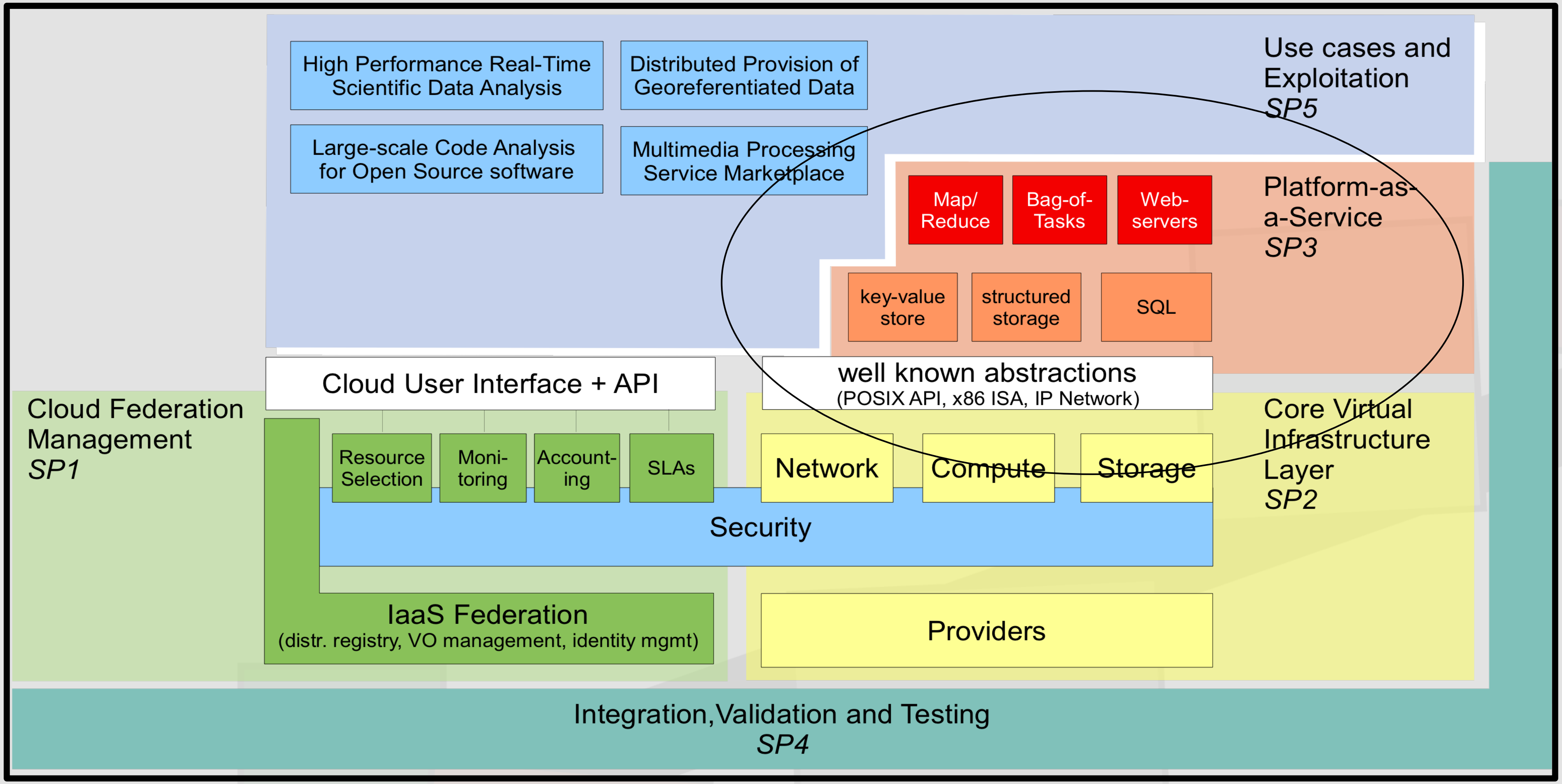
Task Farming in Contrail

Ana Oprescu, *Thilo Kielmann,* Haralambie Leahu
Vrije Universiteit, Amsterdam
Alexandra Vintila, Politechnica University, Bucharest

contrail is co-funded by the EC
7th Framework Programme

The Contrail Project

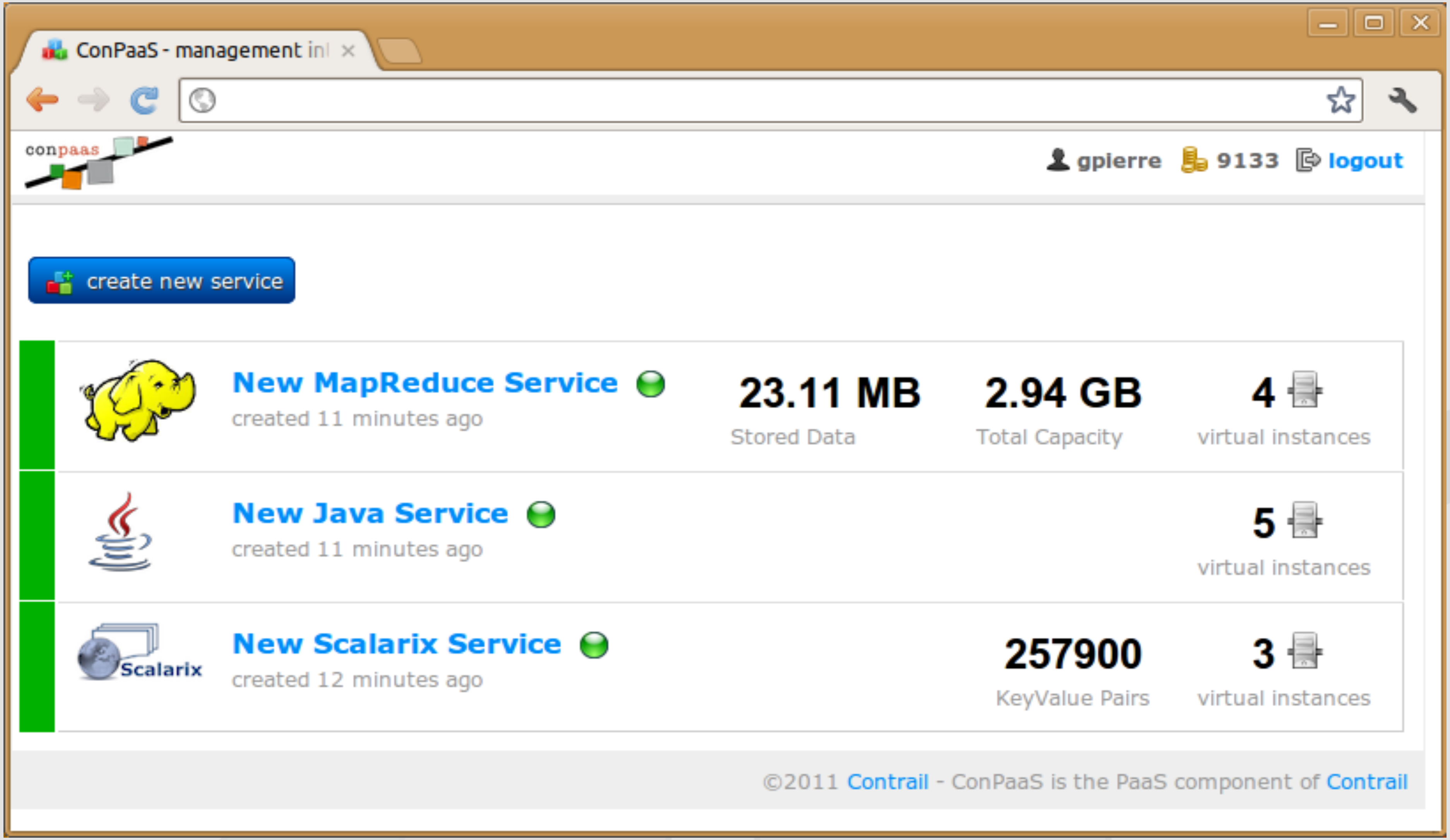


ConPaaS

- Contrail's Platform as a Service
  - PHP-based Web applications
  - MySQL
  - MapReduce
  - ***Task Farming***
  - XtreemFS files system

