Big-data job deadlines
+
Fault-tolerant resource allocation

Peter Bodik
Microsoft Research

Why care about deadlines for big-data jobs?

Important big-data jobs have to finish on time

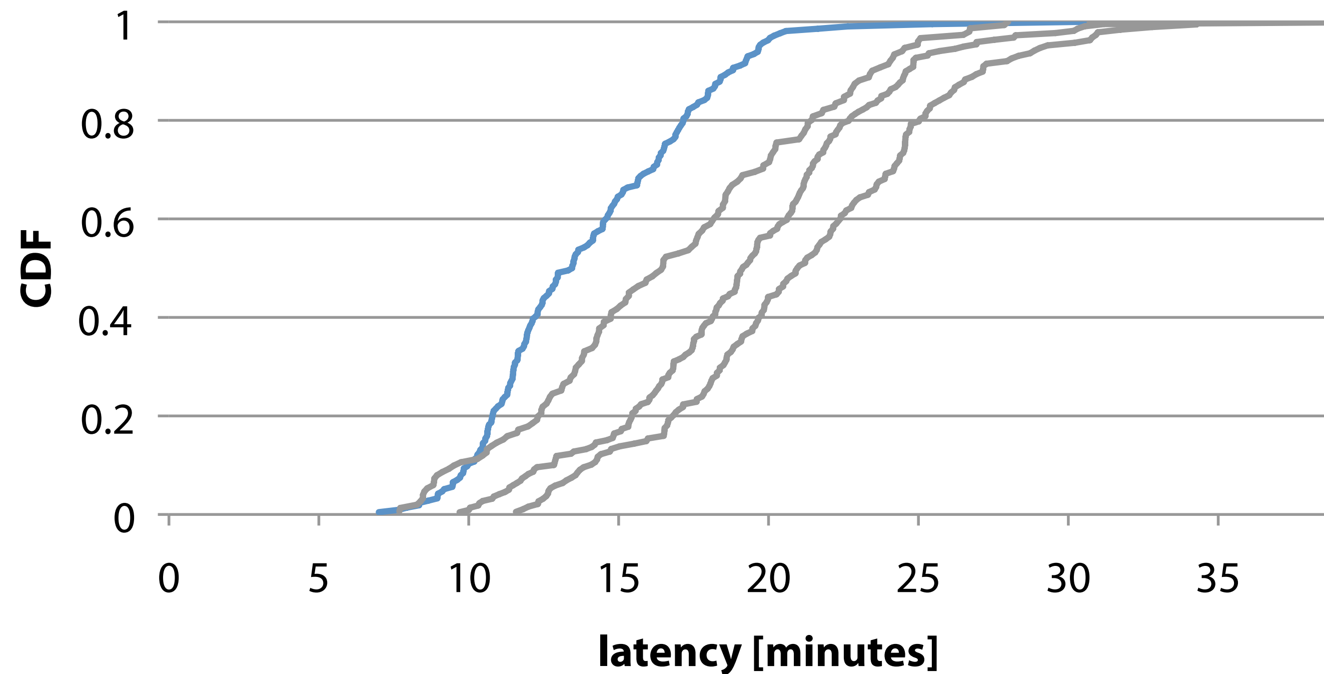
– missed deadline = delayed updates on site, financial penalty, productivity loss

Current clusters

– can't specify deadline in current schedulers

– users don't know how resources map to latency

– noise



Jockey: meeting deadlines for big-data jobs

Cosmos 101

- big-data platform in Microsoft
- job = SQL query + user C# code
- job compiled/optimized using SQL-like optimizer to a DAG of stages/ vertices
- big jobs have 100s of stages, 1M vertices

Jockey

- input: single job with a deadline, past job runs
- offline: builds a job model
- run time: control loop adjusts allocation

erguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, Jockey: Guaranteed Job Latency in Data Parallel Clusters, in *EuroSys*, 2

Job model = past job runs + simulation

Job model

- input: current progress, allocation
- output: remaining time to completion

Example

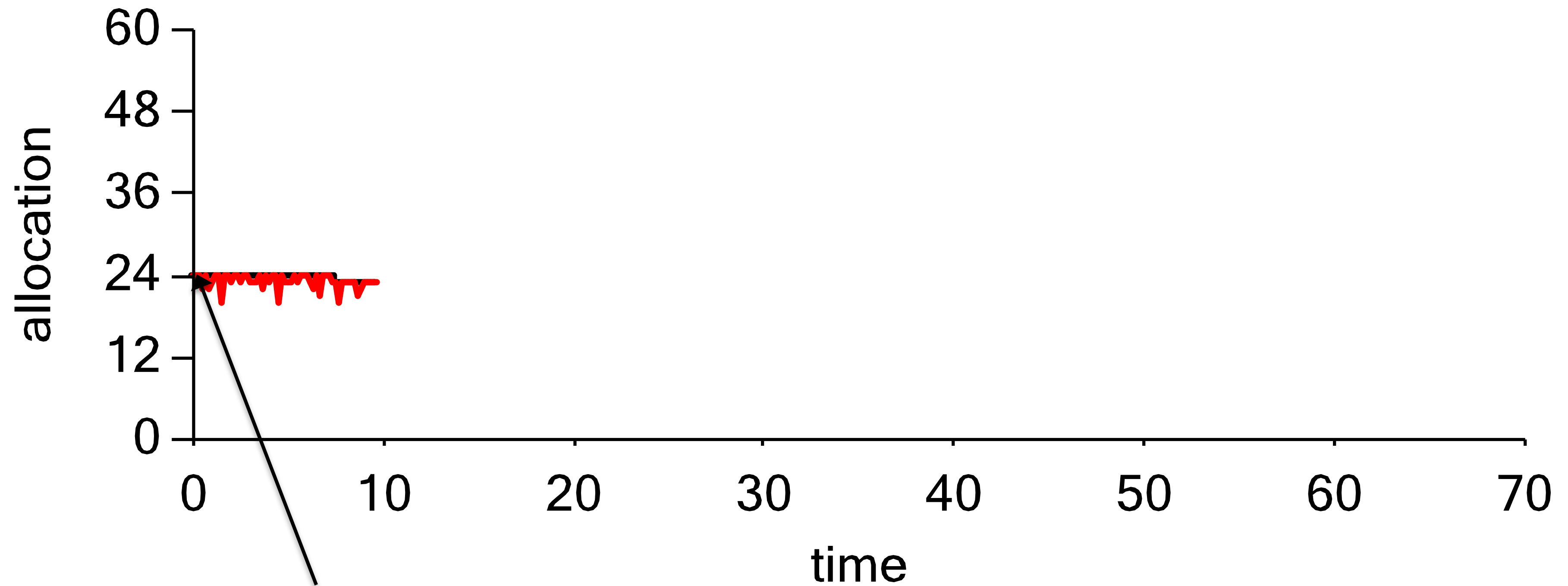
- deadline = 30 min
- after 10 min, completed 50%
- will set allocation to 30 tokens

