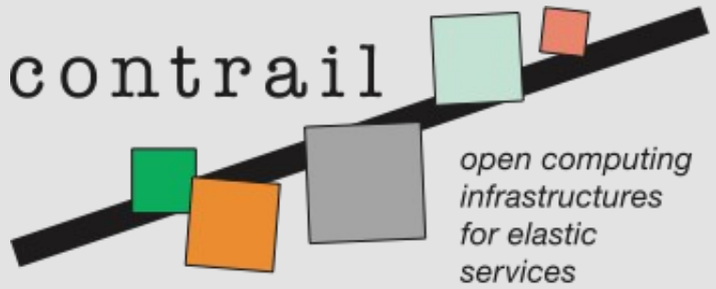
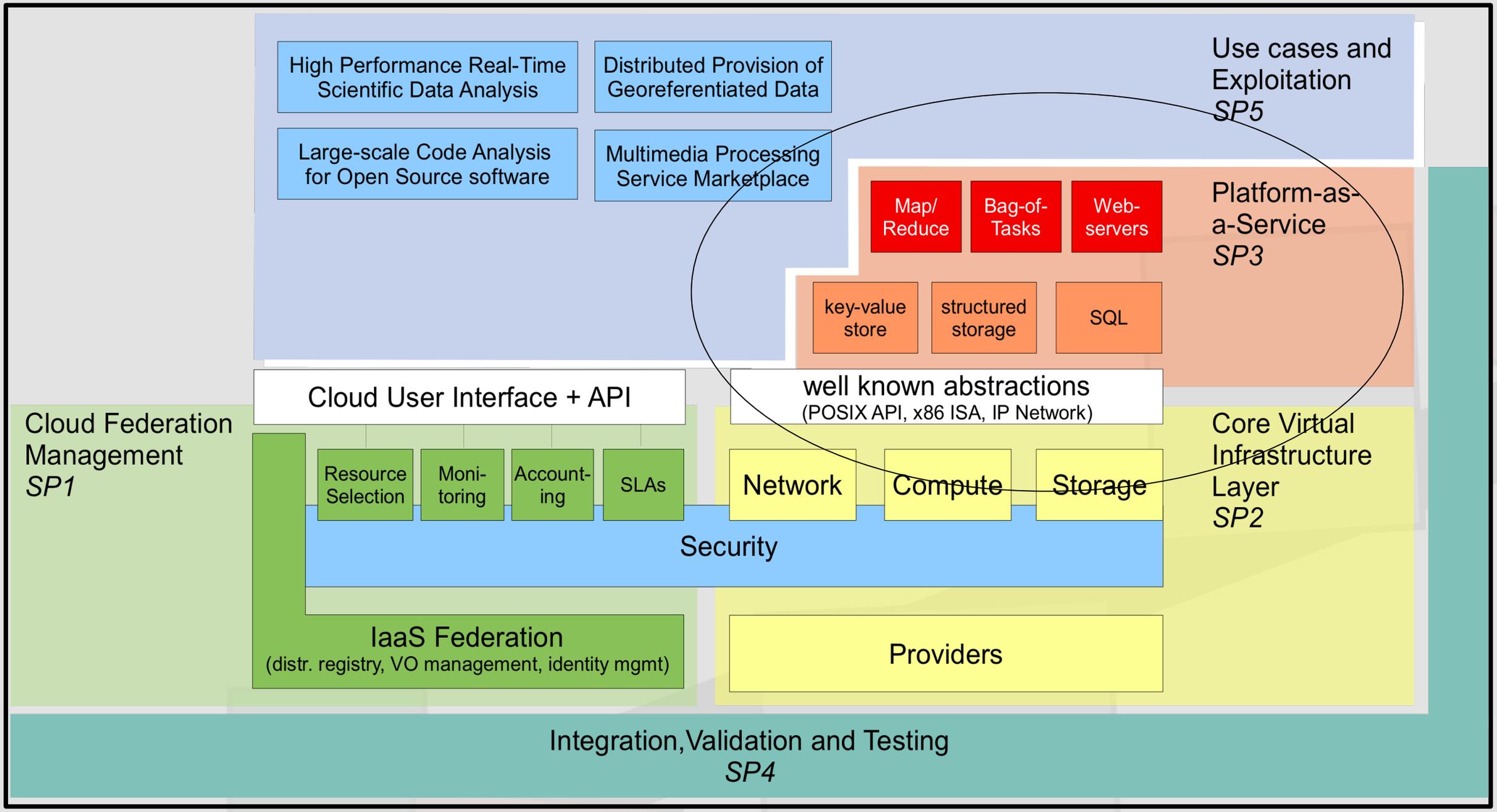


