ERA: A Framework for Economic Resource Allocation for the Cloud

MOSHE BABAIOFF, YISHAY MANSOUR, NOAM NISAN, GALI NOTI, CARLO CURINO, NAR GANAPATHY, ISHAI MENACHE, OMER REINGOLD, MOSHE TENNENHOLTZ, EREZ TIMNAT

ERA - What is it really?

- Economic Resource Allocation (ERA) – “A complete framework for scheduling and pricing cloud resources”.
- Main goal: increase the efficiency of cloud resources usage by allocating resources according to economic principles.
- Thanks to its unique architecture, ERA offers development of scheduling and pricing algorithms in a rather easy way.
- Interfaces with both the cloud resource manager and with the users who reserve resources to run their jobs. More on that, later on.

Current economics mechanisms

- There are 2 main economics mechanisms that cloud resource allocation is controlled by:
  1. Fixed pre-paid guaranteed quotas – mostly used in private cloud frameworks.
  2. On-demand unit prices: the users are charged real money per unit of resource used – Often used for public cloud offering.
- What's wrong with these methods?

Issues in existing Methods

- Pre-paid:
  - Often used for “non-useful” jobs.
  - No real sharing of common resources.
- On-demand :
  - No guarantee of resource availability.
  - Appealing mainly to low-value or highly time flexible jobs.

What’s the problem?

- Let us define our problem:
  - “finding a pricing and a scheduling scheme that will result in highly desired outcome, that of high **efficiency**“.
- But what is efficiency?
  - “Maximize the value that all users get from the system”
- We want to maximize the marginal value.

What's the problem?

- Generally speaking – we should consider the value obtained rather than the resources used, and we should aim to maximize this **value-based** notion of efficiency.
- The Architectural Challenge: provide a common abstraction that consists of all the considerations (scheduling, pricing, resource sharing, etc.).

The Solution!

- The ERA system is designed as an intermediate layer between the cloud users and the cloud infrastructure.
- *ERA Cloud Interface*, hides many of the details of the resource management infrastructure.
- Not all resources in the cloud have to be managed via ERA.

The ERA approach

- ERA presents a pricing model that enables sharing of resources and smoothing of demands for both low and high value jobs. Using the concept of **reservations** and **willingness to pay**.
- It is an approach that focuses mainly on the economic challenges that we face in pricing and scheduling methods.
- ERA manipulates the price dynamically.
- The Mechanism ensures that the final price would be the lowest possible.

How does it work?

- ERA has two main outward-facing APIs as well as a key internal API.
- The user uses “reservation requests” in order to receive resources from the cloud.
- A basic reservation example:
  - *“I need 100 containers (with 6GB and 2cores each) for 5 hours, some time between 6am and 6pm today, and am willing to pay up to $100 for it.”*
- The basic prediction model uses traces of previous runs to estimate future demand.

Bidding Reservation Model with Dynamic Prices

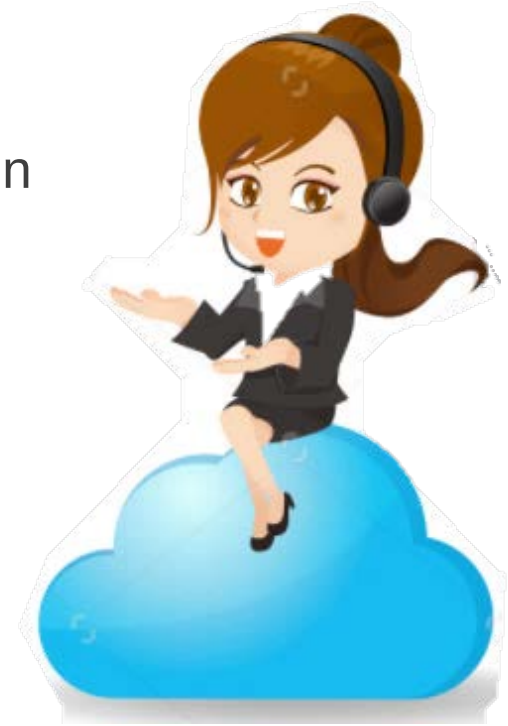
- ERA uses a simple bidding model in which the user attaches to each of his job requests a monetary value specifying the maximal amount he is willing to pay for running the job.
- The price for resources is quoted at reservation time and is dynamically computed according to demand and the changing supply.
- Demand is also algorithmically estimated.
- This mechanism also ensures that at peak times – where demand can simply not be met – the most “valuable” jobs are the ones that will be run rather than arbitrary ones.
- The main idea is that user flexibility in timing is automatically rewarded by lower prices. This behavior encourages the user to be more flexible when making a reservation.