	10 tokens	20 tokens	30 to
10%	60 min	40 min	25
20%	59 min	39 min	24
30%	58 min	37 min	22
40%	56 min	36 min	21
50%	54 min	34 min	20

Issues

- in practice need to trade off between many jobs
- “fraction completed” doesn’t capture the whole run time state

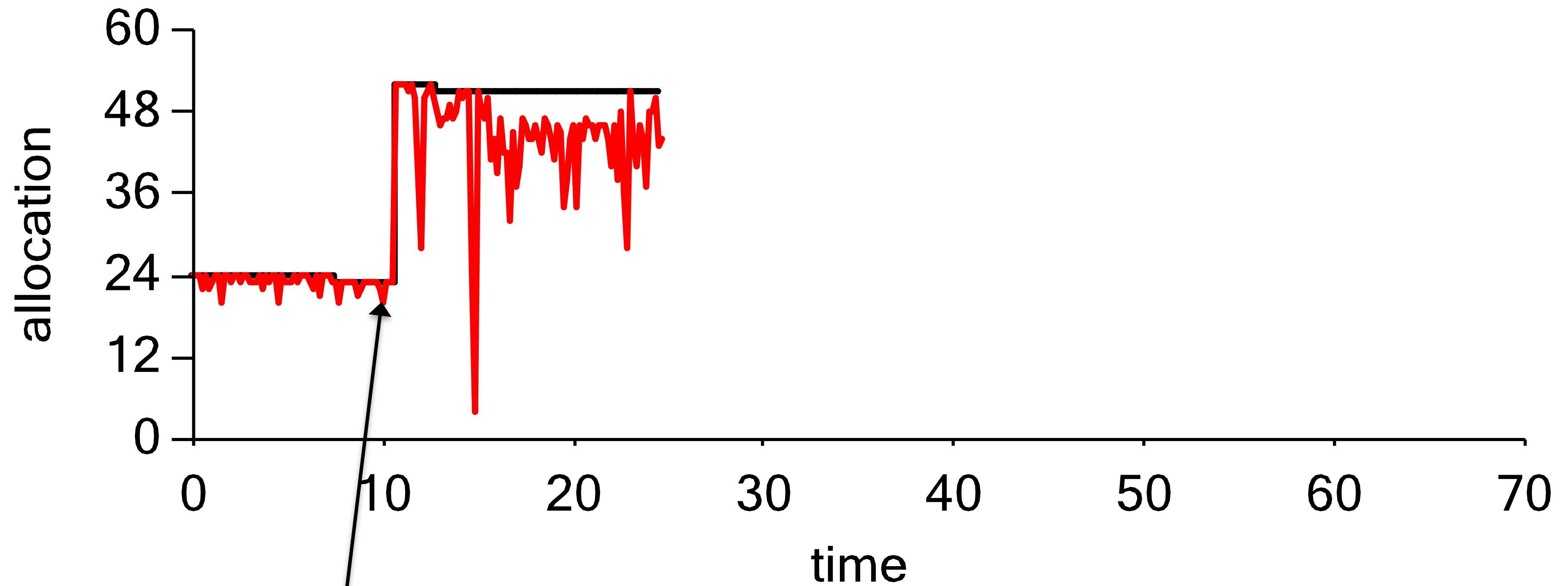