Cost-efficient Task Farming with ConPaaS

Ana Oprescu, *Thilo Kielmann*
Vrije Universiteit, Amsterdam
Haralambie Leahu, Technical University Eindhoven

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 inl

conpaas

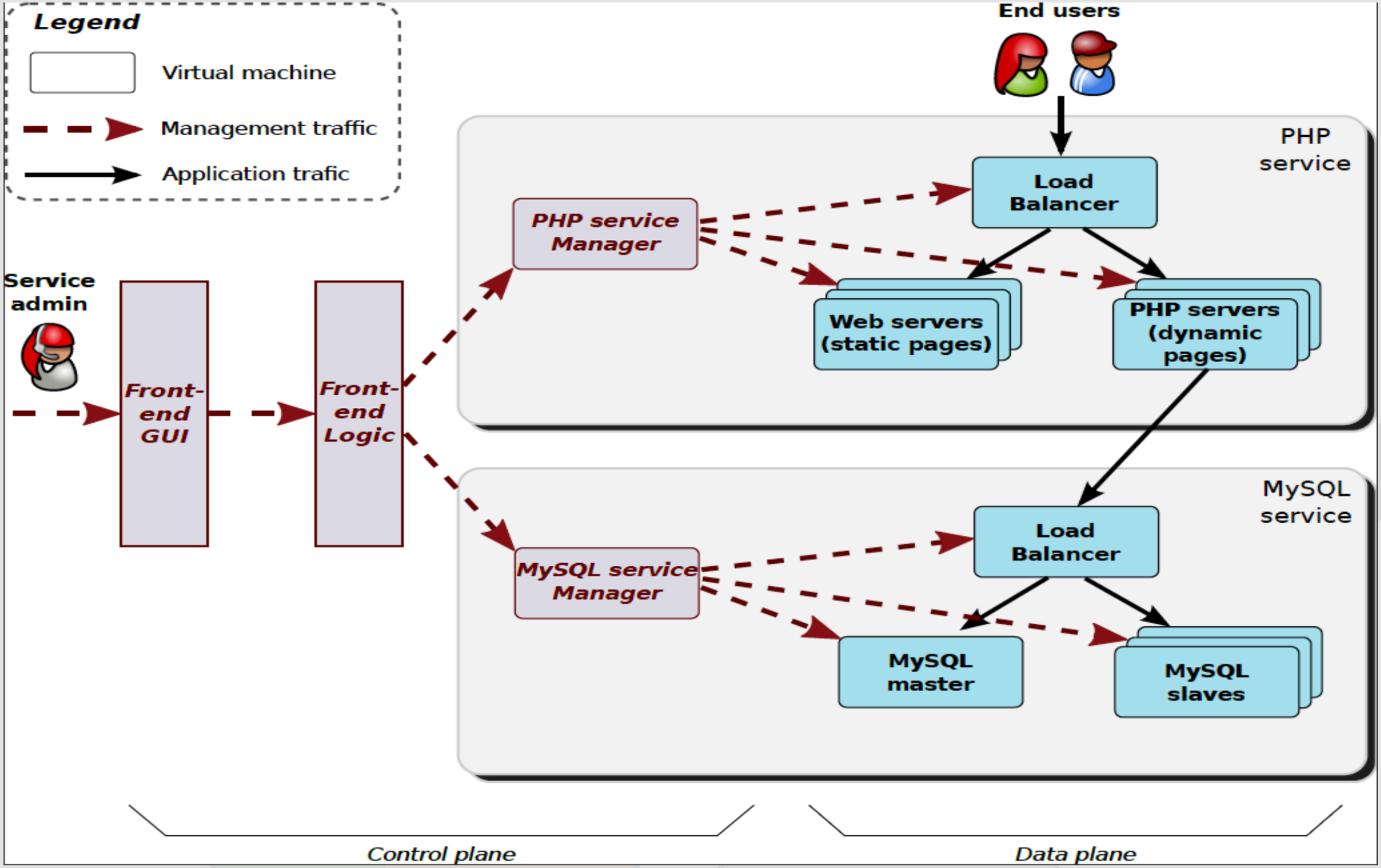
gpierre 9133 logout

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
New Scalarix Service created 12 minutes ago		**257900** KeyValue Pairs	**3** virtual instances

