# Job Coscheduling on Coupled High-End Computing Systems

Wei Tang*, Narayan Desai#, Venkatram Vishwanarth#
Daniel Buettner#, Zhiling Lan*

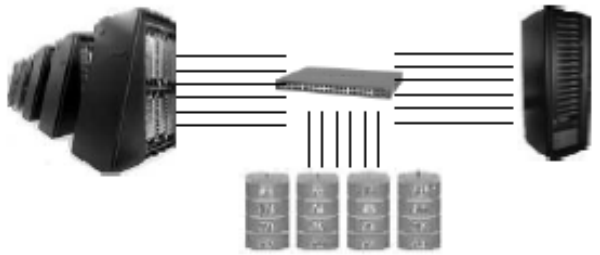
* Illinois Institute of Technolology
# Argonne National Laboratory

# Outline

- Background & Motivations
- Problem Statement
- Solutions
- Evaluations

# Background

- **Coupled systems are commonly used**
  - Large scale system: computation, simulation, etc
  - Special-purpose system: data analysis, visualization, etc
- **Coupled applications:**
  - Simulation / computing applications
  - Visualization/data analysis applications
  - Example: FLASH & vl3, PHASTA & ParaView



# Coupled systems examples

- **Intrepid & Eureka @ANL**
  - Intrepid: IBM Blue Gene/P with 163, 840 cores (#13 in Top500)
  - Eureka: 100-node cluster with 200 GPUs (largest GPU installation)
- **Ranger & Longhorn @TACC**
  - Ranger: SunBlade with 62,976 cores (#15 in Top500)
  - Longhorn: 256-node Dell Cluster, 128 GPUS
- **Jaguar & Lens @ORNL**
  - Jaguar: Cray XT5 with 224, 162 cores (#3 in Top500)
  - Lens: 32-node Linux cluster, 2 GPUs
- **Kraken & Verne @ NICS/UTK**
  - Kraken: Cray XT5 with 98,928 cores (#8 in Top500)
  - Verne: 5-node Dell cluster.
- **And so on … …**

# Motivation

- **Post-hoc execution**
  - Computing applications write data to storage system, and then analysis applications read data from storage system and process
  - I/O time consuming
- **Co-execution is increasingly demanded:**
  - Saving I/O time by transfer data from simulation application to visualization/data analysis (an ongoing project named GLEAN)
  - Co-execution enables monitoring simulations, debugging at runtime
  - Heterogeneous computing

# Problem statement

- System A and B running parallel jobs
  - Job schedulers / scheduling policies are independent
  - Job queues are independent
- Some of jobs on A has associated (mate) jobs on B.
  - Mate jobs are in pairs: one on A, the other on B
- Co-scheduling Goal:
  - **Guarantee the mate jobs in the same pair start at same time on their respective hosting system *without manual reservation***
  - Limit the negative impact of system performance and utilization.

# Related work

- Meta scheduling
  - Managing jobs on multiple clusters via a single instance
  - Moab by Adaptive Computing Inc, LoadLeveler by IBM
  - Our work is more distributed. Different scheduler running on independent resource management domain can coordinate job scheduling.
- Co-Reservation
  - Co-allocation of compute and network resources by reservation
  - HARC (Highly-Available Resource Co-allocator) by LSU
  - Our work doesn't involve manual reservation; co-scheduling is automatically coordinated.

# Basic schemes

- When a job can start to run on a machine while its mate job on the remote machine cannot, it may “hold” or “yield”.
- **Hold**
  - Hold resources (nodes) which cannot be used by others until the mate job can run
- **Yield**
  - Give up the turn of running without holding any resources

# Algorithm

**Algorithm 1:** RunJob($j$, $N$)

**Input**: A scheduled job $j$ with assigned nodes $N$
**Result**: Job $j$ either starts, or holds, or yields. Its remote mate job $k$, if existing, could be triggered to start under certain condition.

```
1  if cosched_enabled then
2      k = remote.getMateJobId(j)
3      if k then
4          mate_status = remote.getMateStatus(k)
5          switch mate_status do
6              case "unsubmitted"
7              case "queuing"
8                  mate_started = remote.tryStartMate(k)
9                  if mate_started then
10                     self.startJob(j, N)
11                 end
12                 else
13                     if self.scheme == "hold" then
14                         self.holdJob(j, N)
15                     end
16                     if self.scheme == "yield" then
17                         self.yieldJob(j)
18                     end
19                 end
20             endsw
21             case "holding"
22                 self.startJob(j, N)
23                 remote.startJob(k)
24             endsw
25             case "unknown"
26                 self.startJob(j, N)
27             endsw
28         endsw
29     end
30     else
31         self.startJob(j, N)
32     end
33 end
34 else
35     self.startJob(j, N)
36 end
```

