
ORACLE®

ORACLE LABS

Callisto: Rethinking the stack for parallel & distributed runtime systems

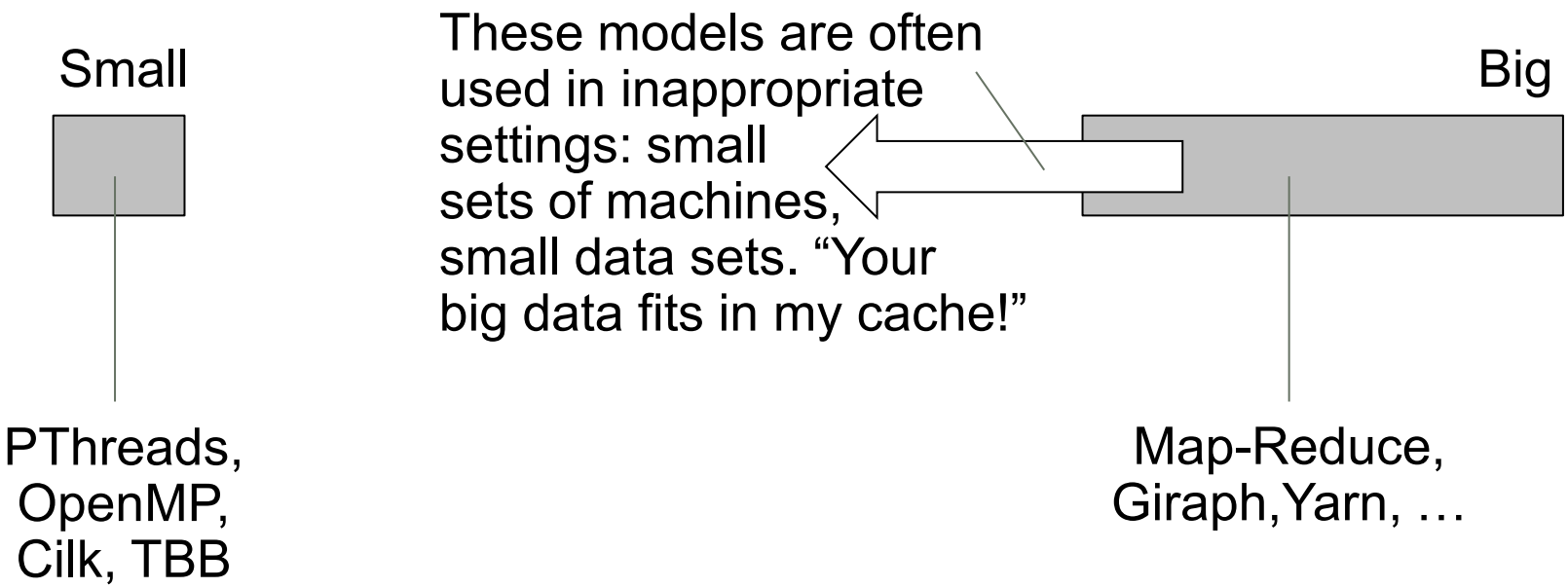
Tim Harris, Architect, Oracle Labs, Cambridge, UK

Meeting in the middle

Research often targets shared-memory multi-processor machines, and “warehouse-scale” distributed systems

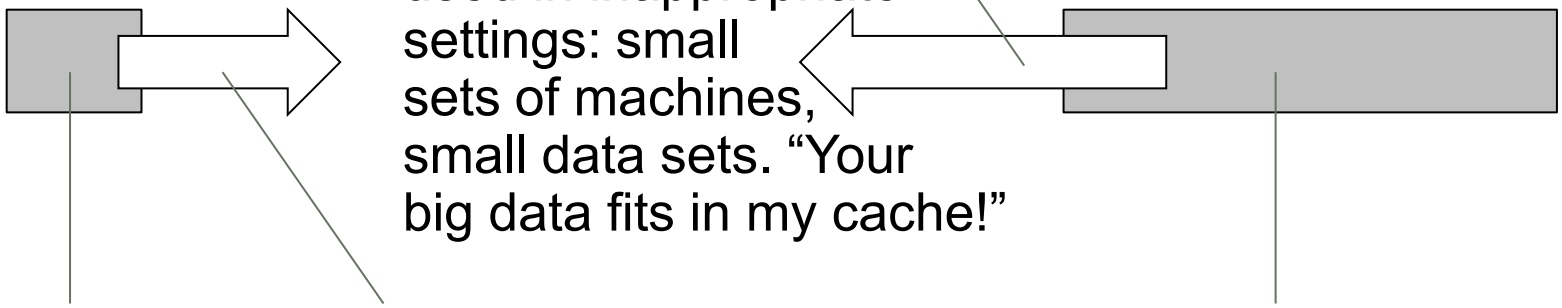
Meeting in the middle

Research often targets shared-memory multi-processor machines, and “warehouse-scale” distributed systems