Jockey in Action



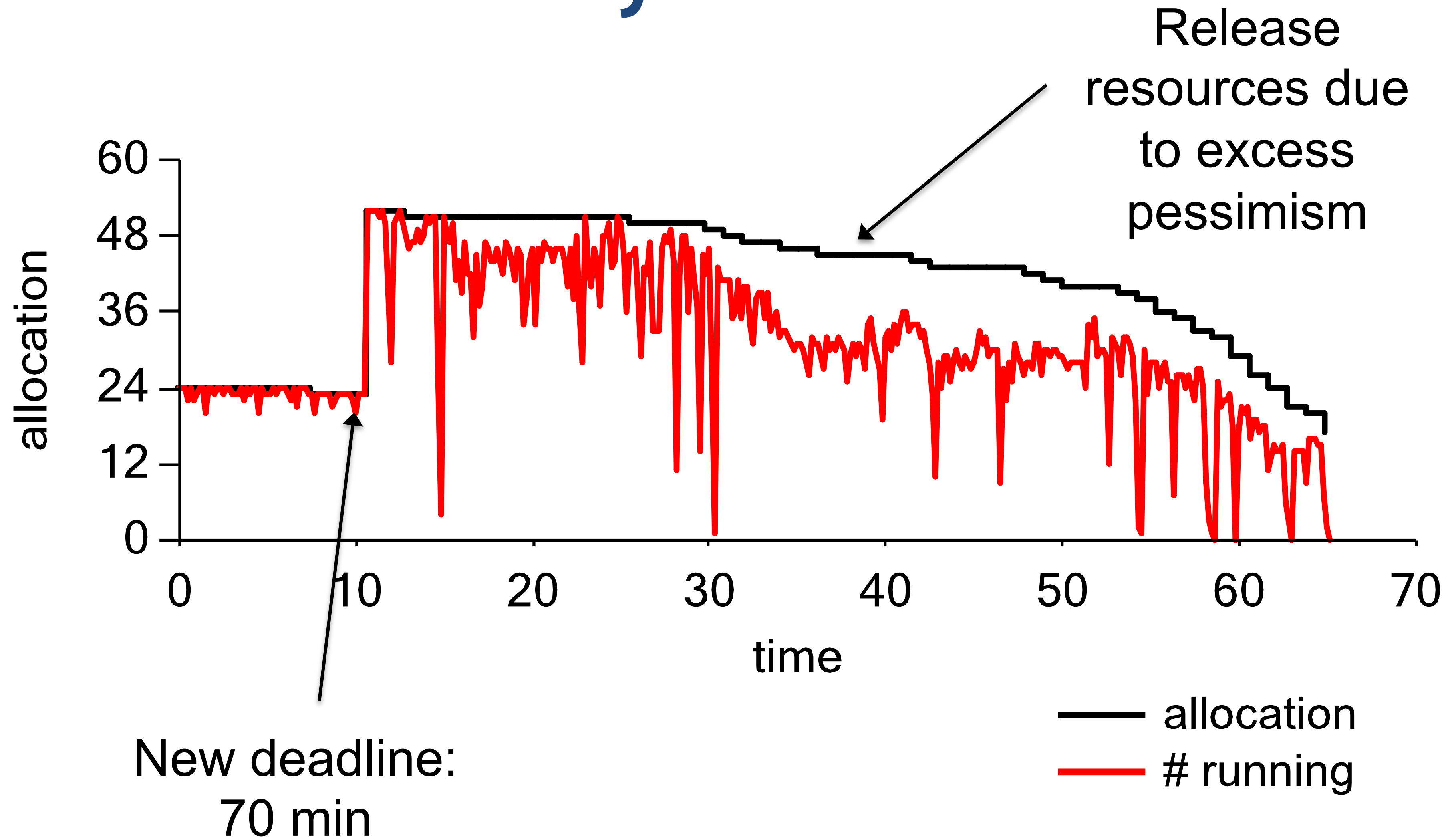
Initial deadline:
140 min

allocation
running

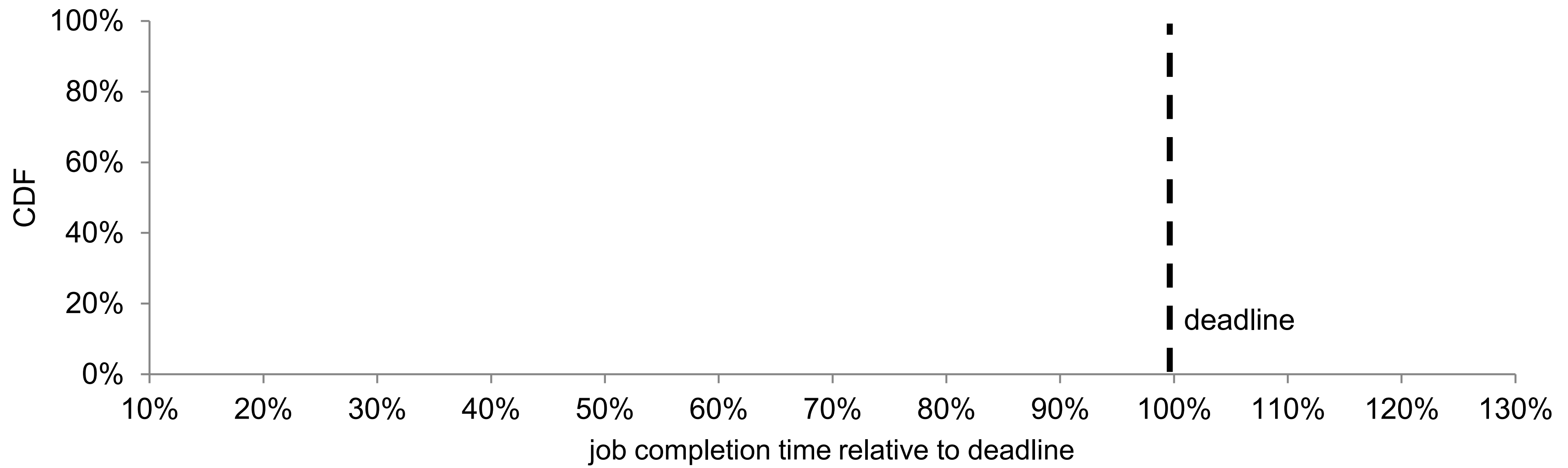
Jockey in Action



Jockey in Action



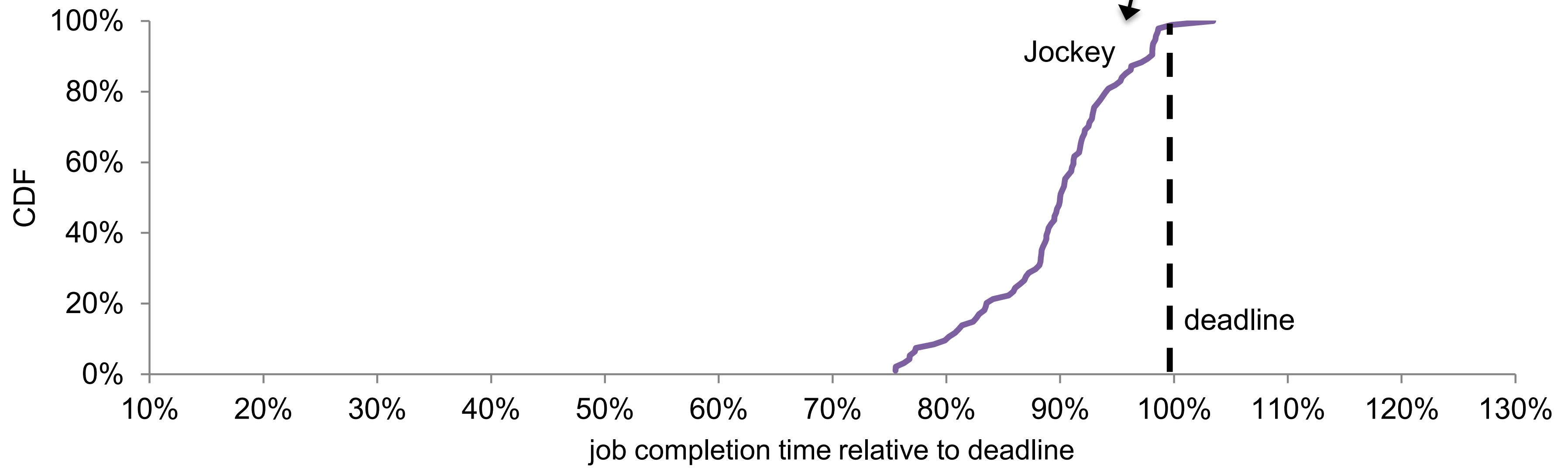
Evaluation



Jobs which met the SLO

Evaluation

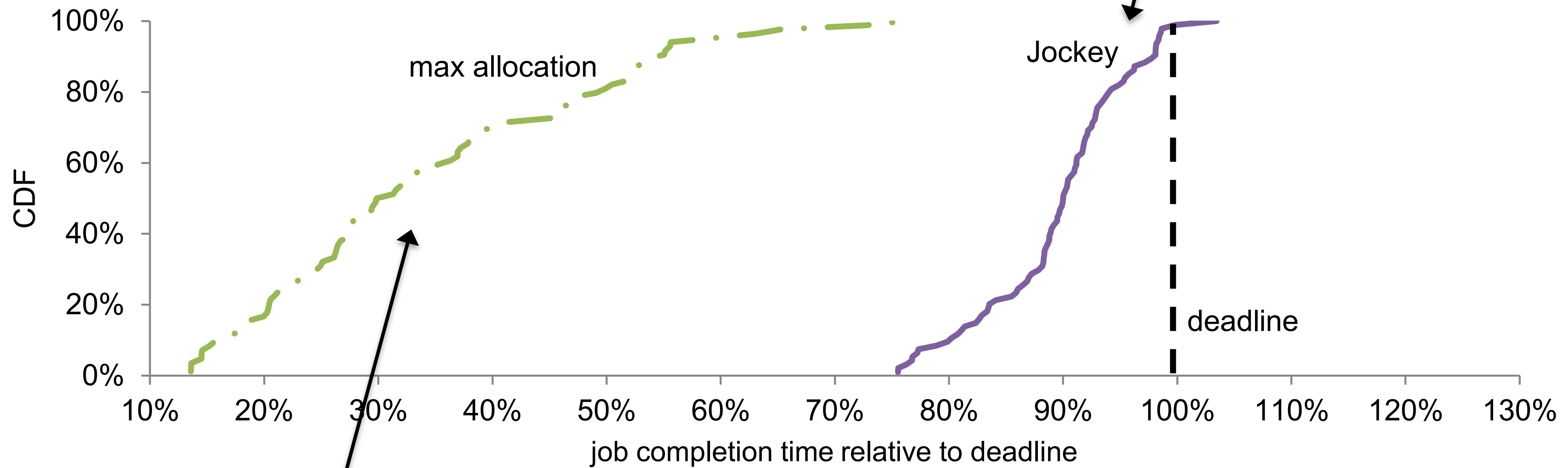
