  - e.g. garbage collection (GC) (important for cloud, not HPC – yet)
  - Other services, e.g. local C/R, analytics, profiling, etc.



(a) Baseline System (no coordination)



(b) Coordinating GC (Stop-the-Universe)

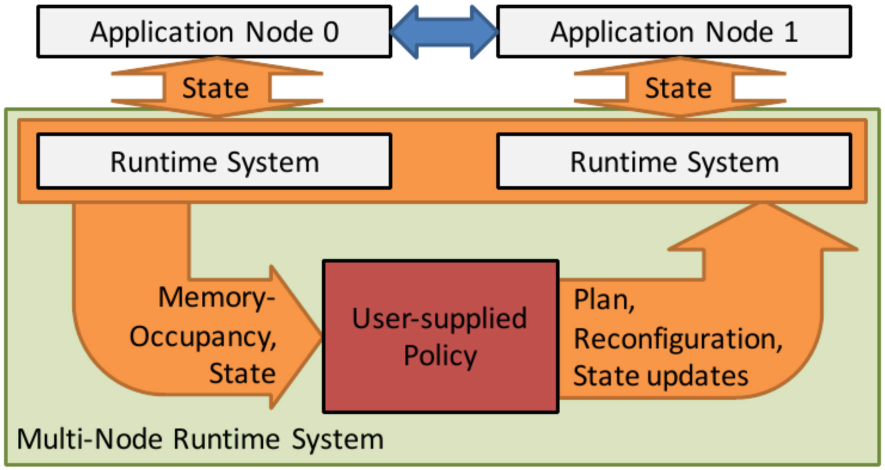
- Taurus prototype
  - Multinode fault-tolerant framework for coordinating distributed shared services
  - No app changes (unless desired)
  - No change to JVM interface
  - First use case: GC in managed languages (Java)

BERKELEY LAB

SWARMLAB UC BERKELEY

Taurus Implementation

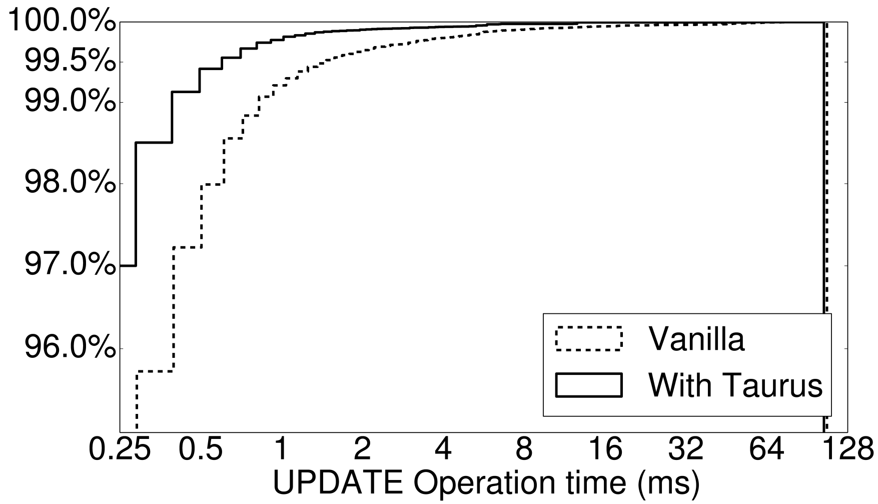
- Multinode runtime for services
  - Simple policy DSL describes strategies for coordinating services
  - Inputs: system & app state
  - Outputs: policy-based plan
  - e.g. when to activate GC given memory usage



- Scalable & fault tolerant
  - Cluster is divided into coordination groups
  - Each group elects a leader that receives inputs, executes policy & distributes the plan
  - Distributed consensus protocol to migrate state & ensure leader exists after node failures