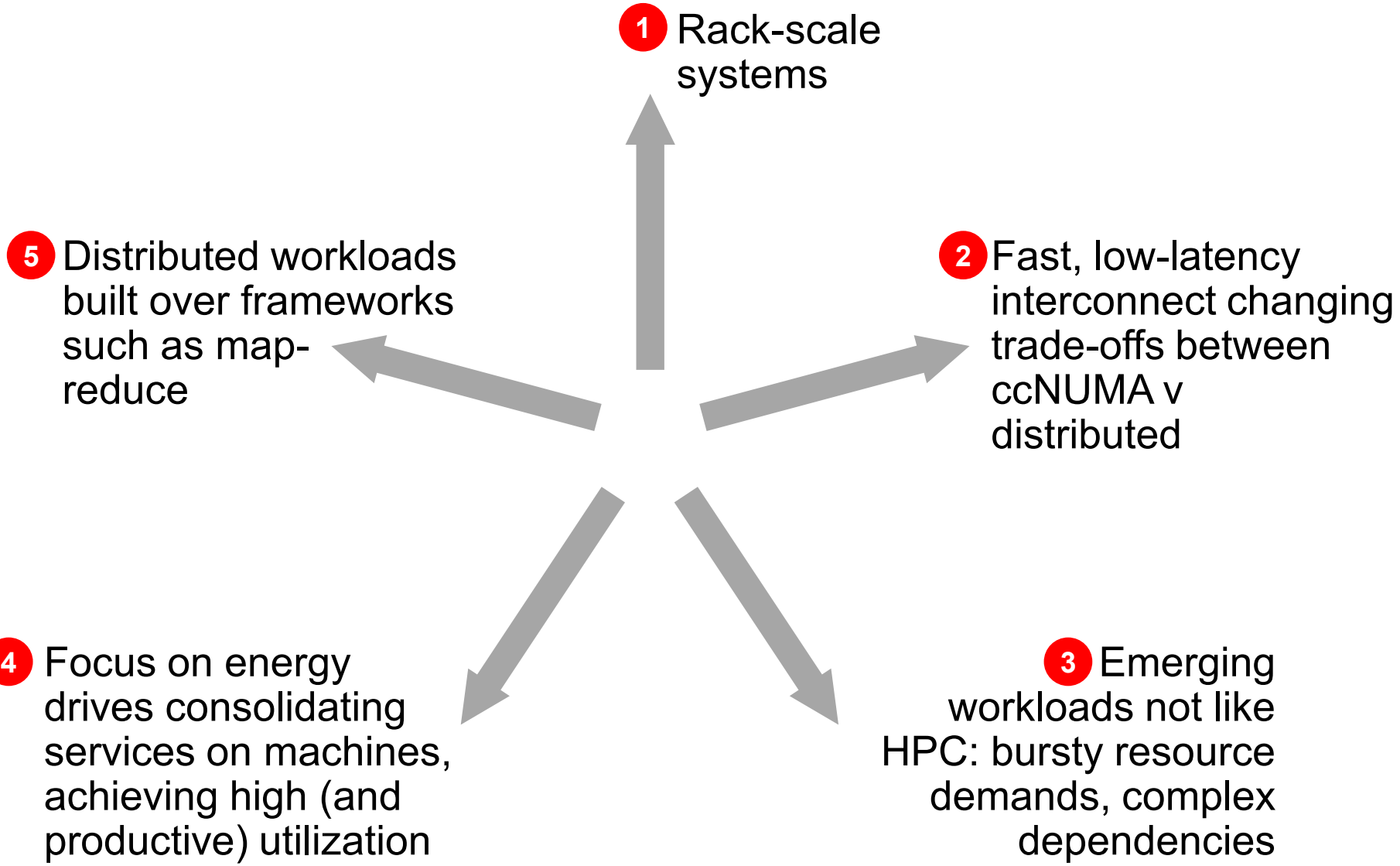


Meeting in the middle

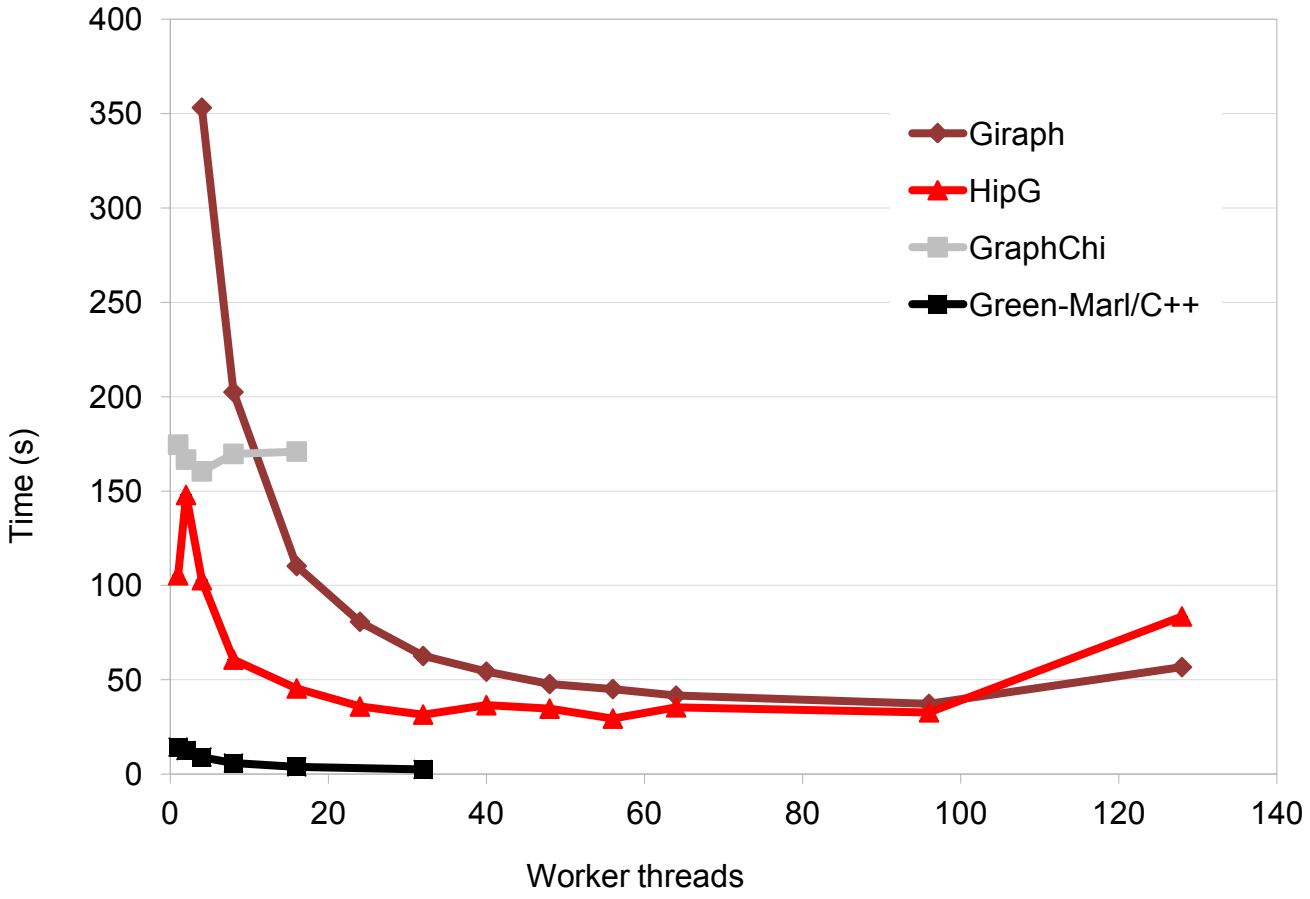
Research often targets shared-memory multi-processor machines, and “warehouse-scale” distributed systems