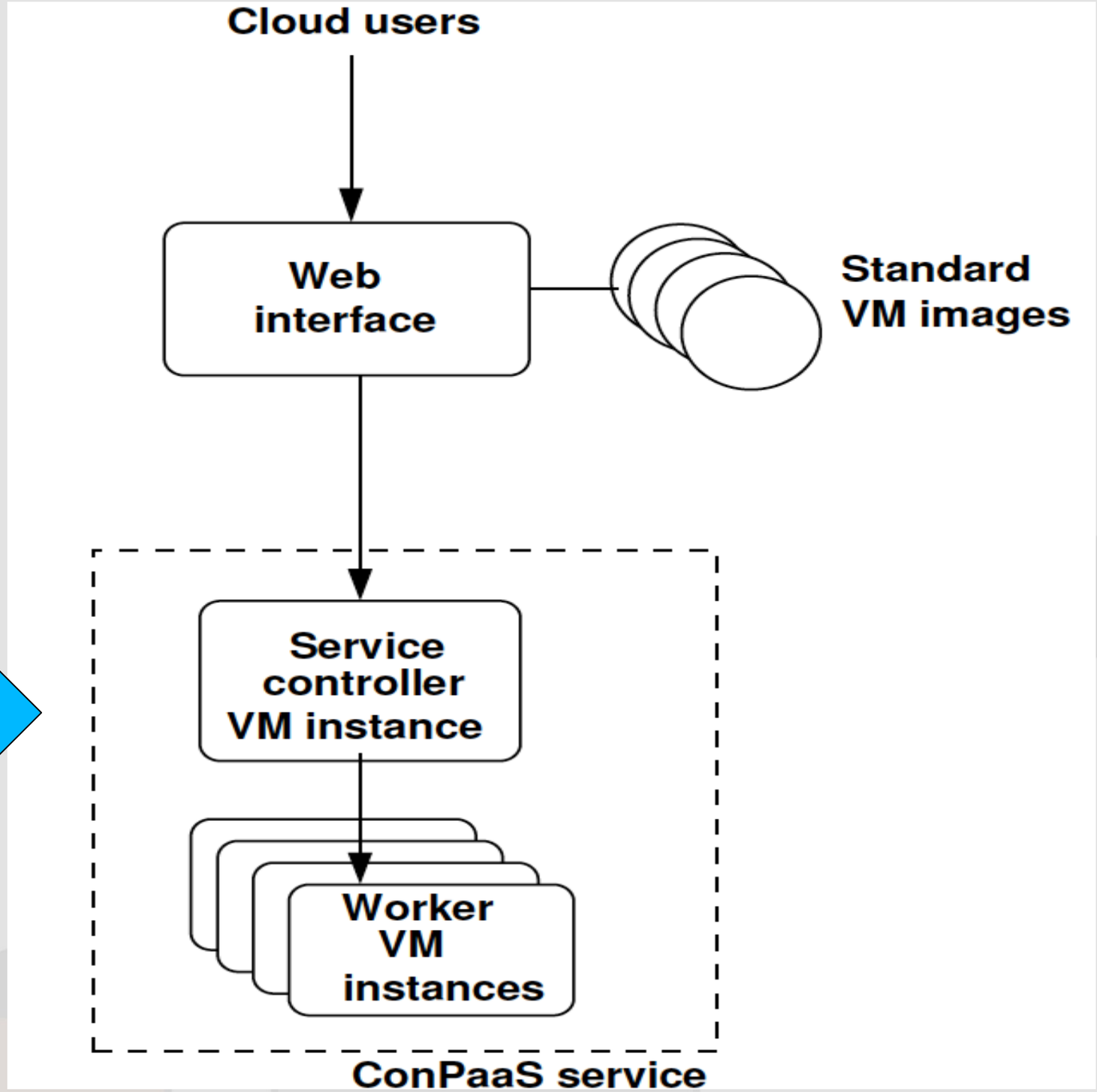
©2011 Contrail - ConPaaS is the PaaS component of Contrail

ConPaaS Web Application



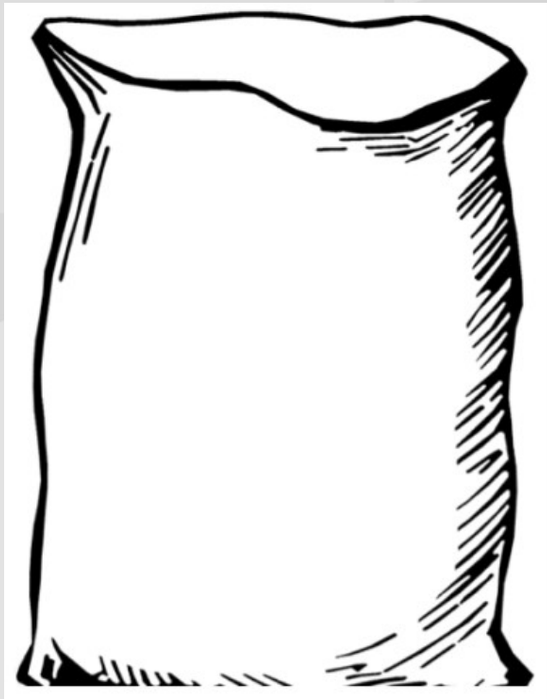
ConPaaS Service Architecture

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, …
- Today: Bags of Tasks; soon: Workflows



