Execution of Bags of Tasks on Multiple Resource Pools

Mark Silberstein, Artyom Sharov
Dan Geiger, Assaf Schuster

Technion – Israel Institute of Technology
2009

Experimental science life cycle

Problem

- BOT – Bag of Tasks
  - Independent tasks
  - Single parallel run
  - All results are required

- Multiple BOTs
  - Different parallel runs
  - From tens to millions of tasks, minutes to hours each

How to run *efficiently* ?

- Resources
  - Opportunistic
  - Non-dedicated
  - Restricted connectivity

- Multiple resource pools
  - Dedicated, collaborative, volunteer, pay-as-you-use
  - No unified management

Zoo of resource pools

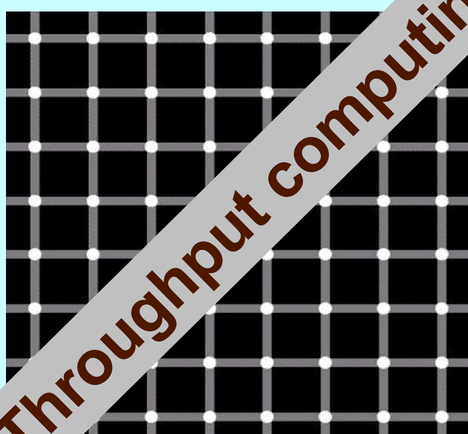
Reliability

Low → High

Performance predictability

Low → High

Opportunistic



Throughput computing

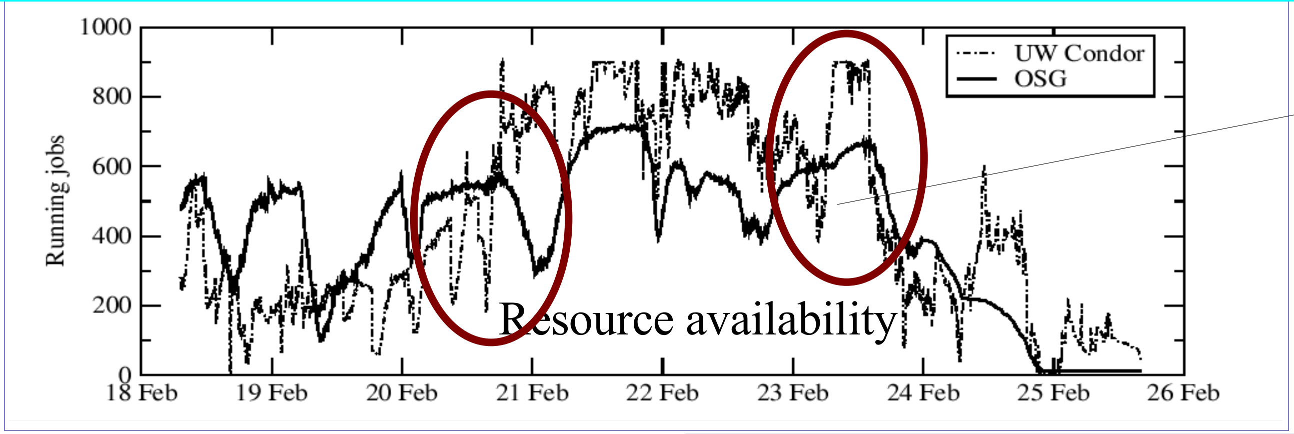
Dedicated

“Burst” computing

Potential amount of available resources

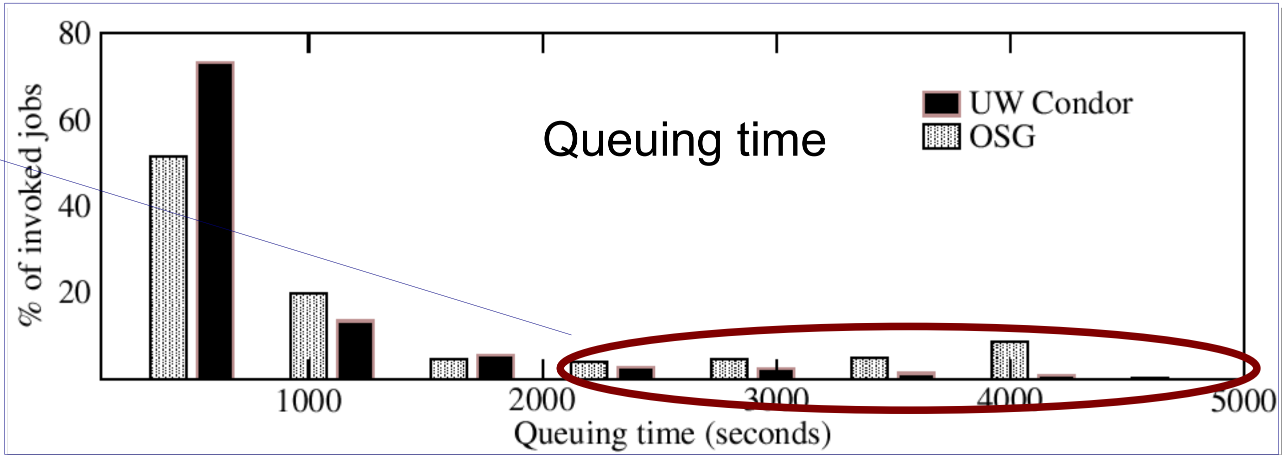
High ← Low

Grids in a wild



Sharp changes

10-20% started after one hour



Grid	#Jobs	Preempted (%)	Failed (%)
UW Madison	96938	20	1%
OSG	60648	10	1%
EGEE	16437	7	2%
Technion **[Dedicated]**	42411	2	0.2%
Community grid (~15,000 hosts)	241364	(♣)0.2	13%