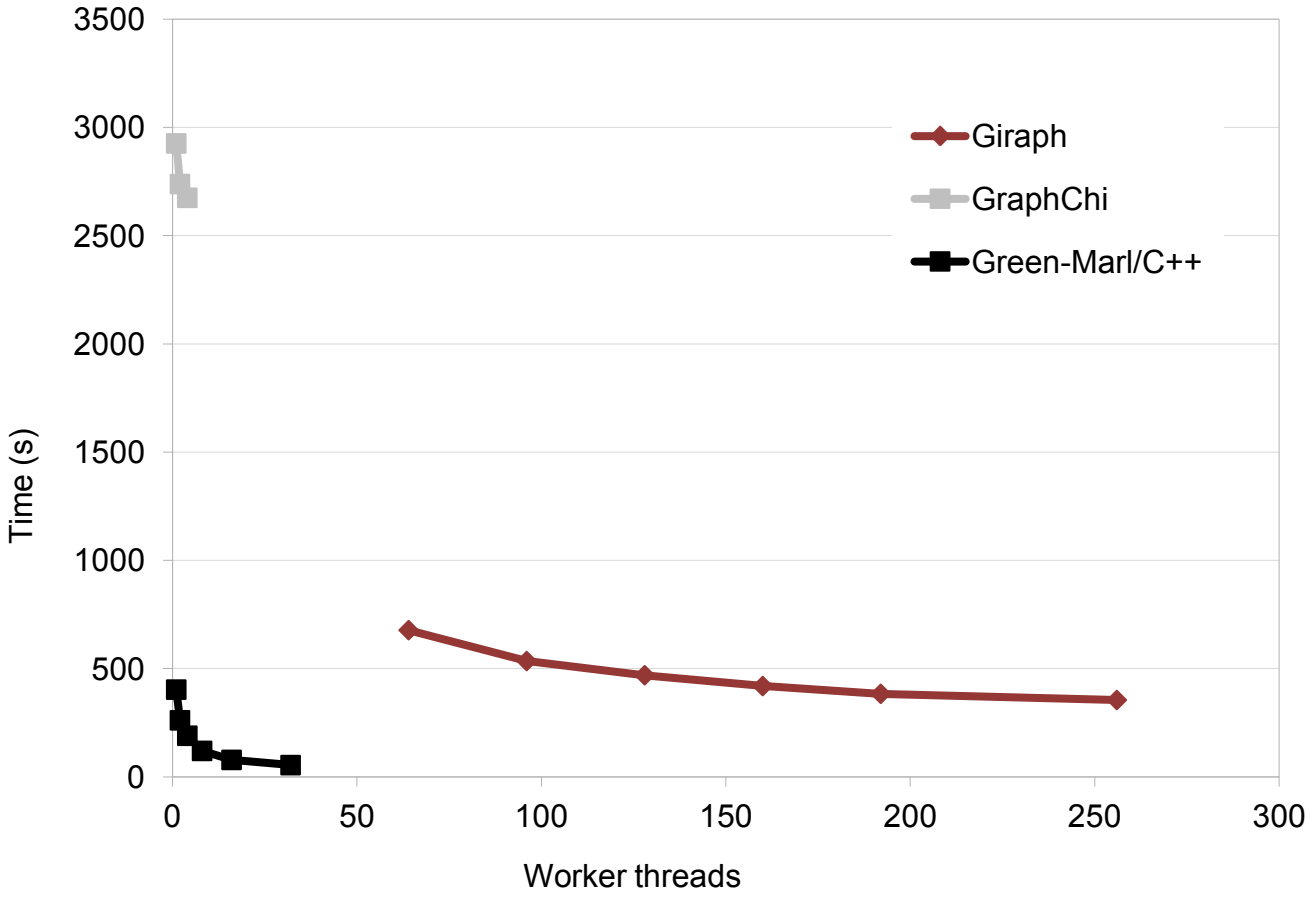
Current trends



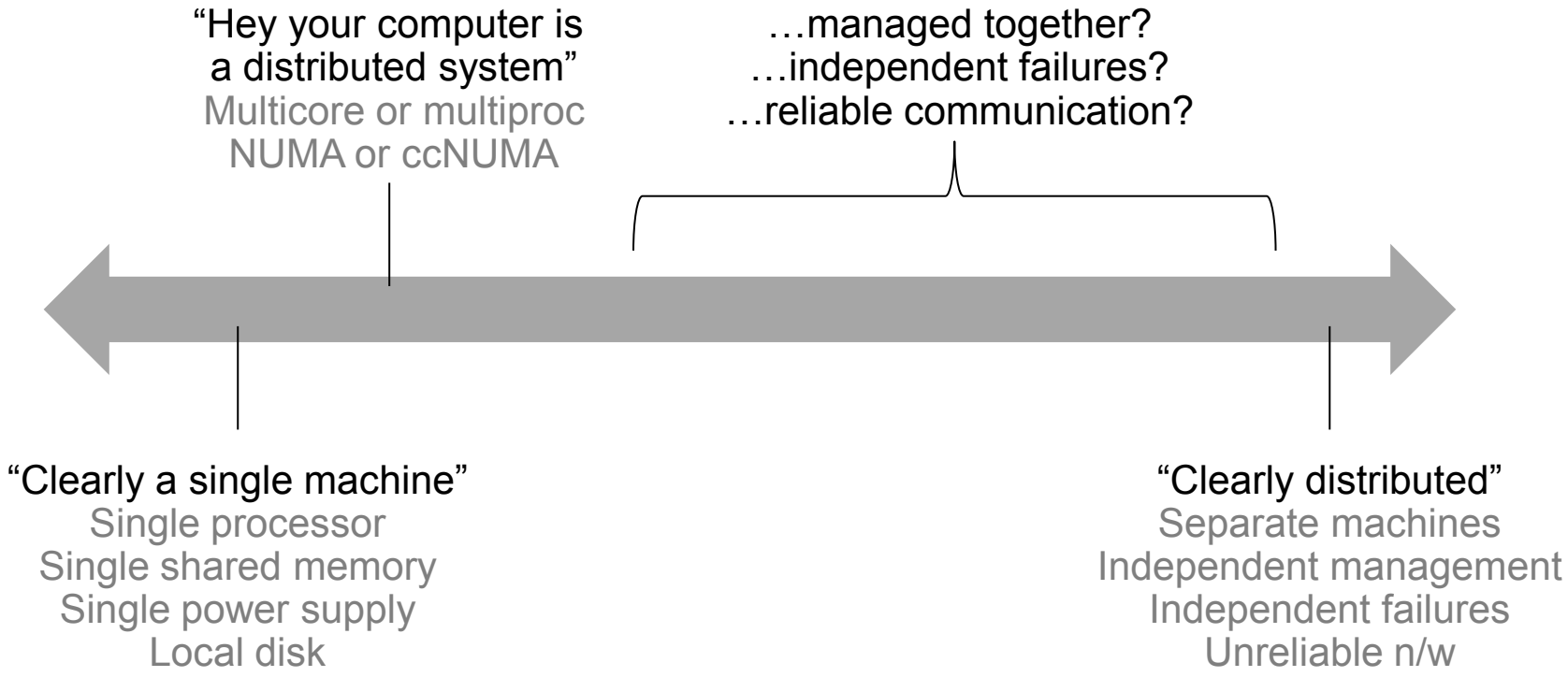
1. PageRank (social-net 4.8M vertices, 69M edges)



1. PageRank (social-net 41.6M vertices, 1.46B edges)



1. What I mean by a cluster



Motivation

1. The line between “single machine” and “distributed system” is getting blurred: shared components, often shared storage, while single machines exhibit partial failures.

