Omega: flexible, scalable schedulers for large compute clusters

Malte Schwarzkopf, Andy Konwinski,
Michael Abd-El-Malek, John Wilkes

Cluster Scheduling

- Shared hardware resources in a cluster
  - Run a mix of workloads on the same set of machines
- **Problem:** Allocation of resources to different "tasks" from a resource pool in a cluster
  - Schedule tasks on machines based on available resources

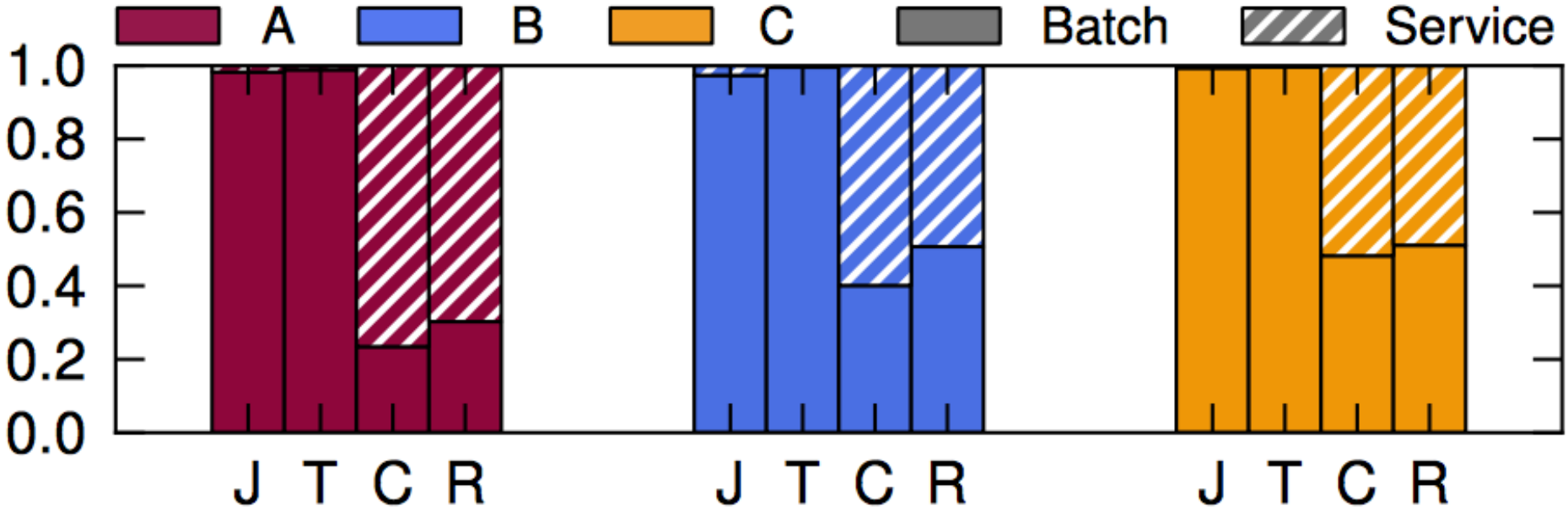
Cluster Scheduling

- … not a new problem
- In HPC community (Maui, Moab, Platform LSF)
- What's new?
  - "Google" scale
  - Need for flexibility (changing policies, constraints)
  - Heterogeneity (hardware, workloads, …)

Characterizing Google's Workloads

- Workload composed of "jobs"
  - Each job composed of multiple "tasks"
- Workload split
  - Long running *service* jobs (e.g., web services)
  - Shorter *batch* jobs (e.g., MapReduce jobs)