Up to 21% failures

Solution in a nutshell

Separate resource allocation from scheduling

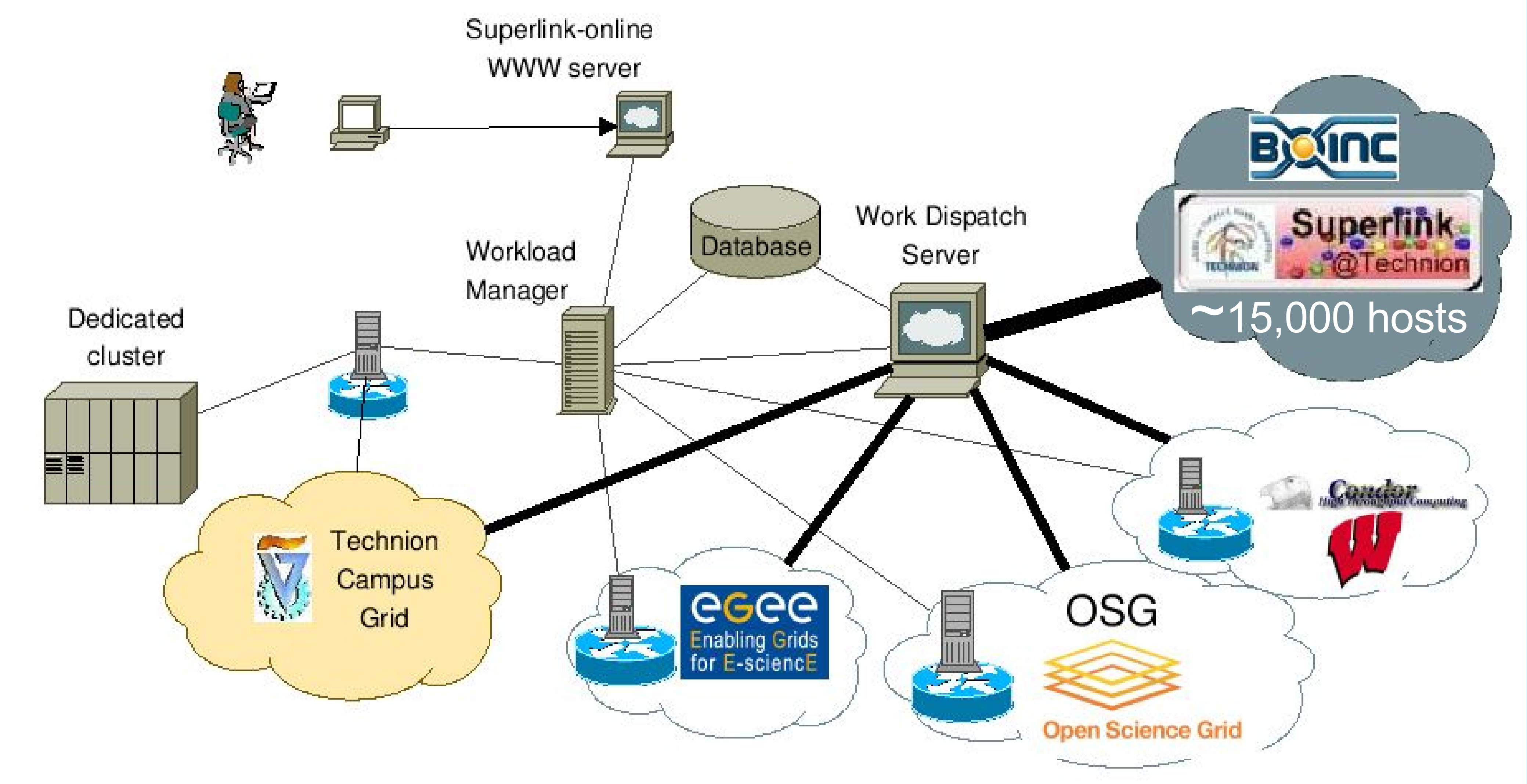
On-demand unified **virtual cluster** over resource pools