2: Shared v distributed memory trade-off

```
do {
   cl.send.ping(&cl, val);
   cl.recv.pong(&cl, &val);
} while (val < ITERATIONS);
```

	Ping-pong latency (cycles)
Using UMP channel directly (threads sharing L3 on AMD 4*4-core)	931
Using event-based stubs	1134
Synchronous model (client only)	1266
Synchronous model (client and server)	1405

2: Shared v distributed memory trade-off

```
for (int iter = 0; iter < ITERATIONS; iter ++) {
  if (rank == 0) {
    MPI_Send(&msg, 1, MPI_INT, next, 0, MPI_COMM_WORLD);
    MPI_Recv(&msg, 1, MPI_INT, prev, MPI_ANY_TAG, MPI_COMM_WORLD, &Stat);
  } else {
    MPI_Recv(&msg, 1, MPI_INT, prev, MPI_ANY_TAG, MPI_COMM_WORLD, &Stat);
    MPI_Send(&msg, 1, MPI_INT, next, 0, MPI_COMM_WORLD);
} }
```

	Ping-pong latency (cycles)
MPI ping-pong (Mellanox ConnectX MT26428 QDR)	560

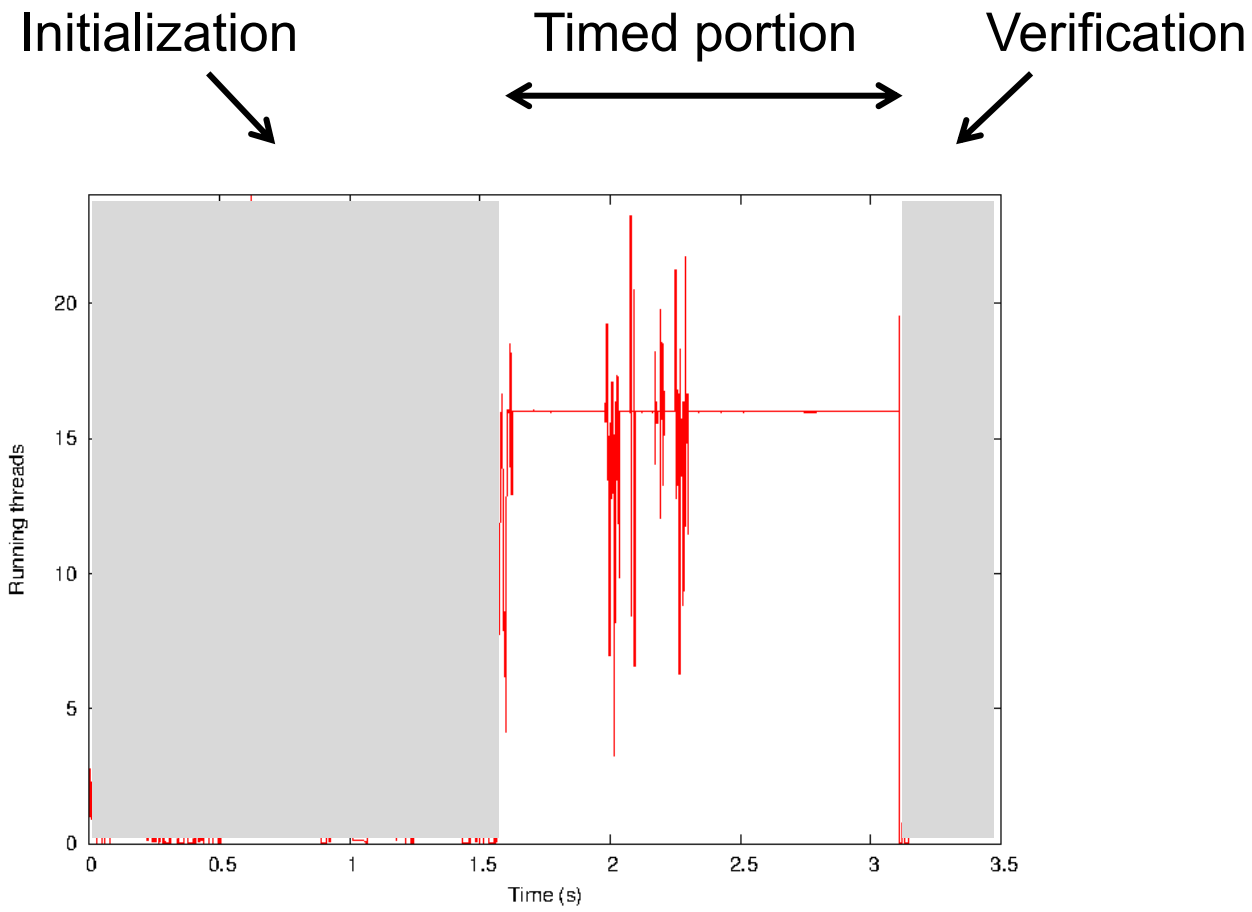
Motivation

1. The line between “single machine” and “distributed system” is getting blurred: shared components, often shared storage, while single machines exhibit partial failures.
2. Communication across these clusters is approx. the cost of communication across traditional big ccNUMA boxes.