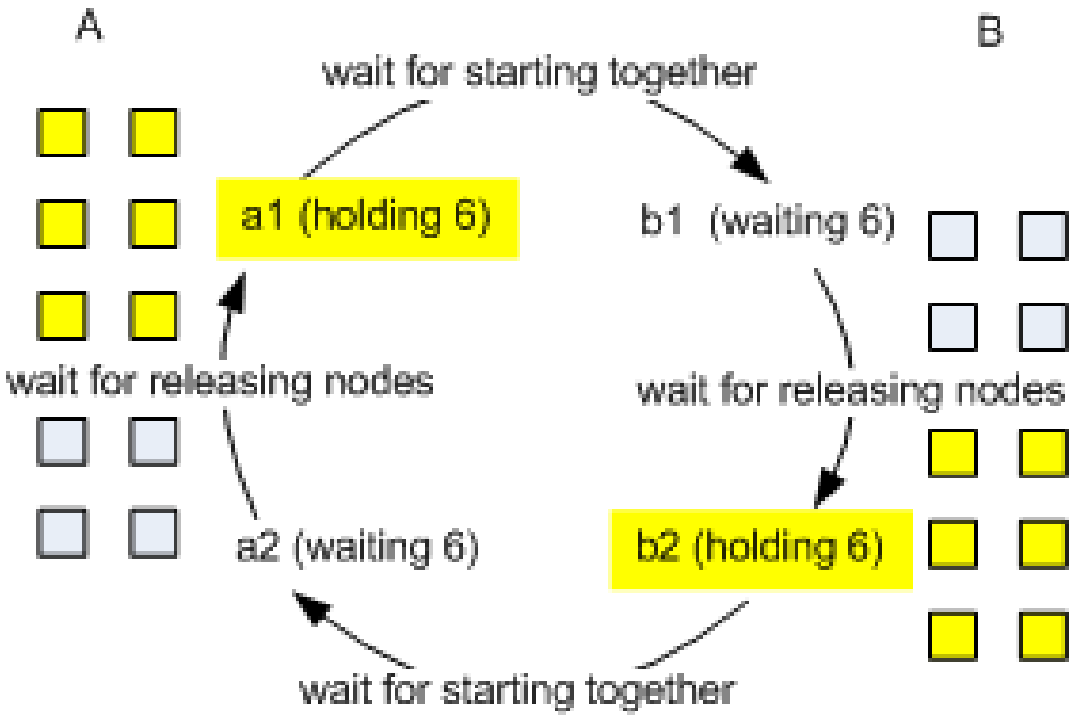

A job (a) can run
Cosched Enabled?
N
Y
Has mate job (b)?
N
Y
Mate status == "hold"?
N
Y
Mate status == "queuing" or "unsubmitted"?
N
Y
Remote.start_mate(b)
Remote.try_run_mate(b)
Mate started?
N
Y
Local Scheme?
Hold
Yield
hold_job (a)
yield_job (a)
start_job (a)

# Strategies combination

- Hold-Hold
  - Good for the sync-up of mated jobs
  - Bad for system utilization
  - May cause deadlock
- Yield-Yield
  - No hurt for system utilization
  - Bad for mated jobs waiting
- Hold-Yield (or Yield-Hold)
  - Behave respectively

# Deadlock

- Coupled systems A & B, both use “hold” scheme
- Circular wait (a1→b1→b2→a2→a1)



# Enhancements

- Solving Deadlock
  - Release all the held nodes periodically (e.g. every 20 minutes)
- Reduce overhead
  - Threshold for yielding times
- Fault-Tolerance consideration
  - A job won’t wait forever when the remote machine is down

# Evaluation

- Event-driven simulation using real job trace from production supercomputers.
- Qsim along with Cobalt resource manager.

# Experiment goals

- Investigate the impact by tuning system load
- Investigate the impact by tuning the proportion of paired jobs.