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**

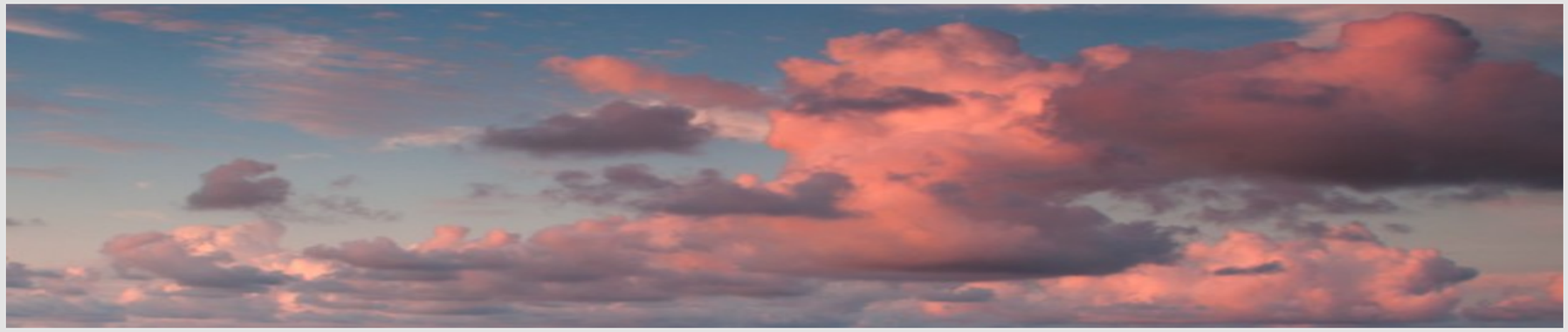
Which one is faster for *my* application???

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...

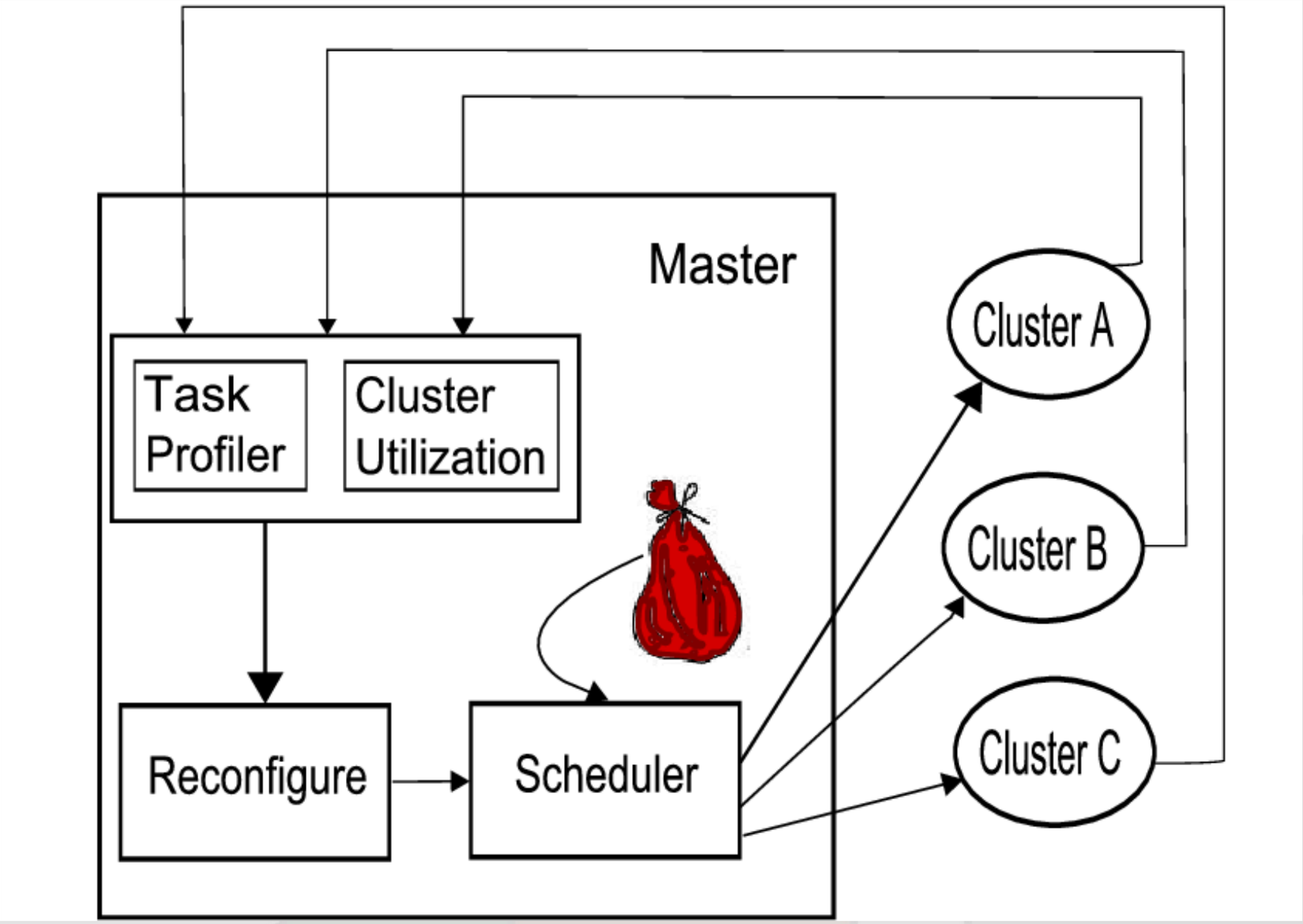
What's the (scheduling) problem?