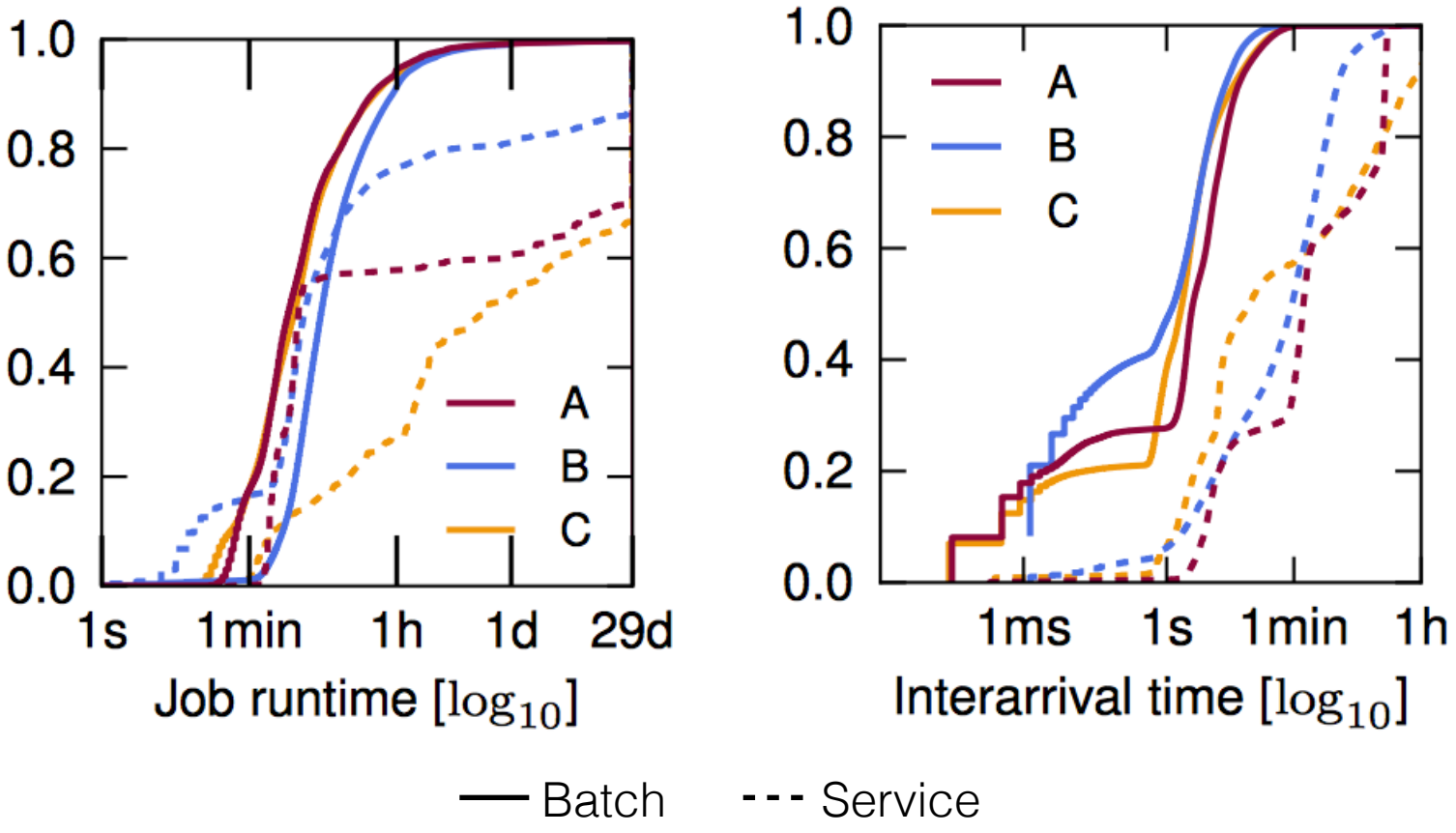
Characterizing Google's Workloads



J=Jobs, **T**=Tasks,
C=CPU-core-seconds, **R**=RAM-GB-seconds

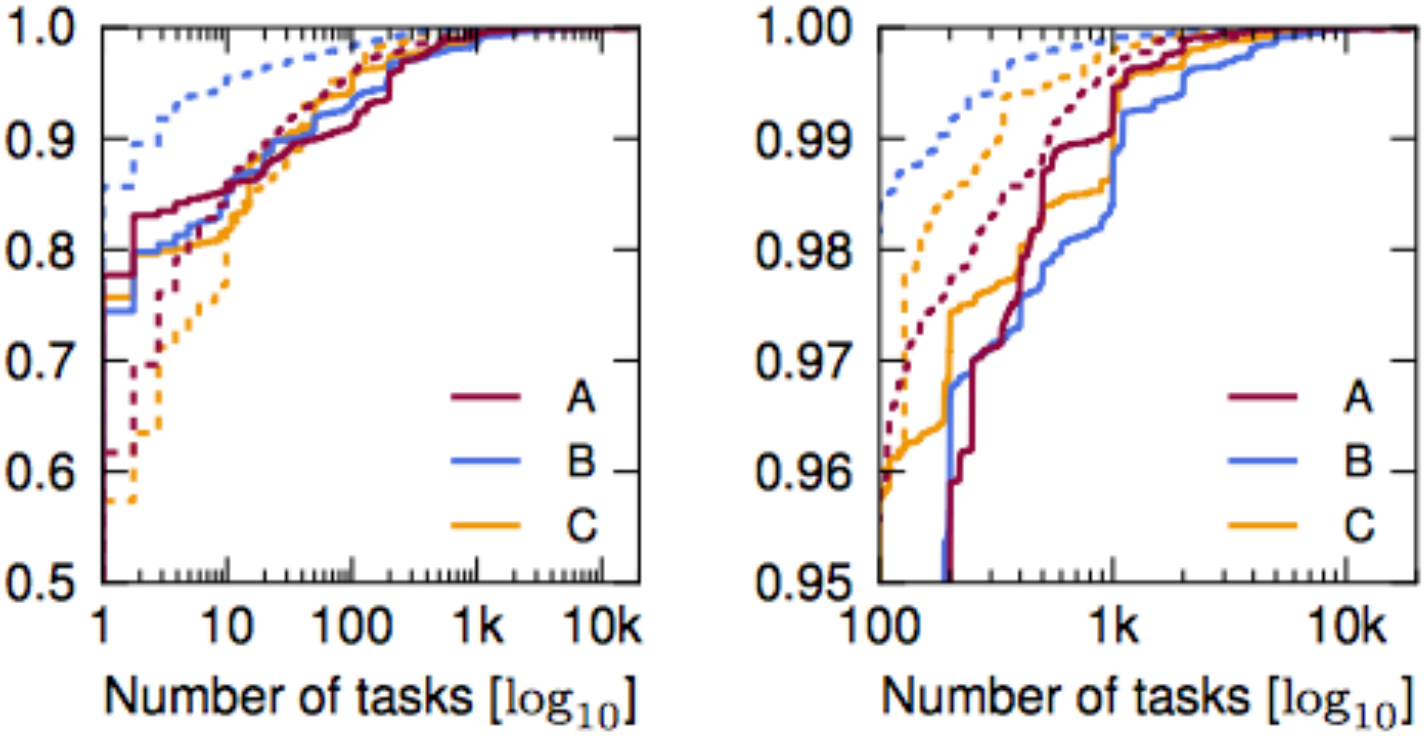
Most jobs are batch, but most resources are consumed by service jobs.

Characterizing Google's Workloads



Service jobs run for much longer than batch jobs

Characterizing Google's Workloads



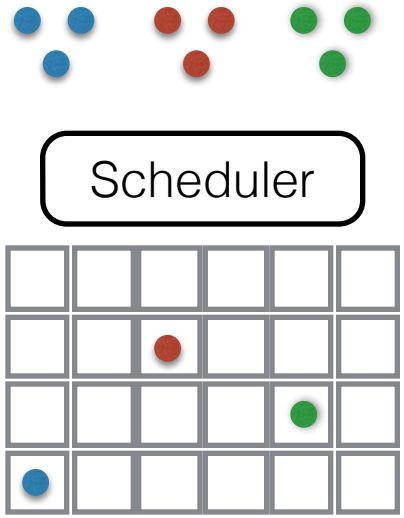
— Batch --- Service

Service jobs have much fewer tasks than batch jobs

Goals

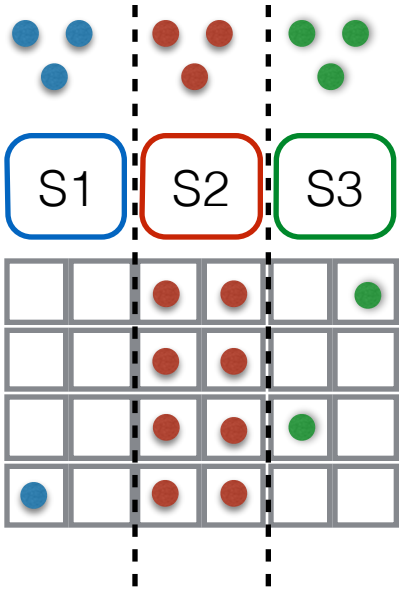
- A cluster scheduling architecture that ensures:
  - High resource utilization (*utilization*)
  - Conformity to user policies, and ability to add new policies (*flexibility*)
  - Should scale to large clusters (*scalability*)

What about existing solutions?