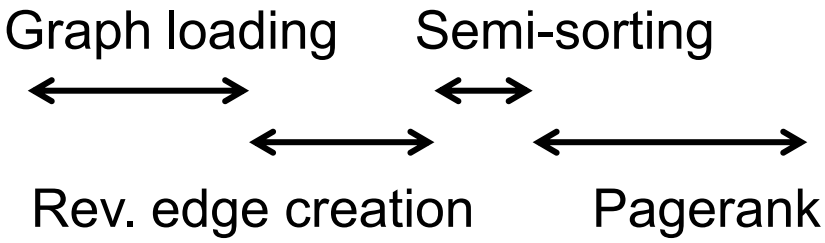
3: Workload demands

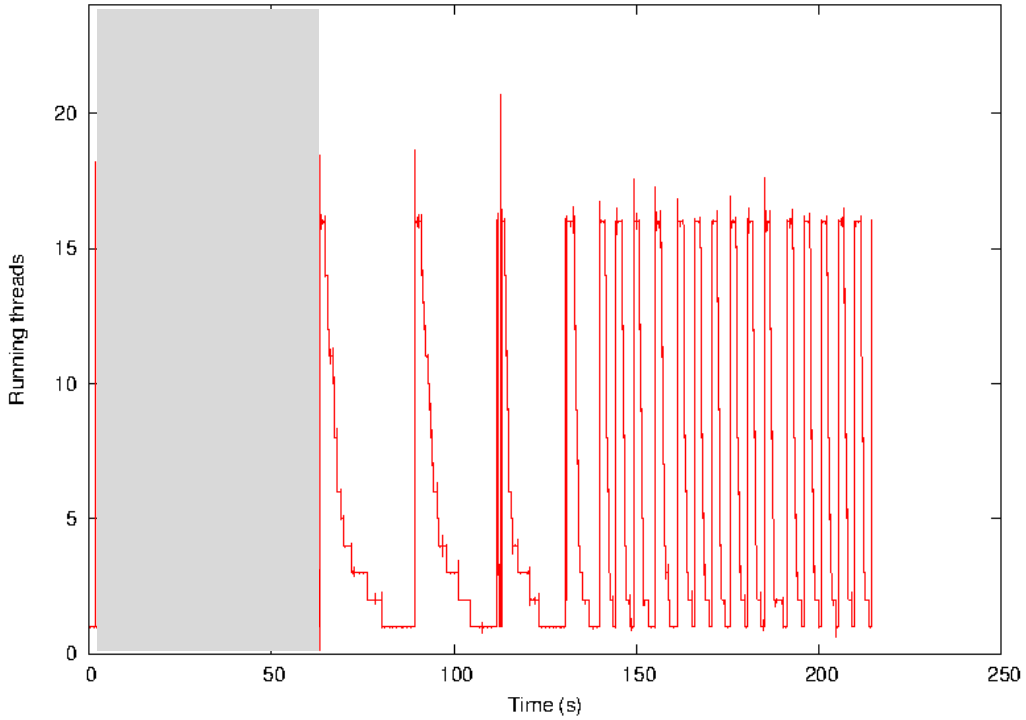
- Scheduler events on x86_64 Linux 2.6.32
  - 2* 8-core Xeon
  - Each core 2-way HT
- Distributed workloads: look at one node on a cluster
- Plot shows number of running threads, mean over 1ms or 10ms depending on run length
- Shaded area hides initialization etc.

3: Workload demands: NAS IS MPI (Integer Sort)



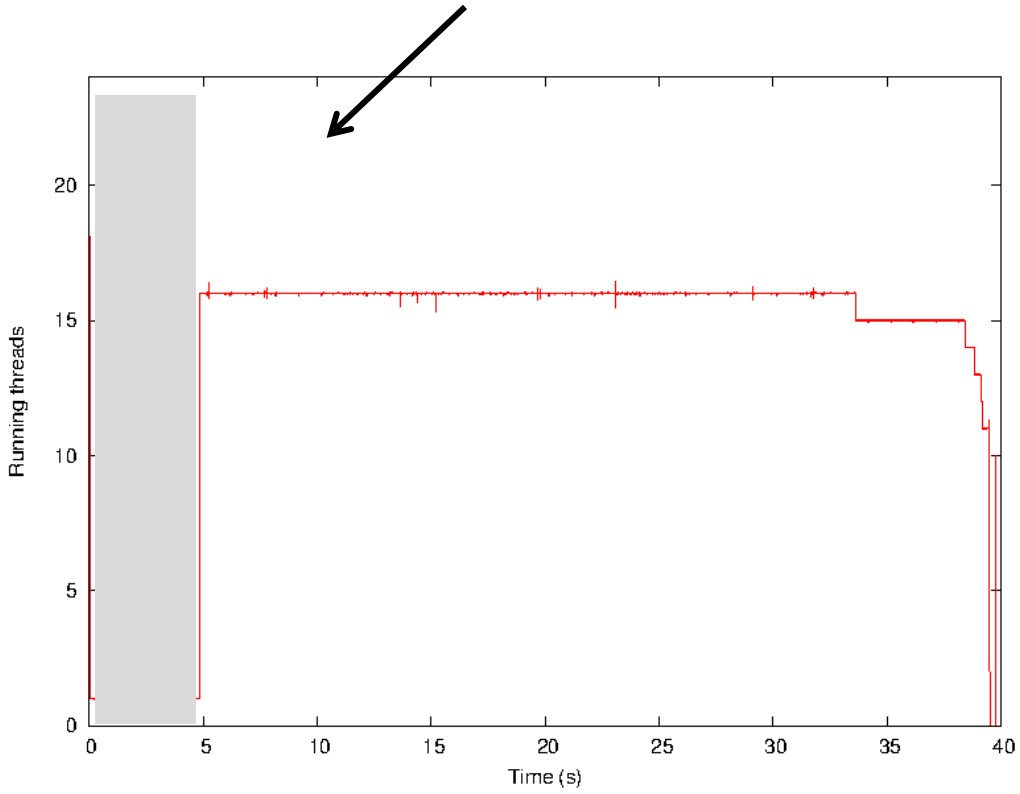
3: Workload demands: Green-Marl (Pagerank)