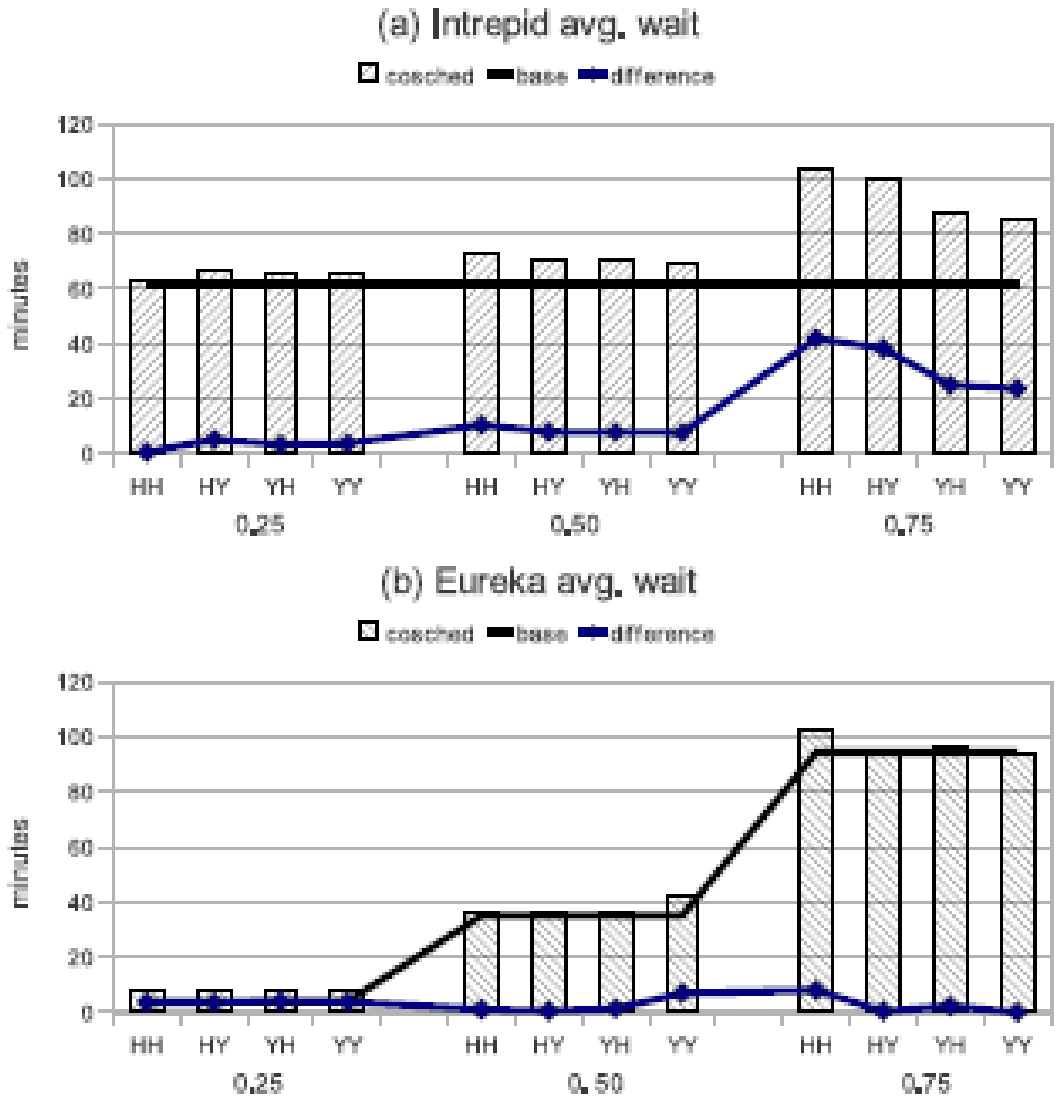
# Job traces

- Intrepid (real trace)
  - One month, 9220 jobs, sys. Util. 70%
- Eureka (half-synthetic,  packed into one month)
  - Trace 1: 5079 jobs, sys. Util. = 25%
  - Trace 2: 11000 jobs, sys. Util. = 50%
  - Trace 3: 14430 jobs, sys. Util.  = 75%
  - Synthetic: 9220 jobs. Sys. Util. = 48%

# Evaluation Metrics

- Avg. waiting time
  - Start time – Submission time
  - Average among total jobs
- Avg. slowdown
  - (wait time + runtime) /runtime
  - Average among total jobs
- Mated job sync-up overhead
  - How many more minutes need to wait in co-scheduling
  - Average among all paired jobs
- Loss of computing capability
  - Node-hour
  - System utilization rate