Missed 1 of 94 deadlines



Jobs which met the SLO

Evaluation

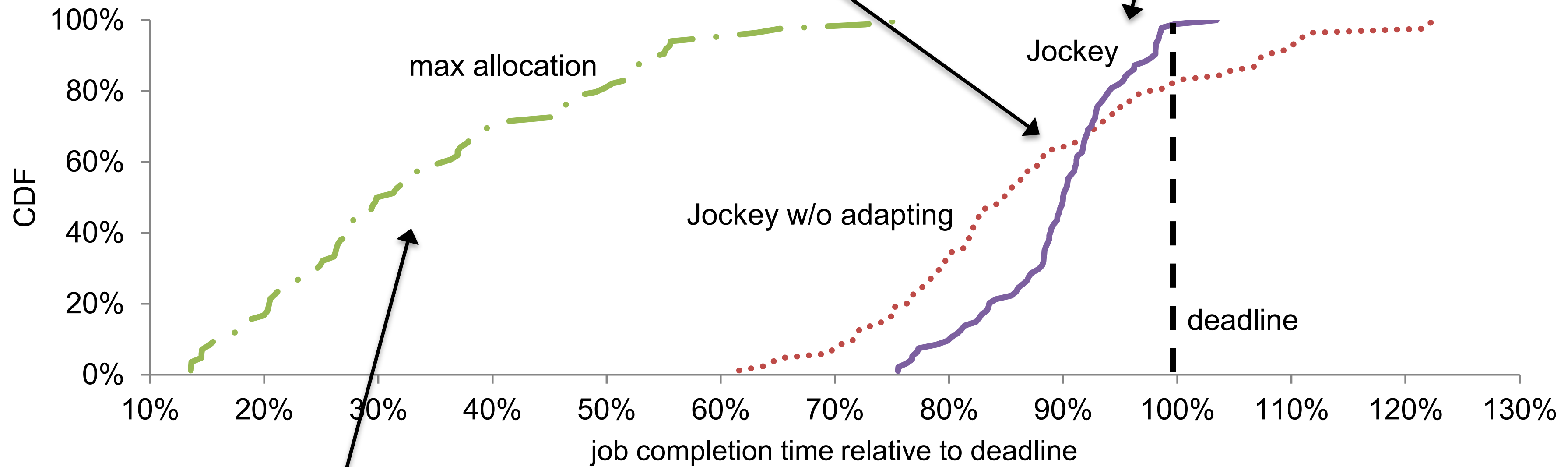
Missed 1 of 94 deadlines



Allocated too many resources

Jobs which met the SLO

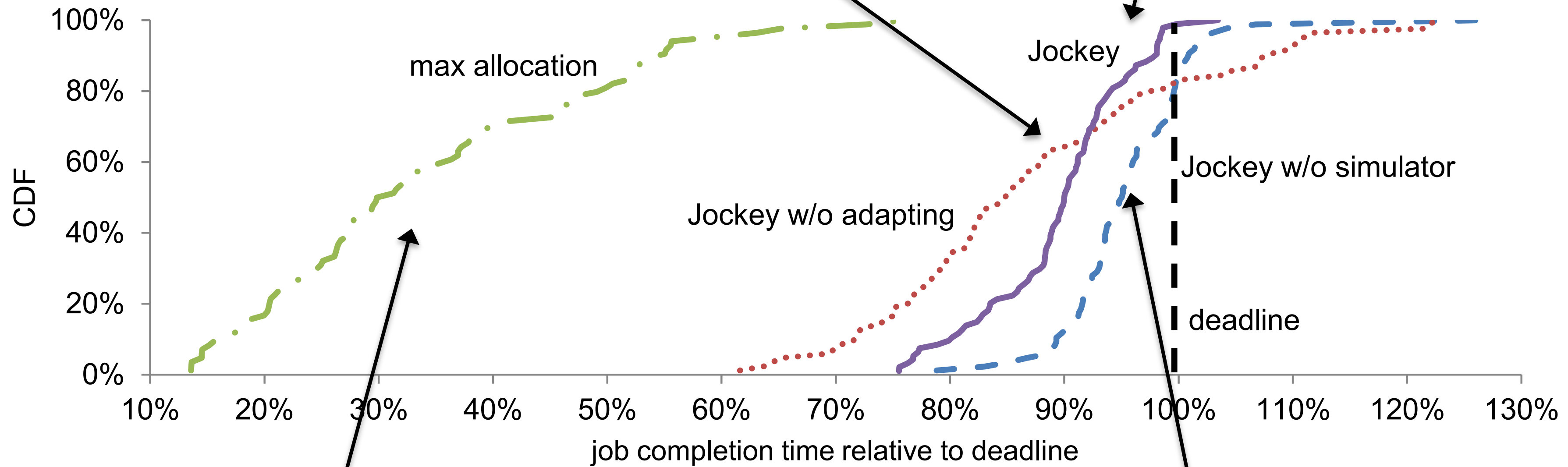
Evaluation



Evaluation

Missed 1 of 94 deadlines