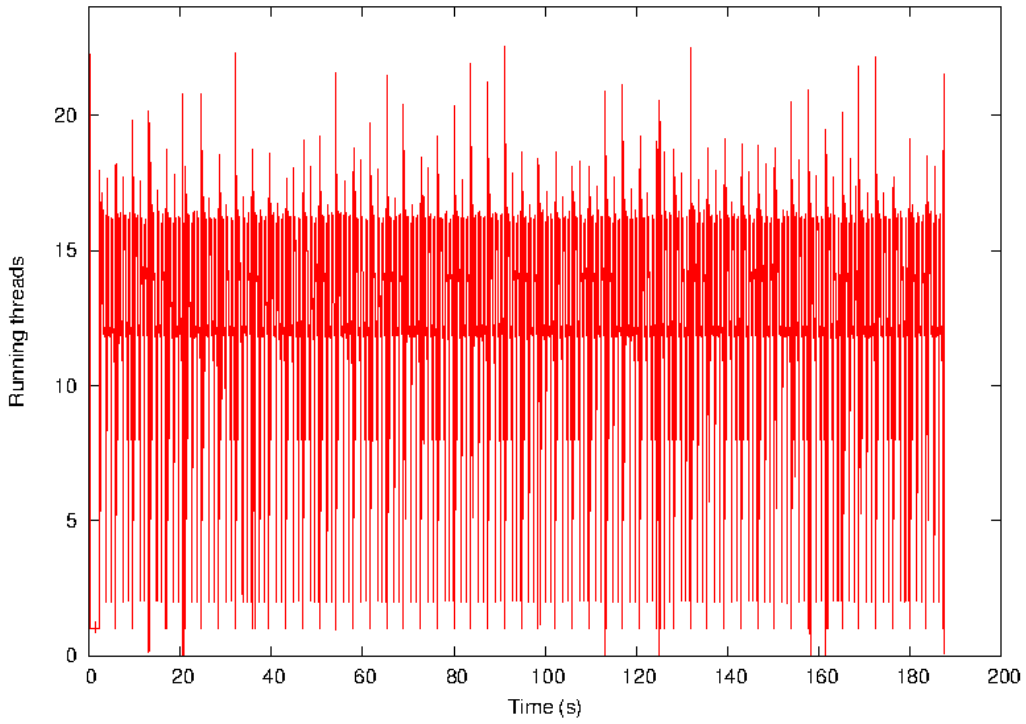
3: Workload demands: OMP (ART – neural network)

Mainly a single Open MP parallel region, dynamically scheduled



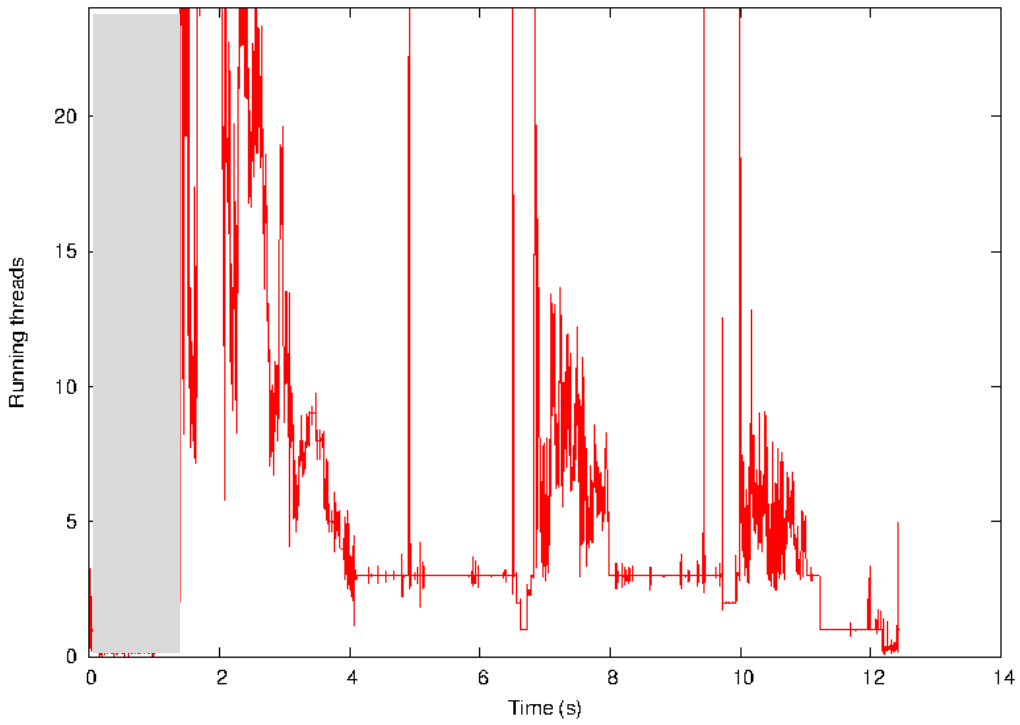
3: Workload demands: OMP (APSI – weather model)

Many parallel regions, linked by short sequential sections. Mean 13.5 running threads



3: Workload demands: DaCapo (PMD – source analysis)