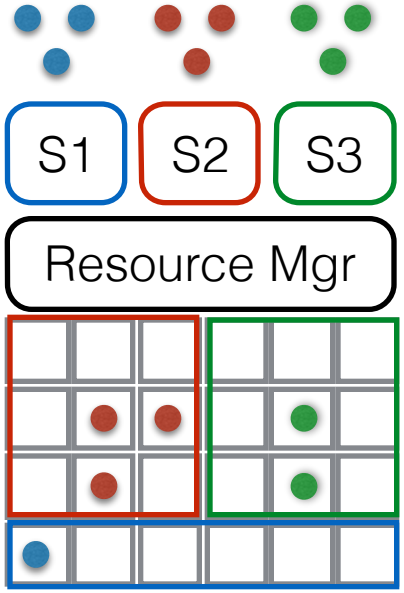
Monolithic

- Hard to add new policies
- Scalability bottleneck



Static Partitioning

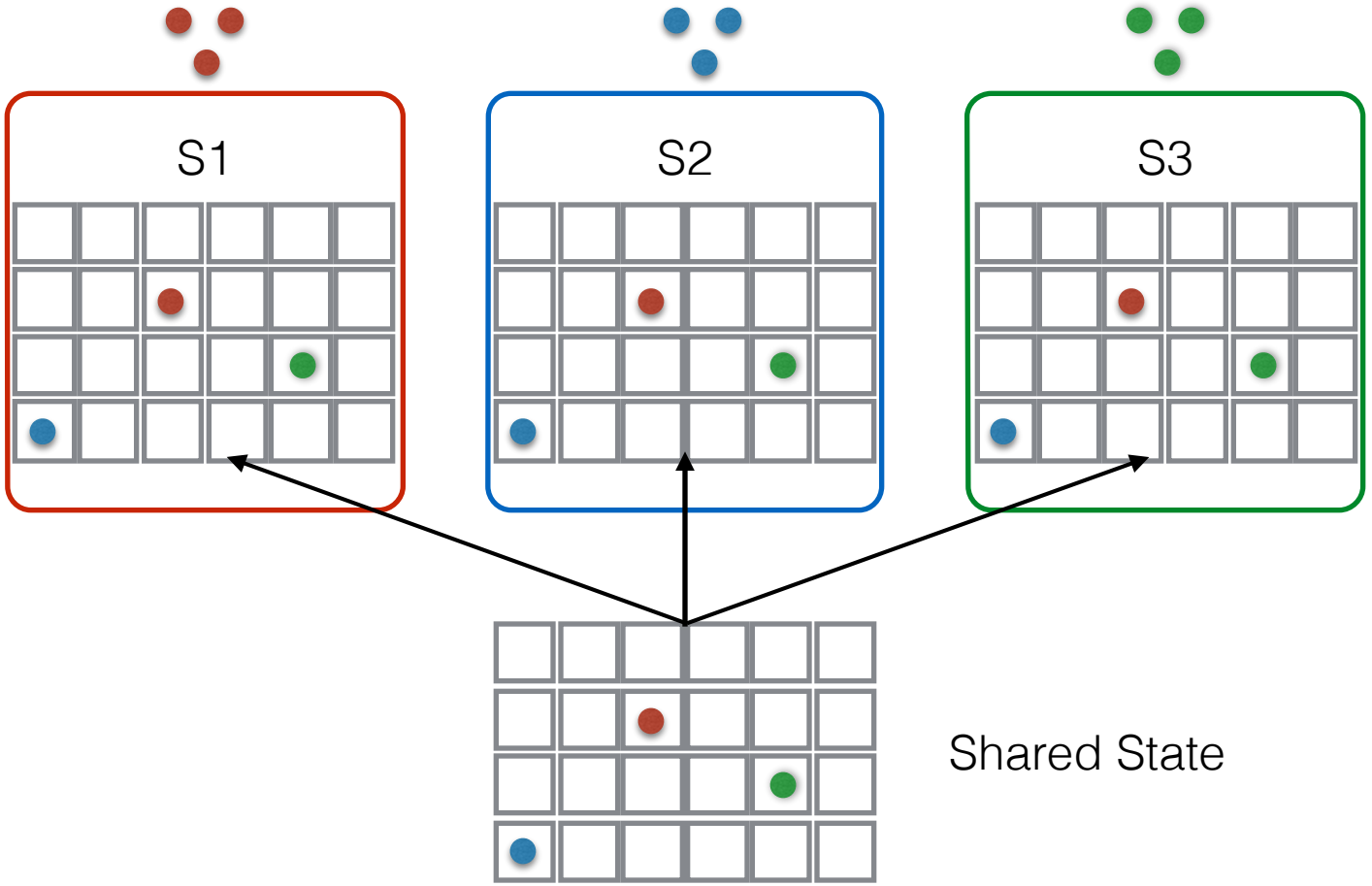
- Poor utilization
- Inflexible allocation



