XARC: Adaptive Resource-Centric Computing for Exascale

Steven Hofmeyr
LBNL

John Kubiatowicz
UC Berkeley

The X-Stack & OSR Principal Investigators (PI) Meeting
Dec 8th 2015

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
  - 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

Exascale Systems will be Complex

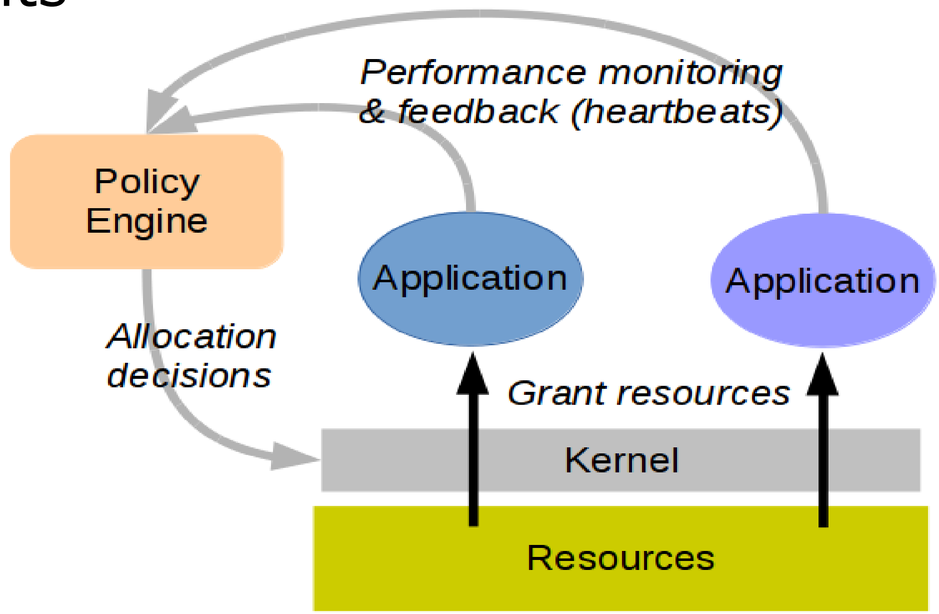
- Applications
  - Multiple components, each with different resource requirements, different scheduling, etc
  - Complex pipelines, e.g. genome assembly
  - In-situ & in-transit analytics and visualization
  - Node-local services, e.g scalable checkpoint/restart
- Hardware
  - Heterogeneity, e.g. fat & thin cores
  - Deep memory hierarchies

Resource allocation will be an ongoing complex optimization problem

This is addressed by the ARCC feedback control loop

ARCC Feedback Control Loop

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



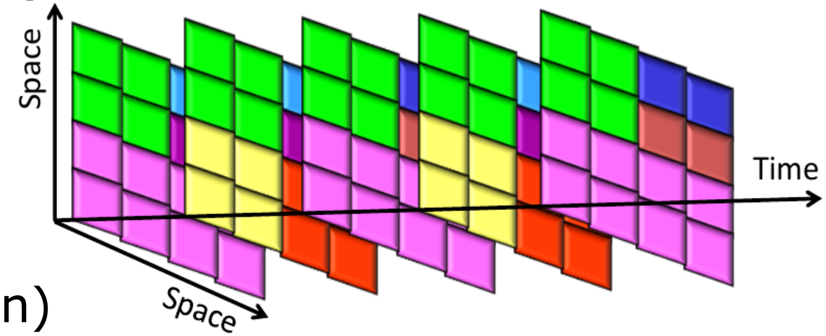
Implemented in the XARC Operating System (OS)

XARC Experimental OS

- Support for running multiple apps on a single node while maintaining performance predictability
  - Cooperative apps, e.g. simulation + in-situ analytics, multicomponent
  - Disparate, competing apps, e.g. system services
  - Improve flexibility & utilization of overall system
- Each app runs in a **cell**:
  - Guaranteed resources & enforced performance isolation
  - Services provide QoS guaranteed access to shared hardware resources
  - Services also run in cells and can use other services
  - Communication between cells via secure channels