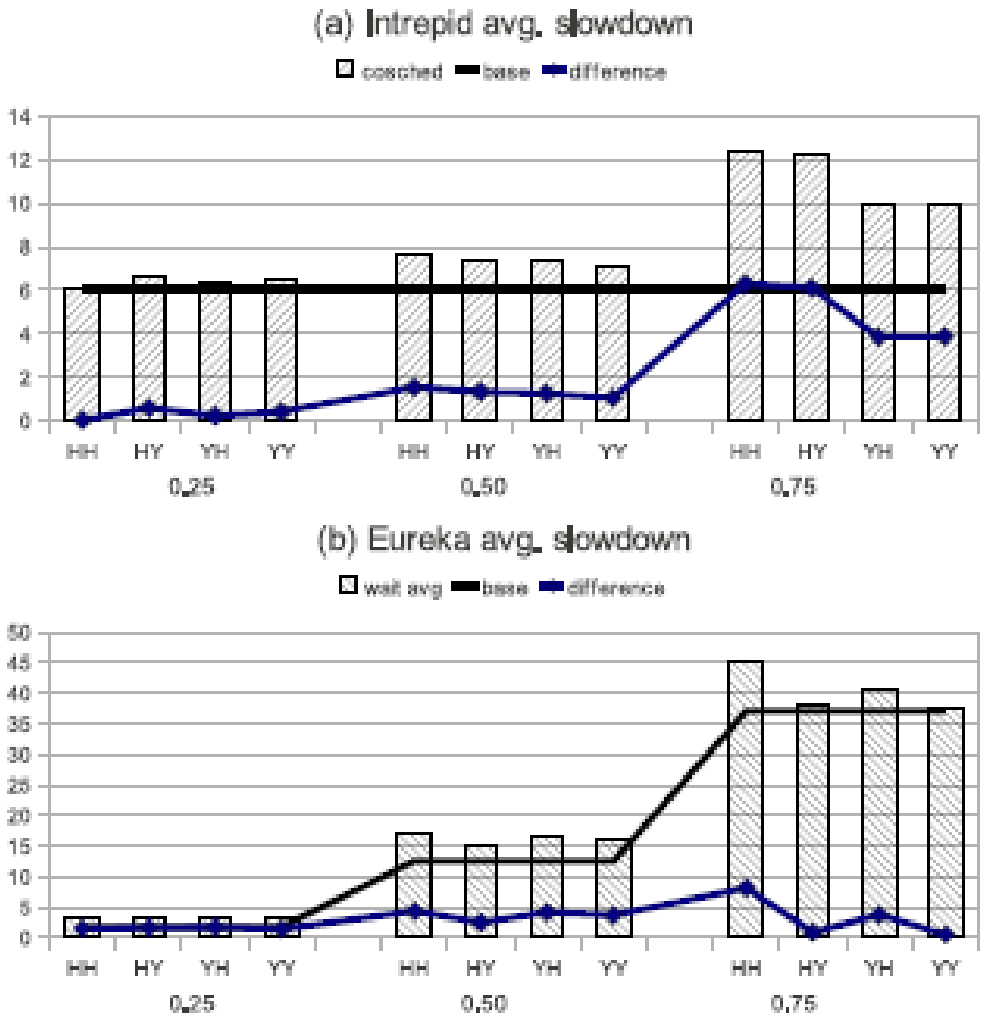
# Average wait by sys. Util.




Scheme on Intrepid-Eureka
HH: Hold-Hold
HY: Hold-Yield
YH: Yield-Hold
YY: Yield-Yield