The Cloud Model

- The cloud is modeled as an entity that sells multiple resources, bundled in configurations, and the capacity of these resources may change over time.
- The cloud is defined by:
  1. A set of formal resources for sale (core, GB, etc.).
  2. A set of resource configurations. A configuration is defined by a bundle of formal resources (e.g., “ConfA” equals 4 cores and 7 GB).
  3. Inventory: the amount of resources (formal and virtual) over time.
  4. Time definitions of the cloud (e.g., the precision of time units that the cloud considers).

ERA-Cloud Interface

- The cloud-scheduler interface offered by ERA is composed of two main methods that allow the cloud to get information about the allocation of resources it should apply at the current time:
  1. The getCurrentAllocation method.
  2. The update method.



The “getCurrentAllocation” method

- The main interface with the actual cloud scheduler.
- The cloud should repeatedly call this method and ask ERA for the current allocations to be made.
- Returns an *allocation*, which is the list of jobs that should be instantaneously allocated resources and the resources that should be allocated to them.
- Example (of a simple case of a single resource):

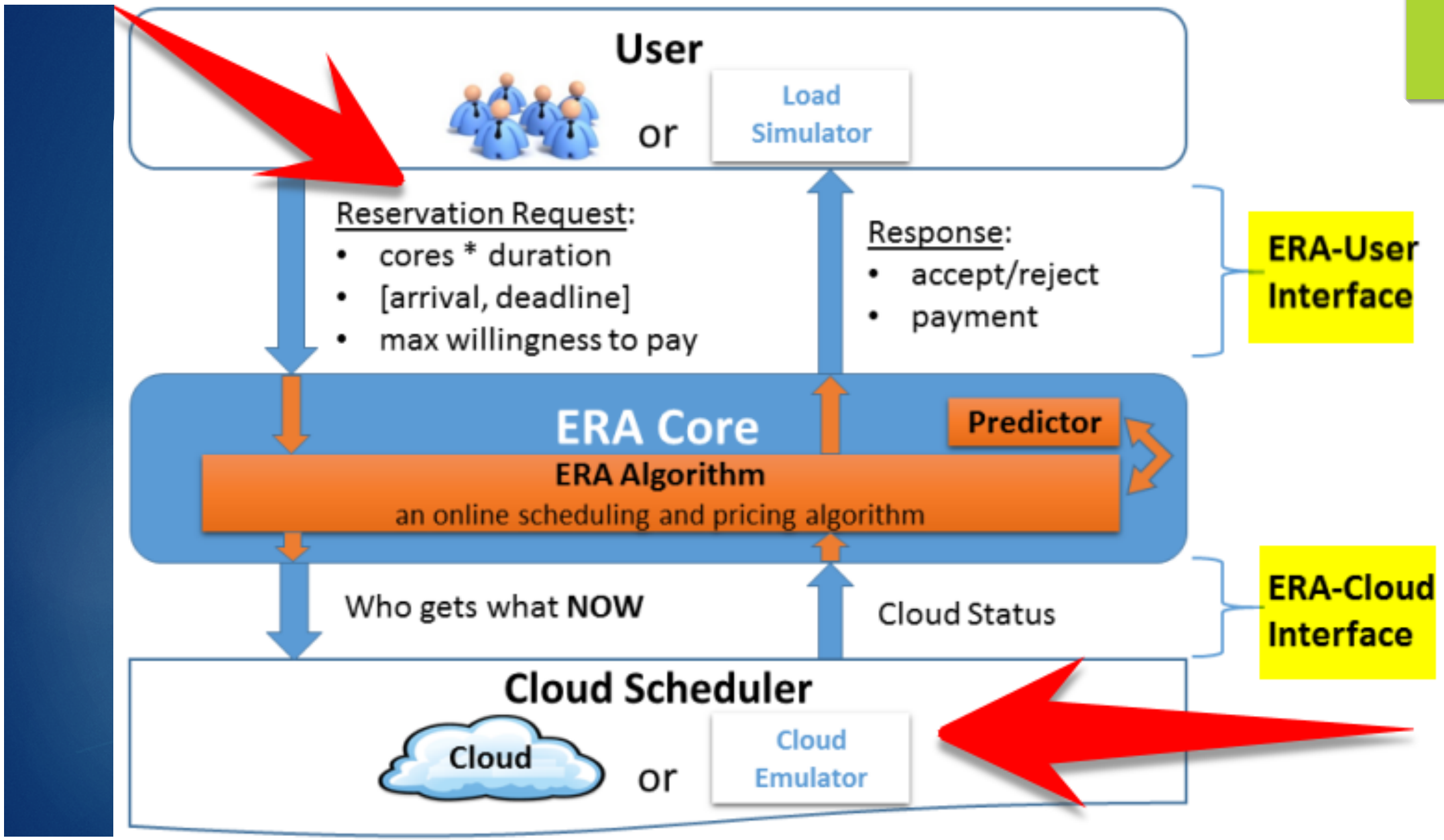
  “*job J should now be getting W resources*”.
- An allocation remains in effect until a future query returns a different allocation. The cloud should poll ERA to determine which allocation should be put in effect.