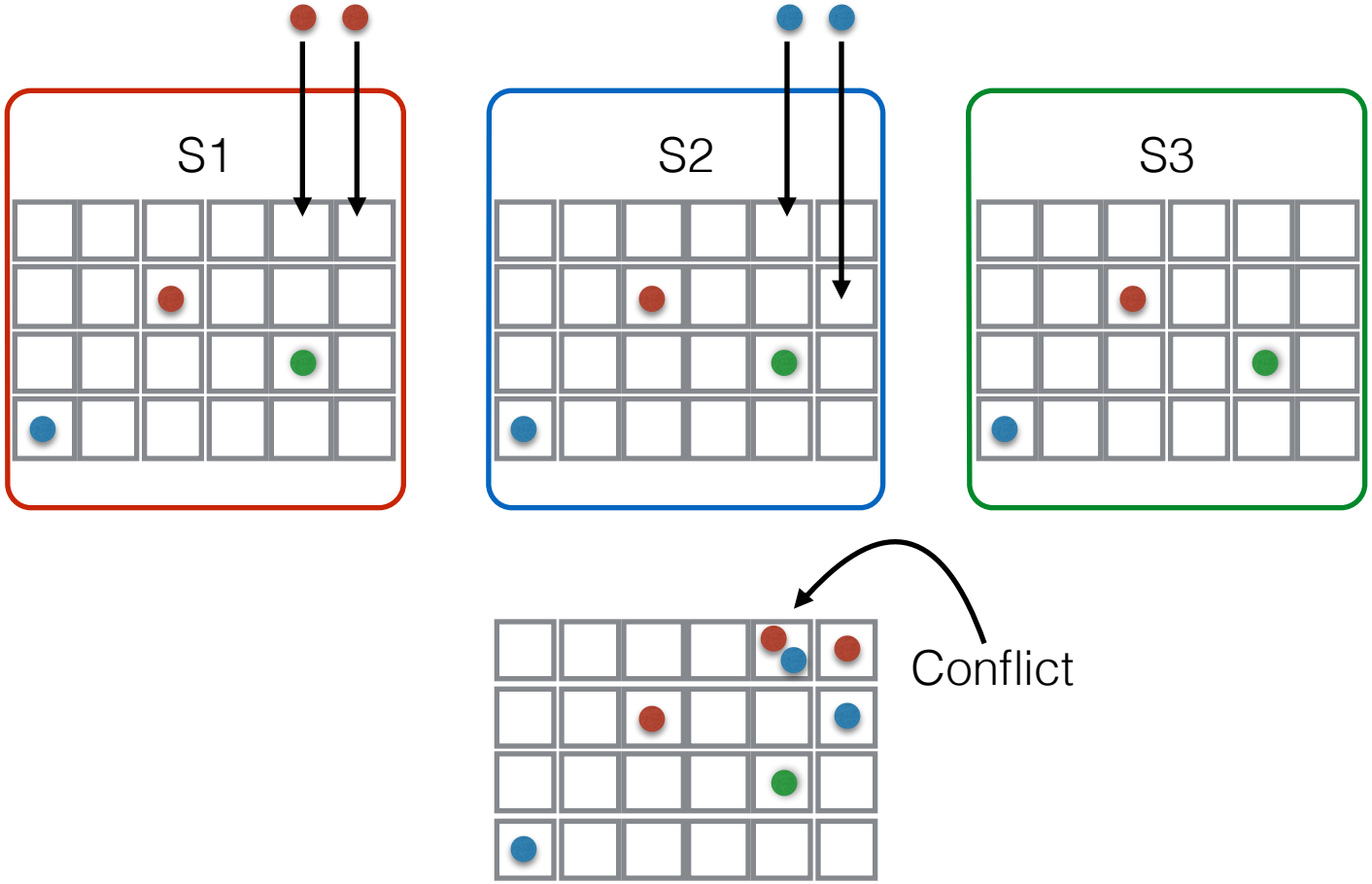
Two-level

- Locks resource during "offer" (*pessimistic*)
- Limited state information

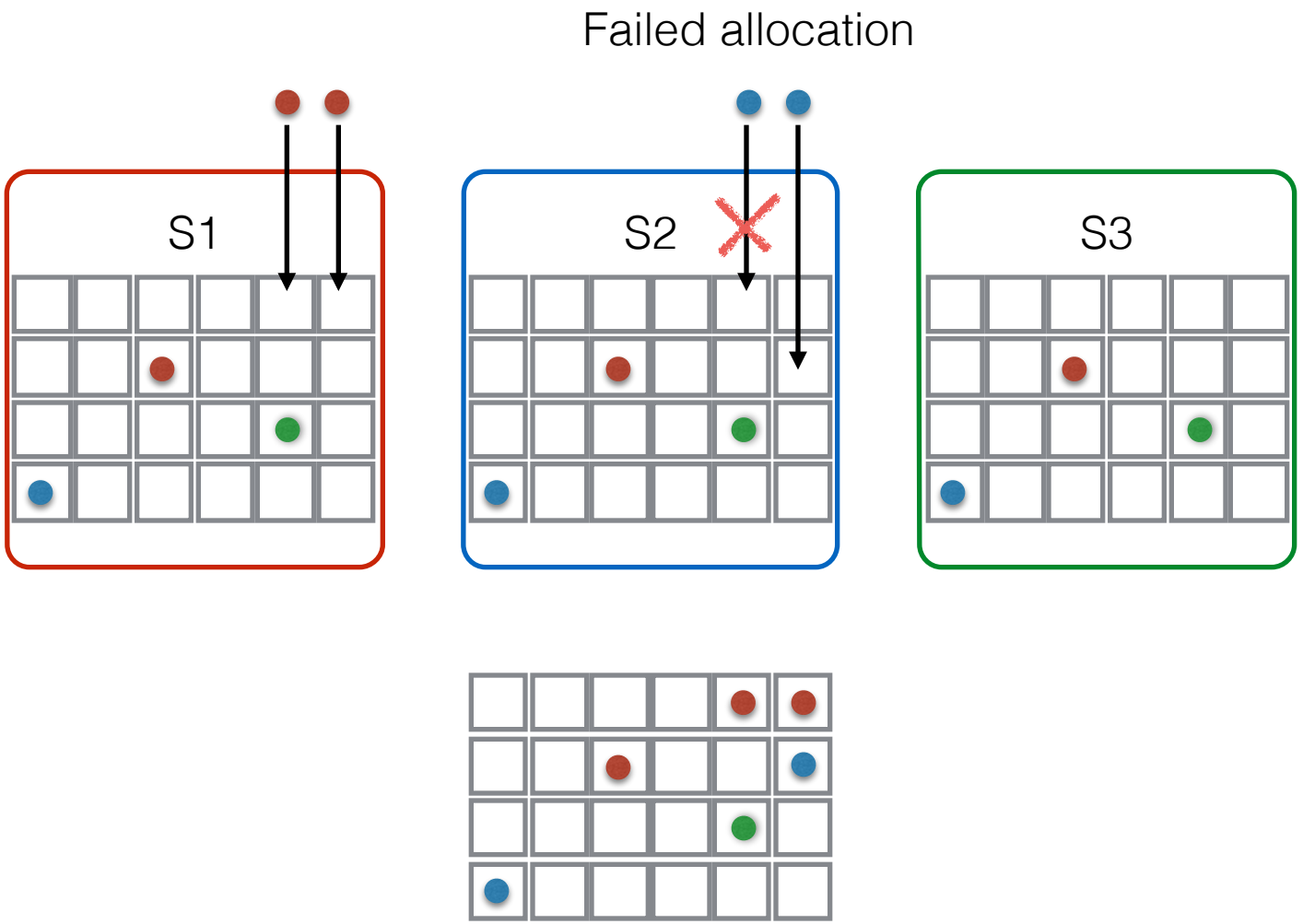
Omega's Approach



Allocation?



Allocation?