Two-Level Scheduling

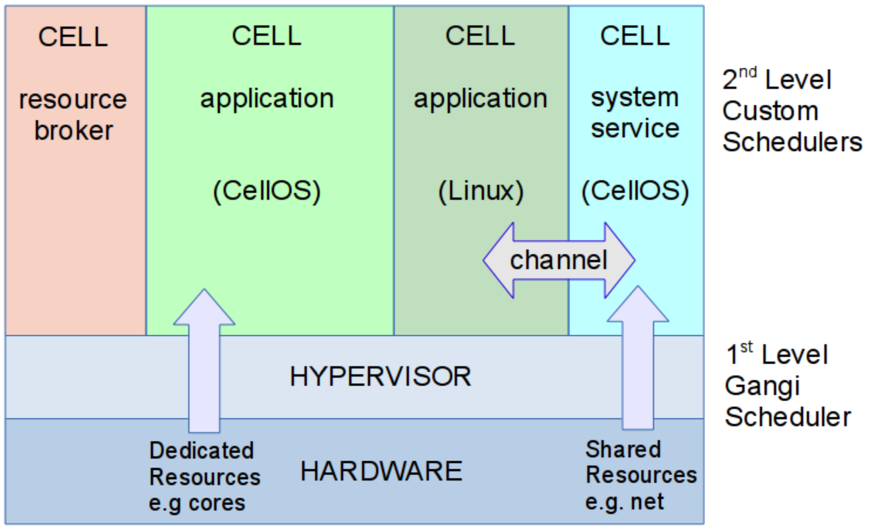
- Separate allocation of resources *to* cells (1^{st} level) from management of resources *within* cells (2^{nd} level)
- First Level (traditional OS role)
  - Manage conflicting resource demands of multiple apps
  - Space-time partitioning with gang-scheduling (predictability & flexibility of resource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

Implementation of XARC

- Lightweight implementation based on XEN VMM
  - 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):
  - CellOS: lightweight runtime based on Xen Mini-OS
  - Customizable scheduling
  - Simple memory management (no virtual memory)
  - Services include networking, file system, block, log & gui



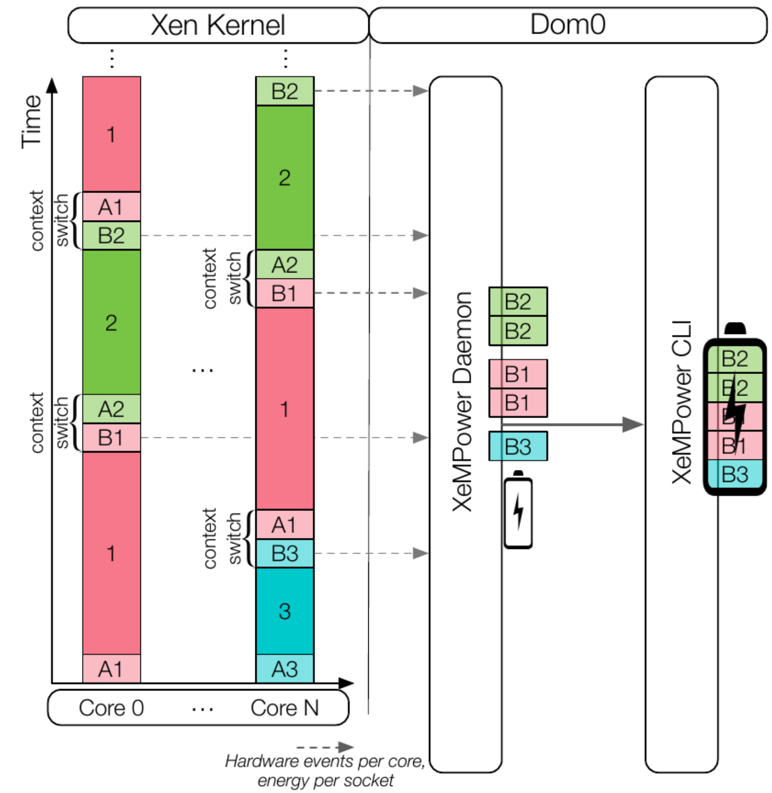
Monitoring Energy Usage in XARC

- 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

Collaboration with M. Feroni & M Santambrogio (Politecnico Milano)