The “update” method

- The cloud may use this method to periodically update ERA with its actual state.
- The simple version of the cloud feedback includes:
  1. changes in the current resources under the cloud’s management (e.g., if some computers crashed);
  2. the current resource consumption.
  3. Termination of jobs.
  4. The number of waiting processes of each job,

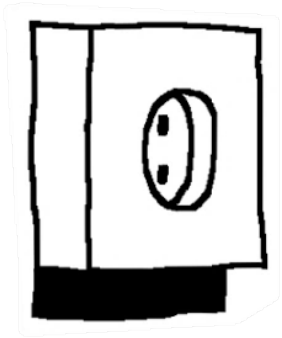
ERA-User Interface

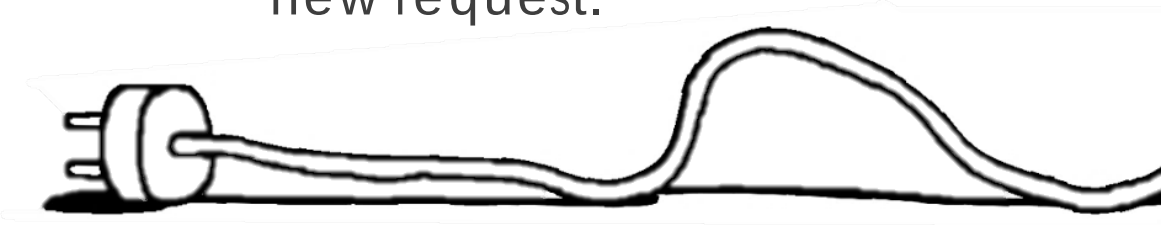
- The interface with the user is done by the "*makeReservation*" method, which handles a job reservation request that is sent by some user.
- Each reservation request can either be accepted and priced or declined by ERA.
- The basic input parameters to this method are the job's bid and the identifier of the job.
- The request is considered fulfilled as long as ERA provides the requested resources within the time window.
- The output: an acceptance or rejection of the request, and the price that the user will be charged for fulfilling his request in case of acceptance.


User
or
Load Simulator
Reservation Request:
• cores * duration
• [arrival, deadline]
• max willingness to pay
Response:
• accept/reject
• payment
ERA-User Interface
ERA Core
Predictor
ERA Algorithm
an online scheduling and pricing algorithm
Who gets what NOW
Cloud Status
ERA-Cloud Interface
Cloud Scheduler
Cloud
or
Cloud Emulator

“Plug and Play” design

- There are 2 separate components that hold the internal algorithmic implementation of ERA: *algorithm* and *prediction* components.
- This design allows to easily change between different implementations to fit different conditions and system requirements. Therefore the name – “Plug and Play”.
- The algorithm component is where the actual scheduling and pricing of job requests are performed.
- The ERA system updates the prediction component online with every new request.