BOT – first class citizen
Run-time policy-driven BOT execution mechanisms

Scalable implementation deployed over multiple resource pools

Production deployment: Superlink-online genetic analysis portal

Check out our online monitoring system: http://tiny.cc/GridBot

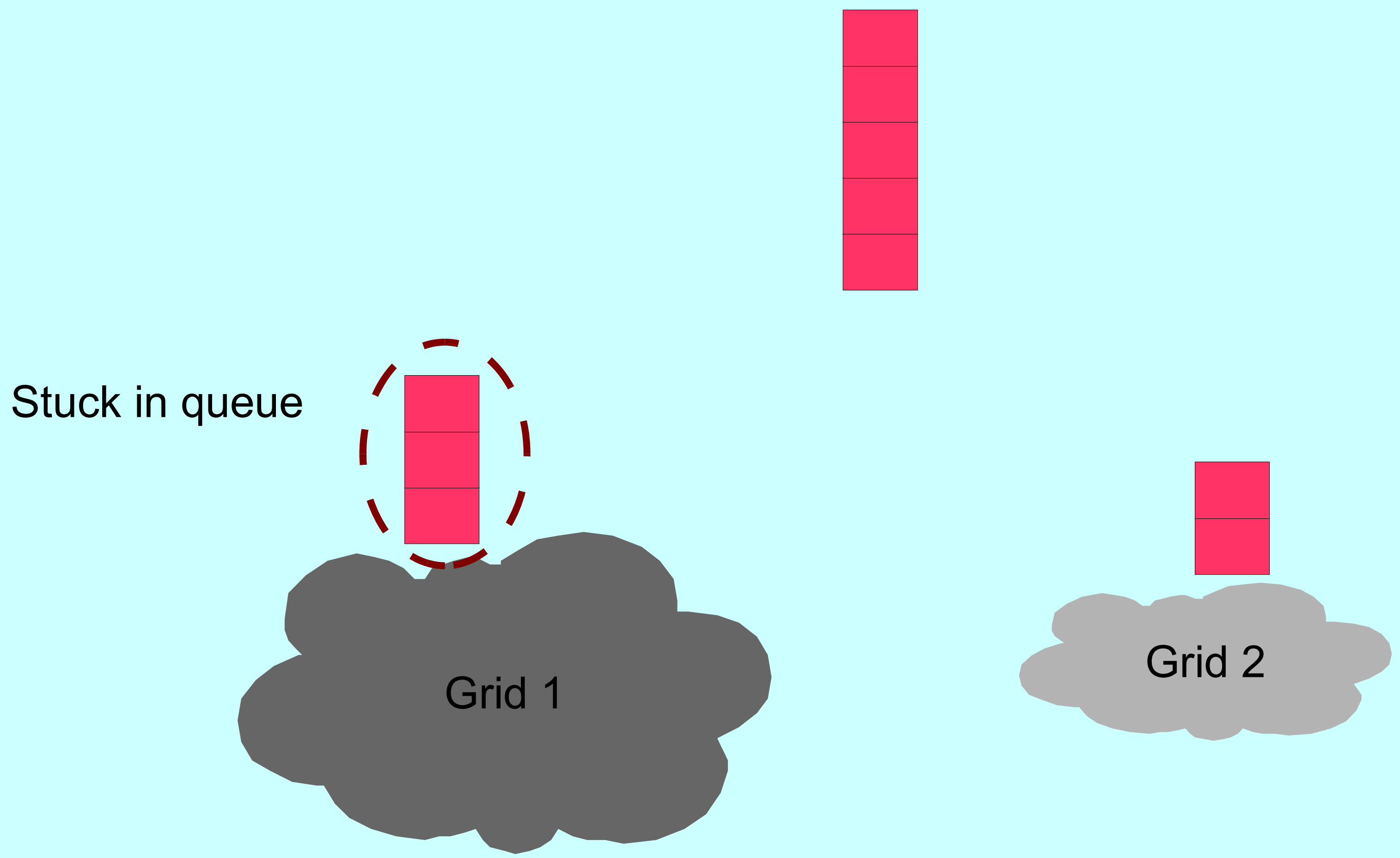


~25,000 active hosts in 3 months

Outline

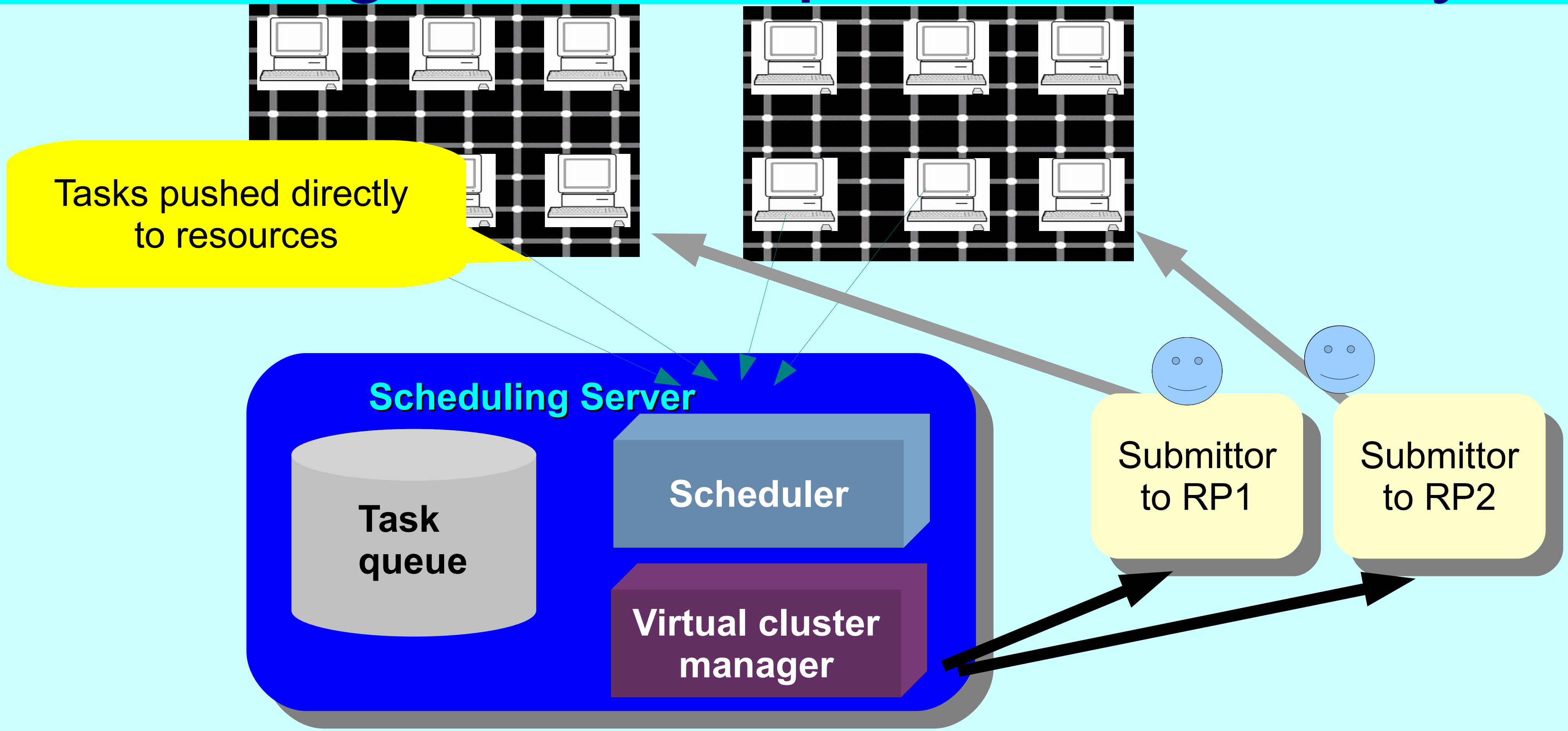
- **Challenges**
- Policy-driven mechanisms
- Implementation
- Experiments
- Conclusions

Challenge 1:
Efficient work dispatch



1. Any static solution will result in load imbalance
2. High scheduling overhead penalty per task

Resource allocation: Gluing resource pools via overlay

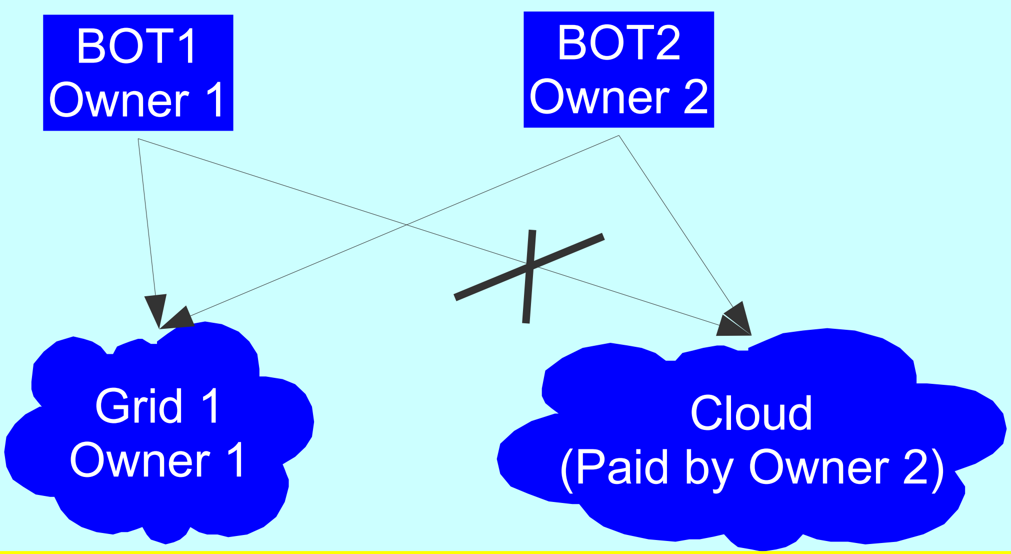


- Naturally scalable to more resource pools
- Dynamic load balancing between resource pools
- Reduced task granularity – no scheduling overhead
- Application-specific scheduling

How to distribute resource requests between grids ?

Challenge 2:
Multi-BOT Multi-RP scheduling

- Different BOTs have different requirements
- Example:
  - Larger BOTs (millions tasks) delay shorter ones
  - Naïve solution: priority queue
- Does it suffice for multiple resource pools?



