SPOTLYTICS: HOW TO USE CLOUD MARKET PLACES FOR DATA ANALYTICS?

TIM KRASKA, ELKHAN DADASHOV, CARSTEN BINNIG

CLOUD IAAS

Idea: Rent virtual machines from and run your software (e.g., DBMS, Spark, etc.)

small

medium

large

extra large

Typical Pricing Models

- **On-demand:** fixed price per hour (e.g., 10 cent/hour)
- **Reserved:** basic fee based on contract over x years + lower hourly rate compared to on-demand

MARKET-BASED IAAS

IaaS providers overprovision their resources

Market-based IaaS: Overcapacity is sold under a dynamic pricing scheme

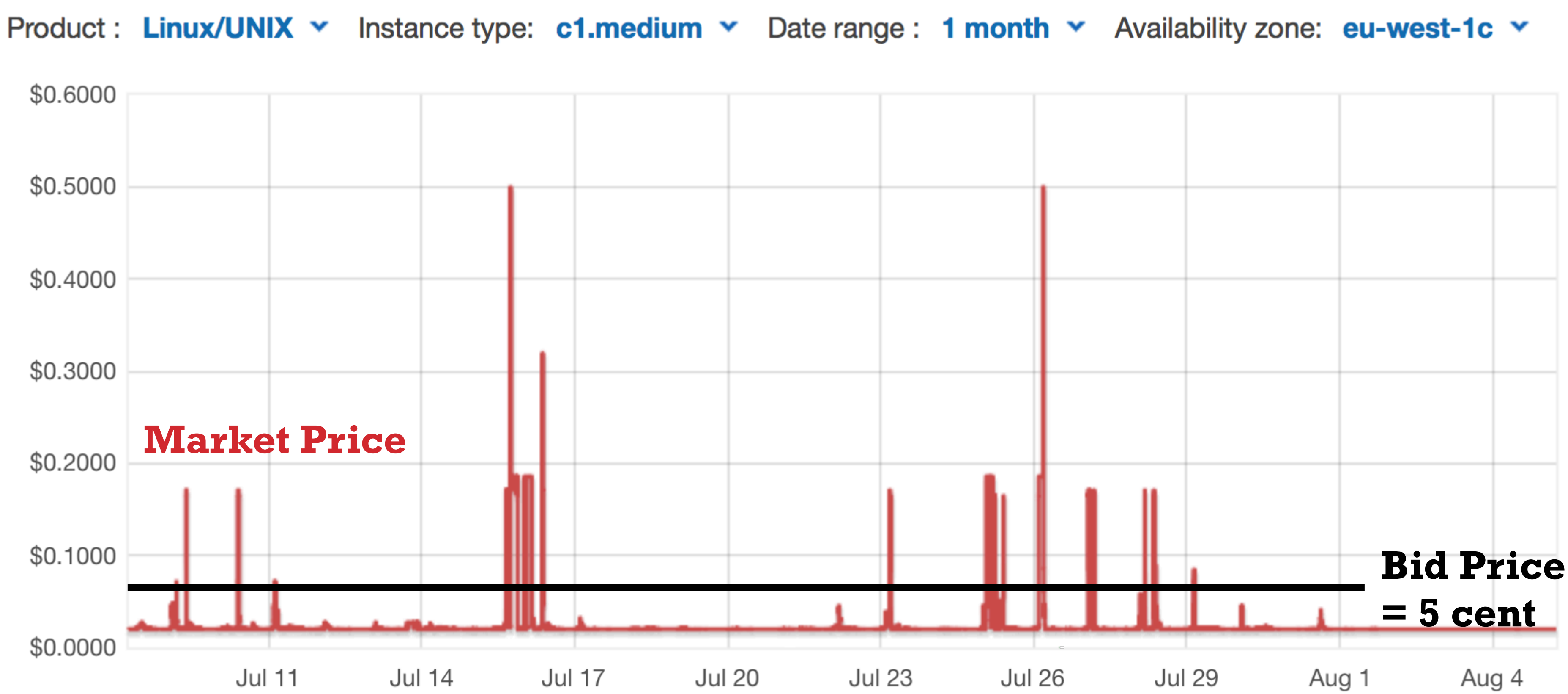
- **High Overcapacity** => Low Price
- **Low Overcapacity** => High Price (BUT also other parameters influence price)