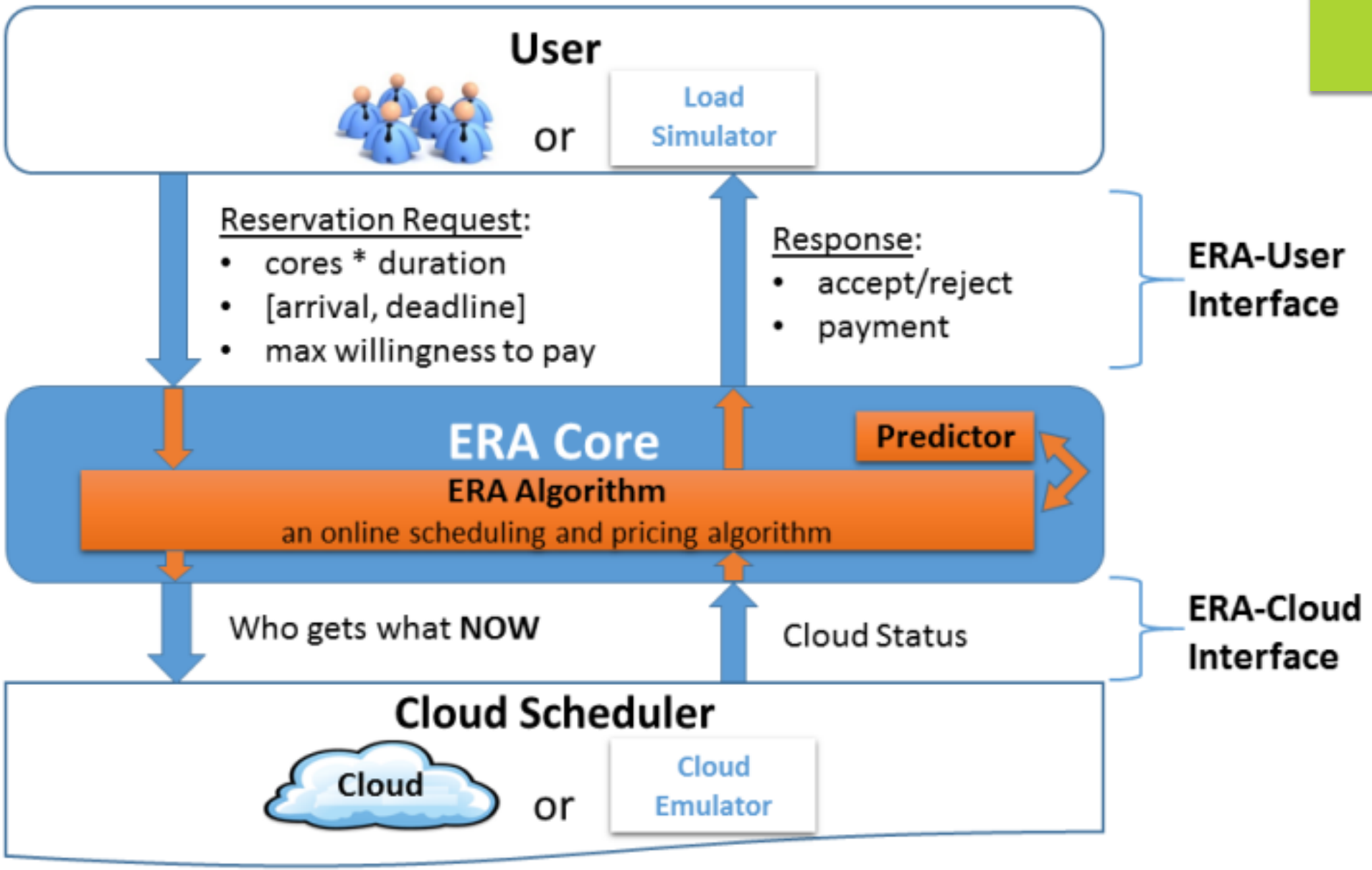


Basic Econ Scheduling

- This is the most basic ERA algorithm.
- In general, the ERA algorithm is an online scheduling and pricing algorithm that provides the logic of an ERA system.
- It is based on “unit prices”:
  - Set a price dynamically for each time and unit in a resource request - calculate seconds * cores.
  - The total price would be the sum over all the “unit prices” for the requested resources.
- The algorithm schedules a job to start at the cheapest time (within its requested window) that fits the request.