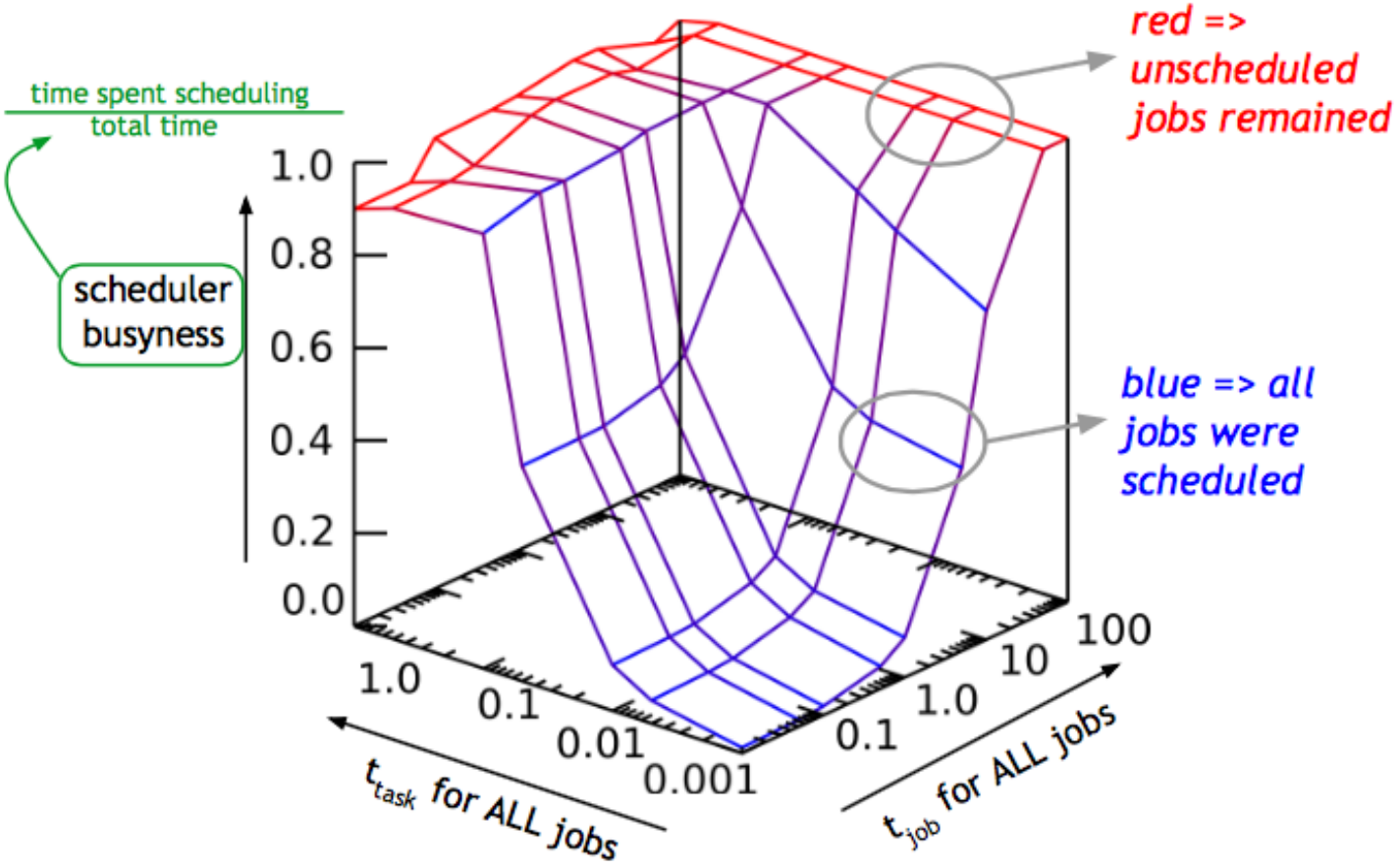


What's the trade-off?

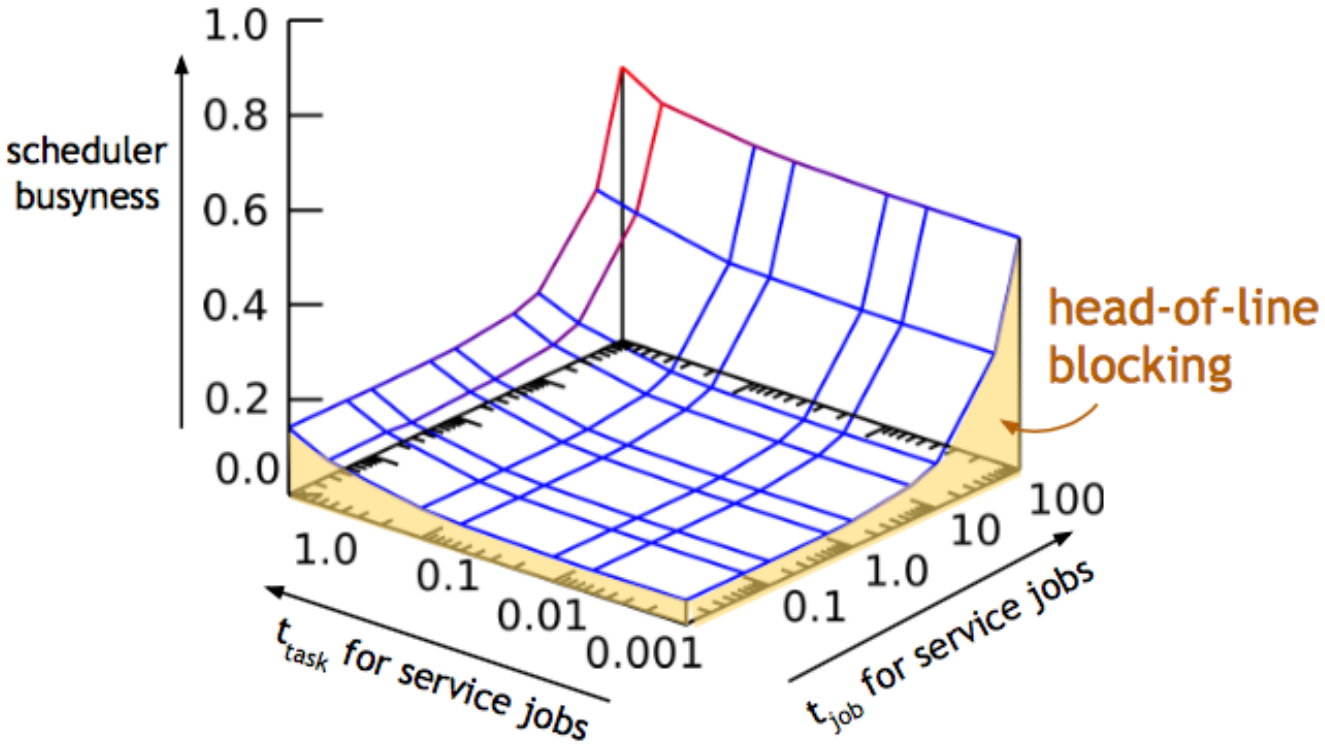
- Two-level scheduling (e.g., Mesos)
  - Limits parallelism (*pessimistic* locking)
  - Schedulers have restricted visibility of resources
- *Shared state* with *optimistic* concurrency control
  - Eliminates the two issues with two-level approach
- Cost: Wasted work when optimistic assumption fails