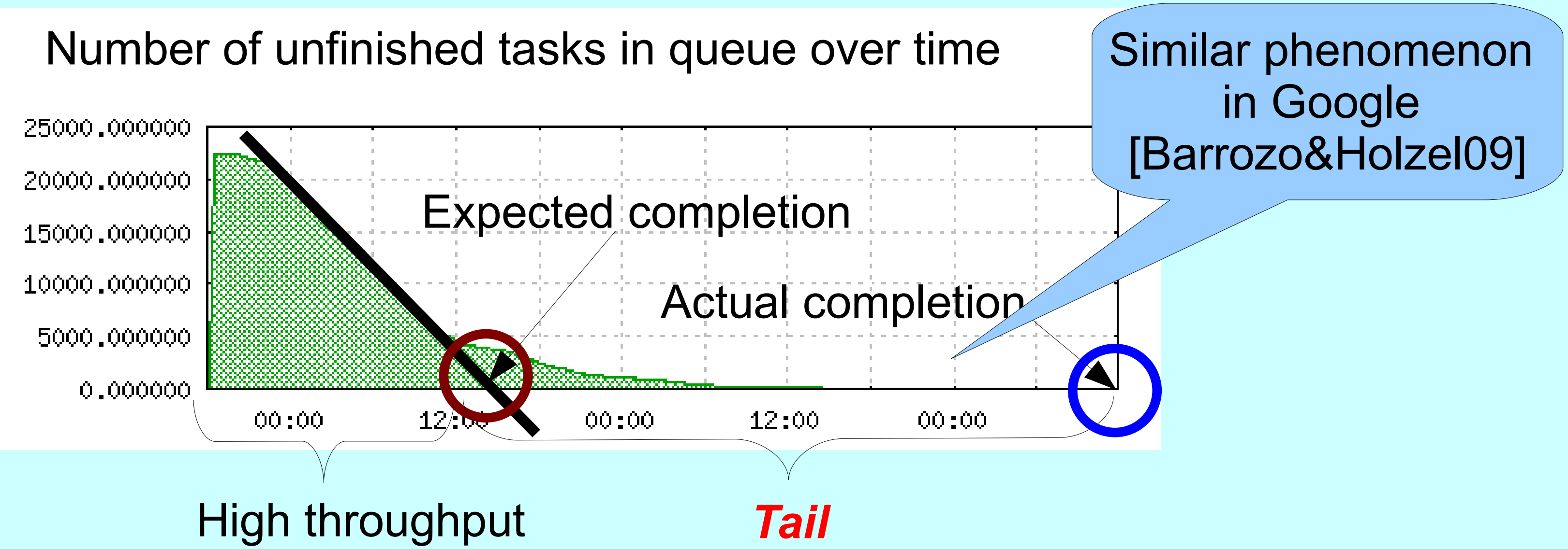
Desired behavior:
BOT1 prioritized on Grid1,
disallowed on Cloud
BOT2 allowed on Grid2

Runtime host-specific policy

Runtime scheduling policies

- Host-specific
  - Execute a task only on reliable hosts (RPs)
- Conditional Task **bundling** (for shorter tasks)
  - Pack 10 tasks to reliable host, 2 to unreliable, 100 to GPU-enabled
- Host-specific priority
  - Multilevel Feedback Queue Scheduling
    - the larger the BOT – the lower its priority

Challenge 3:
Long tail in large-scale systems



Grid	#Jobs	Preempted (%)	Failed (%)
UW Madison	96938	20	1%
OSG	60648	10	1%
EGEE	16437	7	2%
Technion **[Dedicated]**	42411	2	0.2%
Community grid (~15,000 hosts)	241364	(♣)0.2	13%

Need for BOT turnaround-time optimizations

Replication

- Common practice to reduce BOT turnaround in faulty environments
- *Speculatively* invoke r... multiple times
  - Concurr... ple task *replicas,* first ... esult is accepted

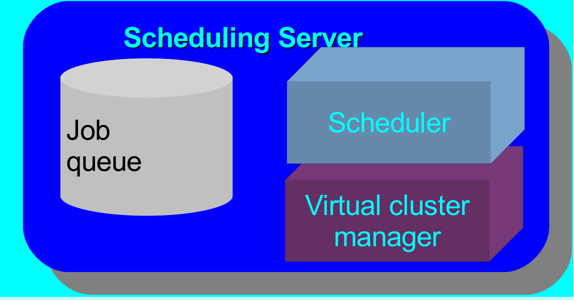
But replication is wasteful!

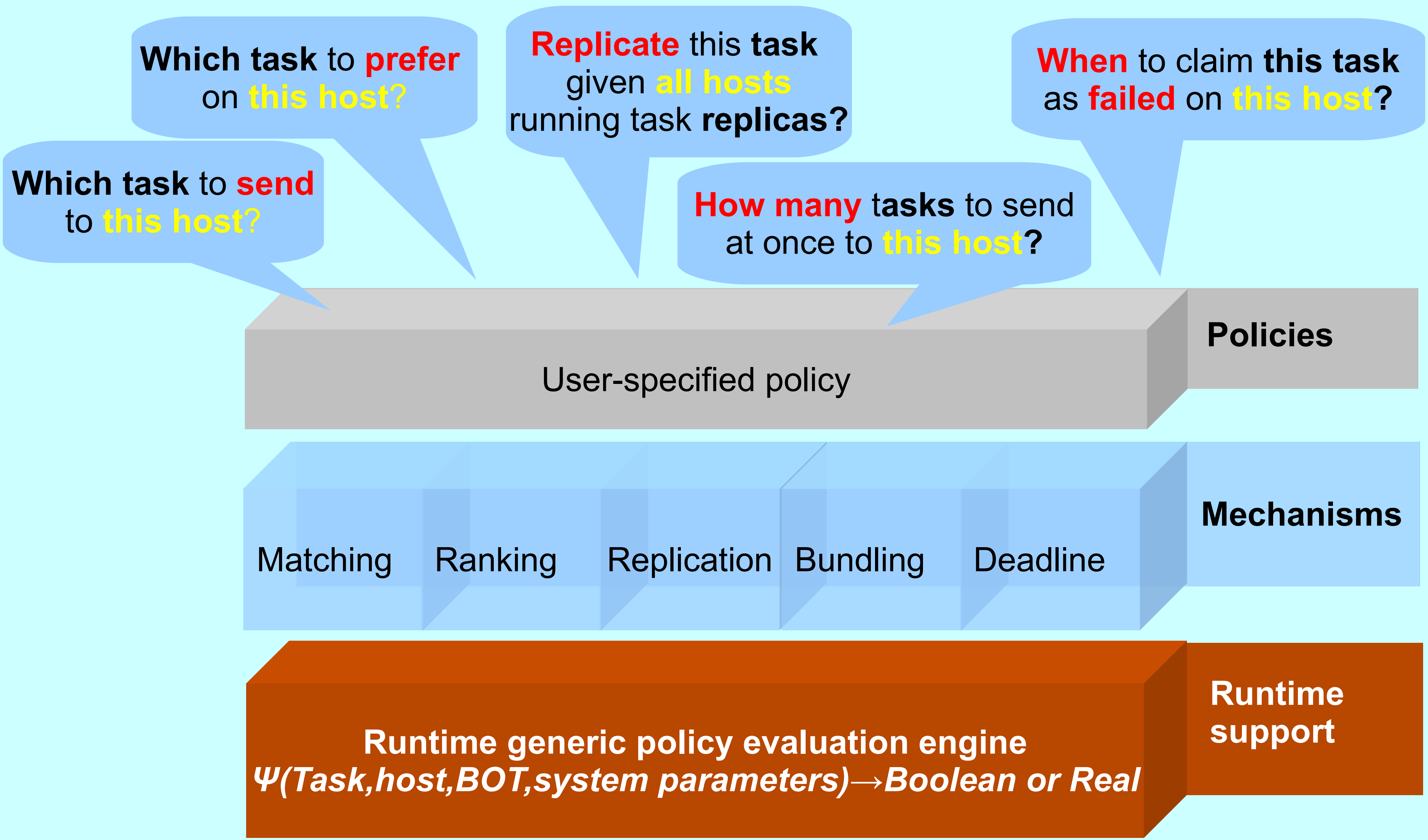
Need to allow BOT-specific replication policies to adjust cost – turnaround time tradeoff