Algorithm 1 Basic-Econ Scheduling

Input: a new job request {W*T in [A,D), V}
Output: accept or reject, and a price if accepted
procedure MAKE RESERVATION
    for each $t \in [A, D)$ **do**
        $demand_t() \leftarrow$ the demand estimate function at t
        for each $i \in [1, W]$ **do**
            $price_t(i) \leftarrow$ the highest price p s.t. $demand_t(p) + promised[t] + i > Capacity$
        $cost[t] \leftarrow price_t(1) + price_t(2) + ... + price_t(W)$
    for each $t \in [A, D - T]$ **do**
        $totalCost[t] \leftarrow cost[t] + ... + cost[t + T - 1]$
    $t^* \leftarrow \arg\min_{t \in [A, D-T]} totalCost[t]$
    if $V \geq totalCost[t^*]$ **then**
        schedule the job to start at t^*
        return accept at cost $totalCost[t^*]$
    else
        return reject
end procedure

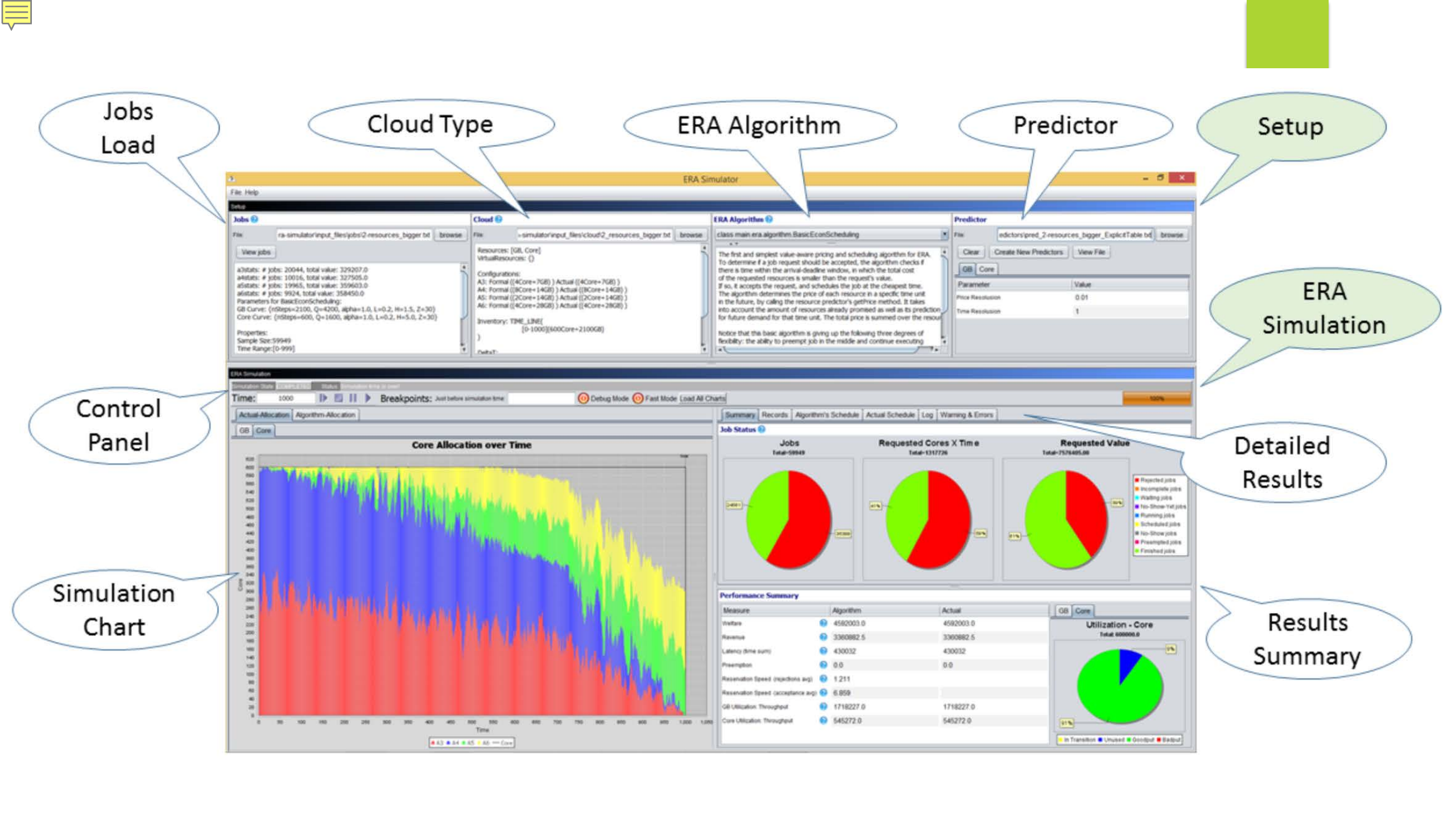

User
or
Load Simulator
Reservation Request:
• cores * duration
• [arrival, deadline]
• max willingness to pay
Response:
• accept/reject
• payment
ERA-User Interface
ERA Core
Predictor
ERA Algorithm
an online scheduling and pricing algorithm
Who gets what NOW
Cloud Status
ERA-Cloud Interface
Cloud Scheduler
Cloud
or
Cloud Emulator

The Simulator

- The simulator provides an evaluation of key metrics, both “system ones” such as loads or latency, as well as “economic ones” such as revenue.
- We have performed extensive runs of ERA within the simulator as well as proof-of-concept runs with 2 resource managers: the full system was interfaced with **Hadoop/YARN** and the C# version of the code was interfaced and tested with **Microsoft’s Azure** Batch simulator.
- These runs show that the ERA algorithms succeed in increasing the efficiency of cloud usage.

The Simulator

- We compare ERA's Basic-Econ scheduling algorithm with a greedy algorithm that does not take into account the values of the jobs.
- The simulation shows that the greedy algorithm populates most of the cluster with the large, low-value jobs and this results in a low efficiency of only 10% of the total requested value.
- ERA's Basic-Econ algorithm achieves 51% of the requested value (note that getting 100% is not possible as the cloud is too small to fit all jobs).