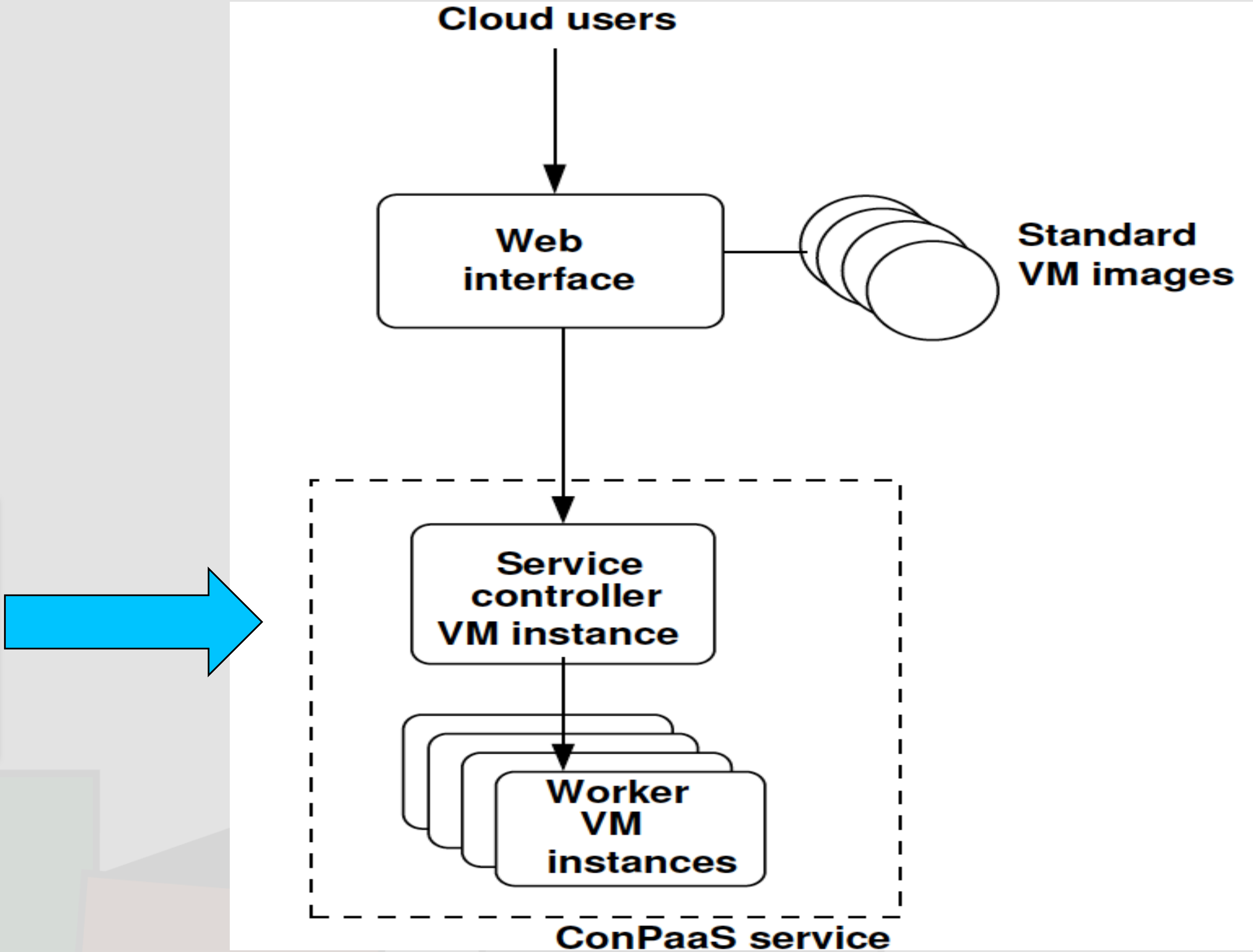
  - Accessible via a common Web GUI

ConPaaS GUI


ConPaaS - management in
gpierre
9133
logout
create new service
New MapReduce Service
created 11 minutes ago
23.11 MB
Stored Data
2.94 GB
Total Capacity
4
virtual instances
New Java Service
created 11 minutes ago
5
virtual instances
Scalarix
New Scalarix Service
created 12 minutes ago
257900
KeyValue Pairs
3
virtual instances
©2011 Contrail - ConPaaS is the PaaS component of Contrail

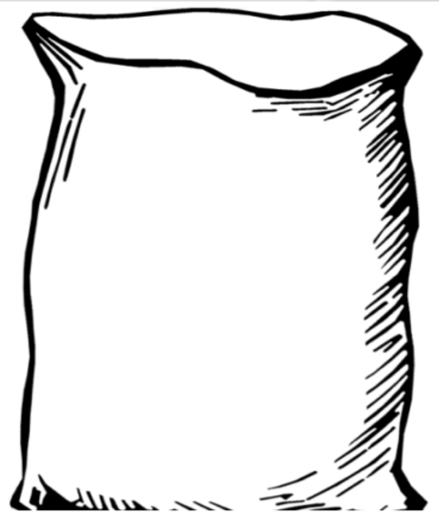
ConPaaS Service Architecture

Today:
Task farming
service

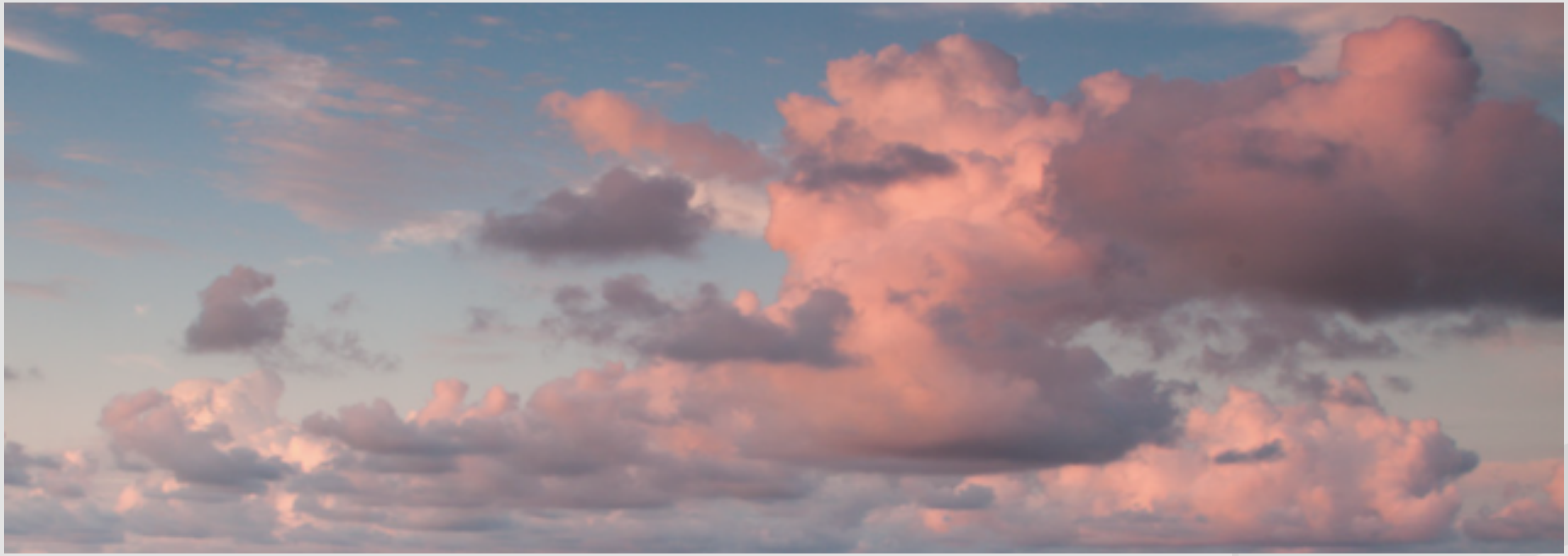


Task Farming

- Dominant application type in grids
  - over 75% of all submitted tasks
  - over 90% of the total CPU-time consumption
  - [Iosup,Epema et al.]
- High-throughput applications (Condor style)
  - Parameter sweep
- Traditional execution model "grab and run"
  - Get as many machines as possible
  - Computation for free, best-effort execution
  - Desktop grids, clusters, …



The promise of the cloud



- Elastic computing, get exactly the machines you need, exactly when you need them...
- Well, did we mention you have to pay for the hour?

“Quality of Service”

- **Small Instance, $0.085 per hour**
  - **1.7 GB of memory, 1 EC2 Compute Unit (ECU)**
- **High-memory extra large, $0.50 per hour**
  - **17.1 GB memory, 6.5 ECU**
- **High CPU medium, $0.17 per hour**
  - **1.7 GB of memory, 5 EC2 Compute Units**

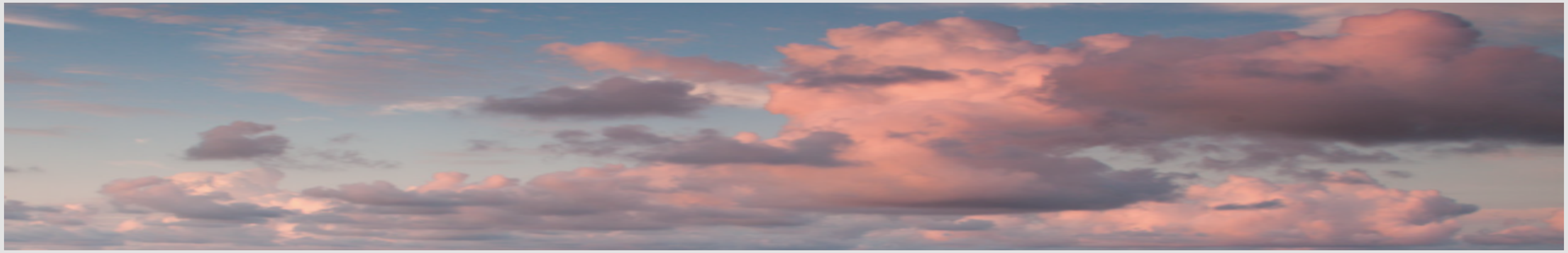
Which one is faster for *my* application???

Which one is cost efficient???