Simulation Study

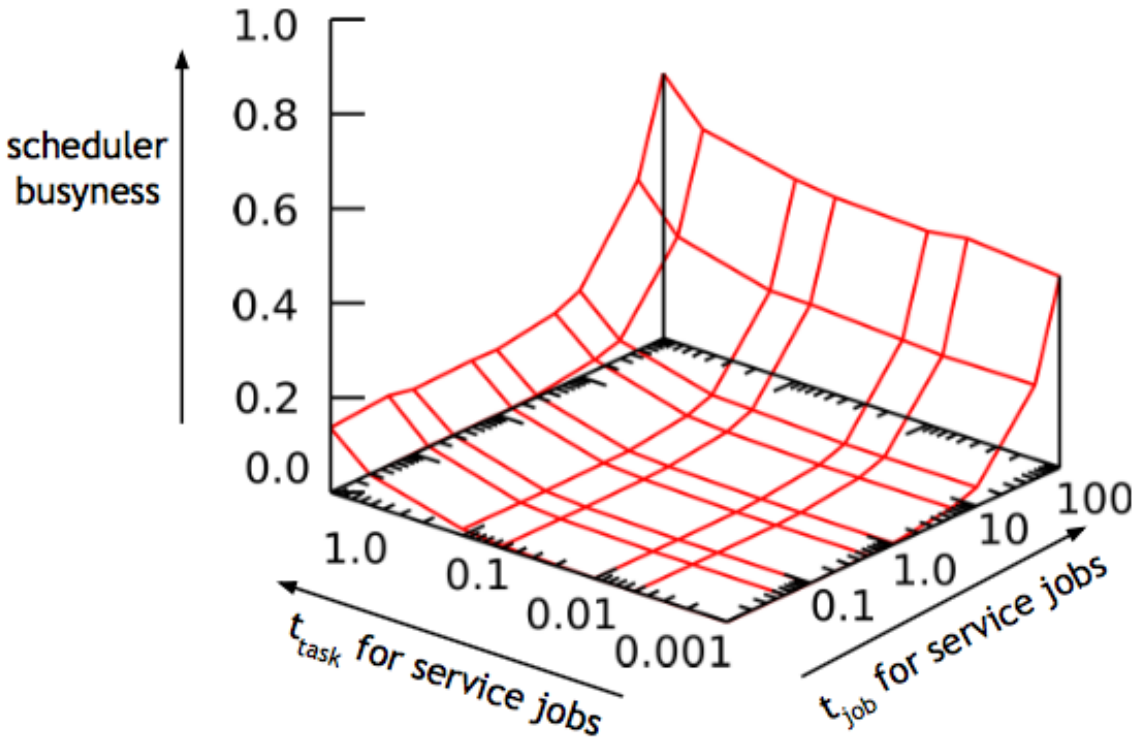
Monolithic, single logic



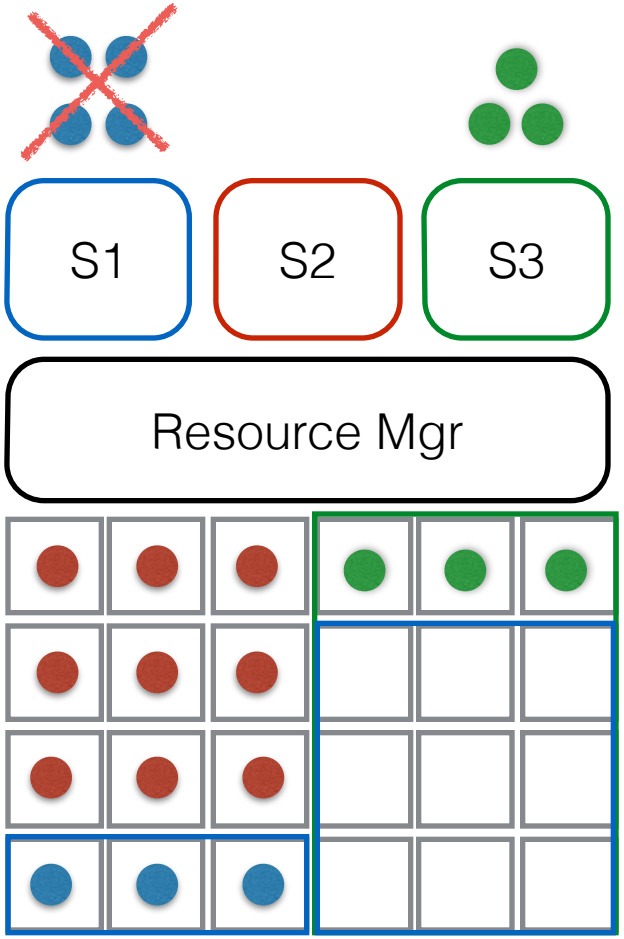
Monolithic, fast-path



Mesos


scheduler busyness
1.0
0.8
0.6
0.4
0.2
0.0
1.0
0.1
0.01
0.001
t_{task} for service jobs
0.1
1.0
10
100
t_{job} for service jobs

What's Going On?



Green receives offer of all available resources

Blue's tasks finish

Blue receives small offer

Offer is insufficient for blue

Blue receives small offer

Offer is insufficient for blue

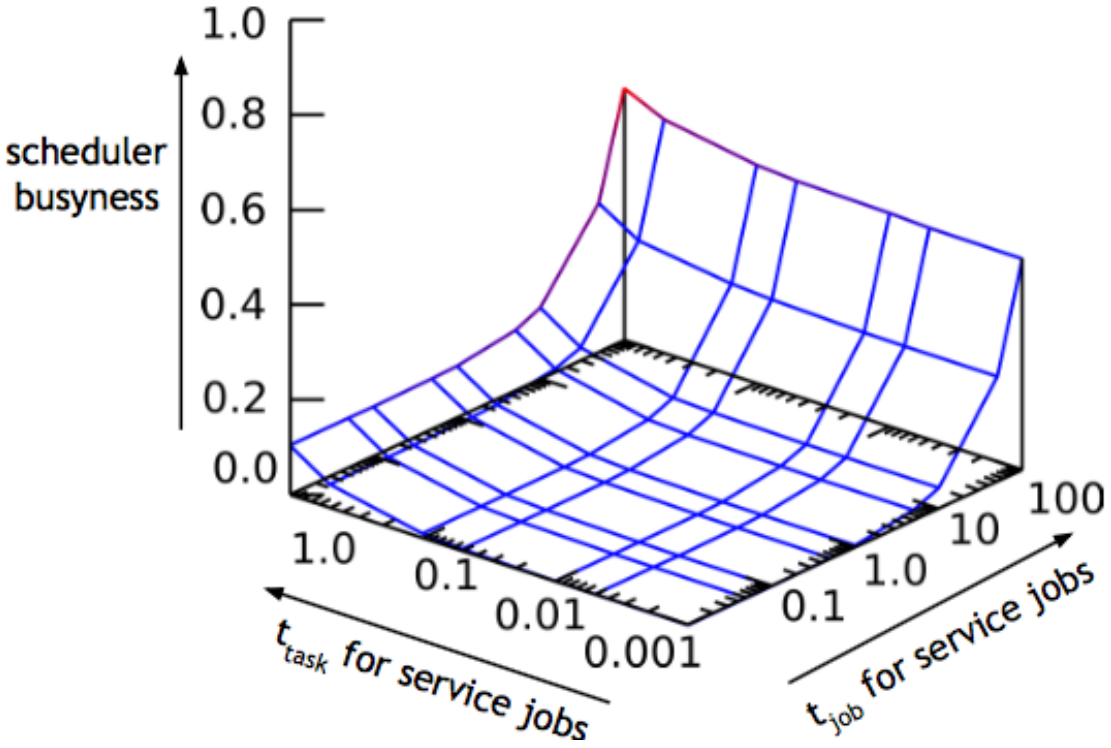
[repeat]