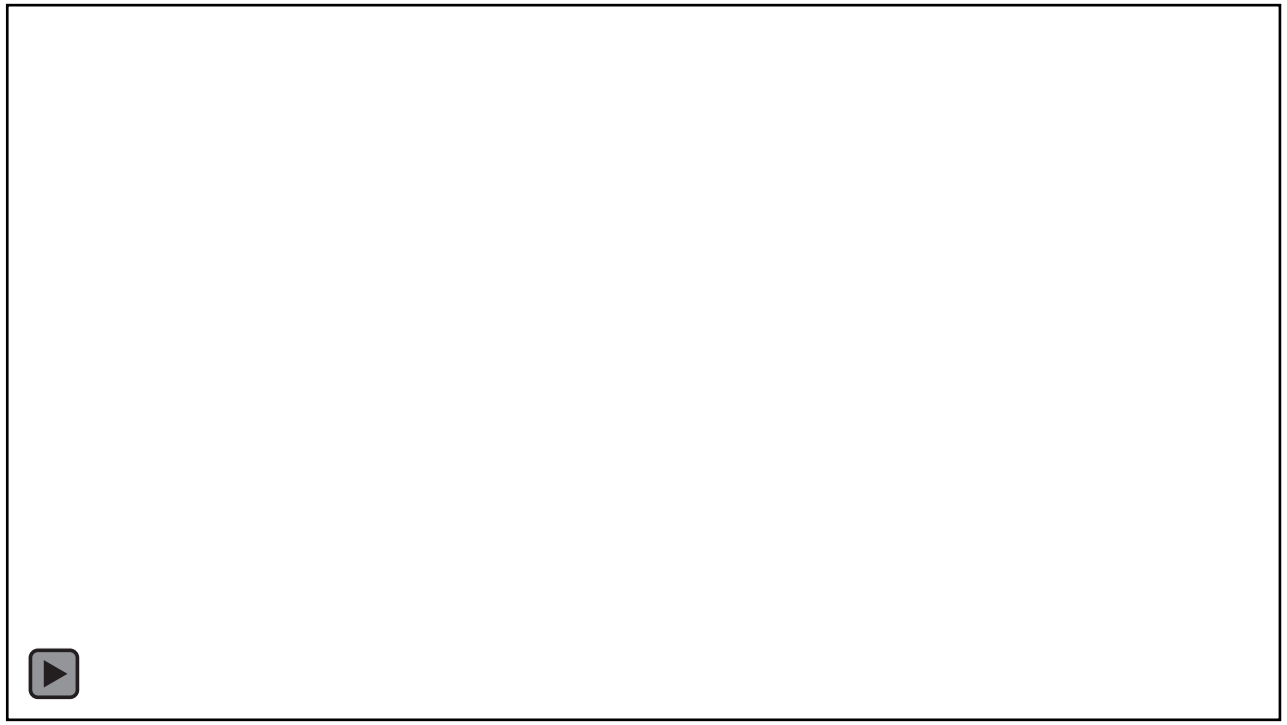

Jobs Load
Cloud Type
ERA Algorithm
Predictor
Setup
ERA Simulator
File Help
Setup
Jobs
Cloud
ERA Algorithm
Predictor
browse
View jobs
Clear
Create New Predictors
View File
GB
Core
Parameter
Value
Price Resolution
0.01
Time Resolution
1
ERA Simulation
Control Panel
Time:
Breakpoints:
Debug Mode
Fast Mode
Load All Charts
Actual-Allocation
Algorithm-Allocation
Core Allocation over Time
Time
Core
Summary
Records
Algorithm's Schedule
Actual Schedule
Log
Warning & Errors
Job Status
Jobs
Requested Cores X Time
Requested Value
Detailed Results
Simulation Chart
Performance Summary
Measure
Algorithm
Actual
Welfare
4582003.0
Revenue
3360882.5
Latency (time sum)
430032
Preemption
0.0
Utilization - Core
Results Summary