Bag Characteristics

- Many independent tasks
  - All tasks are always ready to run
- Runtimes are unknown to the user
- Tasks have some (unknown) runtime distribution
- Simplifications:
  - Tasks can be aborted/restarted
  - No costs of input/output files (ongoing work)
  - No disruptive performance changes across clouds (e.g., with cache sizes that delay some tasks but not the others)

Cloud Characteristics



- A cloud offering provides machines of certain properties like CPU speed and memory
  - All machines in a cloud offering are homogeneous
  - There is an upper limit of machines per cloud that a user can get
- A machine is charged per Accountable Time Unit (ATU); 1 hour, for example
- We call a cloud offering (machine type, price, max. number) a *cluster*
  - We are HPC guys, after all...

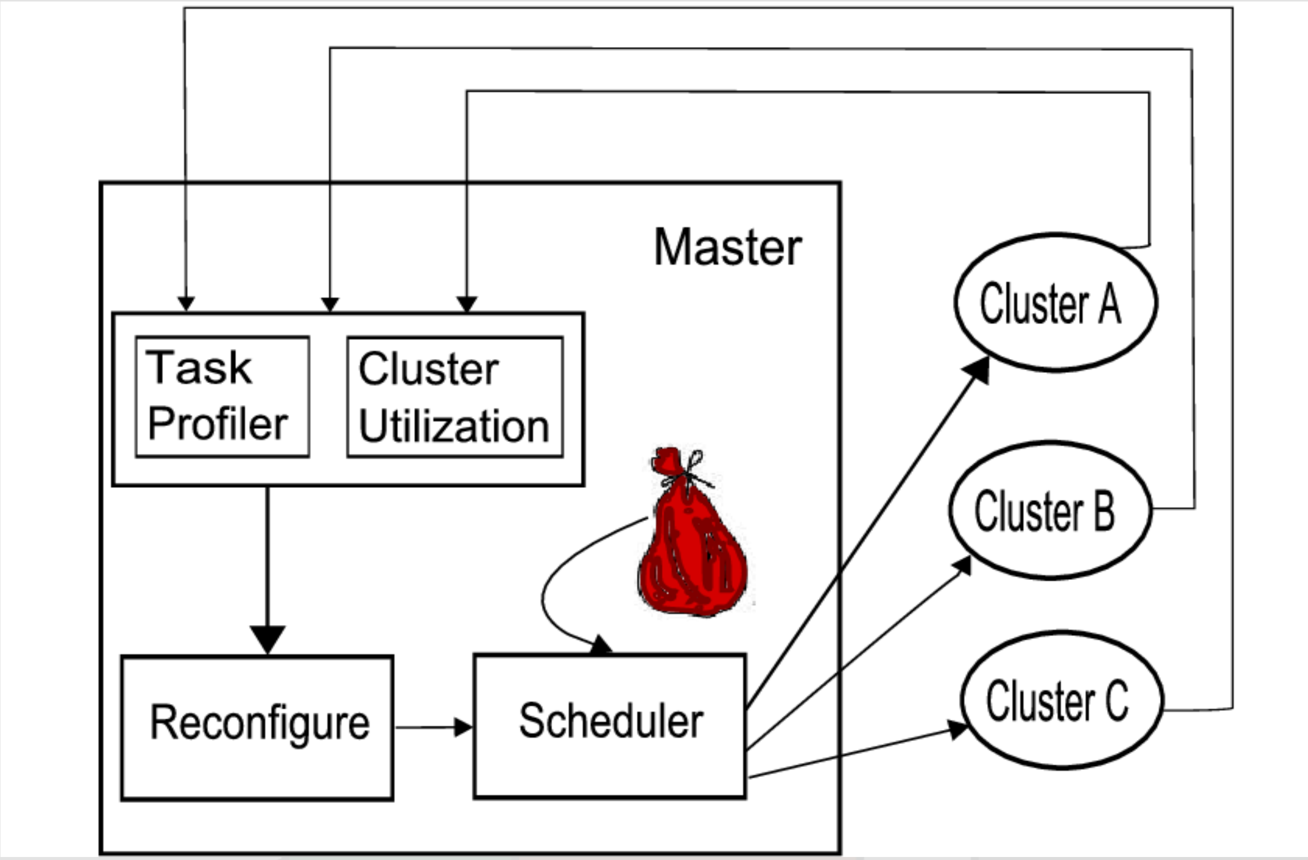
What's the (scheduling) problem?

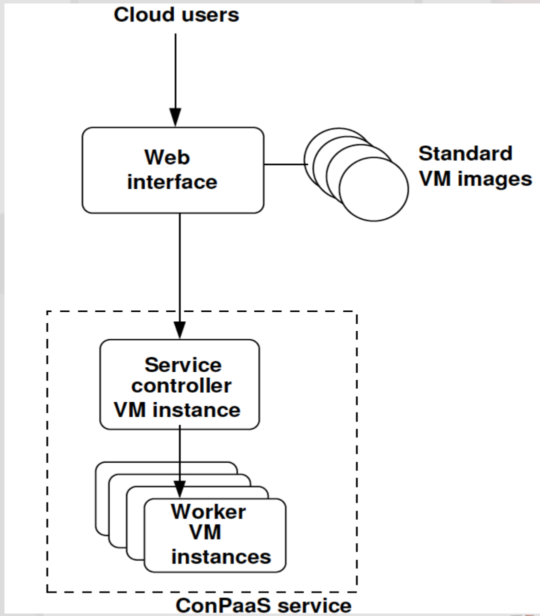
- We are on a budget.
- We know nothing.

- We want to:
  - Run all tasks from our bag on (cloud) clusters, without spending more than our budget
  - Allocate/release machines dynamically while learning how fast our tasks execute on the different clusters
  - If we learn that our budget is too low, give up
  - Minimize makespan of the whole bag, if we can make it within budget

BaTS: Budget-aware task scheduler

- Self scheduling tasks
- Reconfiguring cluster configurations



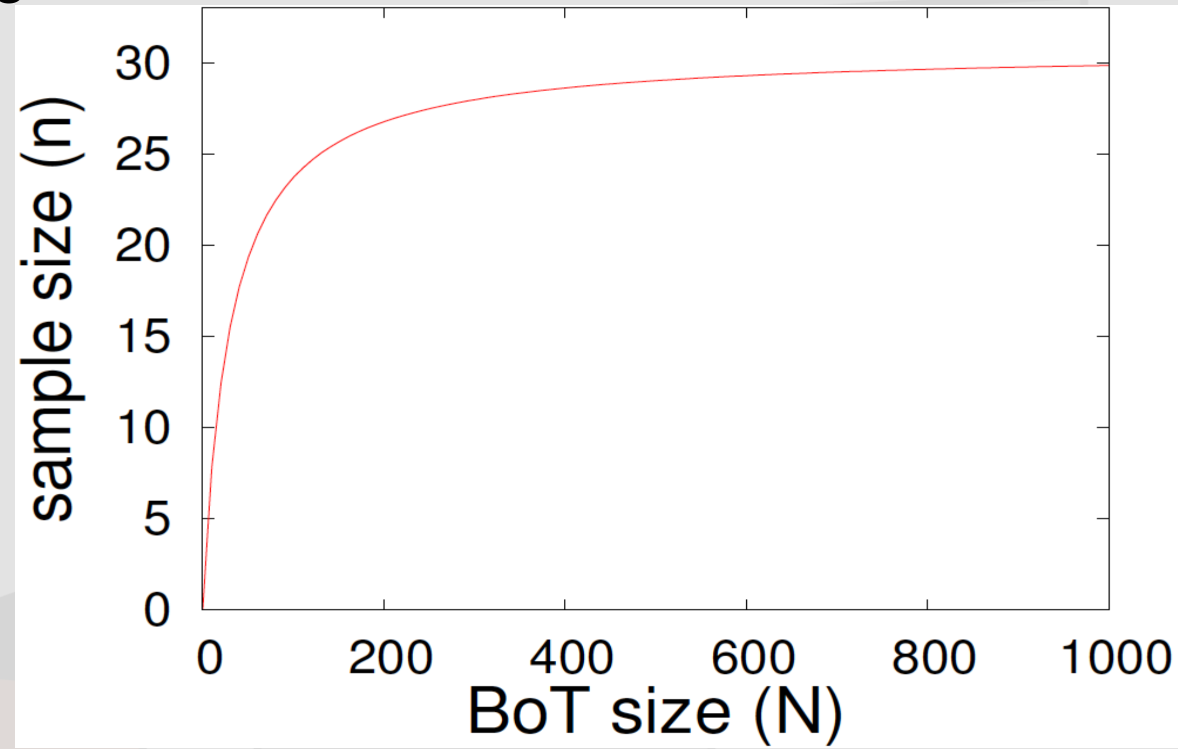


The BaTS Story

- "Every good story has a beginning, a middle part, and an end."
- With BaTS:
  - Runtime and budget estimation
  - Throughput phase
  - Tail phase