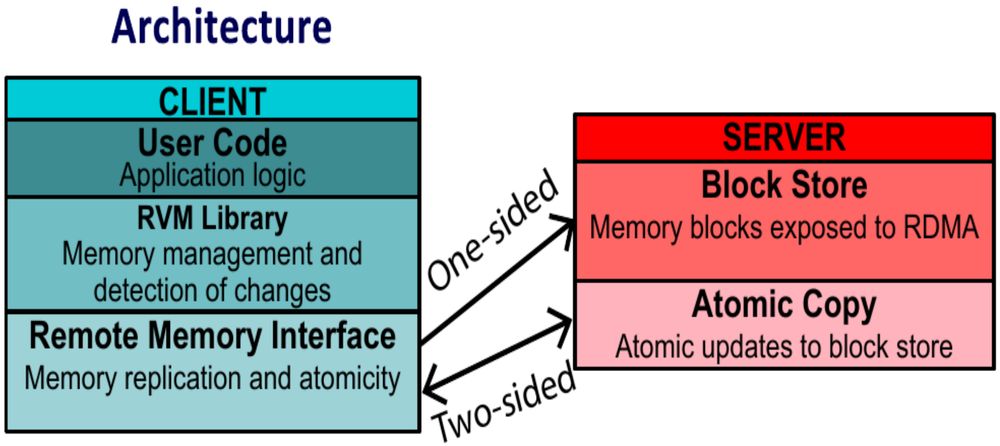
XeMPower Implementation

- 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%
- Connect to MPower energy framework (predictions)



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

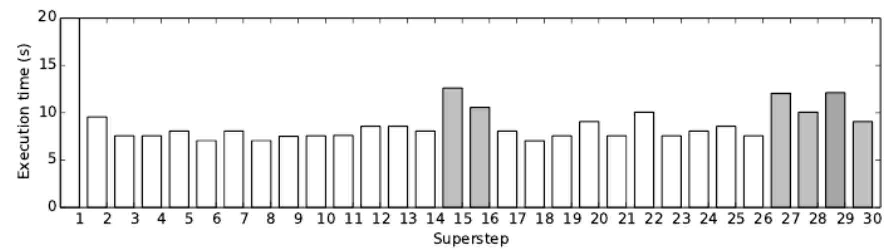


BERKELEY LAB

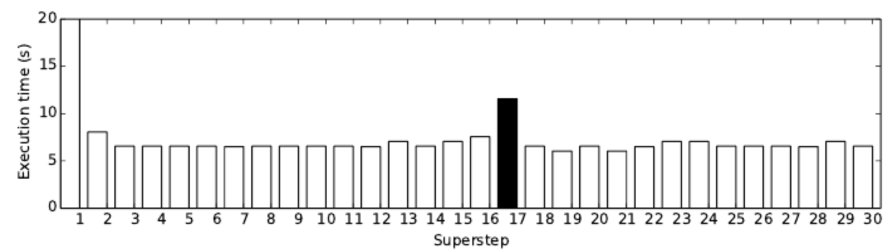
SWARM LAB UC BERKELEY

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)

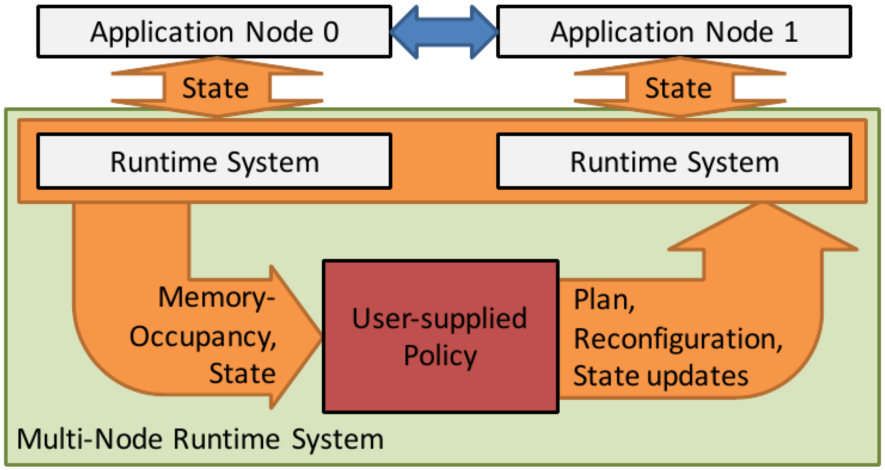
- Holly prototype
  - Multinode fault-tolerant framework for coordinating distributed shared services
  - No app changes (unless desired)
  - First use case: GC in managed languages (Java)