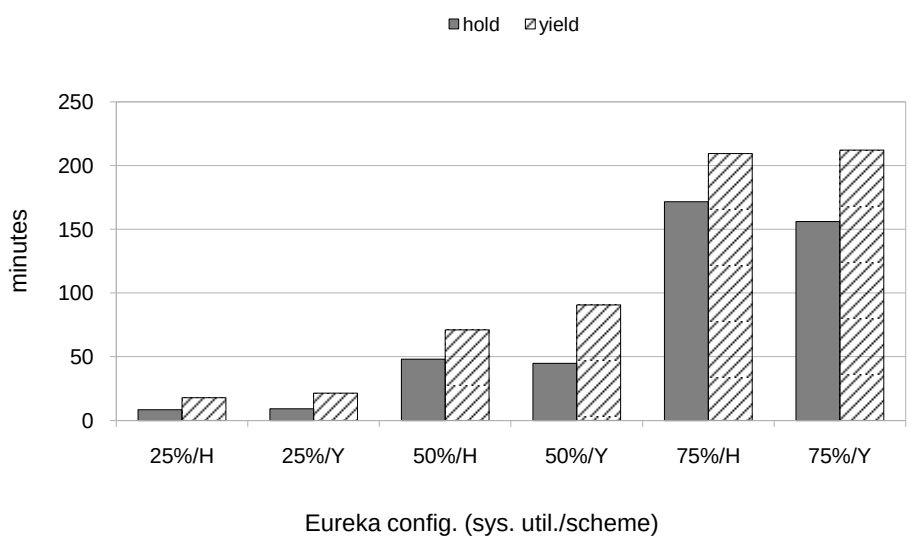
Sys util. on Eureka:
25% 50% 75%

# Slowdown by sys. Util.

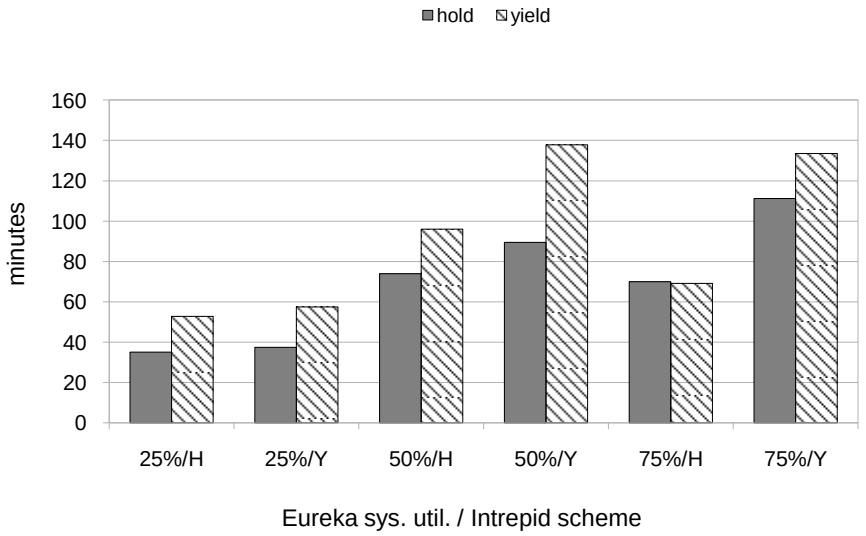


# Coscheduling overhead by sys. Util.

Intrepid job sync-up overhead (average)



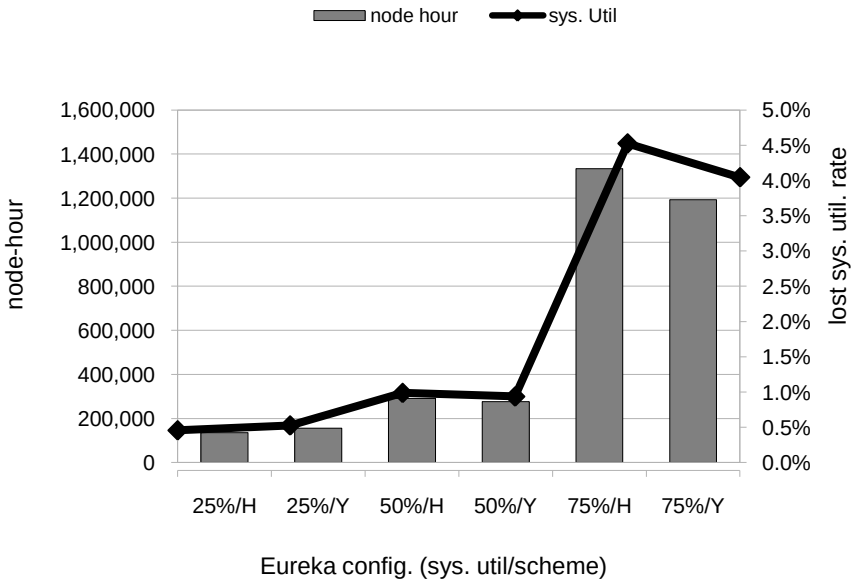

Eureka sync-up overhead (average)