Example - conservative: replicate if all task's other replicas are running too long on unreliable or slow machines

Example - full: replicate every task twice if in Tail

Scheduling: work-dispatch and replication logic





Runtime support: classads

- Schema-less list of name-value attributes
- Attribute can be *a function* of other attributes
- Classad runtime allows on-demand evaluation

```
Host=[  RAM=2048;
        CPUbench=3;
        Performance=RAM*CPUbench;
        Quality=Performance*2.5;
     ];
```

Runtime query: `Host.Quality` yields `15360`

- Convenient for runtime policy specification:
  - Quality, Performance – user specified functions
  - RAM, CPUbench – updated by the infrastructure

Replication policy example

```
Job= [ Name="job1";
       Executable="/bin/hostname";
       NumberOfReplicas=2;
       Replica1= [ Name="job1_1";
                   Host=[ Name="is3.myhost";
                          SentTime=242525;
                          ErrorRate=0.08;]
                 ];
       Replica2= [ Name="job1_2";
                   Host=[ Name="is2.myhost";
                   SentTime=242525;
                   ErrorRate=0.08;]
                 ];
     ];
```

Running replica 1

Running replica 2

```
ReplicationRequirements=
  (NumberOfReplicas<3&&
  Job.Replica1.Host.ErrorRate>0.1);
```

Reference to the properties of the host where the replica is running

Implementation challenge: Connectivity and scalability

- Connectivity
  - Firewalls/private networks
- Support for large number of task requests
  - Too many open connections
  - Scheduling overhead (per request) affects throughput

Implementation

- Enhanced Berkeley Open Infrastructure for Network Computing (BOINC) with support for dynamic policies
  - Production middleware used for establishing community grids (e.g SETI@HOME)
  - Uses HTTP protocol from client to server for control and data
  - Disconnected mode of operation
  - Well-suited for faulty environments
- Use of BOINC allows *for the first time* interoperability between *all types of grids*

Grid Overlay

- BOINC clients are sent to grids as regular jobs
  - Obey local grid policy
- Avoid resource underutilization
  - Self-terminate when idle

Scheduling scalability

- Policy evaluation performed per request for every task (millions) in a queue!
- Optimization 1:
  - *Representative sample*: sample constant number of jobs from every BOT

Consider only these jobs

  - *Complexity: O(#BOTs)*
- Optimization 2:
  - Observation: jobs of a BOT almost always have the same scheduling properties

Evaluate the classad only once per BOT

Replication scalability

- Replication is performed periodically for some subset of ***running*** jobs
- Allowed ***only during BOT Tail*** (more on Tail later) to avoid overload
  - No overhead for large runs
- Task with less replicas prioritized

Tail detection

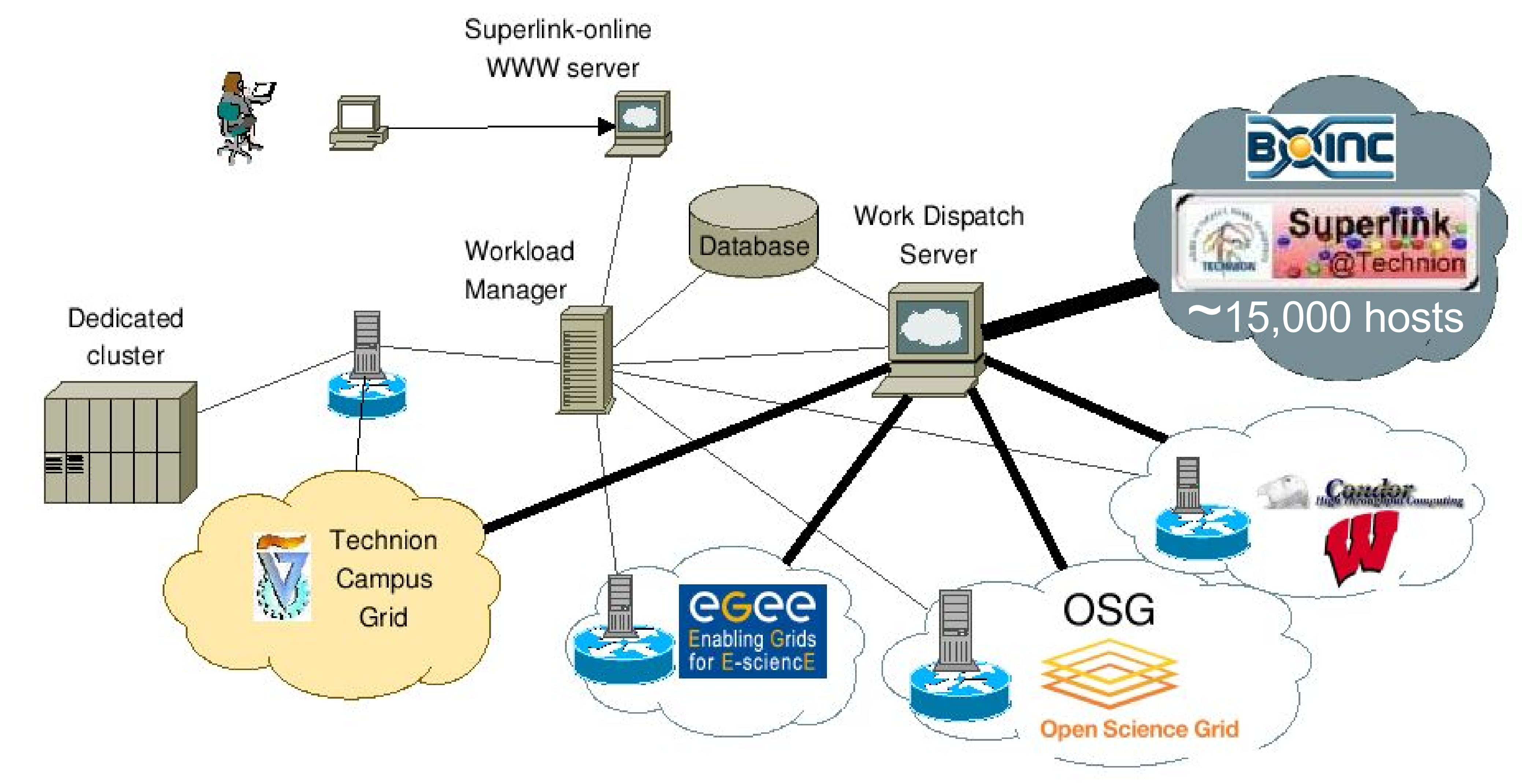
- What is Tail?
  - Load imbalance
  - Occurs in the end of the run due to resource idling
- Determined automatically when no idle tasks for the BOT