BERKELEY LAB

SWARM LAB UC BERKELEY

Holly Implementation

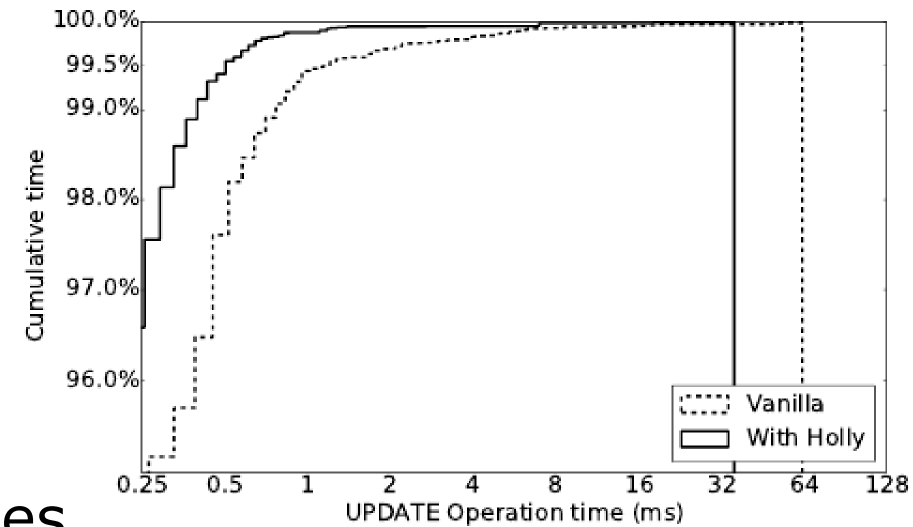
- Multinode runtime for services
  - Simple policy DSL describes strategies for coordinating services
  - Inputs: system & app state
  - Outputs: policy-based plan
  - e.g. when to activate GC given memory usage



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

Holly Performance

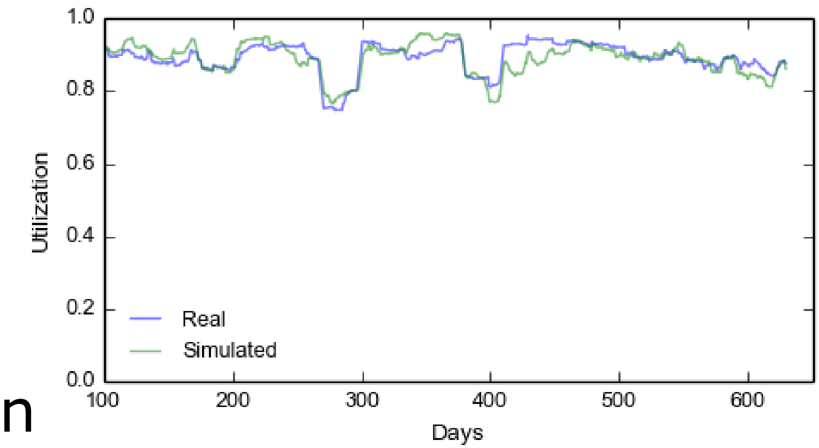
- Experiments with GC in cloud apps (Java)
  - Significant performance improvements in latency & throughput
  - e.g. Spark PageRank, reduce time 21% & tail latency 50%
- Managed language features for HPC
  - Productivity, e.g. automatic memory management
  - New style scientific apps, e.g. genome assembly,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

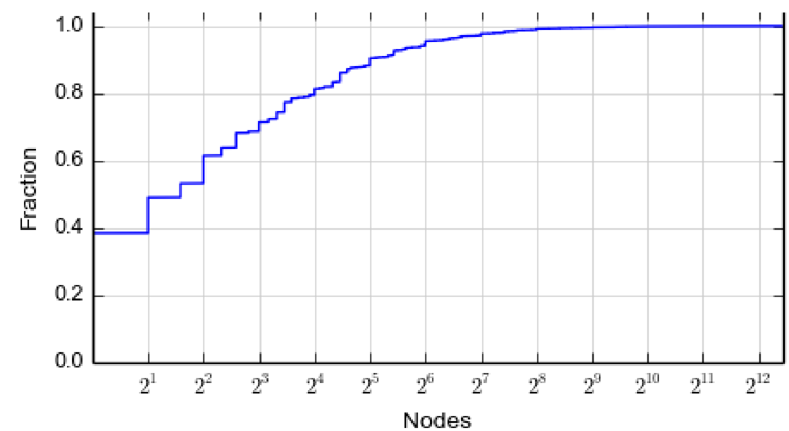


- Beyond managed languages
  - Noise reduction through coordination of services in general
  - Component of cell runtime