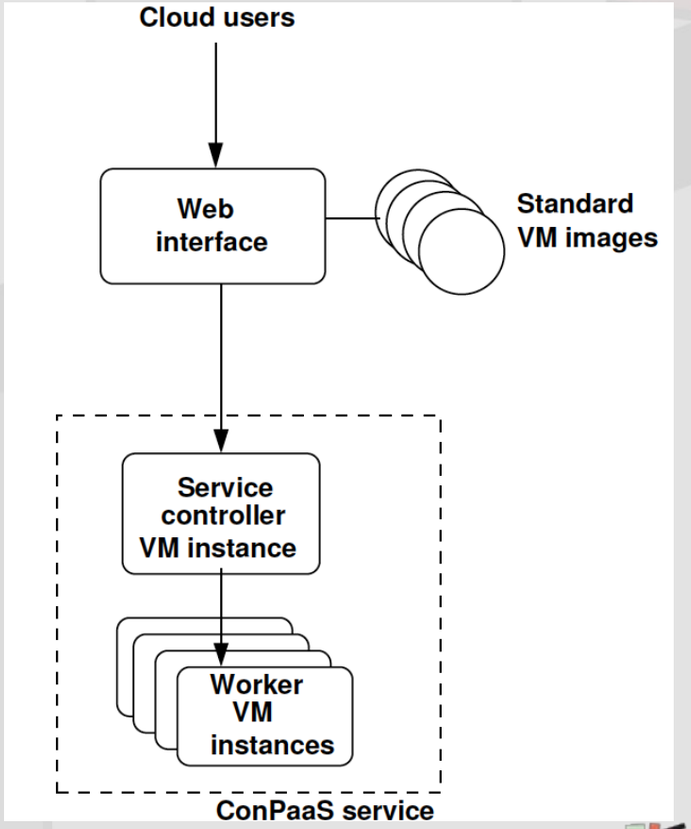
- We are on a budget.
- We know nothing.