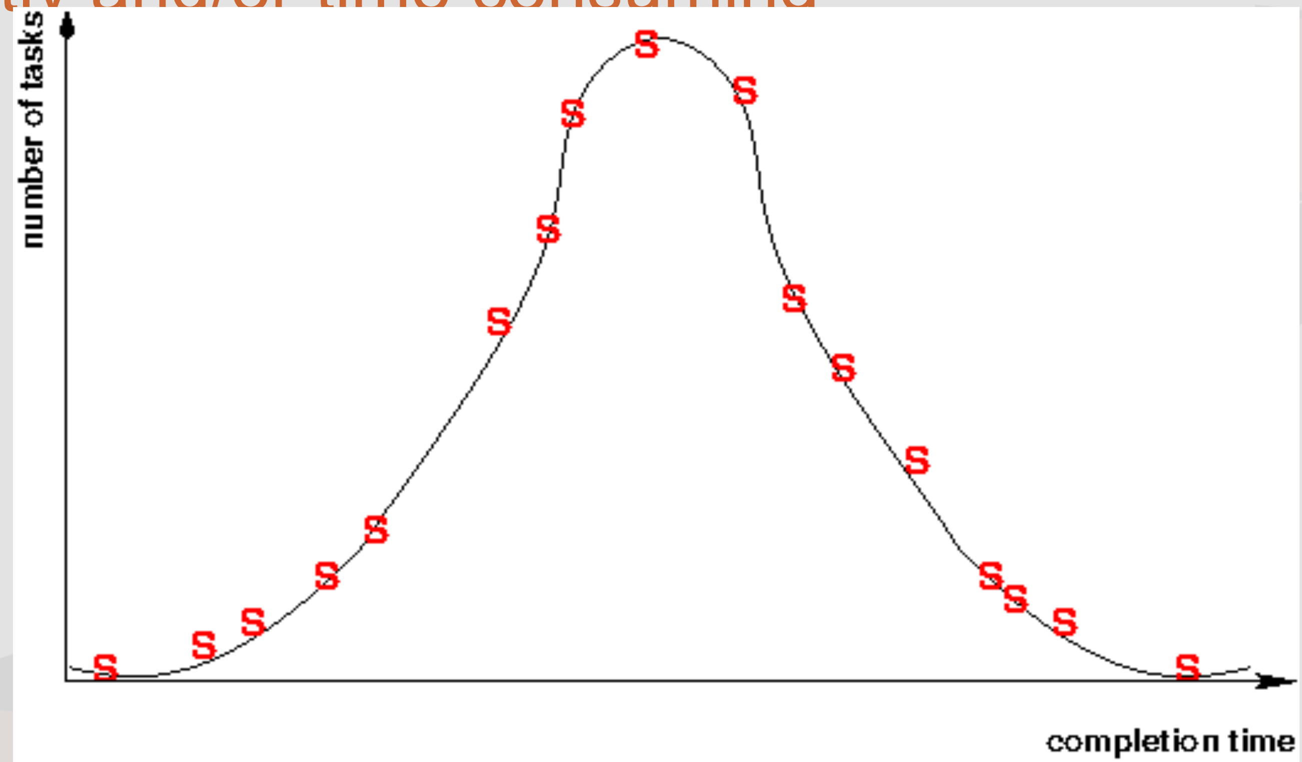
Runtime Estimation

- Statistics for sampling with replacement:
  - Bag of tasks can be described with pretty good accuracy from a small sample
  - We collect average and variance



Runtime Estimation

- For each cluster (cloud machine type) we need a sample of +/- 30 completed tasks
  - (drawn at random)
- This might be costly and/or time consuming



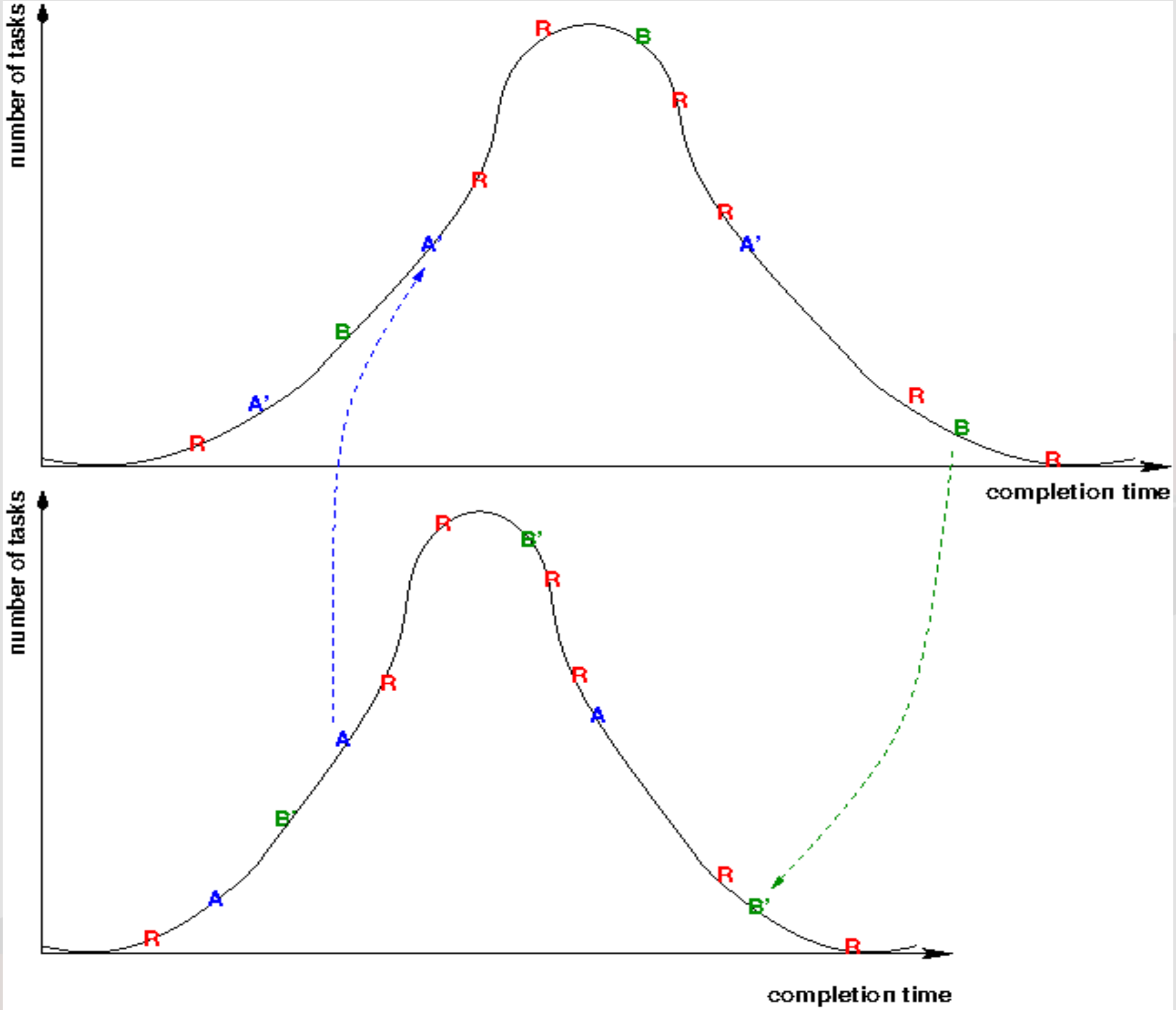
Compact Sampling

Assume:
g(x) = a * f(x)+b

Linear Regression:
Replicate 7 tasks

Distribute rest of sample (30-7=23) over all clusters

Map samples to other clusters



Cluster Configuration

- From the average speed of each cluster, (in tasks per minute) we can compute estimates for makespan (Te) and cost (Be) for a configuration from nodes of multiple clusters:

$$T_e = \frac{N}{\sum_{i=1}^{C_{nc}} \frac{a_i}{T_i}} \quad ; \quad B_e = \left\lceil \frac{T_e}{ATU} \right\rceil * \sum_{i=1}^{C_{nc}} a_i * c_i$$

- We minimize Te while keeping Be <= B using a modified Bounded Knapsack Problem (BKP)
  - The BKP can be solved in pseudo-polynomial time, as a 0-1 knapsack problem via linear programming
- BaTS chooses the configuration with minimal Te for Be <= B