The system updates ***Tail*** classad attribute -

Dynamically affects scheduling policy

Production deployment: Superlink-online genetic analysis portal

Check out our online monitoring system: http://tiny.cc/GridBot



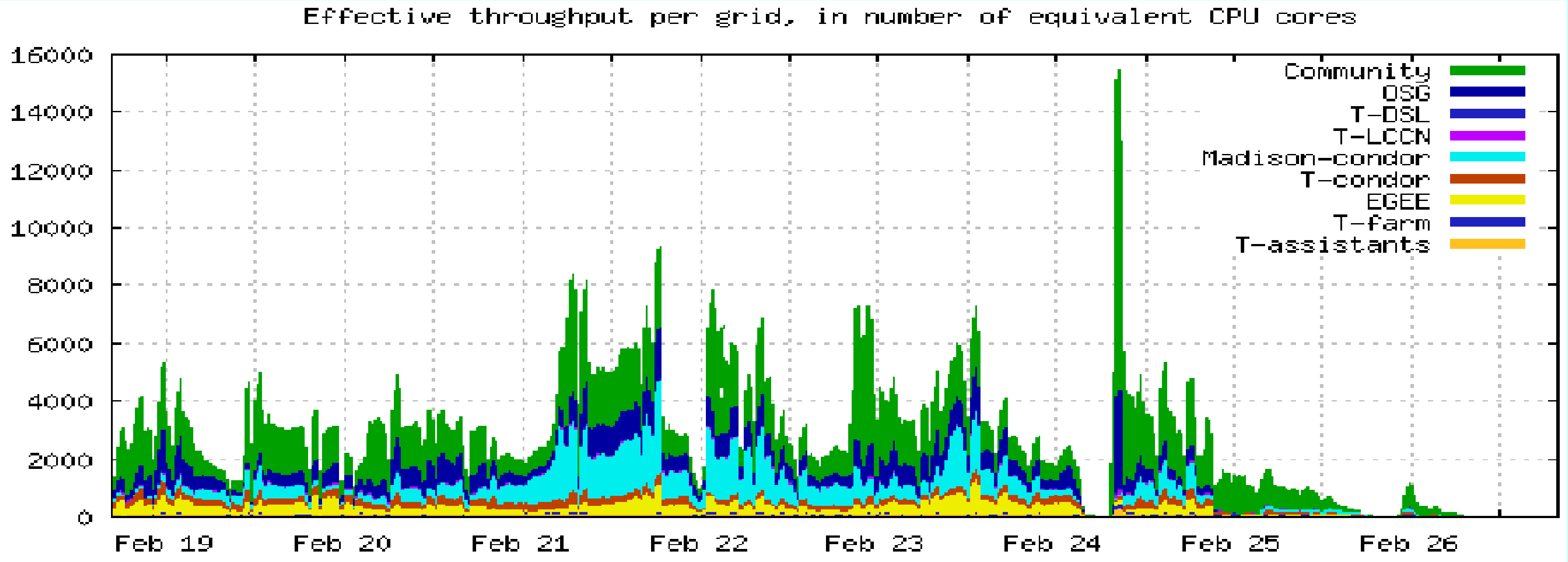
~25,000 active hosts in 3 months

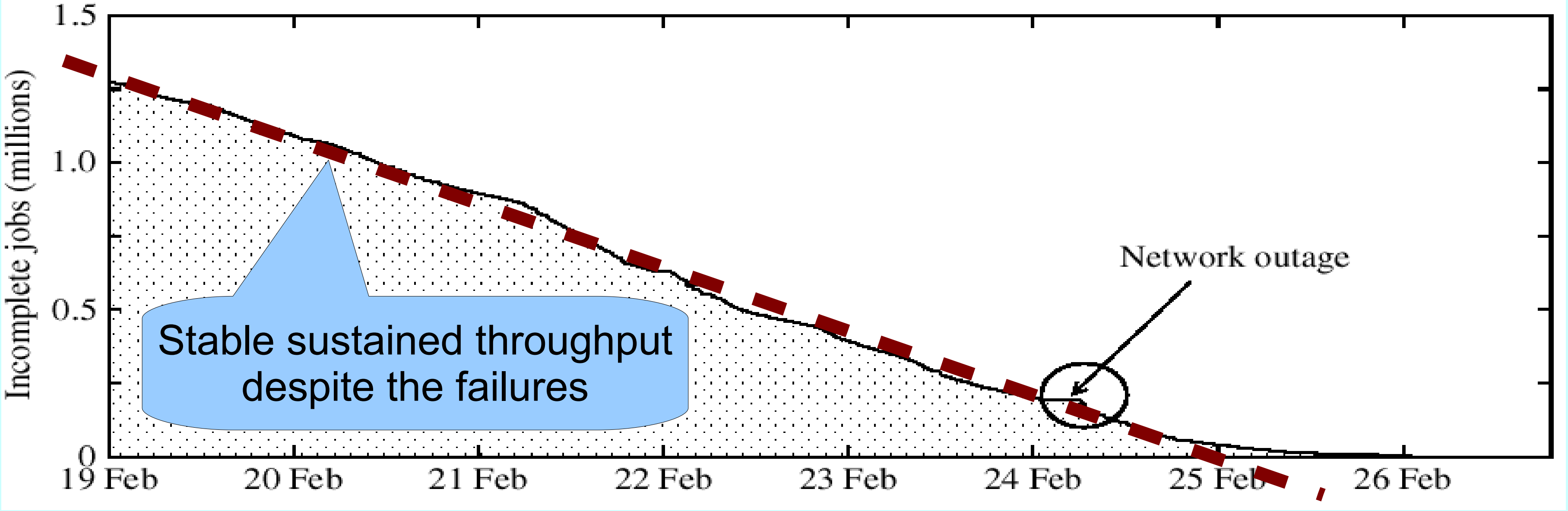
Experiments

- Plain BOINC overlay vs. GridBot
- GridBot vs. Condor
- Scalability
  - #Jobs in queue, # requests/sec, #BOTs
- Replication and policies