Simulator made good predictions: 80% finish before deadline



Allocated too many resources

Jobs which met the SLO

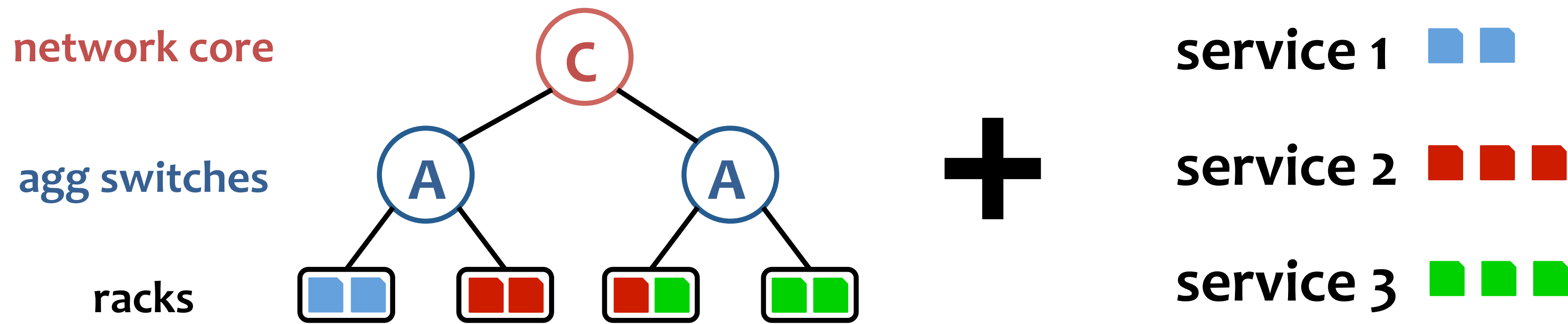
Control loop is stable and successful

What's missing?

- multiple jobs/deadlines, multiple pipelines
- better representation of job state