Main provider: Amazon Spot Instances

AWS INSTANCES SPOT: USAGE MODEL

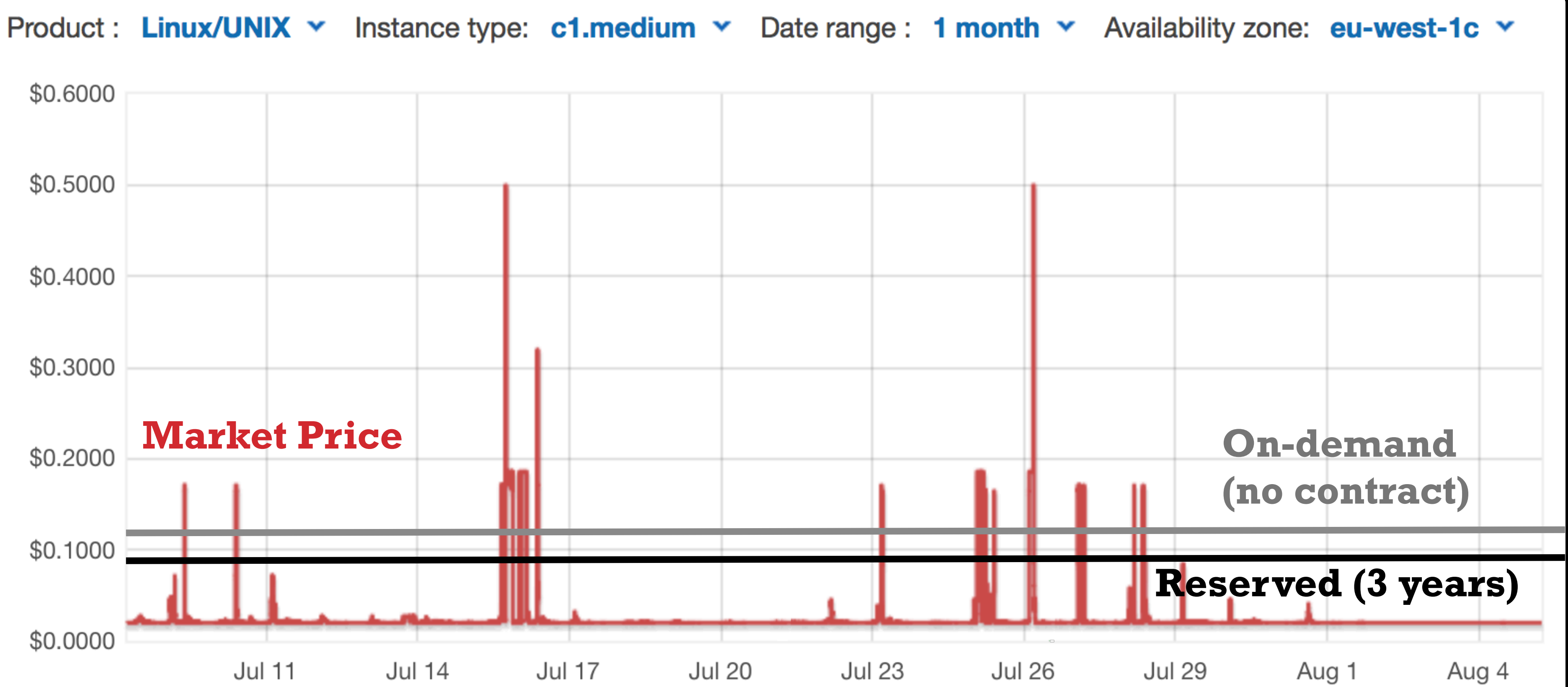
Bid Price ≥ Market Price: instance is granted