- 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



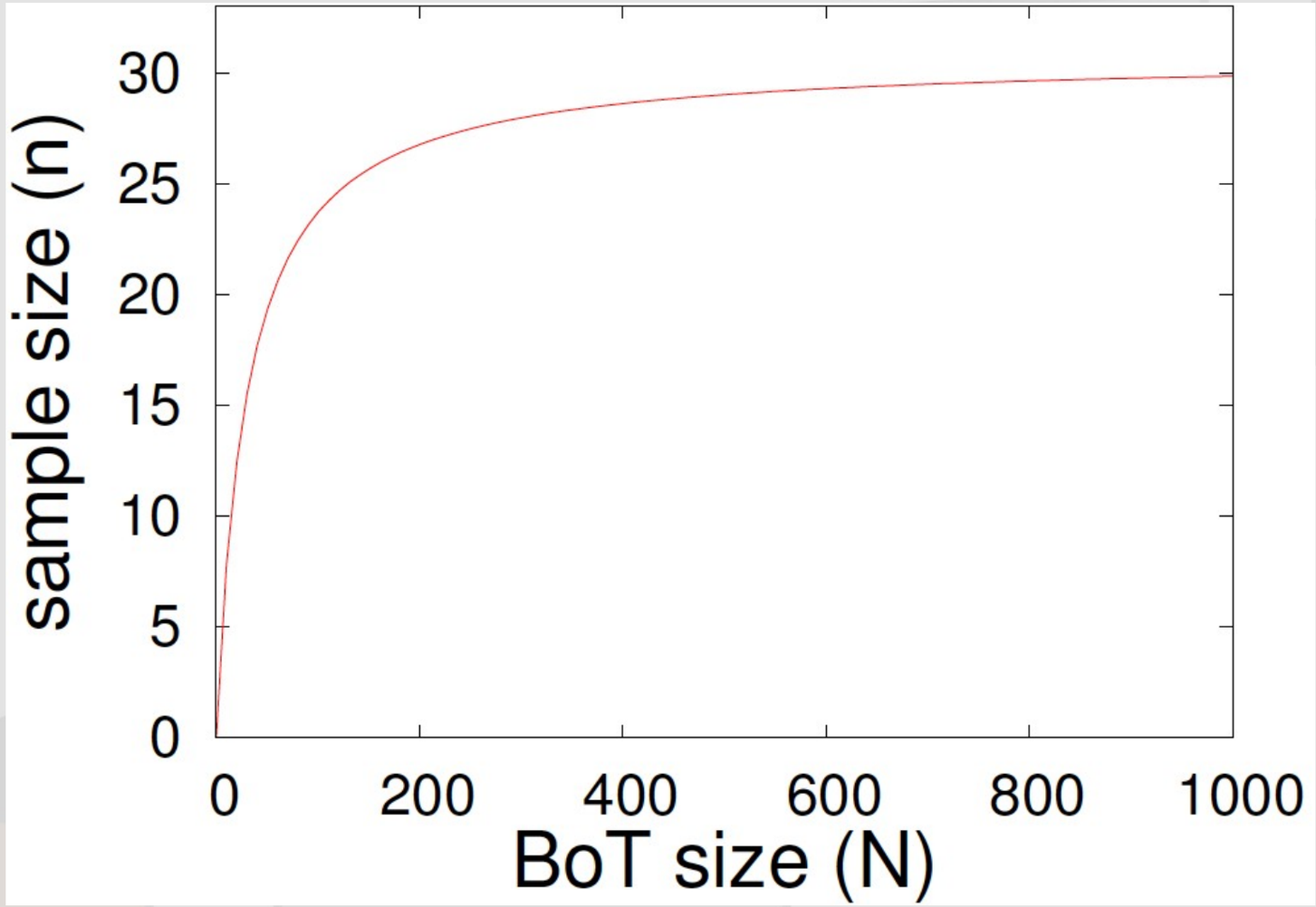


The BaTS Story

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

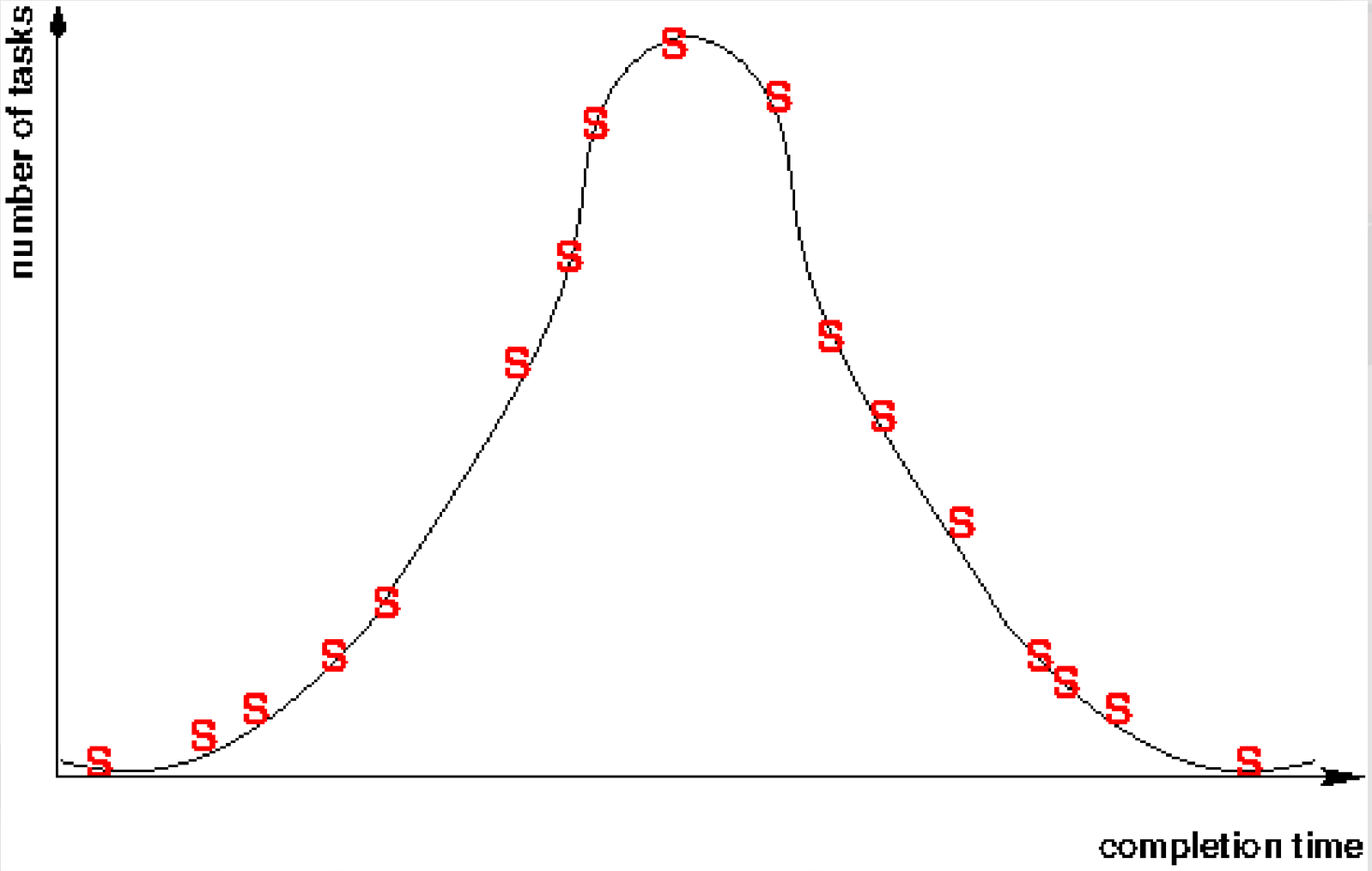
Runtime Estimation