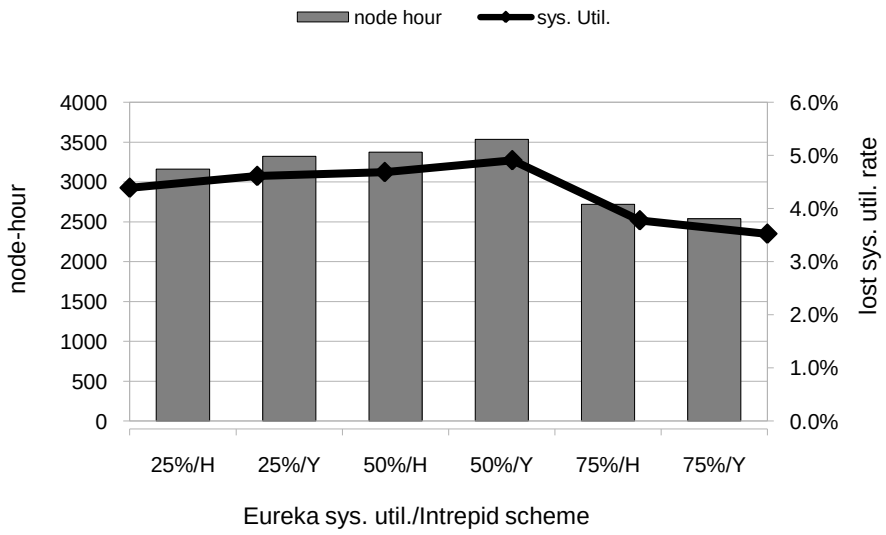

Using yield costs more sync-up overhead than using hold

# Loss of computing capability by sys. Util.

Intrepid loss of computing capability

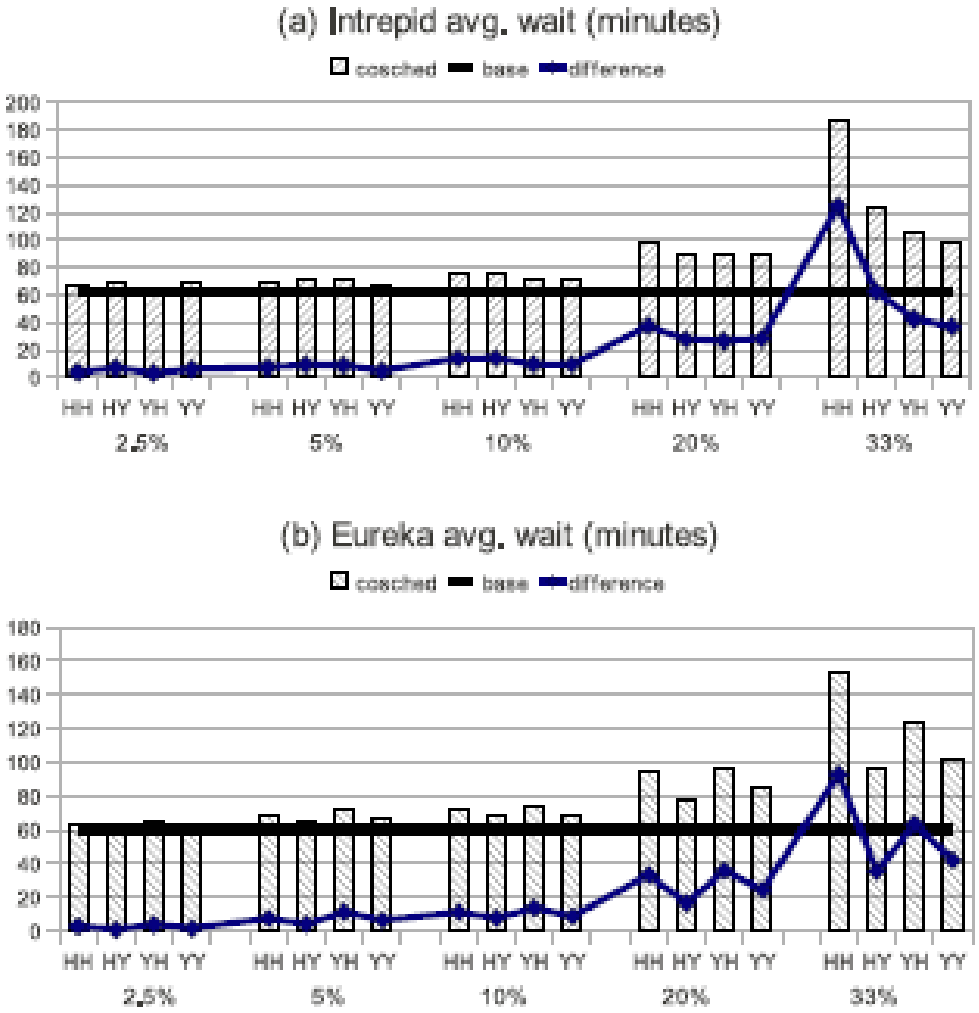