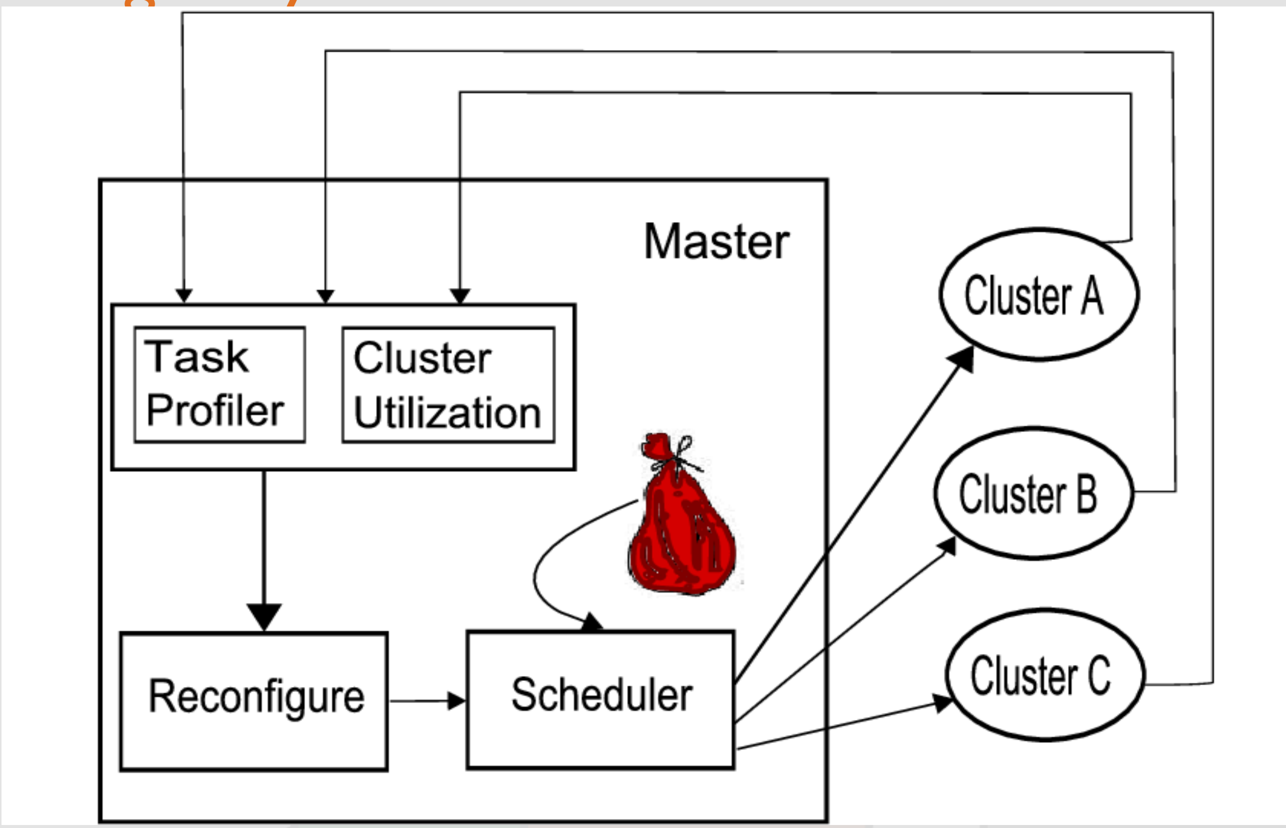
Budget Estimation

- User must make the trade-off between cost and completion time
- BaTS provides the user with choice ***(cost, time)***, using cluster configurations computed from the sampling phase:
    - Cheapest makespan
    - Cheapest makespan +10/20% cost
    - Fastest makespan -10/20% cost
    - Fastest makespan

    - (more options are possible)
- Each configuration (in fact) consists of the numbers of machines per cluster

BaTS: Throughput Phase

- Self scheduling tasks
- Reconfiguring cluster configurations regularly



Progress Monitoring

- BaTS starts from the user-selected, initial configuration
- At regular intervals (e.g., 5 minutes), BaTS re-evaluates the configuration
    1. Update average and variance per cluster
    2. Re-evaluate the machine configuration
- Execution on real machines adds some complexity:
    - Individually requested from the cloud provider(s), startup time before being ready
    - Each machine has its own end of the next ATU

Re-evaluate the machine configuration

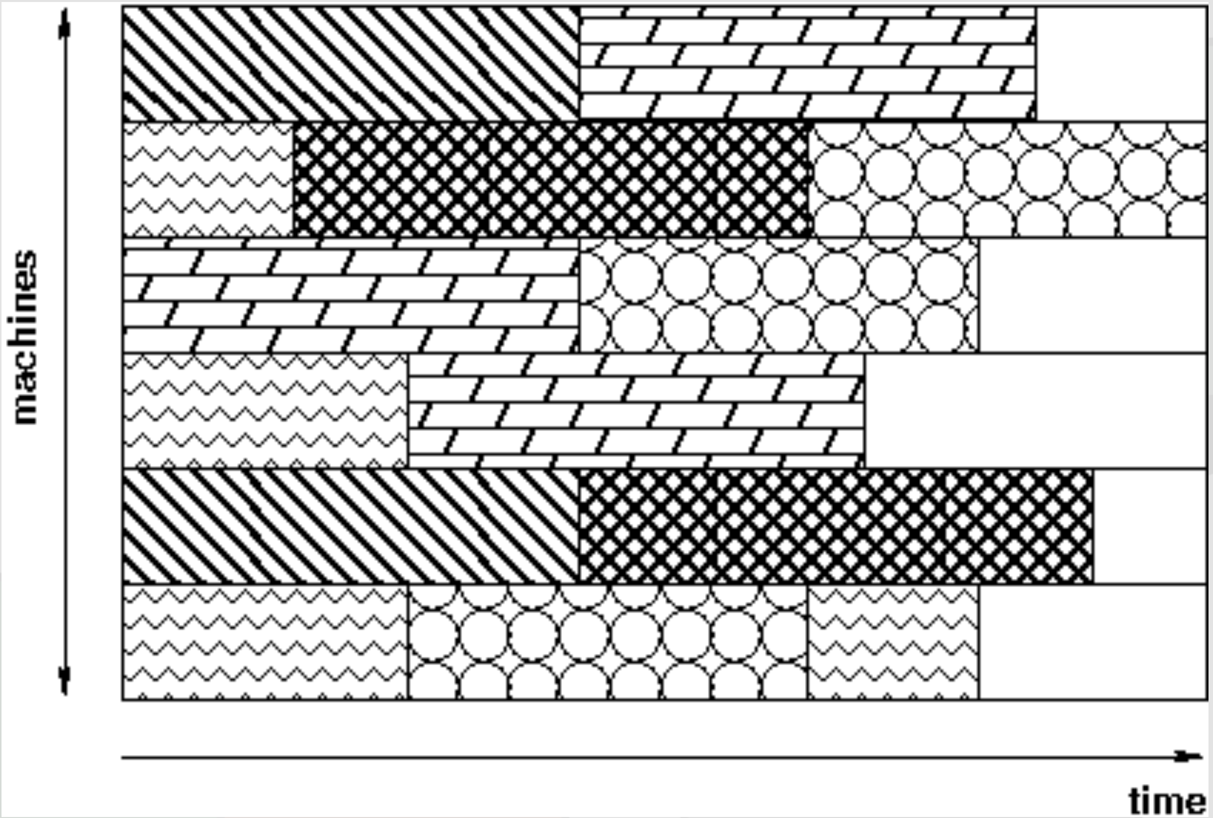
- Solve the remaining problem
  - Less tasks
  - Less money left
  - Track already-paid time left on machines
- If budget violation expected, get more machines with better price/performance ratio, and drop others
- If makespan violation expected, get more fast machines, and drop others
- If both budget and makespan violations expected, call ~~mummy~~ the user

Fluid vs.Discrete Models

- BaTS (the BKP solver) allocates machines per full ATU
- Assumes a "fluid" model of computing time

Fluid vs.Discrete Models

- Tasks, however, are sequential, cannot be split across "leftover" cycles
- Tasks on machines in final ATU:



The End is Near!

- The tail phase needs some special consideration
- Bags with high variance may overrun predicted makespan (and thus budget)
- Even without overrunning, towards the end machines remain idle

BaTS' Tail Phase

- As soon as a machine can not be assigned a task, BaTS switches to the *tail phase:*
  - Replicate running tasks onto idle machines
- Which task to replicate?
  - The one that will terminate last!
- OK, how do we know?
  - Estimate completion time based actual runtime:
    - "Task *i* is running for 12 minutes now, what is its expected completion time, given the observed average and variance of the bag?"
  - Estimate completion time onto the idle machine (starting from scratch)
  - If shorter, replicate

BaTS' Tail Phase (2/2)

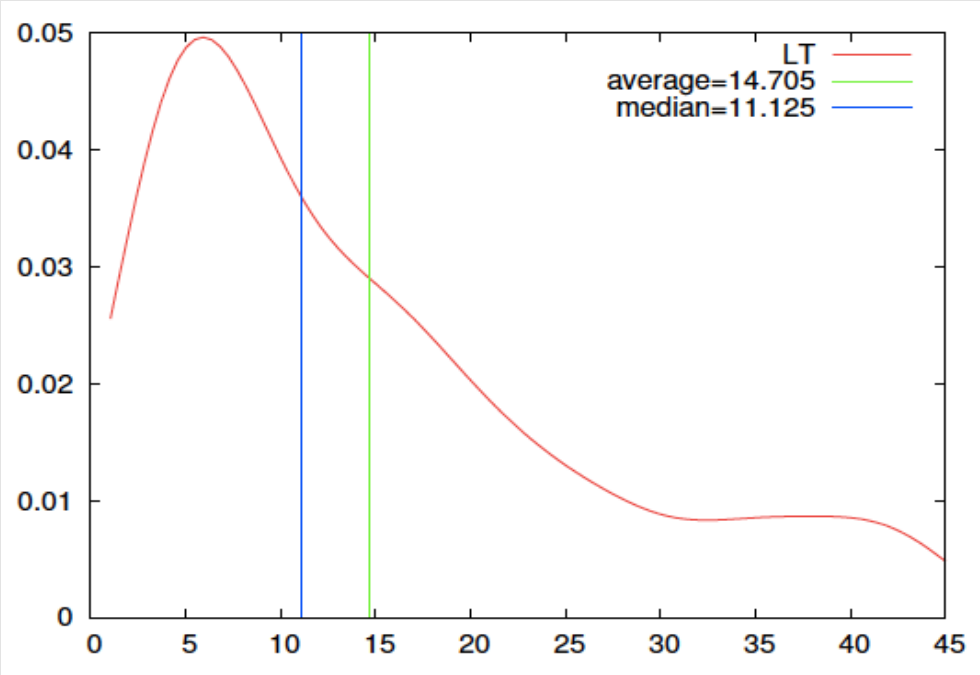
- Do we need to start earlier?
- In the throughput phase, the average runtime determines the speed.
  - According to the *central limit theorem*, this no longer holds, once the population is smaller than a threshold (the same as the sample size in the beginning, +/- 30)
- With the threshold reached, BaTS migrates tasks to faster machines.
  - Same as replication, but original task is killed.
  - This frees a slow machine for a hopefully shorter task.