  - Replication
  - Multi-BOT scheduling

Real data of Superlink-online runs used in all experiments

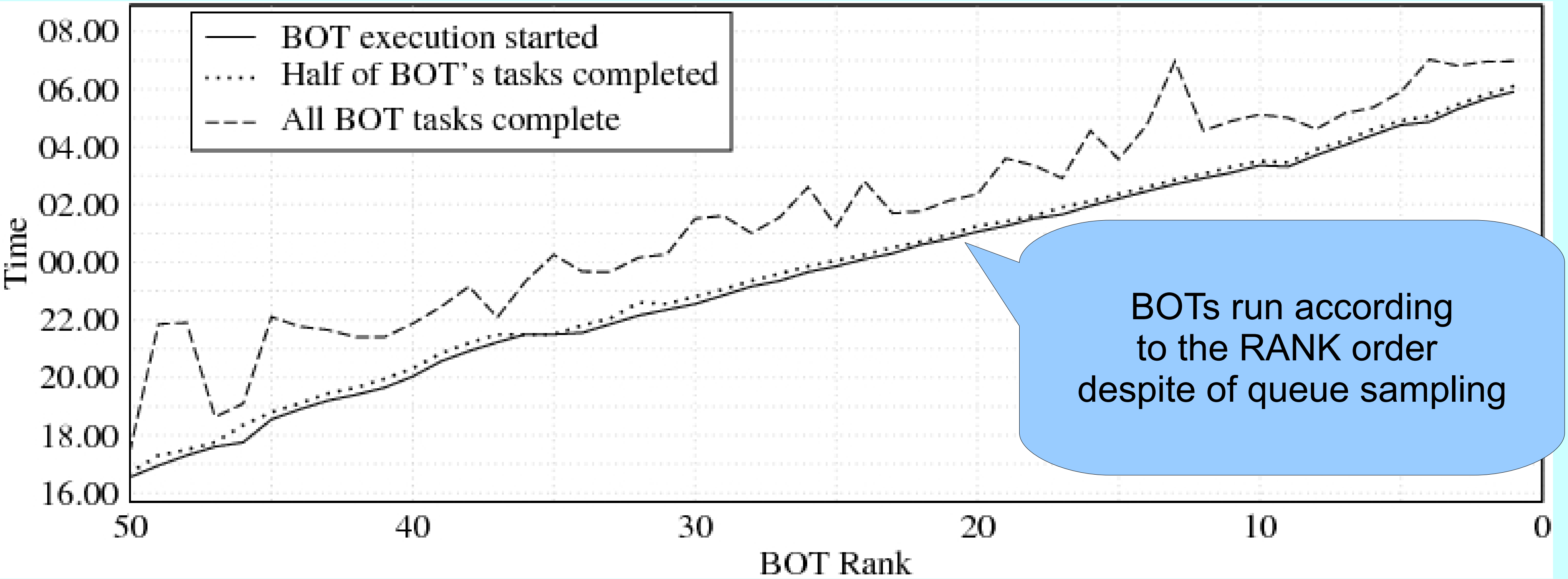
Scalability benchmark: #Tasks in Queue 2.2 mln tasks ~40 min each





Scalability benchmark: #BOTs in Queue

- 50 BOTs submitted at once, 1000 tasks each
- Each BOT *i* has Rank=*i*



BOTs run according to the RANK order despite of queue sampling

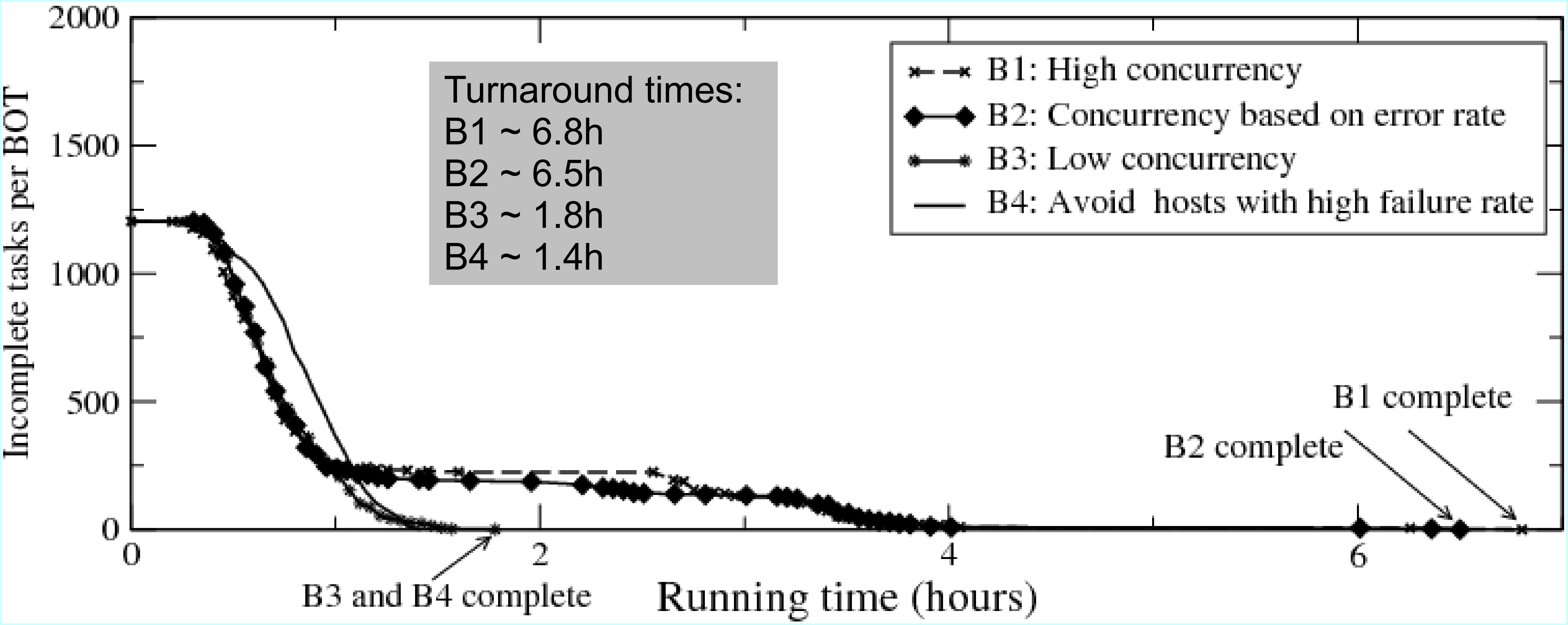
Throughput is not affected!