FAULT-TOLERANT RESOURCE ALLOCATION

How to allocate services to physical machines

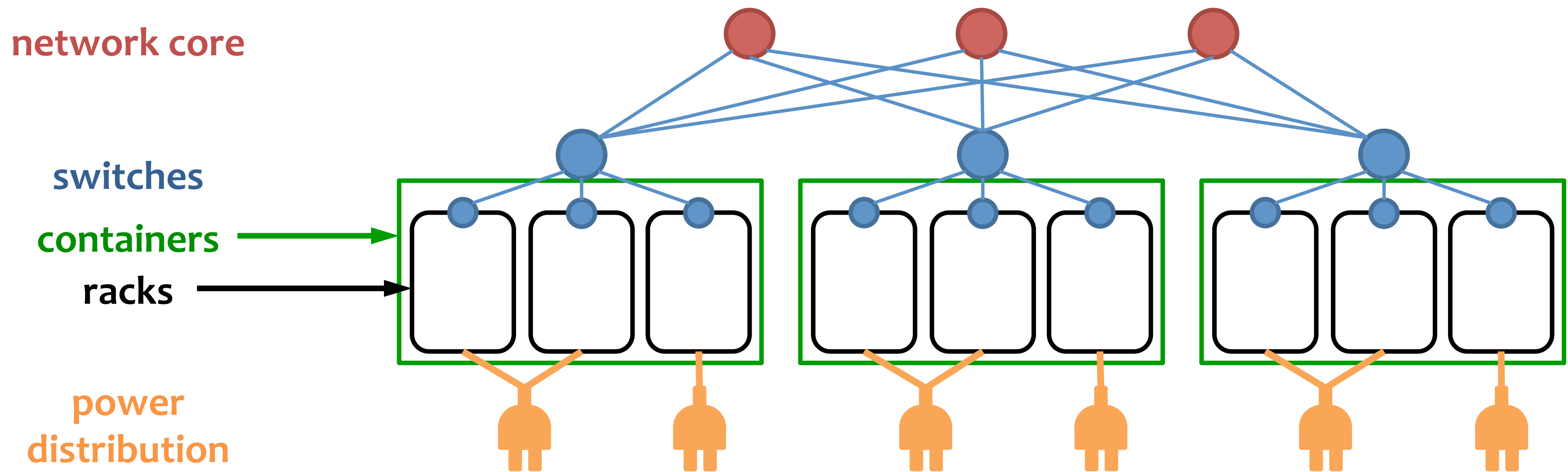


Three important metrics considered together

- FT: service fault tolerance
- BW: bandwidth usage
- #M: # machine moves to reach target allocation

OMM 2012, Surviving Failures in Bandwidth-Constrained Datacenters
Bodik, Ishai Menache, Mosharaf Chowdhury, Pradeepkumar Mani, David A. Maltz, and Ion Stoica

: Improving fault tolerance of software servic

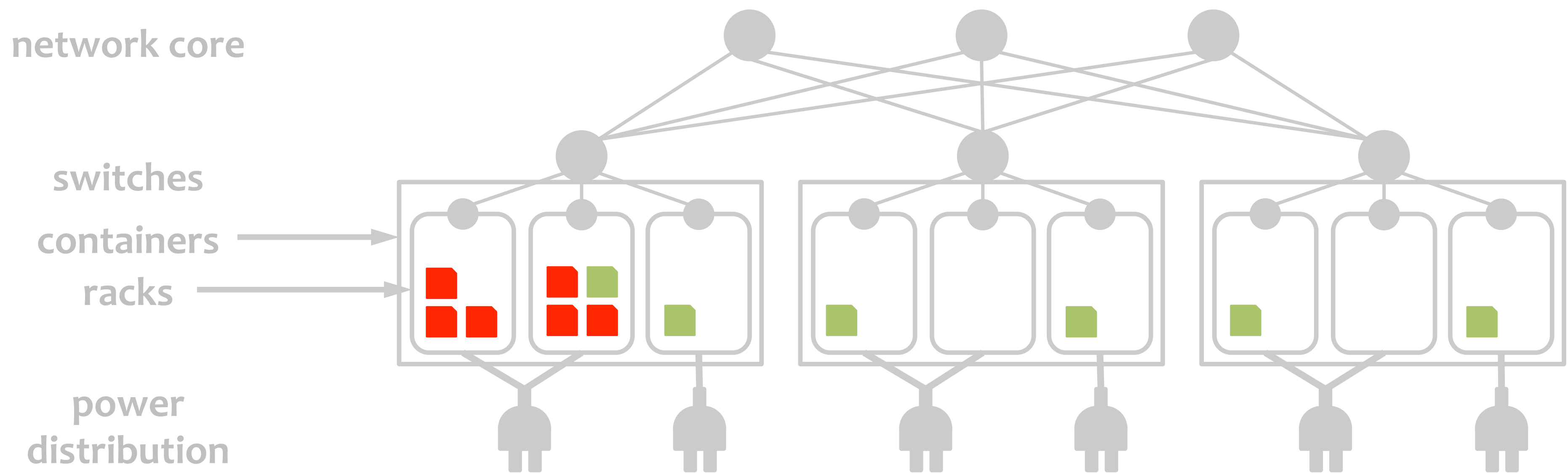


Complex fault domains: networking, power, cooling

Worst-case survival = fraction of service available during single worst-case failure

– corresponds to service throughput during failure

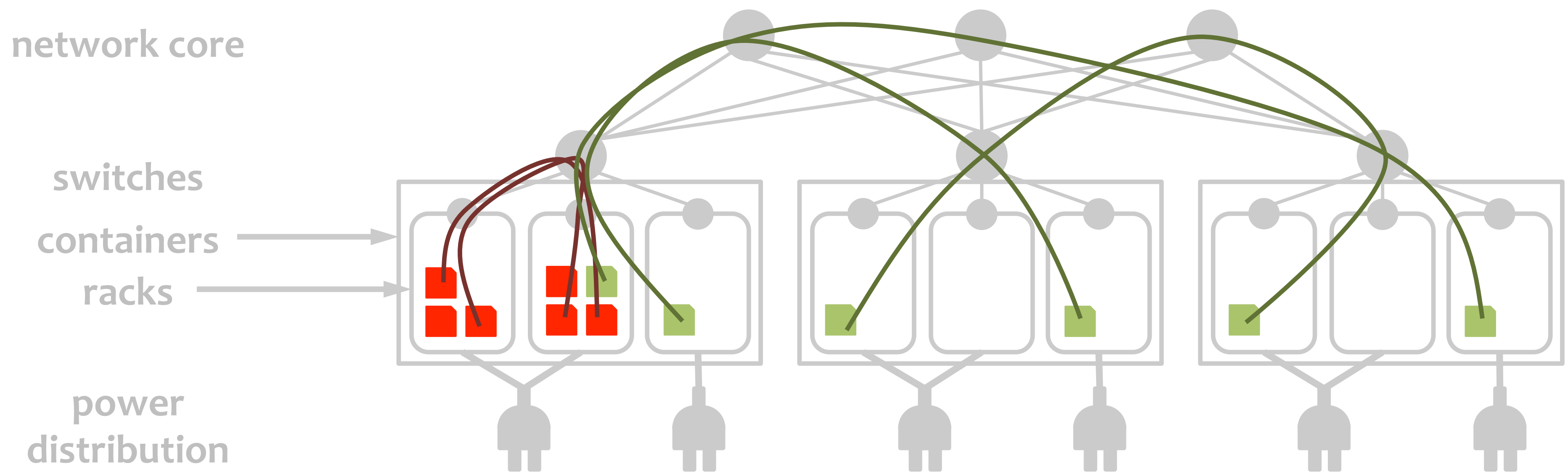
: Service allocation impacts worst-case surviv