Bid Price < Market Price: instance is not granted / revoked



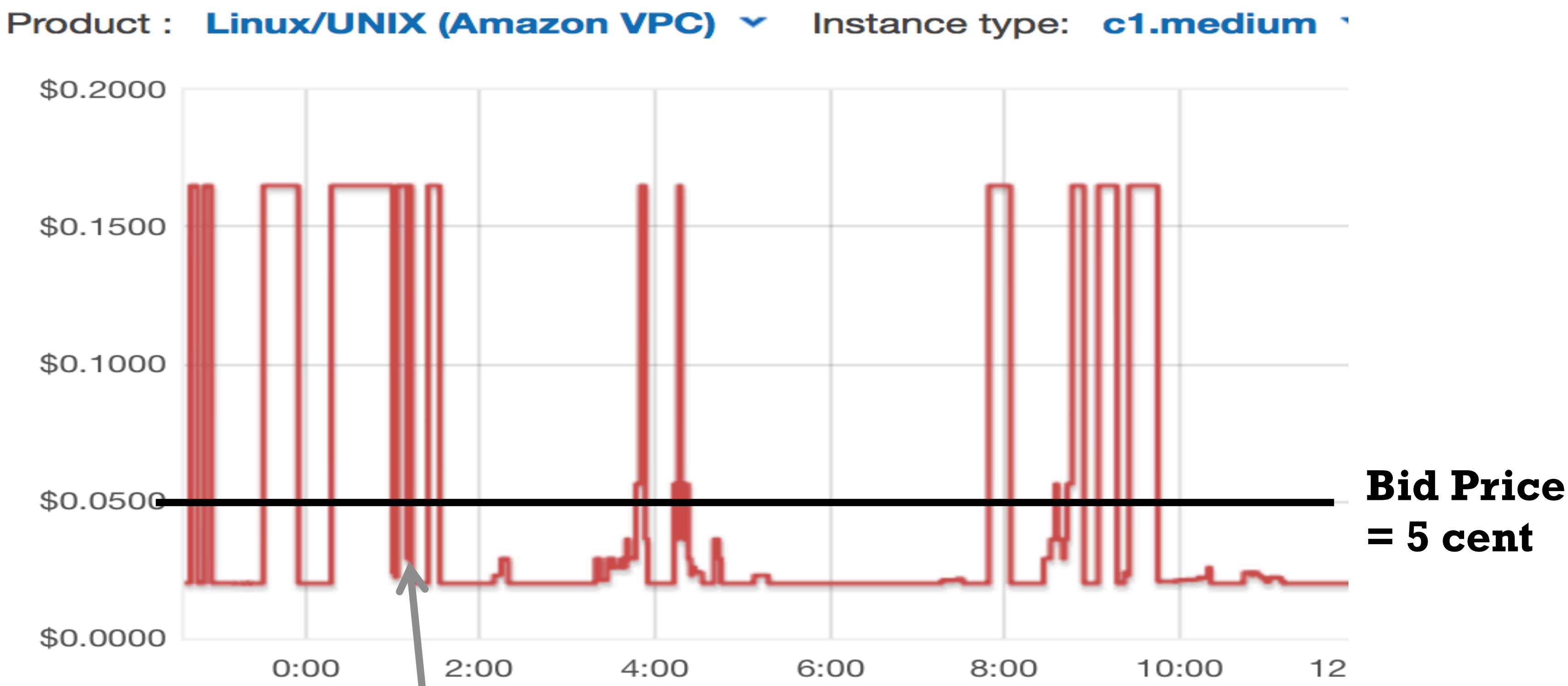
AWS SPOT INSTANCES: PRICE MODEL

Prices are different per **instance type + region + zone**



AWS SPOT INSTANCES: BILLING

Billing is based on an **interval** ε (1h for Spot)



Costs: price at launch time*intervals (re-evaluated every interval)

Discount: for non-full intervals if instance is terminated by provider

CHALLENGES FOR **ANALYTICS** ON SPOT

Main goal should be to save monetary cost

Fault-tolerance of systems plays a key role

Other Peculiarities:

- all machines of the same type fail together
- weird almost binary (high price, low price) behavior
- price fluctuations for some types suddenly stopped
- abnormally high spikes
- etc.

PROBLEM STATEMENT

- **Given job J** (e.g., Map-Reduce program, a SQL query) and a **fault-tolerance strategy FT**
- Find the best deployment strategy to **minimize the overall monetary cost** of executing Q