XARC at Scale

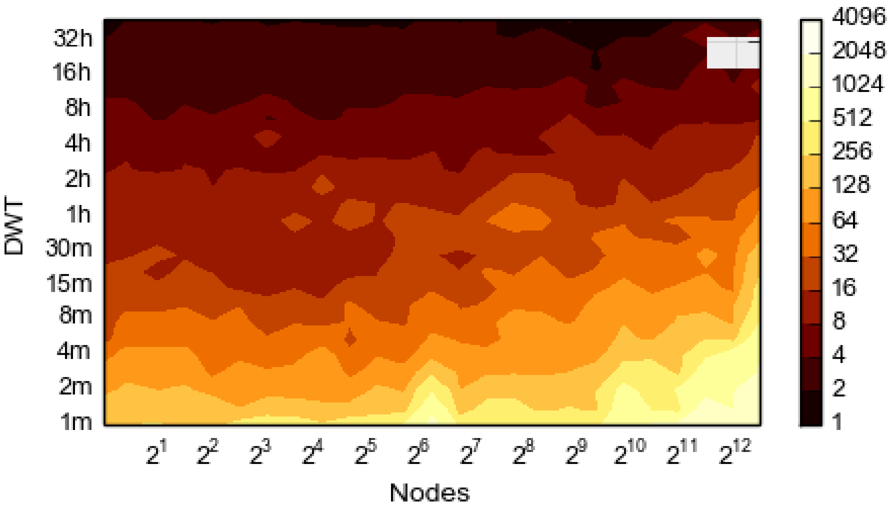
- Are XARC design principles meaningful at scale?
  - When does it make sense to share resources on a node?
  - Coupled apps, services, …?
- Explore issues with simulation
  - Space-time partitioning & gang-scheduling vs batch scheduling
  - Noise, heterogeneity, etc
- Job data from Edison
  - 2.6 million jobs over 620 days
  - High utilization ~90%
  - Many small, short jobs: 90% < 32 nodes and < 2 hrs





Exploring Global Fairness

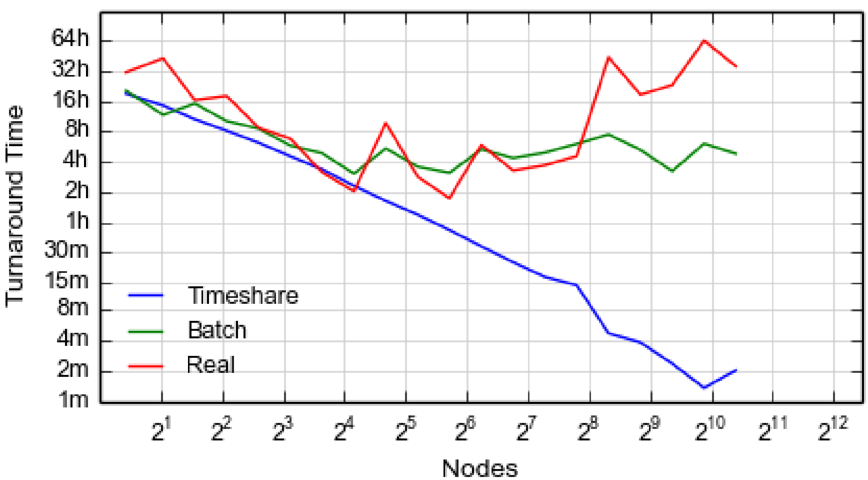
- Comparing batch & timesharing
  - Batch is FCFS + backfill (simulation very close to real system)
  - Timeshare assumes bulk sync & gang-scheduling
  - Similar utilization (90%)
- Measuring QoS/fairness
  - Stretch = turnaround / DWT (normalized turnaround)
  - Batch scheduling: longer-running, smaller jobs have lower stretch
  - Timeshare: constant stretch