Worst-case survival:

- red service: 0% -- same container, power
- green service: 67% -- different containers, power

BW: Reduce bandwidth usage on constrained links

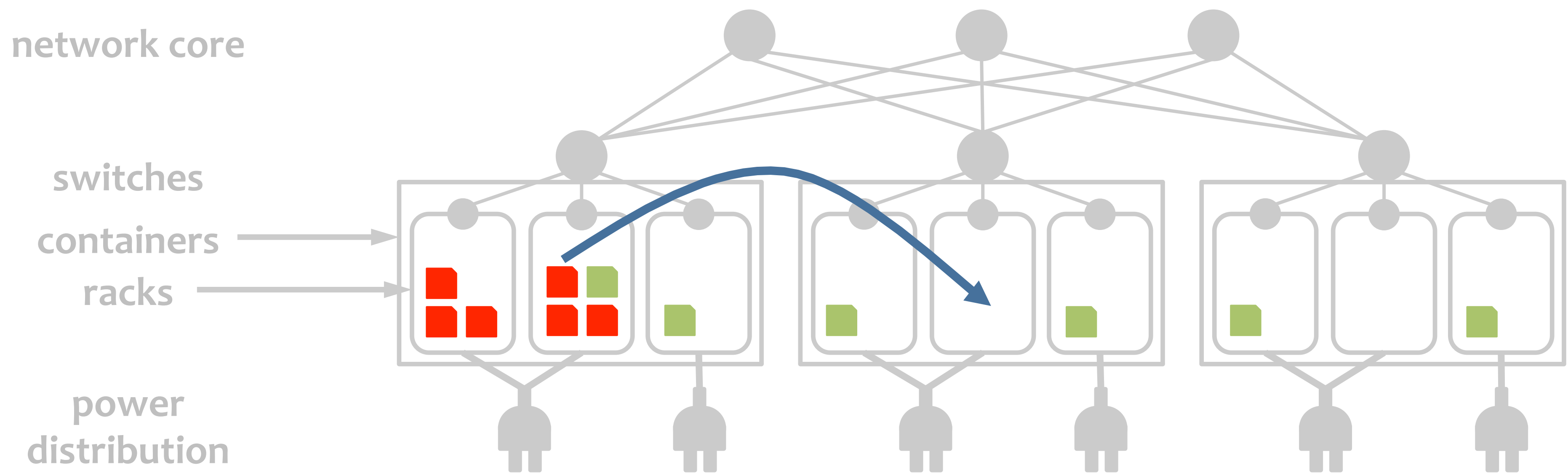


BW = bandwidth usage in the core

Goal

– reduce cost of infrastructure
– consider other service location constraints

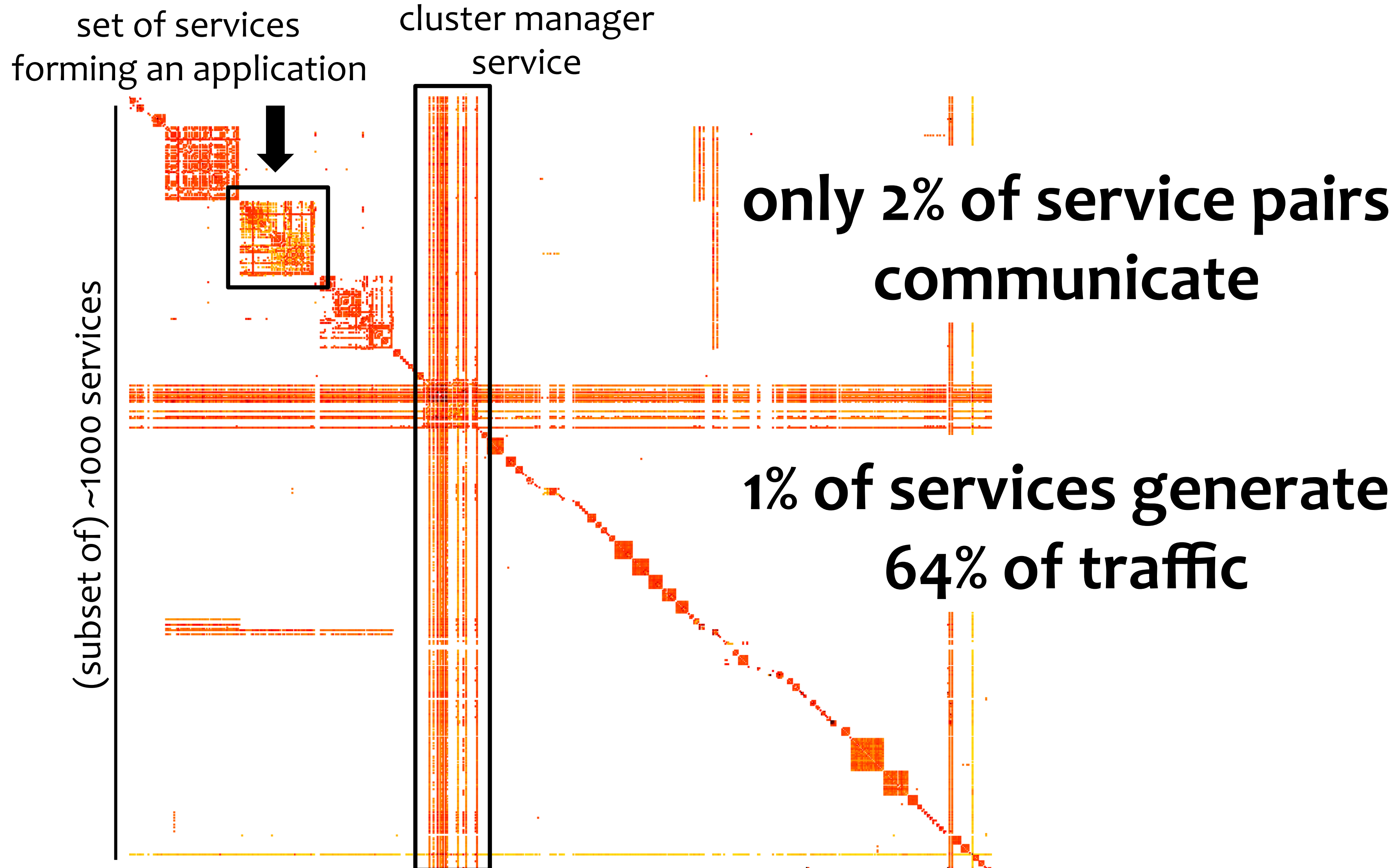
#M: Need incremental allocation algorithms