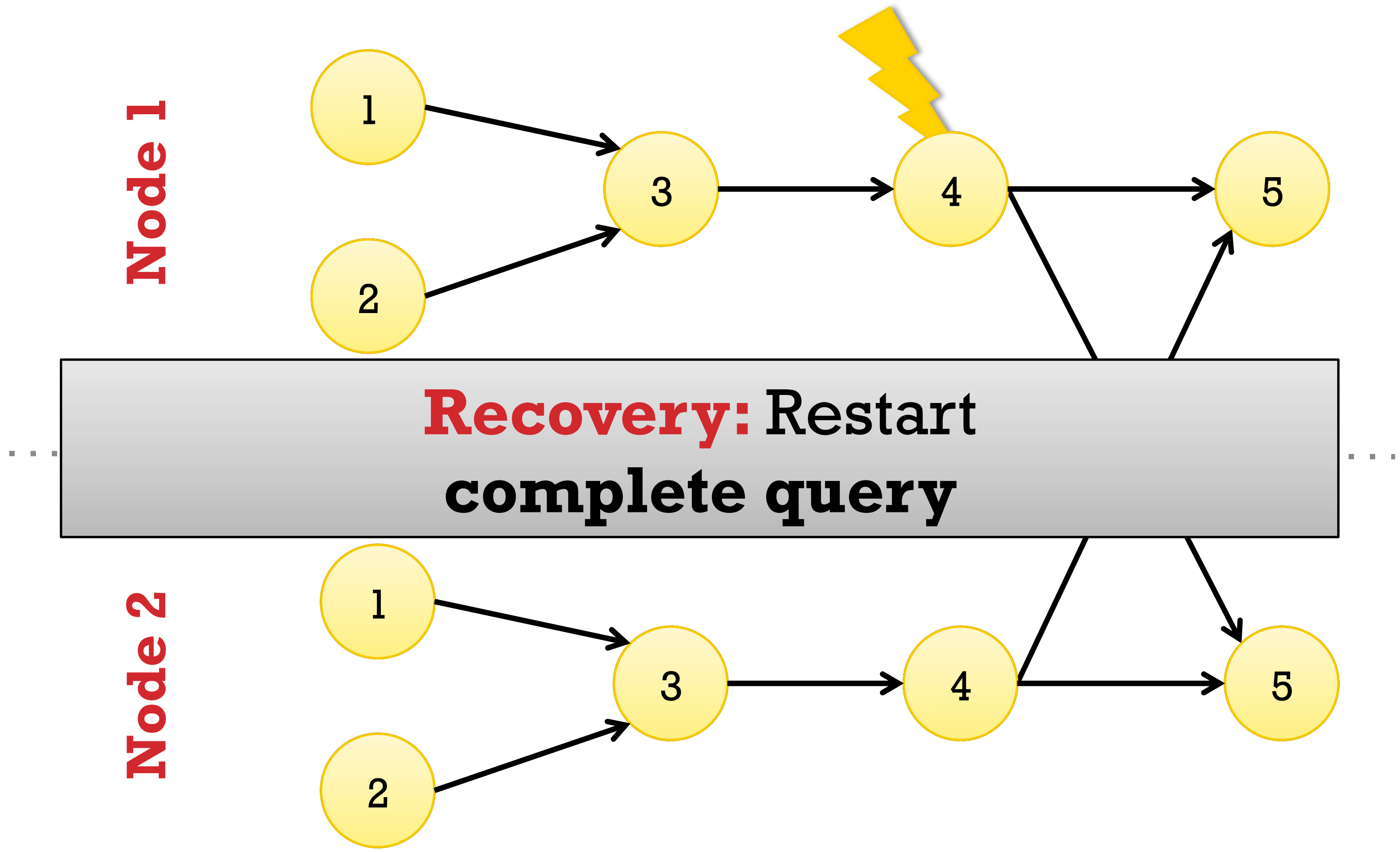
Deployment Strategy?