Green finishes scheduling

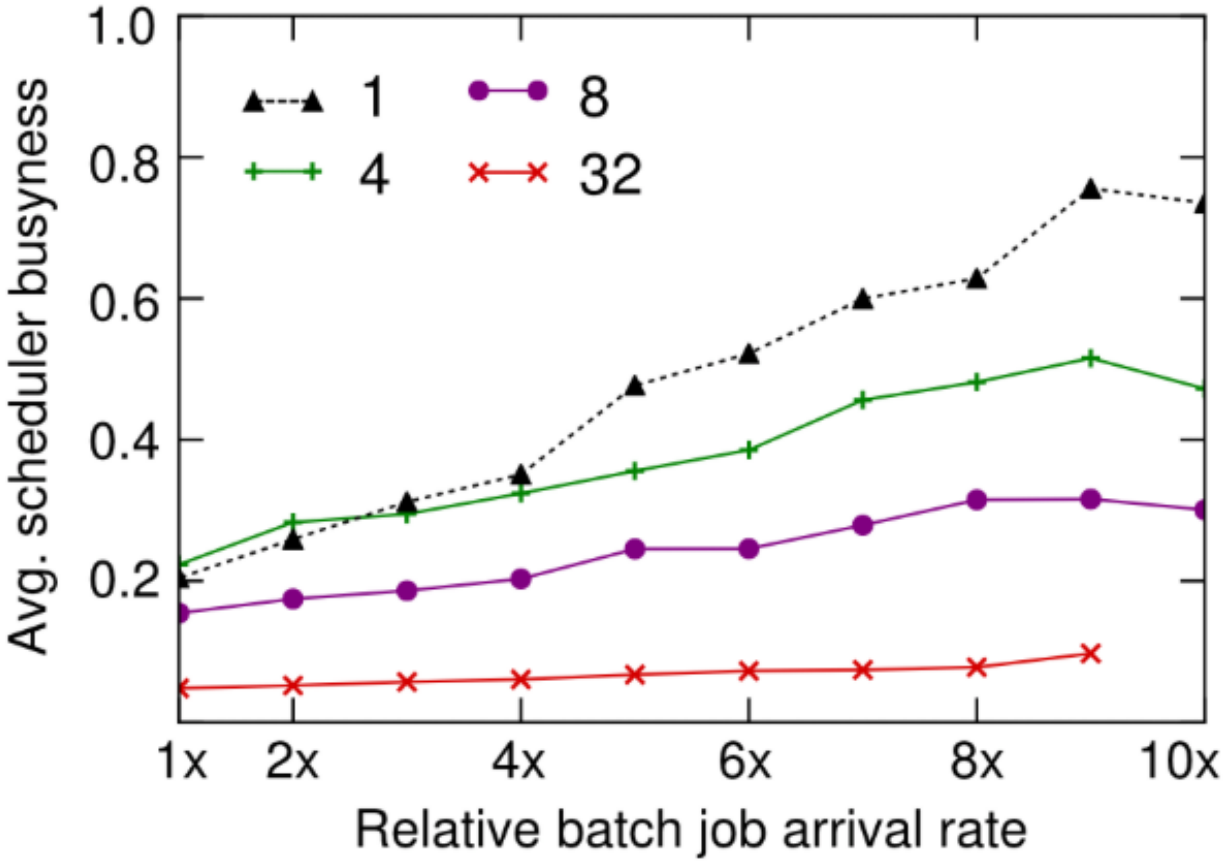
Blue receives larger offer

By now, blue has given up

Omega


1.0
0.8
0.6
0.4
0.2
0.0
scheduler busyness
1.0
0.1
0.01
0.001
t_{task} for service jobs
0.1
1.0
10
100
t_{job} for service jobs

Effect of Parallelism

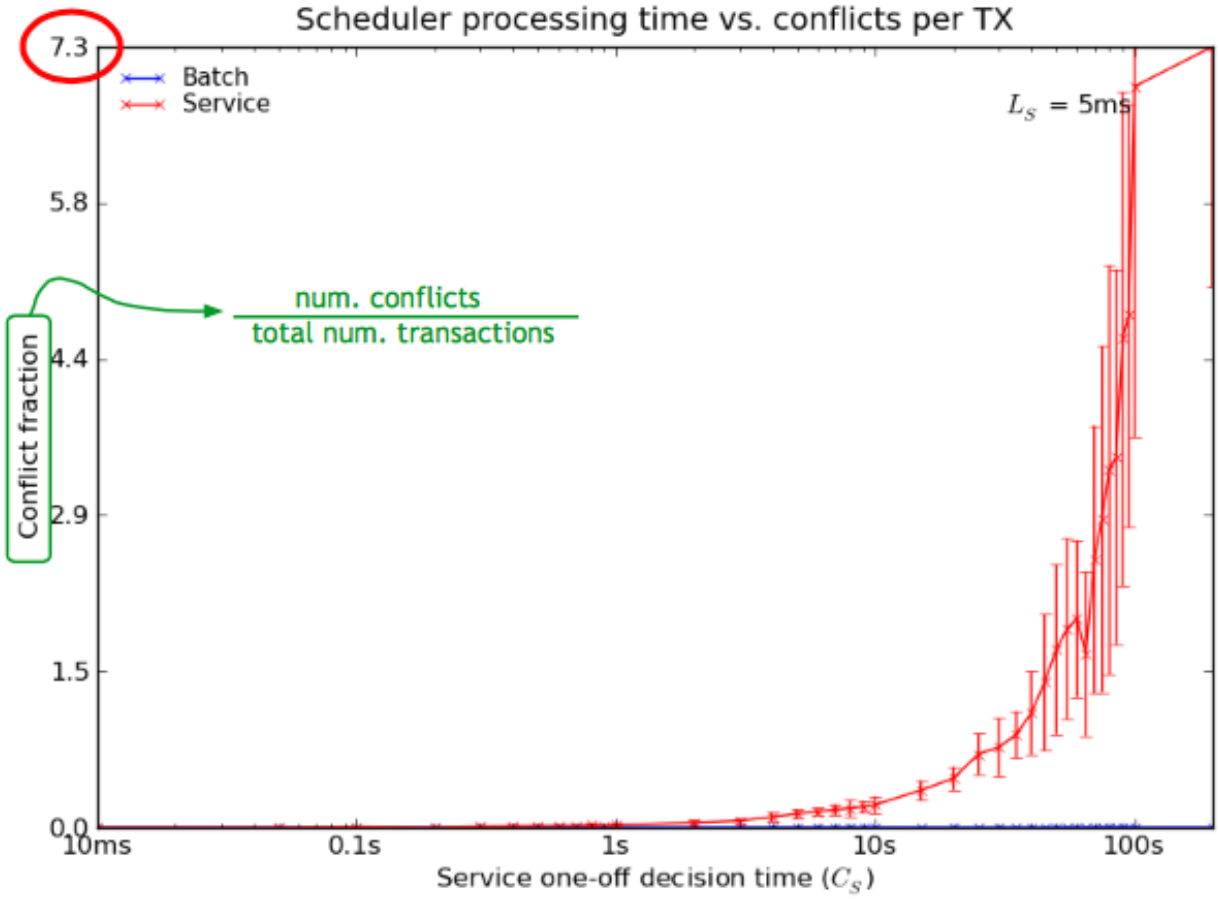


Takeaway?