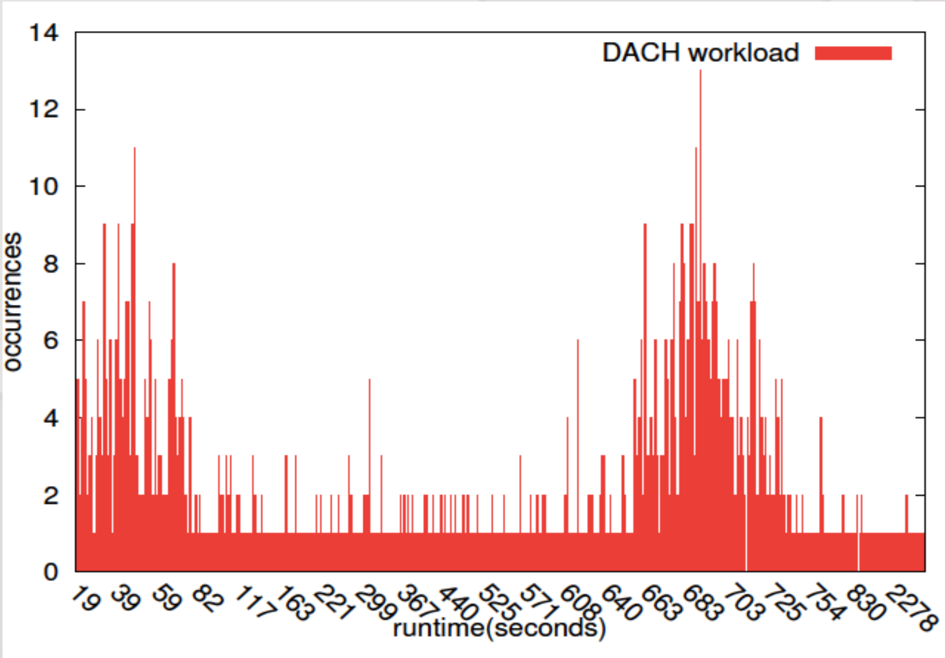
BaTS' Tail Phase Evaluation

- We compare the following options:
  - No tail phase optimization.
  - Stochastic replication (based on completion time prediction)
  - Replication with perfect knowledge (theoretical optimum)
  - Replication with random task selection (no knowledge)
  - Replication plus migration

Types of Bags Used

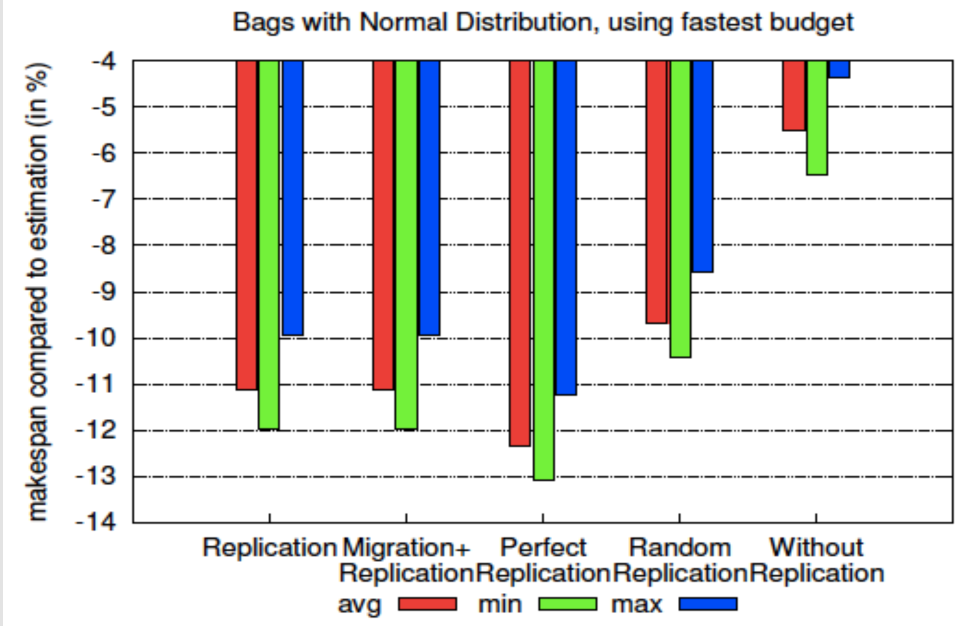
- Normal distribution
- Truncated Levy distribution (heavy tailed)
- Multi-modal distribution (real world data)

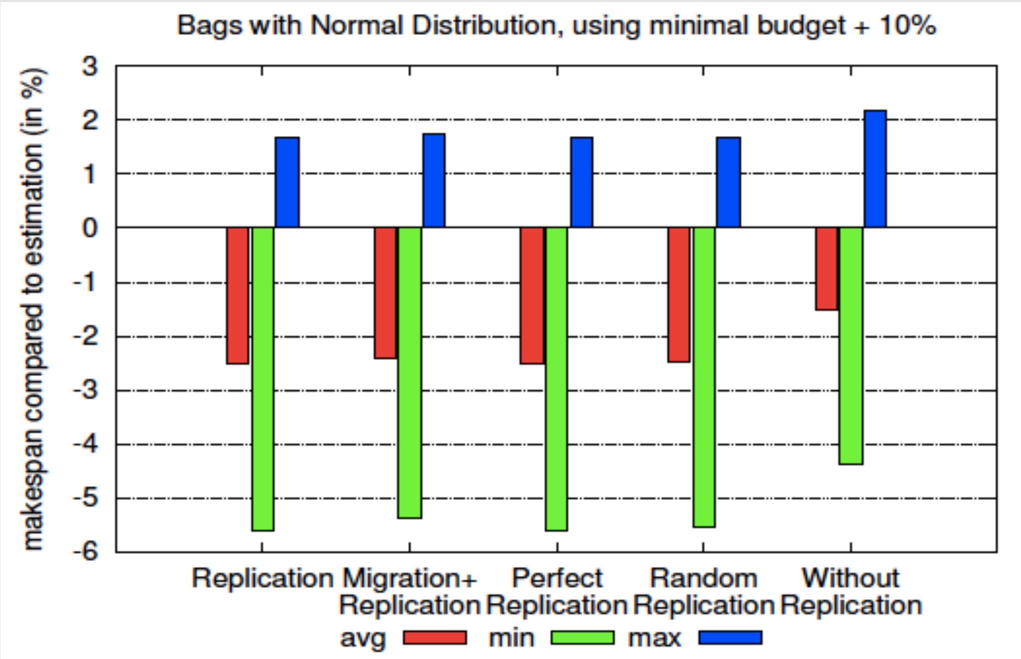




Normal Distribution

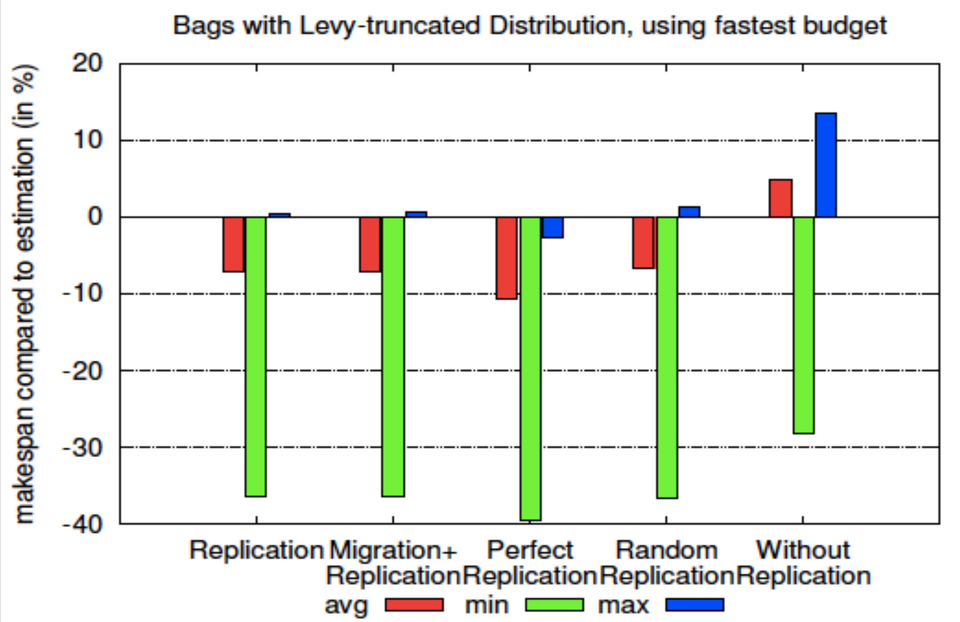
- Simulator runs
- 30 bags each
- 30 runs each