Price: 5c / hour

Type: 3 x m4.large

COARSE-GRAINED RESTART

Scheme implemented in a Distributed DBMS

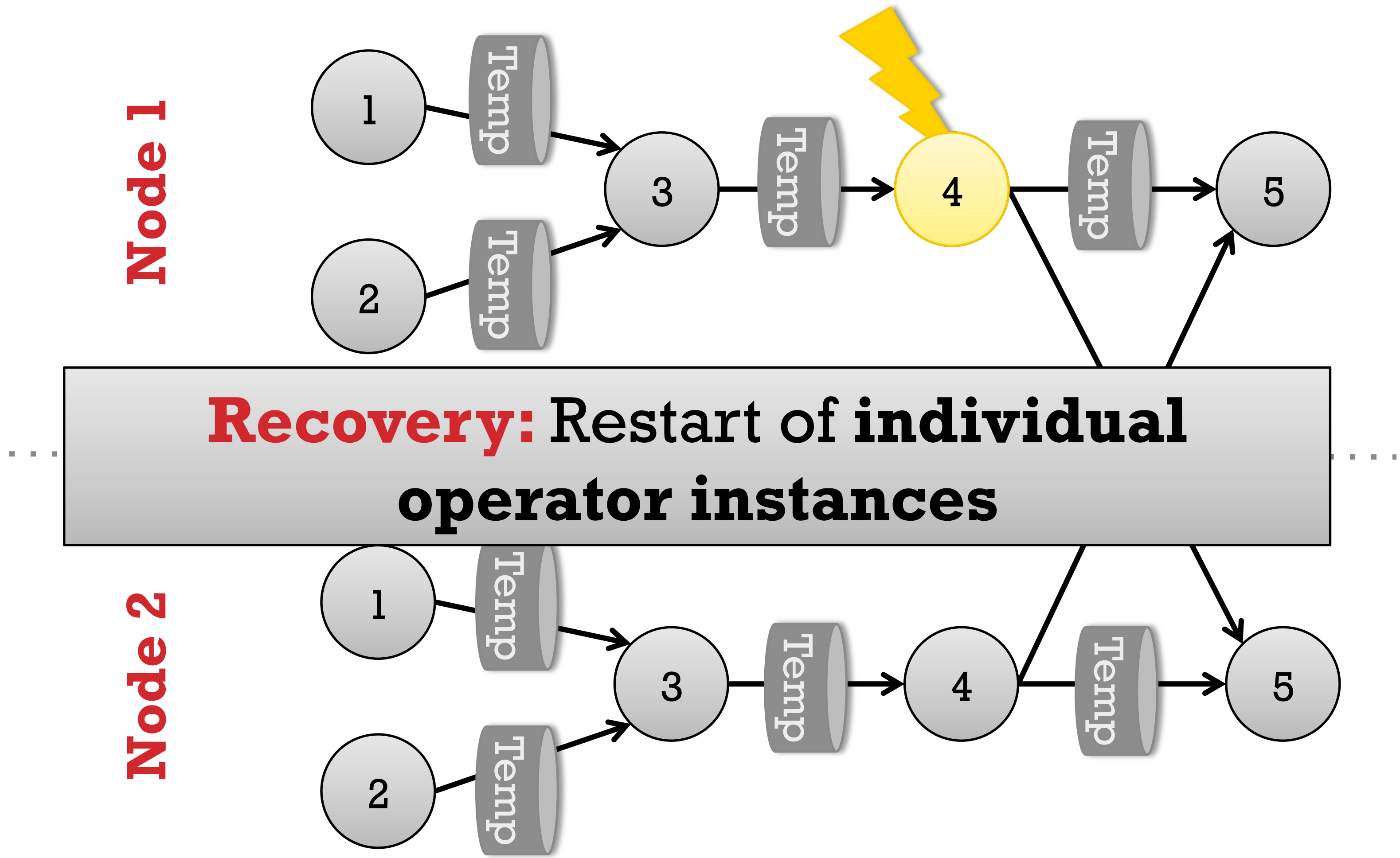


Recovery: Restart

complete query

FINE-GRAINED RESTART + CHECKPOINTS

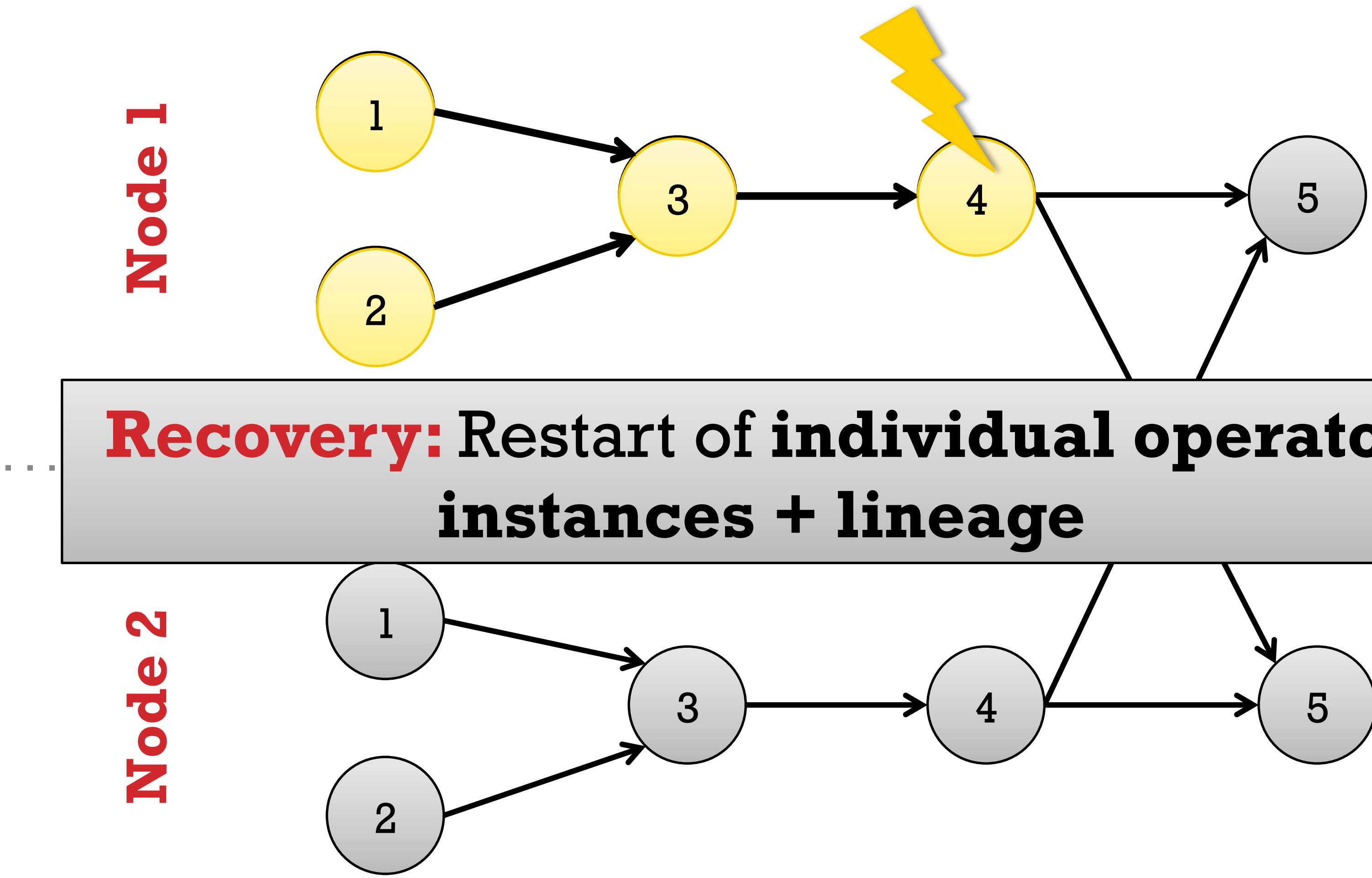
Scheme implemented in Hadoop



Recovery: Restart of **individual operator instances**

FINE-GRAINED RESTART + LINEAGE

Scheme implemented in Spark



Recovery: Restart of **individual operator instances + lineage**

CONTRIBUTIONS OF THIS PAPER?

Cost analysis for different fault-tolerance strategies

- Coarse-grained Query Restart
- Fine-grained Restart / Check pointing
- Fine-grained Restart / Lineage

Result 1. It is never beneficial to shut down an instance before the end of the billing interval ε .

COARSE-GRAINED RESTART

Runtime costs of a job J (wo failure)

- Job is composed of multiple tasks
- Runtime of task on one instance: **R**
- Runtime of task on n instances: **R/n**

On failure: Complete Restart

Result 2. Running a job in a single billing interval ε is cheaper than running the job with fewer resources over several intervals

Result 2. Running a job in a single billing interval ε is cheaper than running the job with fewer resources over several intervals

- Assume that $\boldsymbol{q \cdot m}$ is the number of **machines** to run the job in exactly **one billing interval**
- Then $\boldsymbol{m}$ the number of **machines** to run the job in $\boldsymbol{q}$ **intervals**
- Thus, **cost for a successful run are equal**
- However, **probability for failure increases** with runtime k

$$cn\left(e^{\frac{\lambda}{\varepsilon}}-1\right)\sum_{k=1}^{\frac{\varepsilon R}{n}}\left(\underbrace{e^{-\frac{\lambda(k-1)}{\varepsilon}}\left\lfloor\frac{k}{\varepsilon}\right\rfloor(1-\gamma)}_{i}+\underbrace{e^{-\frac{\lambda(k-1)}{\varepsilon}}\frac{k}{\varepsilon}}_{ii}\right)$$

COARSE-GRAINED RESTART

Runtime costs of a Job J (wo failure)

- Job is composed of multiple tasks
- Runtime of task on one instance: $\mathbf{R = R_{CPU} / I_{CPU}}$ (R_{CPU}: Total Cycles, I_{CPU}: Cycles of instance in one ε)
- Runtime of task on n instances: **R/n**