- 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



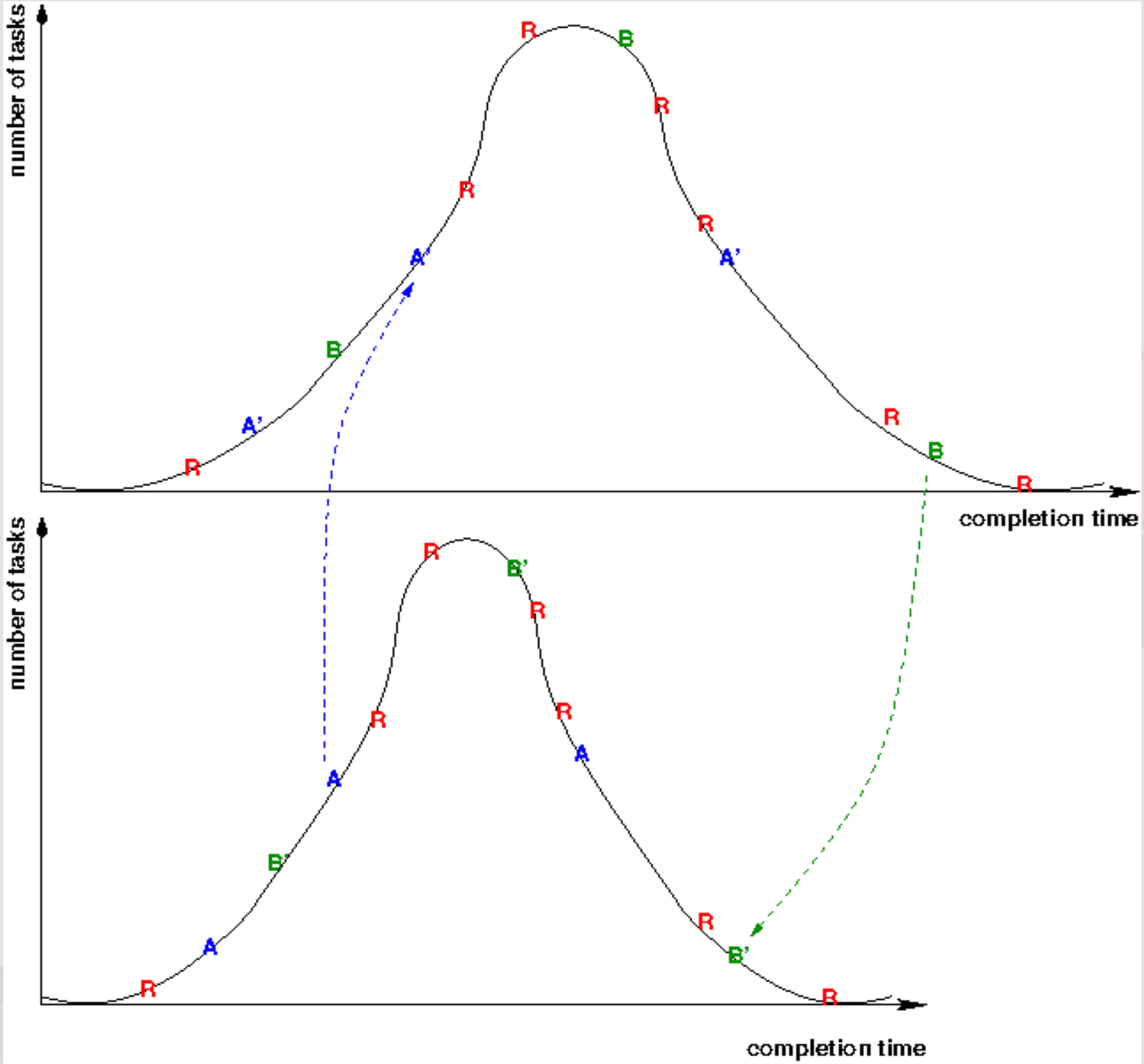
Compact Sampling

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 0-1 knapsack problem via linear programming