Scalability benchmarks
Task request rate

BOT	#Tasks	Time/Task	Dispatched Tasks/sec	Throughput
A	42,200	10-50 min	Up to 20	~3,700 cores
As A, each task split in 5	211,000	2-10 min	Up to 93	~3,700 cores

Throughput is not affected!

Scheduling policies and short runs
(1200 tasks/BOT, 3 min/task)



Replication policies
Collaborative grids alone

- 30,000 tasks, 10-15 min each

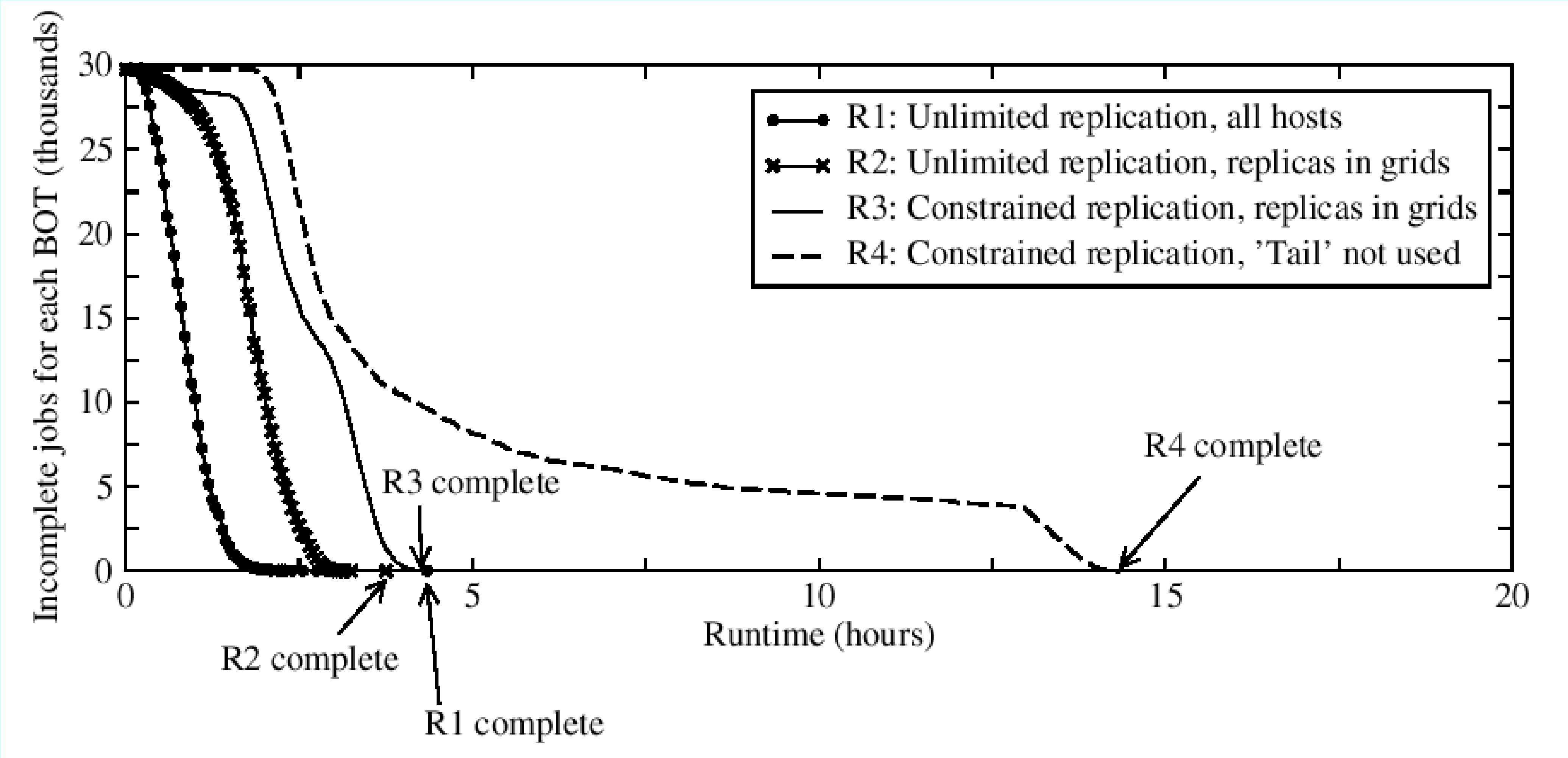
Replication policy	Scheduling policy	Replicas (%)	Waste (%)	Runtime (h)
Up to 5 replicas/task	Any host	58	30	4.1
Only unreliable host or no response for >30 min	Only reliable	11	7	3.2
Up to 5 replicas/task	Only reliable	73	57	4.2
Disabled	Only reliable	0	0	5.1
Disabled	Any host	0	0	5.8

Replication policies
All grids including community grid

- 30,000 tasks, 10-15 min each

Replication policy	Scheduling policy	Replicas (%)	Waste (%)	Runtime (h)
Up to 5 replicas/task	Any host	188	105	4.2
Up to 5 replicas/task	No community grid resources in Tail	129	75	3.8
15 min between replicas and one of them on unreliable host	No community grid resources in Tail	49	25	4.3
No replication until below 2000 tasks in BOT	As above, Tail statically recognized when below 2000 tasks in BOT	35	21	14.2

Replication policies
All grids including community grid



Conclusions

- Contributions
  - Policy-driven mechanisms for efficient BOT execution
  - Scalable implementation in a production system
  - Large-scale experiments over multiple production grids including community grid with various policies
- Future work
  - Incorporate supercomputers (TSUBAME)
  - Different overlay establishment target functions
  - Cost-efficient combination of grids and clouds
  - Data-intensive computing in clouds

Acknowledgments

- Miron Livny, UW Madison
- David Anderson, UC Berkeley