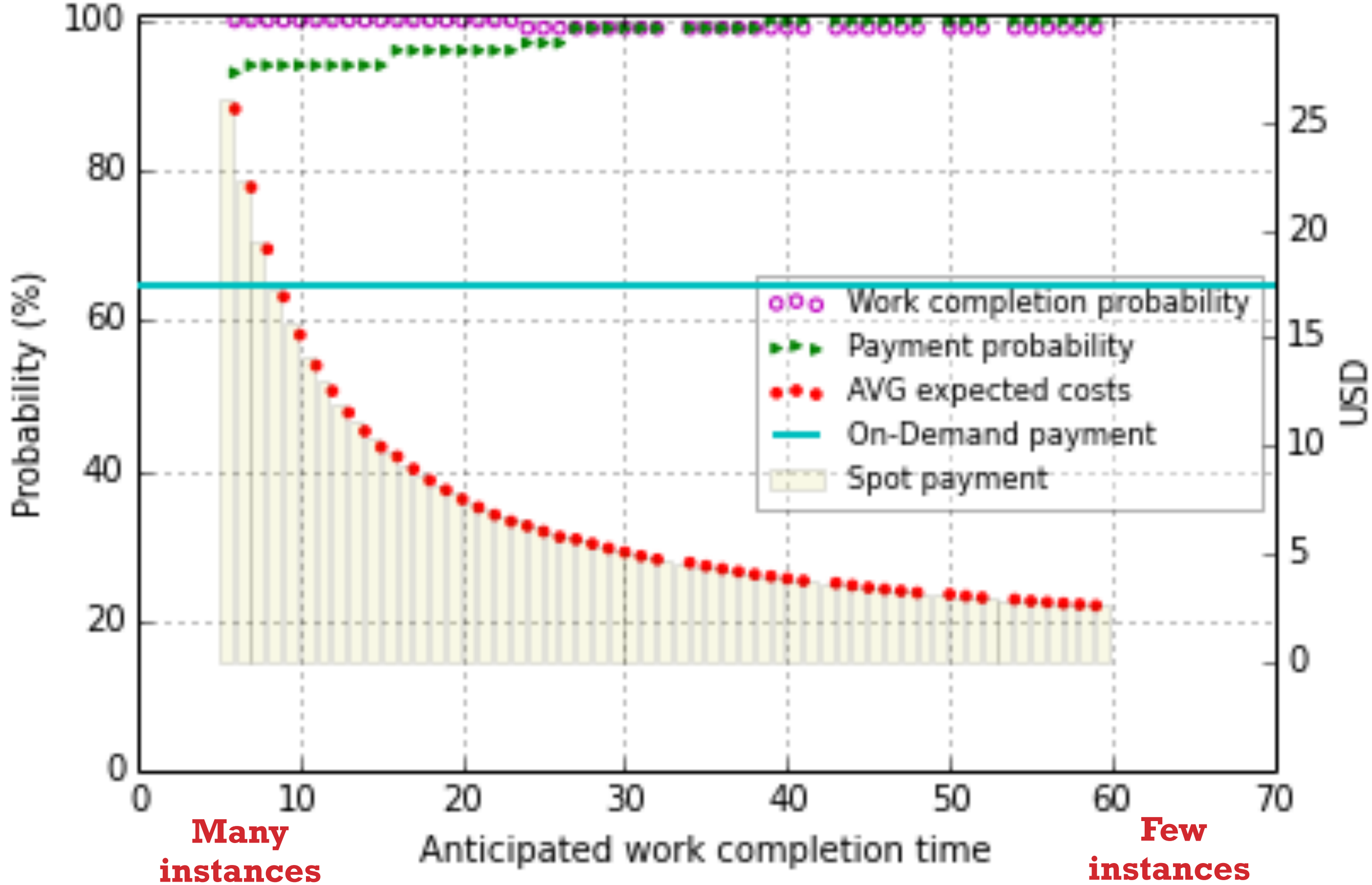
On failure: Complete Restart

Result 2. Running a job in a single billing interval ε is cheaper than running the job with fewer resources over several intervals

Result 3. Using more machines to finish early can be beneficial (depending on the failure rate λ).

EXP: VARYING # OF MACHINE

Low Failure Rate (λ =0.75 -> every 800 minutes)