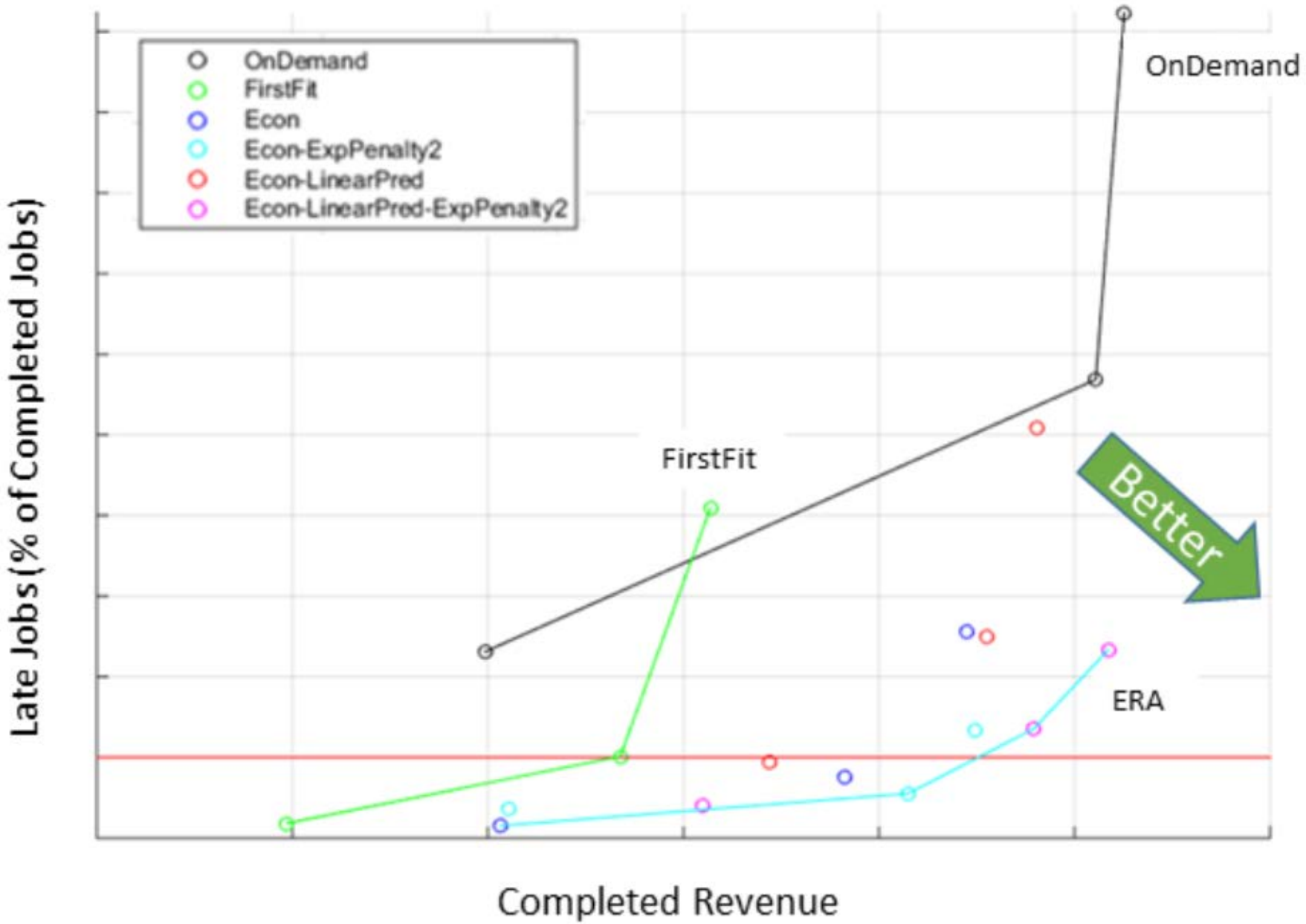
Testing Azure Batch

- The simulations were of a datacenter consisting of 150K cores.
- ERA was given access to 20% of the resources and the remaining 80% were allocated to non-ERA requests, which were modeled using the standard Azure jobs.
- The test was ERA vs. 2 other algorithms:
  1. On-demand algorithm - accepts jobs if there are enough available resources to start and run them. Charges a fixed price.
  2. Greedy (“FirstFit”) algorithm.

Testing Azure Batch

- In a typical cloud environment, we cannot expect one instance of ERA to have complete control of millions of cores.
- The goal is to evaluate whether ERA will work with a subset of cores in a region, even while the underlying resource availability is constantly changing.