Taurus Performance

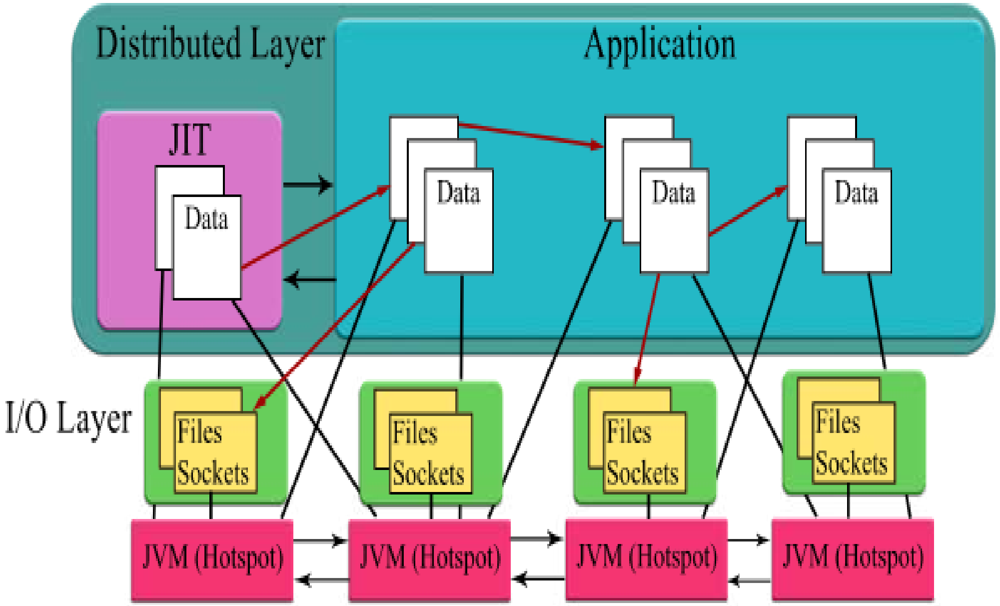
- Experiments with GC in cloud apps (Java)
  - Significant performance improvements in latency & throughput
  - e.g. Spark PageRank, reduce time 21% & tail latency 50%
- Managed language features for HPC
  - Productivity, e.g. automatic memory management
  - New style scientific apps, e.g. genome assembly
  - Machine learning (Spark is a premier ML framework)



- Beyond managed languages
  - Noise reduction through coordination of services in general
  - Component of cell runtime

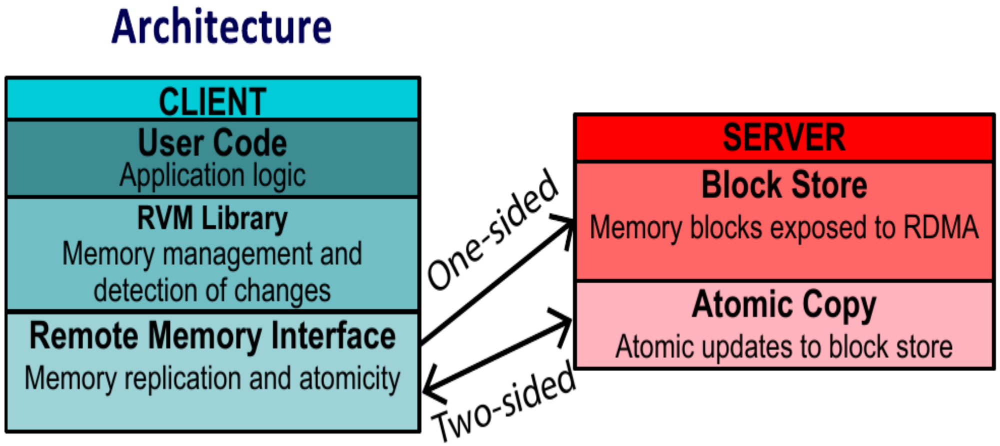
DJ Distributed Runtime

- Exploring multinode runtimes
  - Java platform that enables objects to be relocated and remotely accessed
  - Appear as single JVM
  - Transparent to programmers
- Functioning prototype
  - Overlay layer on top of JVMs
  - Performance tuning still needed



Advanced Memory Features

- Nephele recoverable memory
  - Detects changes to recoverable memory regions
  - Replicates memory to remote nodes using RDMA
- Simple API:
  - Funcs for allocation
  - Func to mark consistency points
  - Minimal app changes
  - Implement in cell runtime, e.g. barrier → consistency point
- Efficient (even unoptimized)
  - Replication 5x faster & recovery 10x faster than BLCR



Conclusions

- X-ARCC: discover & demonstrate potential mechanisms for an exascale OS
- Sharing nodes between applications can be beneficial and be done with low noise
- Implemented lightweight runtimes (CellOS, DJ), advanced scheduling (Gangi), distributed resource management (Taurus), power monitoring (XeMPower) & modeling (MARC), recoverable memory (Nephele)