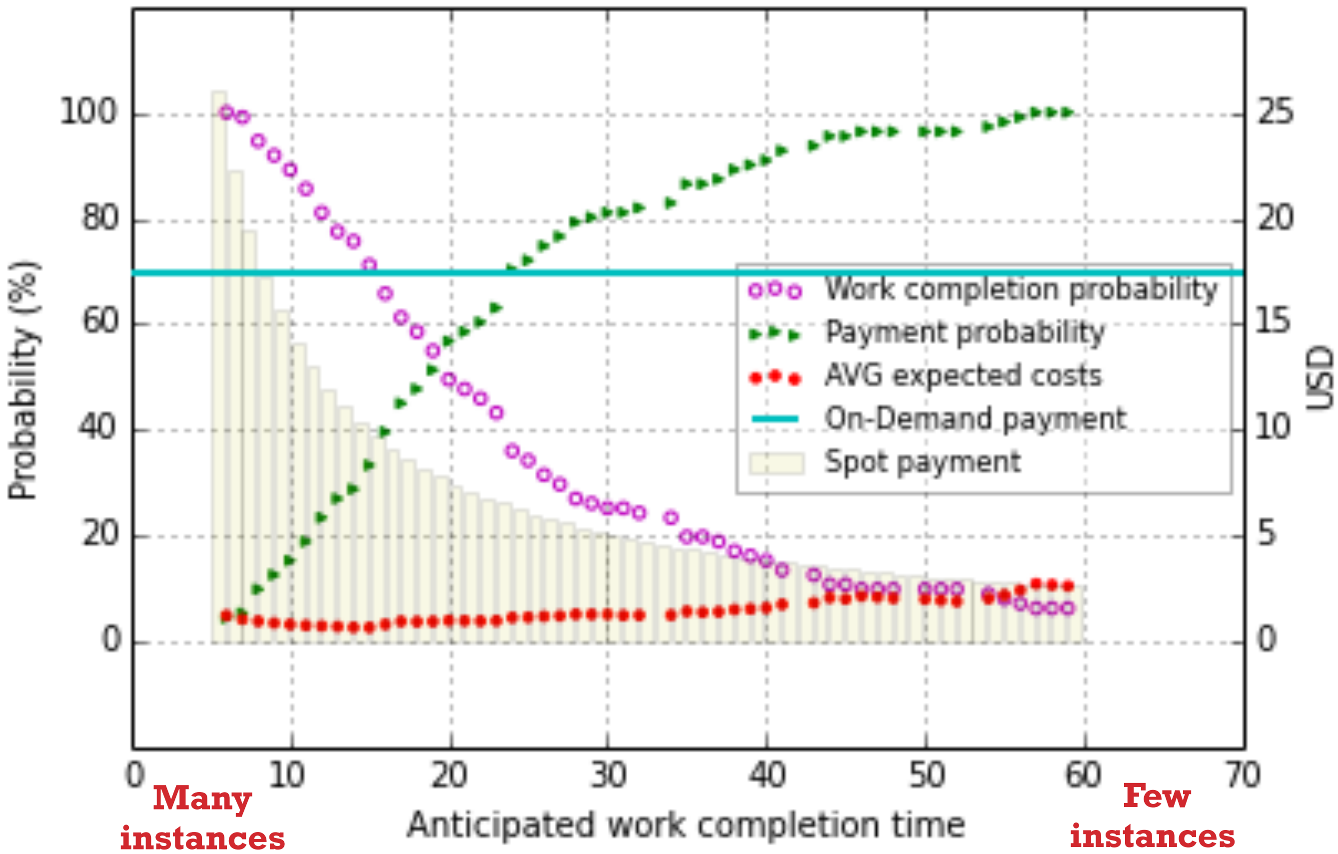
Many instances

Few instances

Setup: *us-east-1c–m1.large–Linux instance type with on-demand price of $0.175 and a bid price of $0.0263 (15% of on-demand price)*

EXP: VARYING # OF MACHINE

High Failure Rate (λ =1.8 -> every 33 minutes)



Setup: *us-east-1c–m1.large–Linux instance type with on-demand price of $0.175 and a bid price of $0.0263 (15% of on-demand price)*

FINE-GRAINED + CHECKPOINT

Result 4. The expected cost of using n or $2 \cdot n$ machines for a job is the “same” with check-pointing

Intuition:

- Checkpointing allows to resume work “w/o loosing” invested work
- Doubling machines reduces runtime by half but increases cost per billing interval by two

FINE-GRAINED + CHECKPOINT

Result 4. The expected cost of using n or $2 \cdot n$ machines for a job is the "same" with check-pointing

Intuition:

- Checkpointing allows to resume work "w/o loosing" invested work
- Doubling machines reduces runtime by half but increases cost per billing interval by two

Result 5. Using a single instance to finish a job in a single check-pointing interval is the cheapest and most risk-averse option.

Intuition:

- High variance for one interval (i.e., pay nothing or all)
- Less variance for more intervals

EXP: ONE VS. MANY MACHINES

Medium of the prices from 4 years as the bid- price

	1 instance for 100 hours		100 instance for 1 hours	
	$\bar{\mu}$ ($)	$\bar{\sigma}$	$\bar{\mu}$	$\bar{\sigma}$
m2.2xlarge	9	12	17	15
m2.4xlarge	15	18	32	30
m2.xlarge	5	5	11	7

Setup: three machine types, m2.2xlarge, m2.4xlarge, and m2.xlarge all from the us-east-1a data center

FINE-GRAINED + LINEAGE

Result 6. Same as Coarse-grained Query Restart on Spot Instances if we do not mix instance types

CONCLUSIONS

Market-based IaaS for Data Analytics

Main Contributions: Cost Analysis for different FT schemes

- Query Restart: Get more machines to pay less
- Fine-grained / Checkpointed (Hadoop): One machine saves most
- Fine-grained / Lineage (Spark): Same as query restart

Future work:

- Mixing instance types, bid prices for deployment
- Minimize runtime for given budget