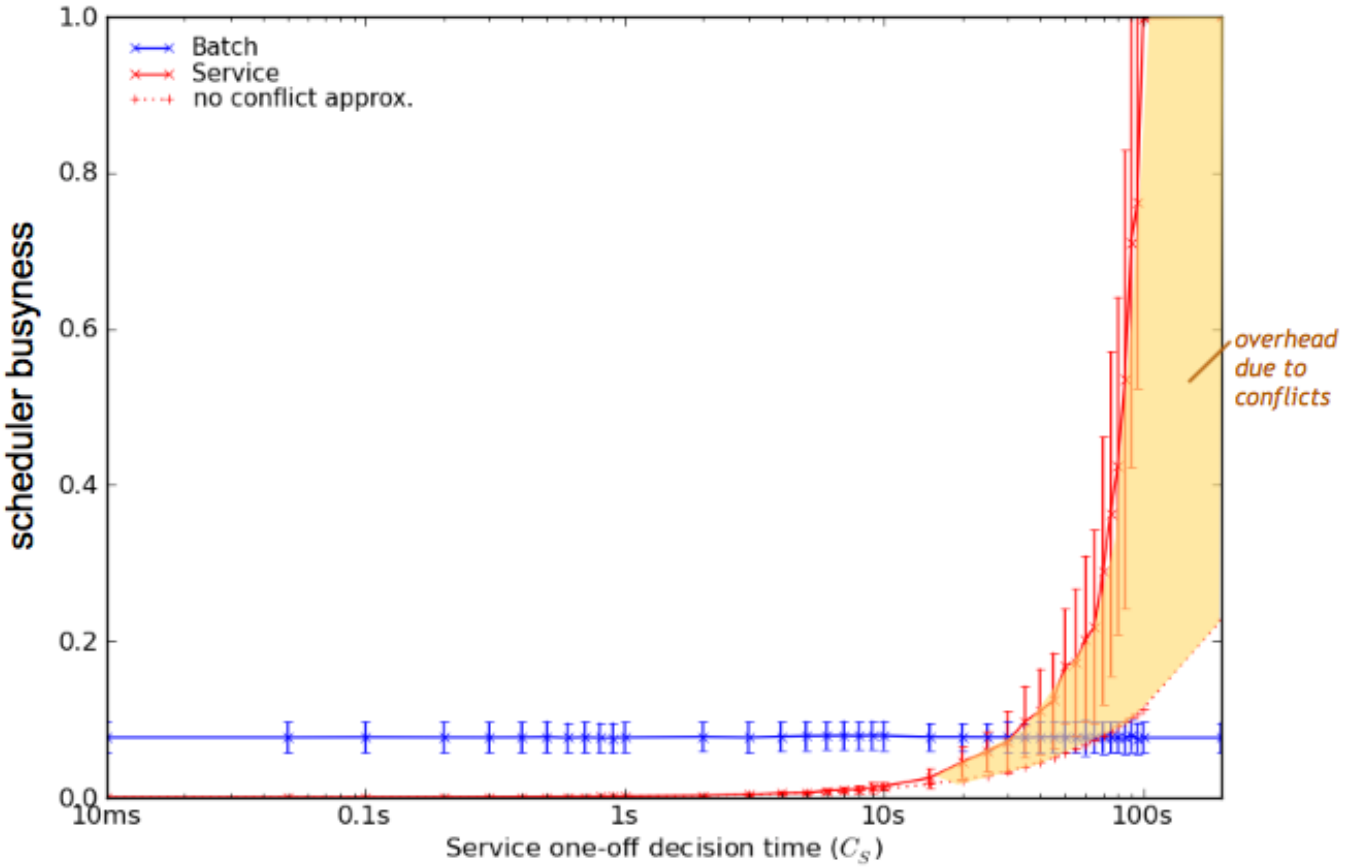
- Omega's shared state model
  - Performs as well as a complex monolithic multi-path scheduler
  - Can overcome its scalability issues by using multiple schedulers

Effect of Conflicts?

Conflict Fraction



Scheduler Busyness



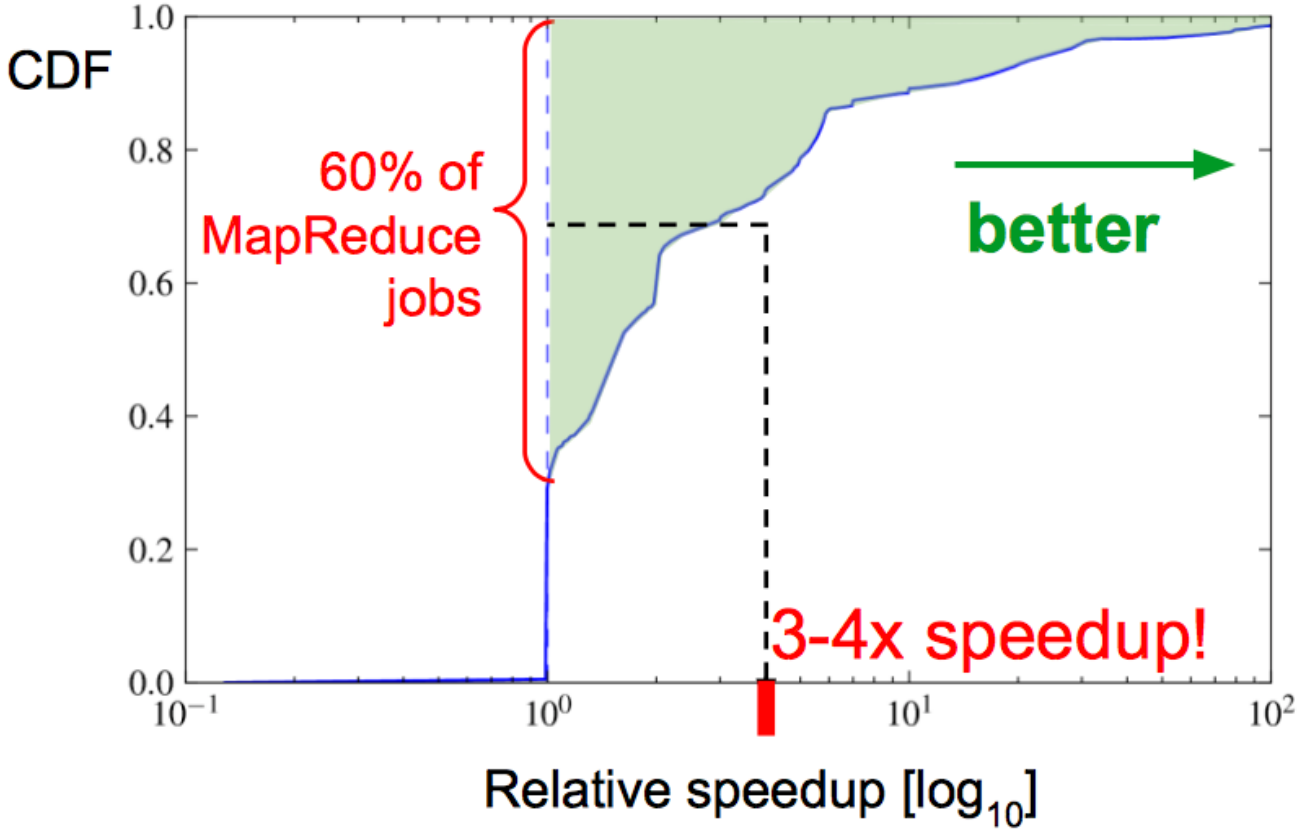
Interference is high for real-world settings

Case Study: Specialized MapReduce Scheduler

MapReduce Scheduler

- Opportunistically add more mappers/reducers as long as benefits are obtained
- *Max-parallelism* approach
- More policies in paper

Benefits of opportunistic allocation



Conclusions

- Cluster scheduling architecture with
  - Parallelism
  - Shared State
  - Optimistic Concurrency control
- Enables
  - Scalable scheduling
  - Flexibility in scheduling policies
  - Visibility to complete cluster state
    - Potentially more efficient scheduling

Questions

- Is it okay to ignore certain global policies like fairness?
  - *"… it helps that fairness is not a primary concern in our environment: we are driven more by the need to meet business requirements."*
- An underlying assumption is that decision times for schedulers can grow quite high (due to complicated scheduling); is this valid?

Questions

- Is the design too “Google-specific”?
- E.g., OCC works well when contention is low.
- In this context, contention is high if:
  - Resources are few (small clusters)
  - Too many tasks in a given time (high arrival rate)
  - Number of schedulers is large (high parallelism)
- When does high contention become a bottleneck?
  - Perhaps not an issue for Google’s clusters, but in general…
- Google can afford to “over-provision” resources, is it possible in general?