Eureka loss of computing capability

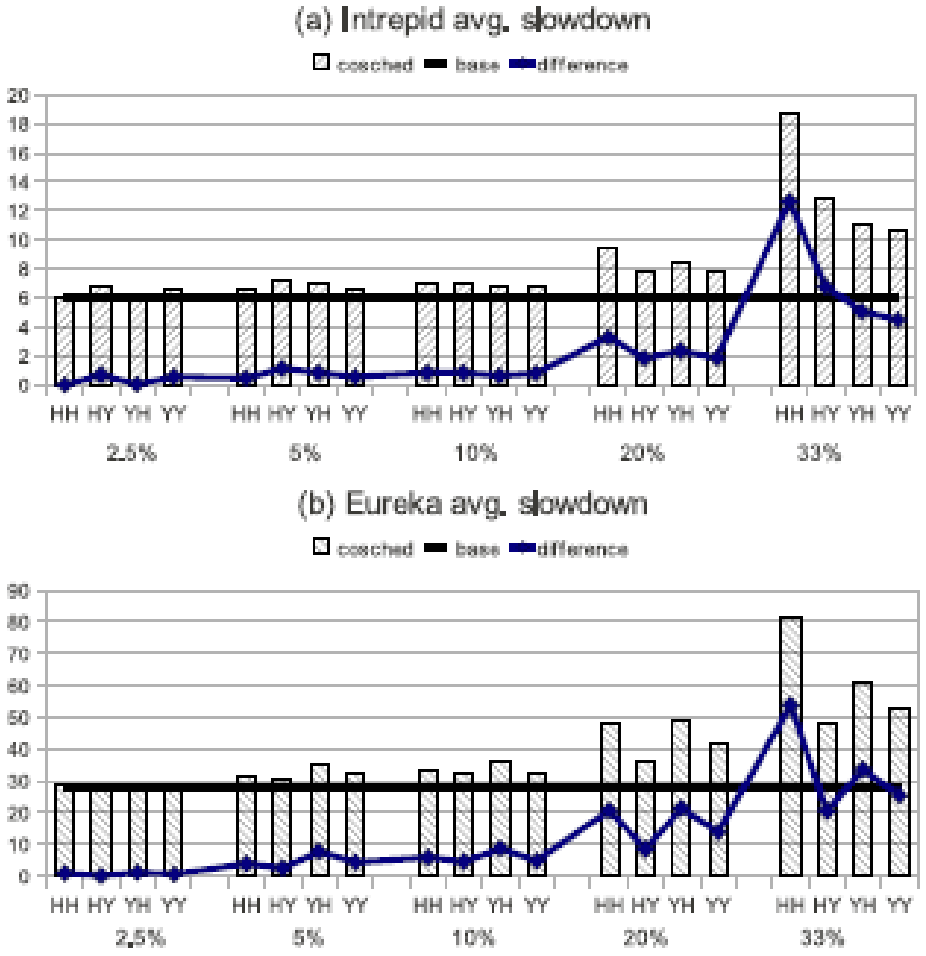



Util loss is caused only by using “hold”

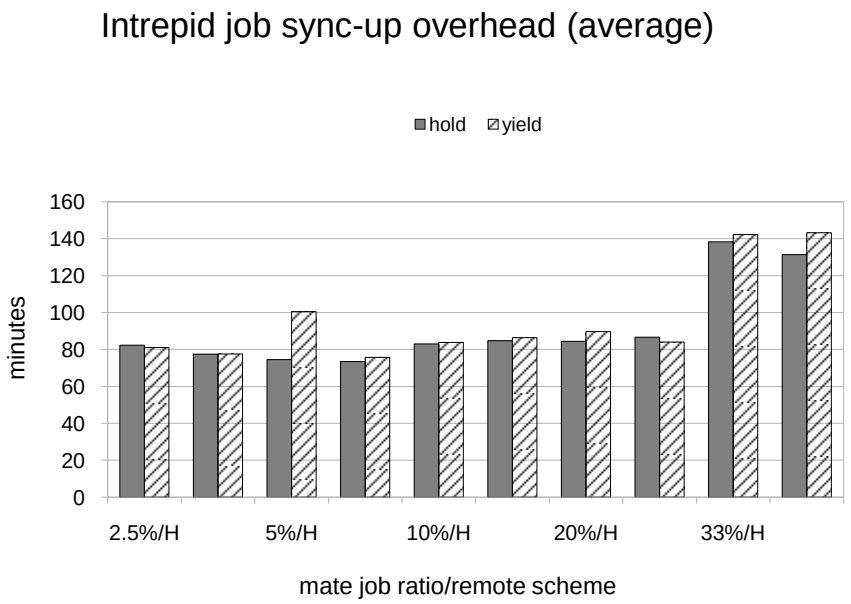
# Avg. wait by proportion of paired jobs