Each of three iterations



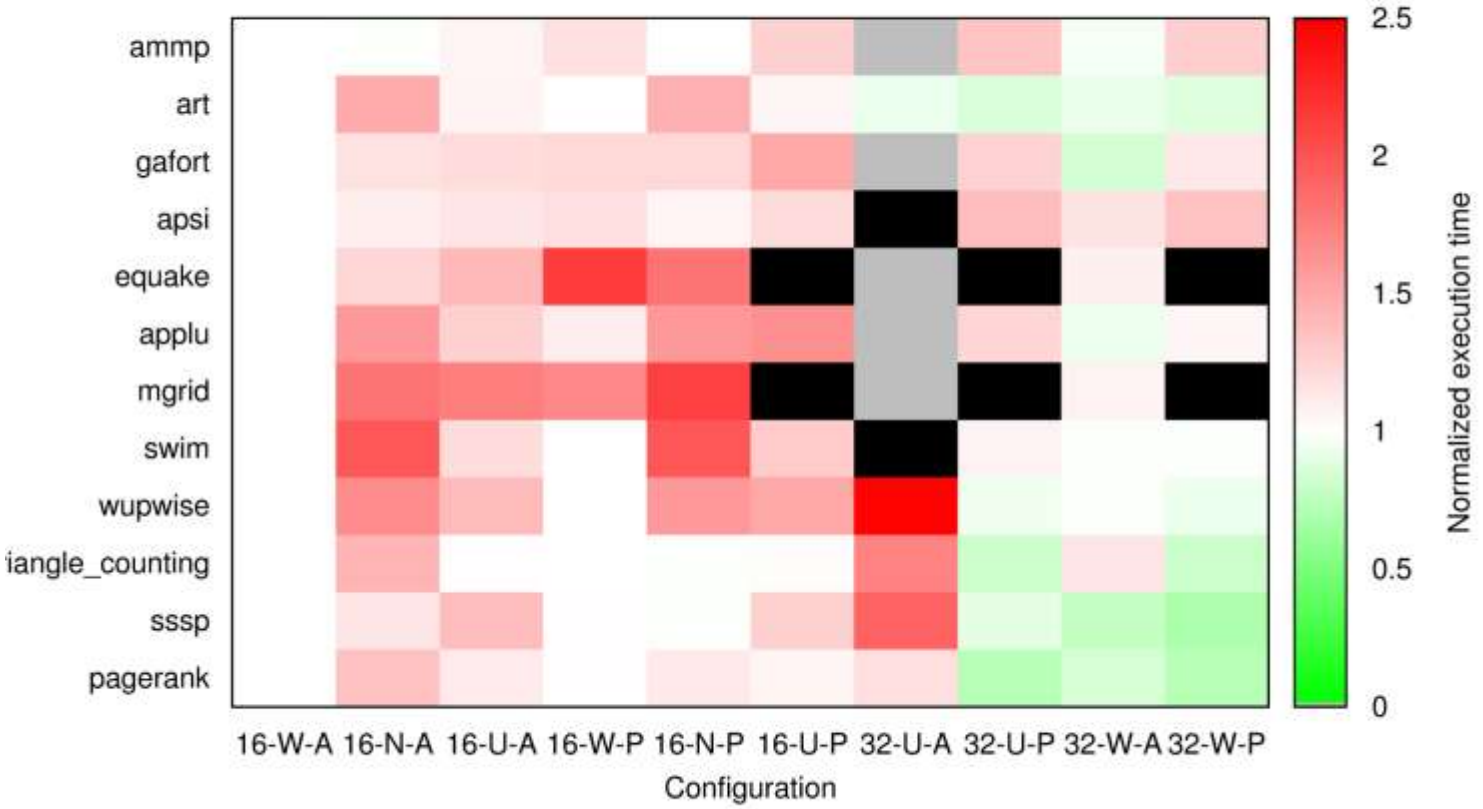
Motivation

1. The line between “single machine” and “distributed system” is getting blurred: shared components, often shared storage, while single machines exhibit partial failures.
2. Communication across these clusters is approx. the cost of communication across traditional big ccNUMA boxes.
3. Workload demands are burstier than traditional HPC and servers. We don’t get smoothing via handling of independent clients. Fluctuations on sub-second basis.

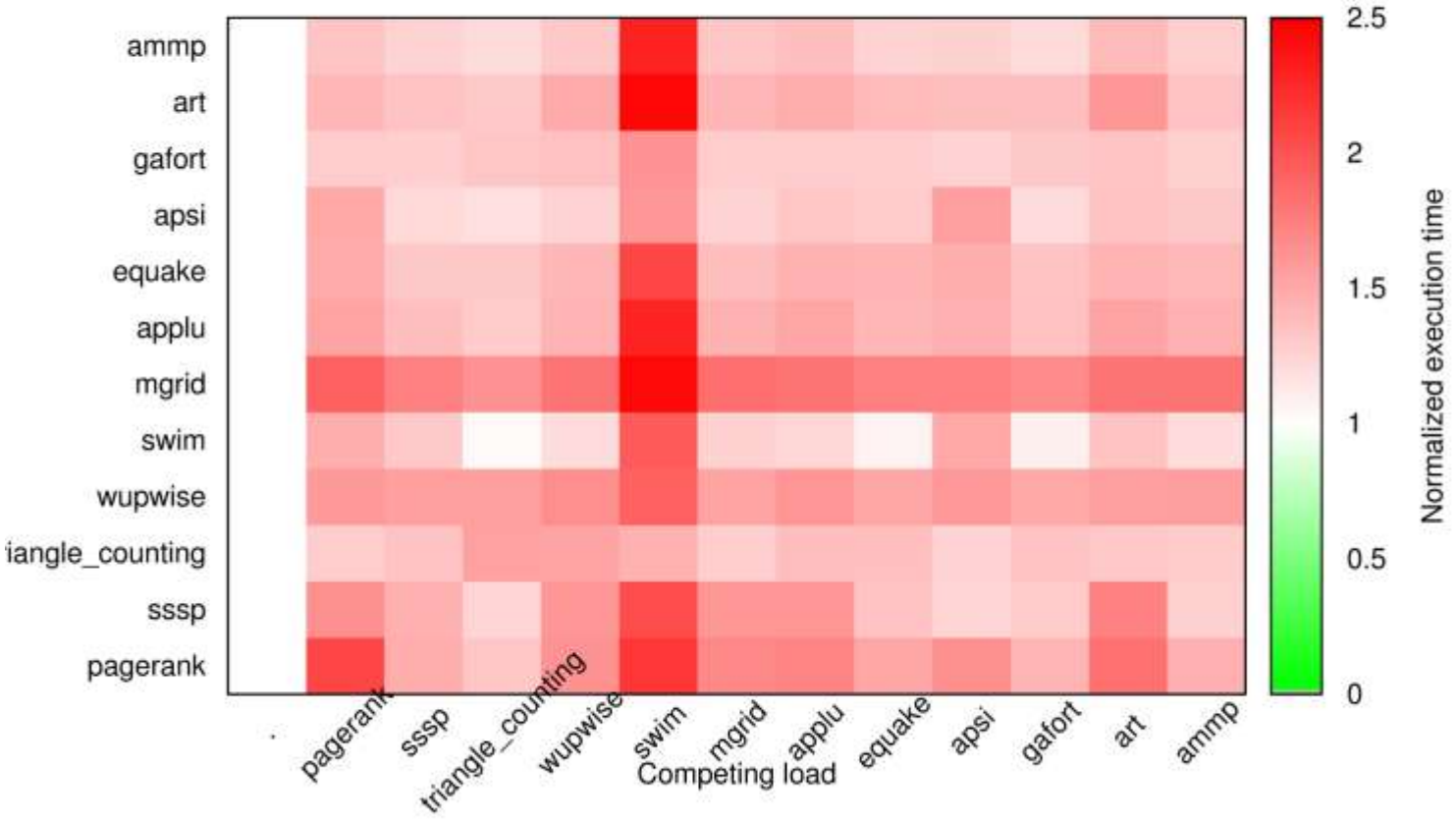
4: Consolidating workloads

- Look at a range of parallel workloads
  - Currently single-machine Green-Marl (soon distributed)
  - Currently SpecOMP 2001 on larger input sets (soon OMP 2012 and CPU)
- Run together on Bunch machines
  - 2-socket * 8-core * 2-way-HT
  - How well does the current OS and runtime system work under different configuration settings?
- Scheduler configurations:
  - 16-W-A
  - W (bound, wide), N (bound, narrow), U (unbound)
  - A (active waiting, spin), P (passive waiting, block)