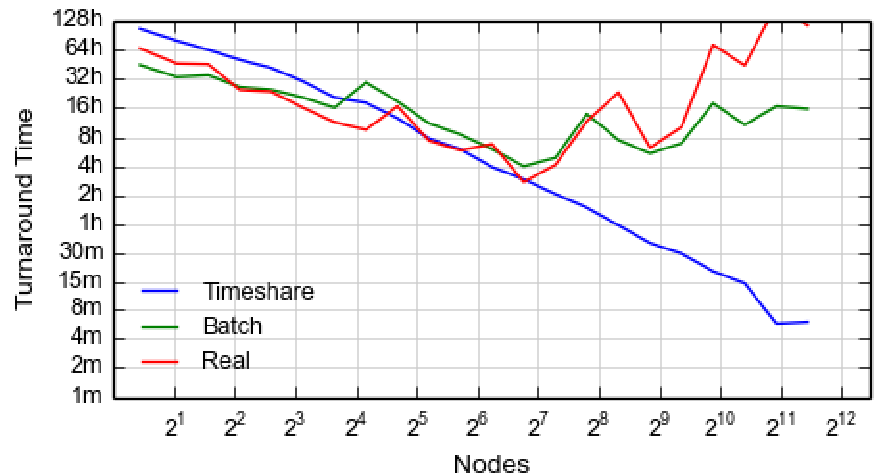
Scaling Implications

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

HipMer



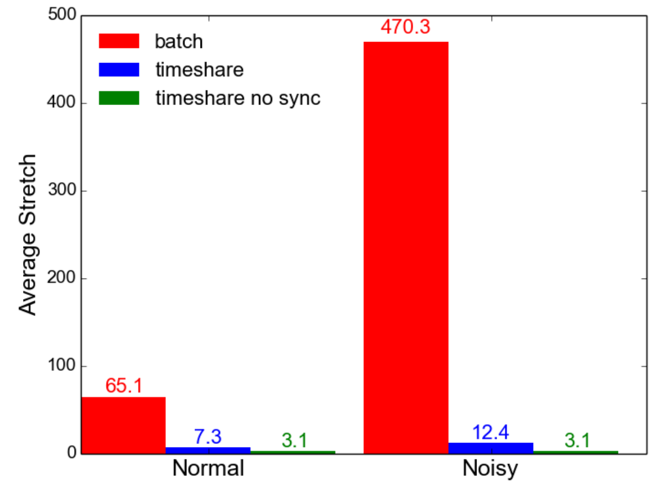
GTC

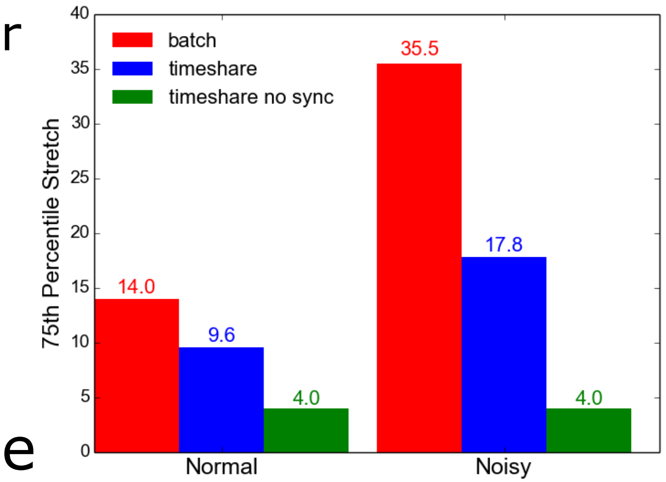


- Batch: turnaround doesn't scale due to bias in stretch
- Timeshare: turnaround scales (as expected)

Impact of Noise

- Simple noise model
  - Each minute, 1/1000 prob. of each node running between (½, 1) speed
  - More benign than turbo-boost?
  - Big increase in the long-tail of batch scheduled jobs
- Noise and programming model
  - Relax assumption about bulk sync for timesharing, e.g. async tasking
  - Noise tolerant & halves stretch
  - Even if async prog models are less efficient, overall system utilization & turnaround could still be better
- Next step: extrapolate to exascale





BERKELEY LAB

SWARMLAB UC BERKELEY

Conclusions

- XARC: discover & demonstrate potential mechanisms for an exascale OS
- Several approaches hold promise: energy measurement, scheduling distributed services, advanced memory management
- Simulation at scale can illustrate consequences of different resource allocation strategies and programming models