- Tried to evaluate 2 measure metrics:
  1. Late-job percentage: the percentage of jobs that finished later than their deadlines.
  2. Accepted revenue: as we can charge only for jobs that are accepted, the better the algorithm, the more jobs we can accept.

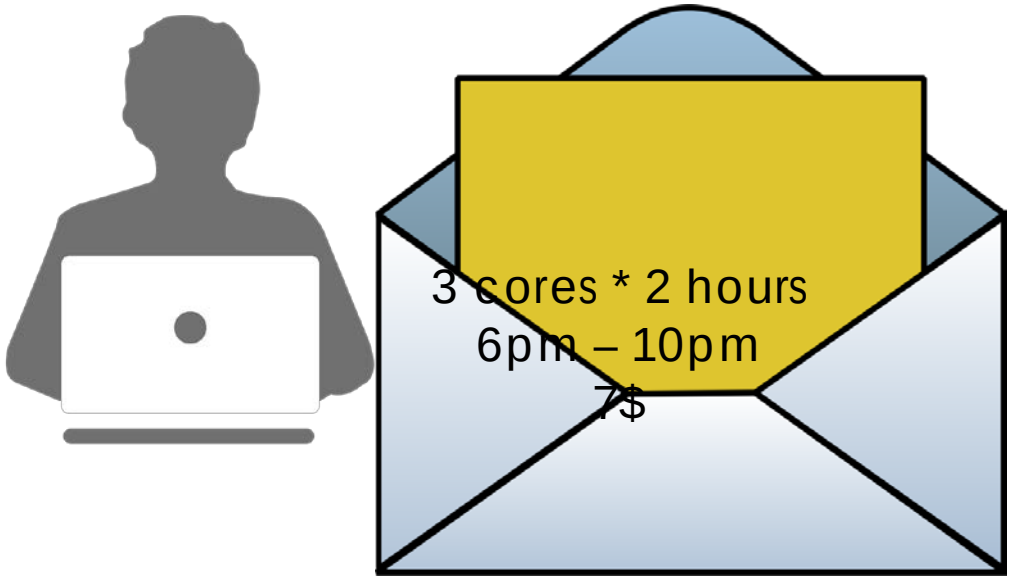



OnDemand
FirstFit
Econ
Econ-ExpPenalty2
Econ-LinearPred
Econ-LinearPred-ExpPenalty2
OnDemand
FirstFit
Better
ERA
Late Jobs (% of Completed Jobs)
Completed Revenue

Grand Challenges

- The main challenge is to get the ERA system integrated in a real cloud system, and interface with real paying costumers.
- Pricing
- Learning
- Robustness

Thanks for listening!



ERA