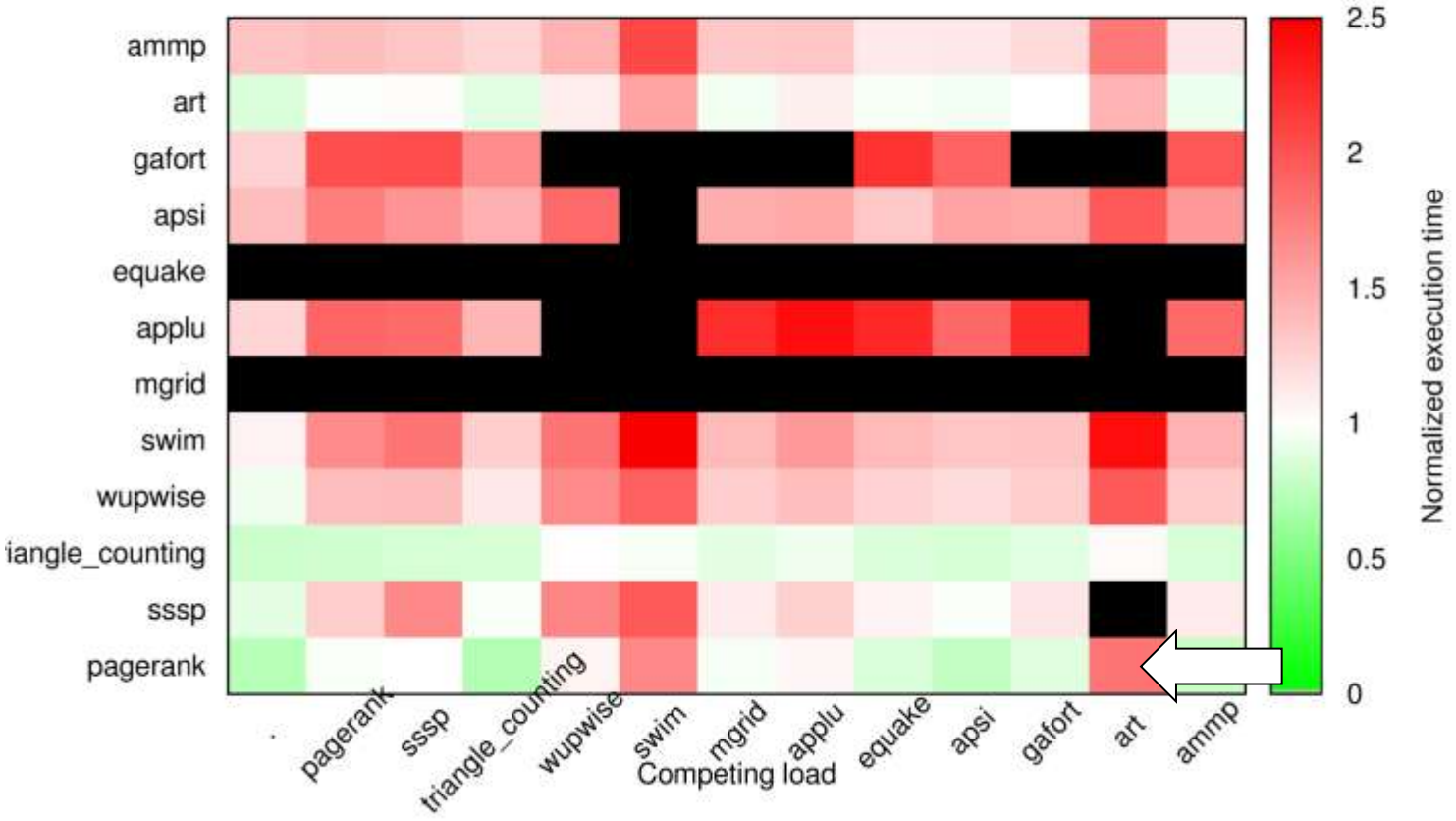
4: Consolidating workloads, running alone



4: Consolidating workloads, one hyperthread each



4: Consolidating workloads, 32-U-P



Motivation

1. The line between “single machine” and “distributed system” is getting blurred: shared components, often shared storage, while single machines exhibit partial failures.
2. Communication across these clusters is approx. the cost of communication across traditional big ccNUMA boxes.
3. Workload demands are burstier than traditional HPC and servers. We don’t get smoothing via handling of independent clients. Fluctuations on sub-second basis.
4. We can’t just run workloads together: uncontrolled pre-emption introduces stragglers. Current mechanisms for expressing policy ineffective (e.g., nice).

5. Distributed runtime systems

Cluster-OS work in the 80s-90s

- Exploit PCs and emerging LAN technology
- Explore new OS structure, services, and programming models
  - Often in the same project
- Software (re-)developed using new mechanisms e.g.,
  - IPC
  - Distributed object models
  - DSM
- Abstraction of diverse and evolving h/w

Today

- Common frameworks enable distribution
  - E.g., Hadoop
- In other domains, a shift to DSLs for which we can build distributed implementations
  - E.g., Green-Marl
- OS itself is off the data path in high-perf software
  - Direct access to virtual n/w hardware
  - Access to storage over the n/w
  - Ref early ideas from Nemesis & ExoKernel

Motivation

1. The line between “single machine” and “distributed system” is getting blurred: shared components, often shared storage, while single machines exhibit partial failures.
2. Communication across these clusters is approx. the cost of communication across traditional big ccNUMA boxes.
3. Workload demands are burstier than traditional HPC and servers. We don’t get smoothing via handling of independent clients. Fluctuations on sub-second basis.
4. We can’t just run workloads together: uncontrolled pre-emption introduces stragglers. Current mechanisms for expressing policy ineffective (e.g., nice).
5. An opportunity: so long as we support the required dependencies we do not need to develop new programming models.

Summary

- Clusters have traditionally supported HPC…
  - Workloads assigned to fixed machines / partitions of machines
  - Size often set when starting a job (e.g., MPI)
  - Early use of map-reduce style frameworks can look similar to HPC: long-running “embarrassingly parallel” tasks with occasional communication between them (or no “reduce” at all)
- …and long-running distributed / replicated workloads
  - Replication for HA, with monitoring and control of replicas
  - Changes in size over long timescales via management interfaces to add/remove nodes
- These workloads are handled well, but
  - Emerging workloads are not handled well
  - …and they make poor use of modern clusters

ORACLE

ORACLE IS THE INFORMATION COMPANY