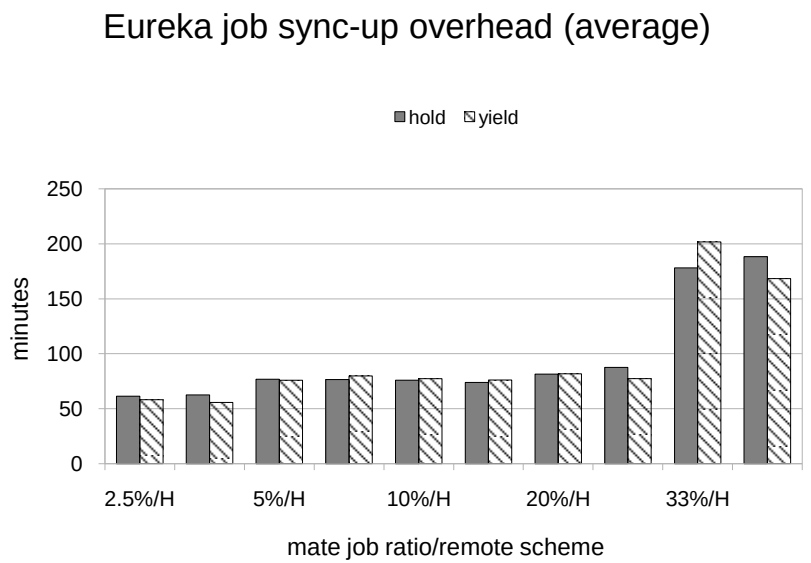


# Slowdown by proportion of paired jobs



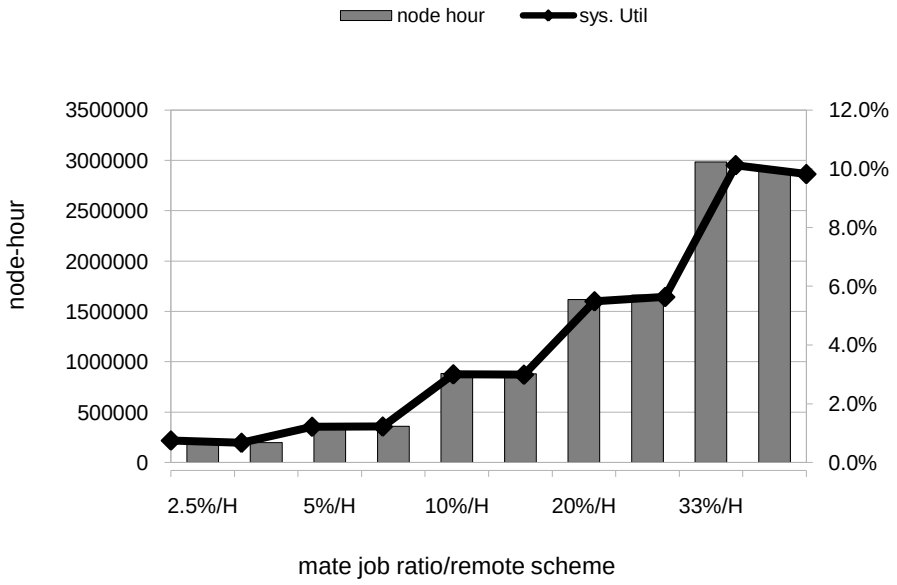
# Overhead by proportion of paired jobs



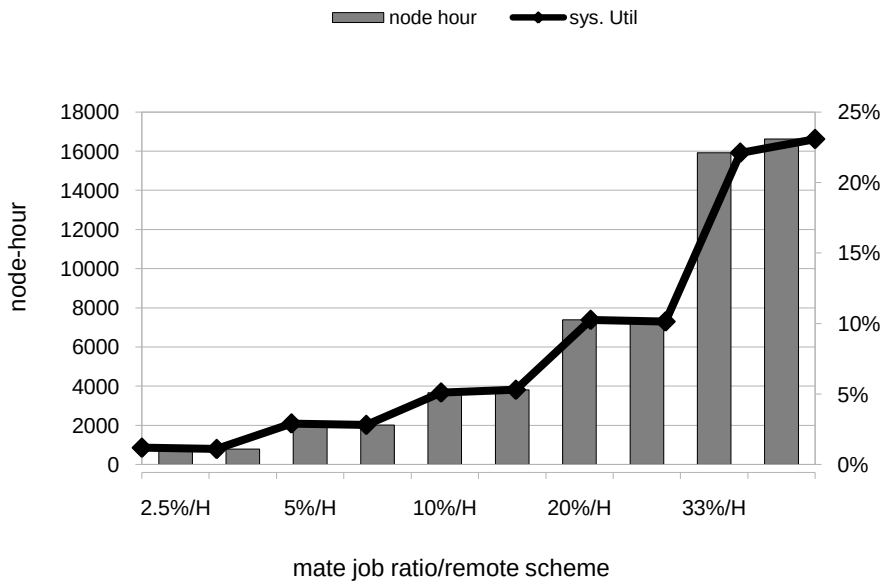



# Loss of computing capability by proportion of paired jobs

Intrepid loss of computing capability




Eureka loss of computing capability



# Summary

- Designed and implemented coscheduling algorithm to start associated at the same time in order to fulfill multiple needs of certain applications, such as reducing I/O overhead in coupled HEC environment.
- Evaluated the coscheduling impact on system performance and overhead for jobs needing co-scheduling.
- Conclusion: coscheduling can work with some acceptable overhead under different system utilization rate and proportion of mated jobs.

Thank you!