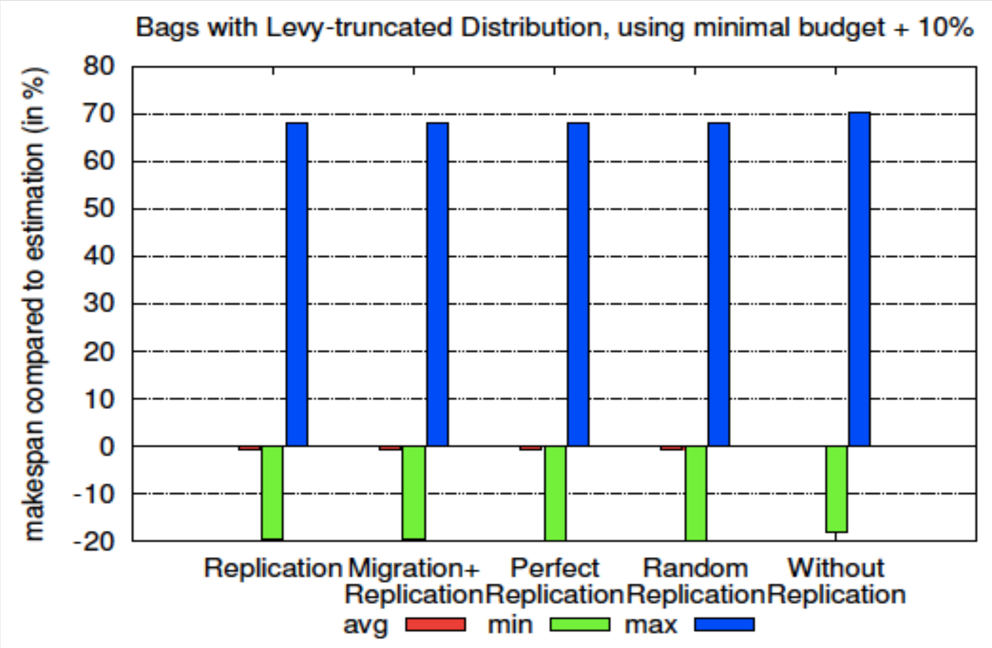




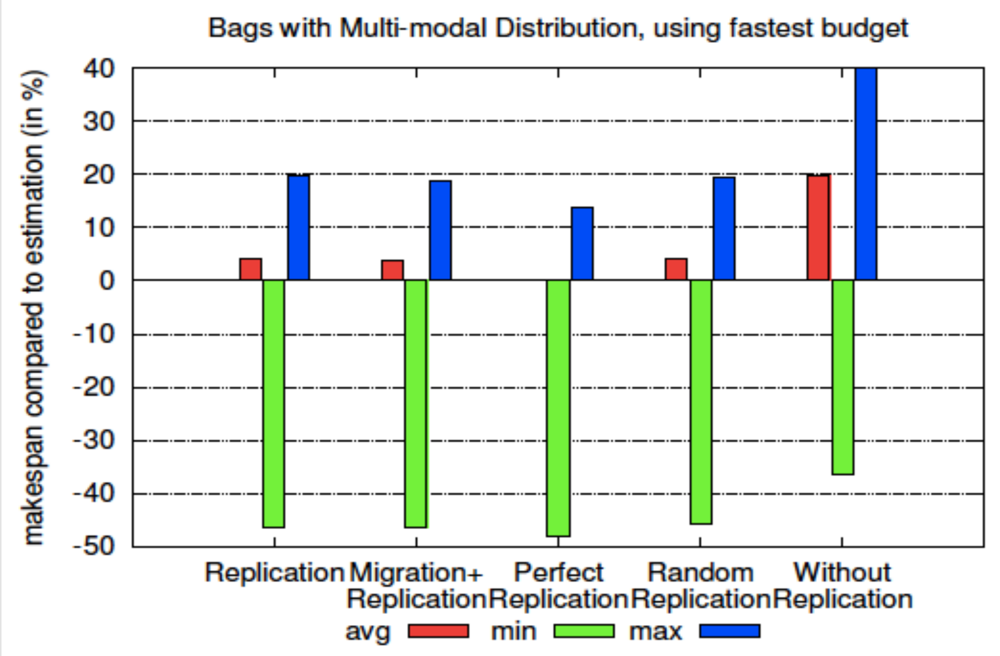
"low is good"

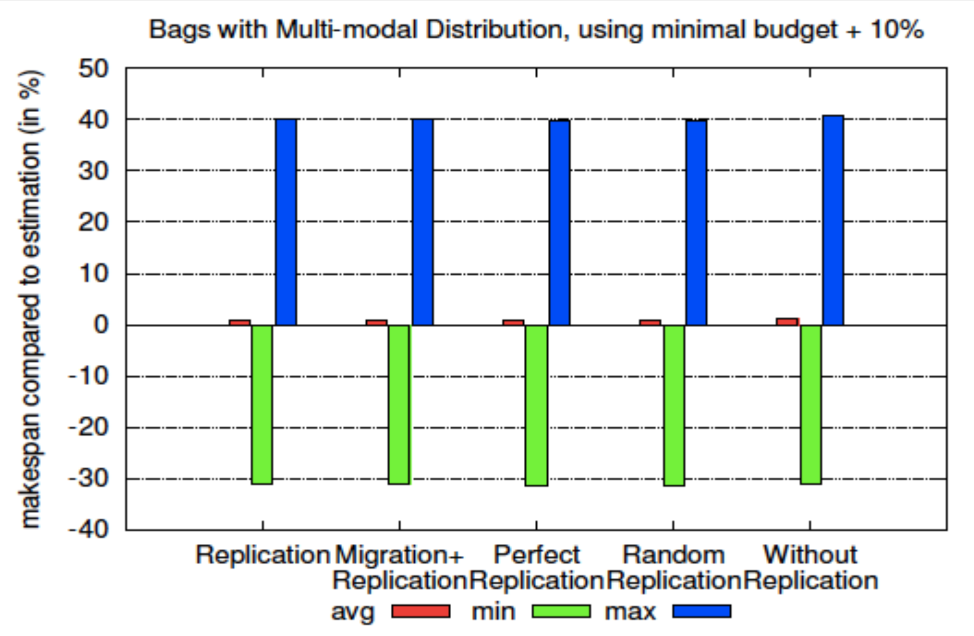
Heavy-tailed Distribution





Multi-modal Distribution





Tail Phase Findings

- Doing “nothing” is the only bad option
- Replication works fine
  - Even with random selection
    - But has higher error rate
- Additional migration seems not to be worth the effort
  - The price we pay (kill running task) seems to outweigh the benefits

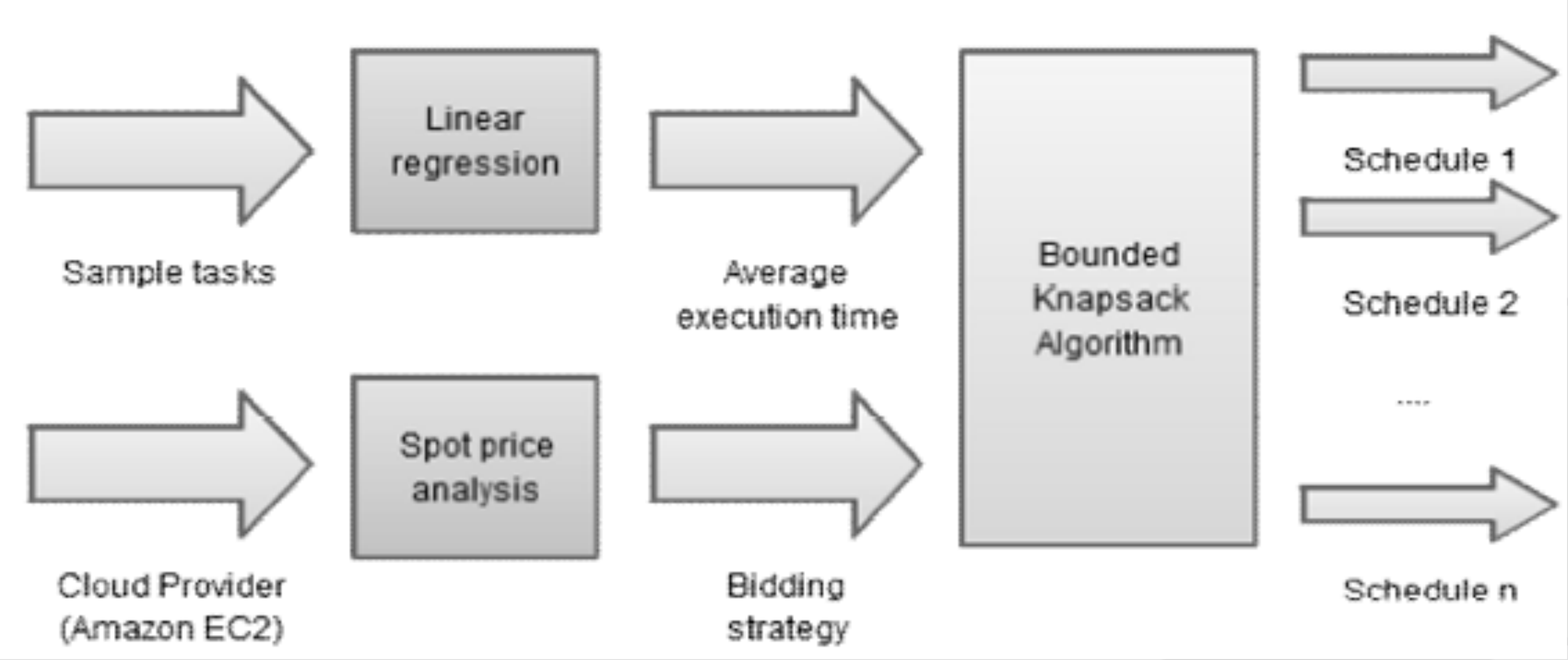
BaTS on the Amazon Spot Market

- So far, we used “on demand” instances
  - Fixed price per hour
- Amazon spot market:
  - Same (on demand) machine types at different prices
    - Users “bid” a price for a machine (of a type)
    - If the bid is >= the current spot price, user gets the machine
    - If the spot prices exceeds the bid, the user is kicked out without prior notice
      - (and is reimbursed for the aborted hour)

Spot Market: pros and cons

- Pro:
  - We might get machines cheaper
  - In practice, spot prices hardly ever change (boring)
- Con:
  - Tasks might get aborted
    - (we also do this ourselves, no problem)
  - Total budget fluctuates
  - Getting a spot instance takes +/- 8 minutes (before the booting starts)

BaTS Sampling for the Spot Market



New research problem:
What is a good bidding strategy for spot machines?