High cost of machine move

- need to deploy potentially TB of data
- warm up caches
- could take tens of min, impact network

Service communication matrix is very sparse and skewed



Formulate as convex optimization

Spread machines across all fault domains

$$\min \alpha BW + \sum_{s} c_s \sum_{f} w_f \cdot z_{s,f}^2$$

number of machines of service s in domain f

service weight

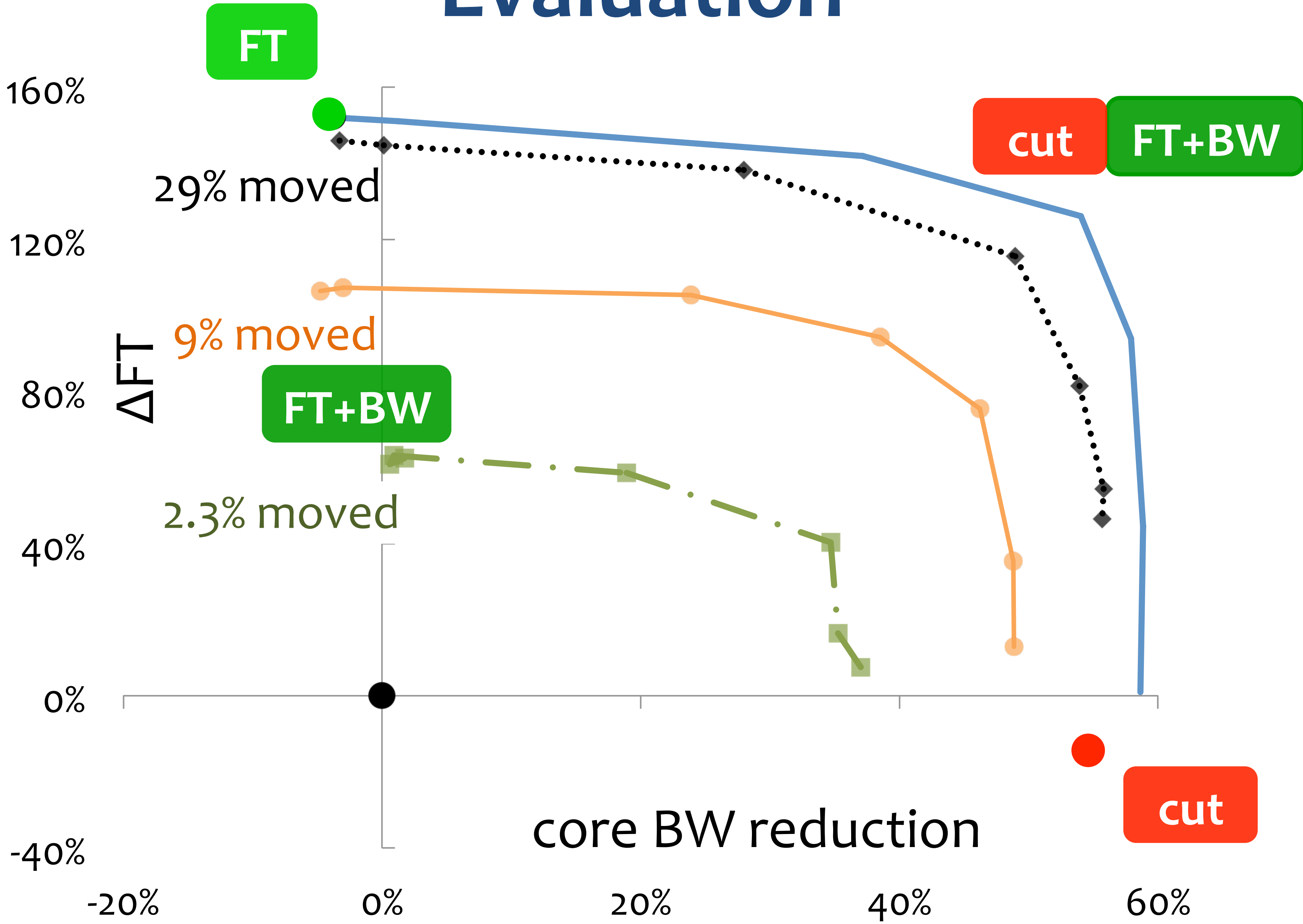
fault domain weight

Advantages of convex cost function

– local actions (machine swaps) lead to improvement of global metric

– directly considers #M

Evaluation



Potential future work

Deadline scheduling

- multiple deadline jobs (with different penalties for missing deadline)
- dependencies across jobs
- maximize total *utility*
- adapt to new jobs arriving, reduces capacity, …

Resource allocation

- consider *structure* of the applications, eg, data partitions and replications
- dependencies across services