Bidding Strategies

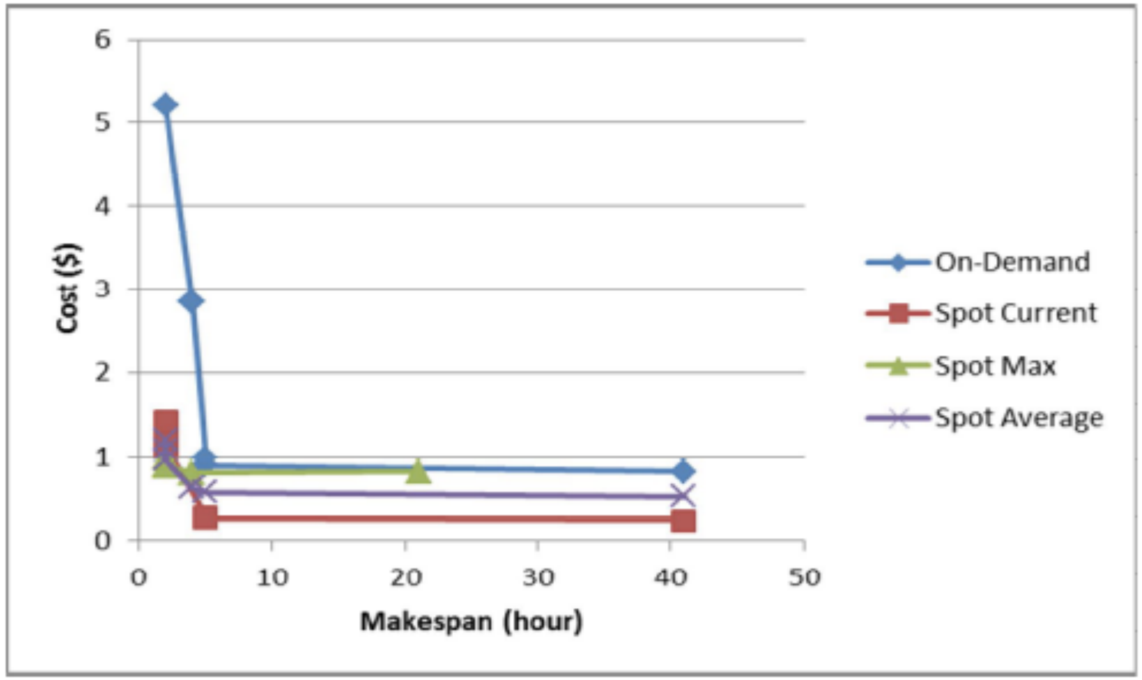
- Maximum price
  - Determine the max price at which a spot instance is more cost efficient than the most profitable on-demand instance:

$$Max_i = \frac{T_p}{T_i} * c_p - \varepsilon$$

- Current price
  - Always get spot instances, the cheapest option at the moment of execution
- Average price
  - Literally the average between “current” and “maximum”, in between the two extremes

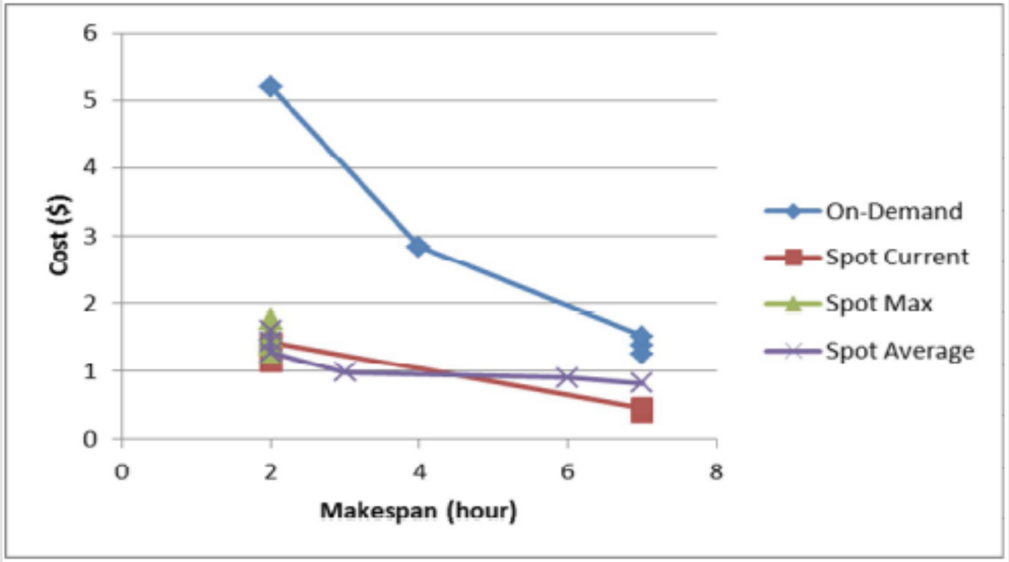
Spot Market Estimations

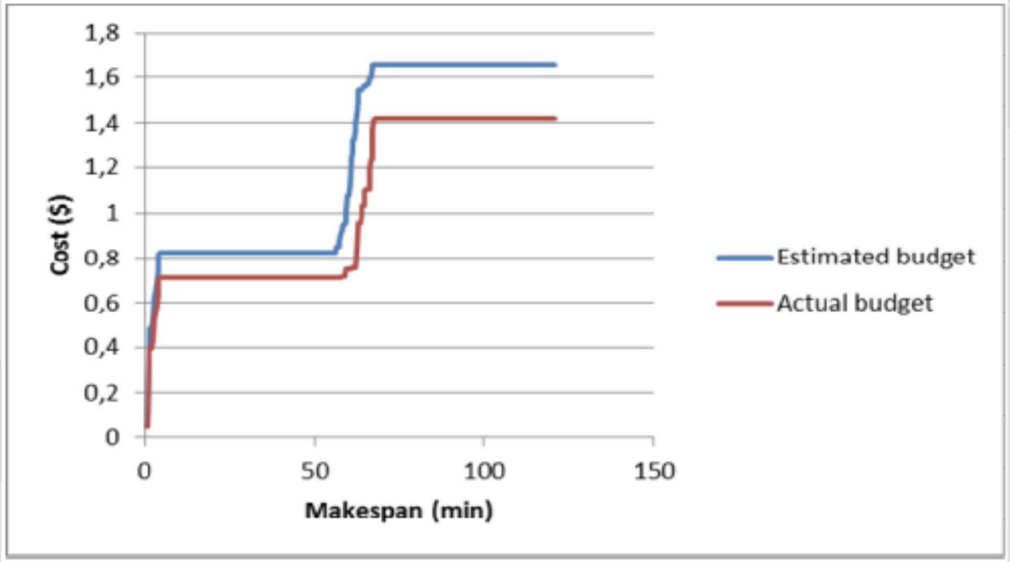
- Using max. 10 instances each of t1.micro, m1.small, m1.medium
- Bag with 18000 tasks (average 32, 15, and 8 seconds)
- Max. bid used: $0.02 for t1.micro, $0.007 for m1.small and $0.015 for m1.medium



No clear "winner" strategy. The user simply gets more options…

Spot Market Runs





Spot Market Findings

- It is too early for final conclusions.
- Opens more choices for the cost-savvy user.
- The current implementation only uses the current spot prices (no history)
- Taking long-term spot prices into account, user might opt for a hard cost limit:
  - Place a low bid and wait until the price drops
  - Interrupt the whole computation if price goes up during the computation

Conlusions

- **BaTS gives the user control over and choice from several cloud offers**
    - **Run cheaper and longer**
    - **Or run faster with higher budget**
- Learning stochastic properties of tasks works well in the absence of runtime estimates
- Next steps:
    - Fully integrate file I/O
    - Handle fluctuating node performance (ongoing)
    - Support workflows (tasks with dependencies)
    - ~~Fault tolerance~~ Resilience
    - Dig deeper into spot mark[illegible]ns


Questions?

contrail

open computing infrastructures for elastic services

contrail is co-funded by the EC 7th Framework Programme

Funded under: FP7 (Seventh Framework Programme)

Area: Internet of Services, Software & virtualization (ICT-2009.1.2)

Project reference: 257438

Total cost: 11,29 million euro

EU contribution: 8,3 million euro

Execution: From 2010-10-01 till 2013-09-30

Duration: 36 months

Contract type: Collaborative project (generic)