- BaTS chooses the configuration with minimal Te for Be <= B

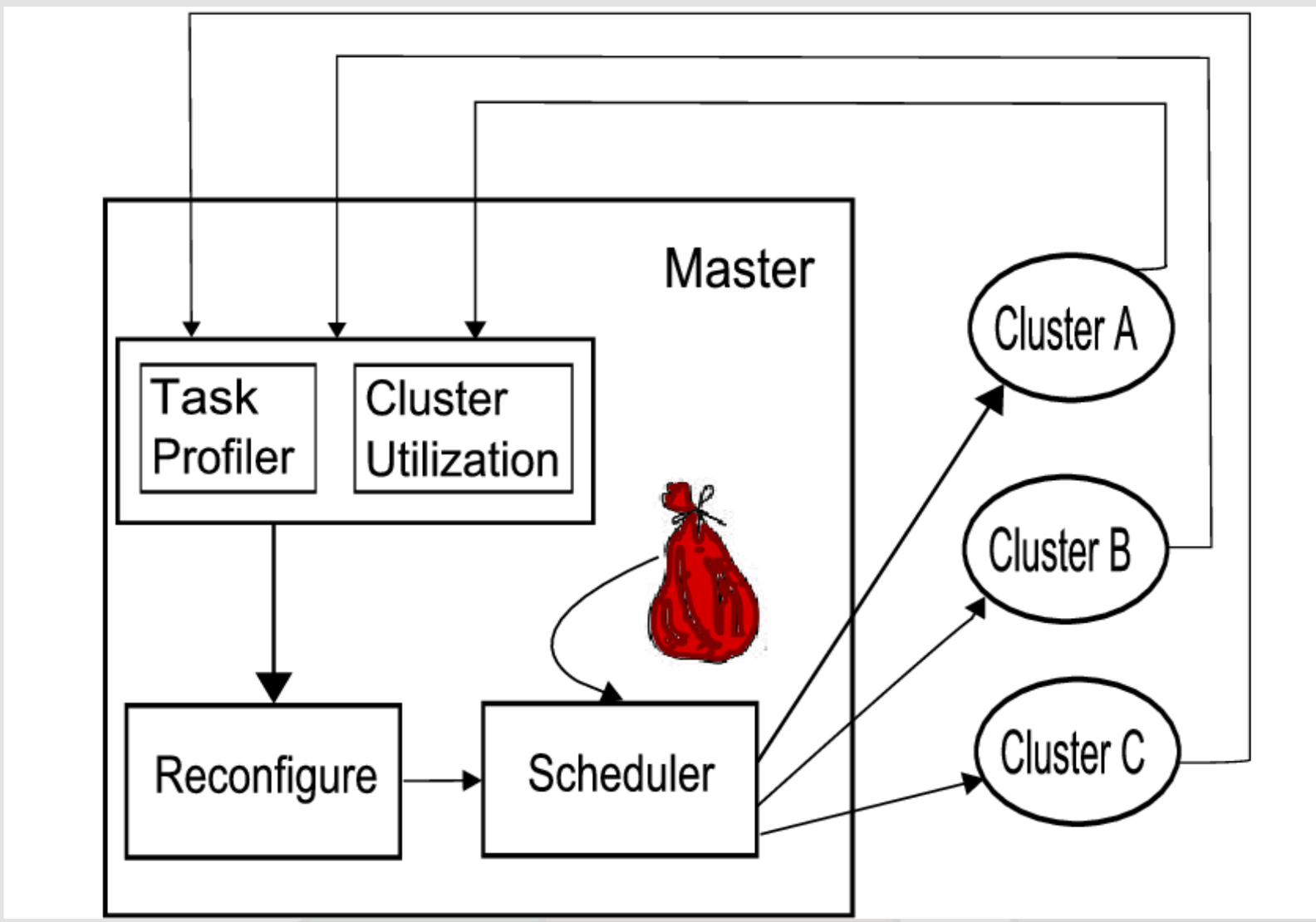
Budget Estimation

- 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 +20% cost
  - Fastest makespan -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

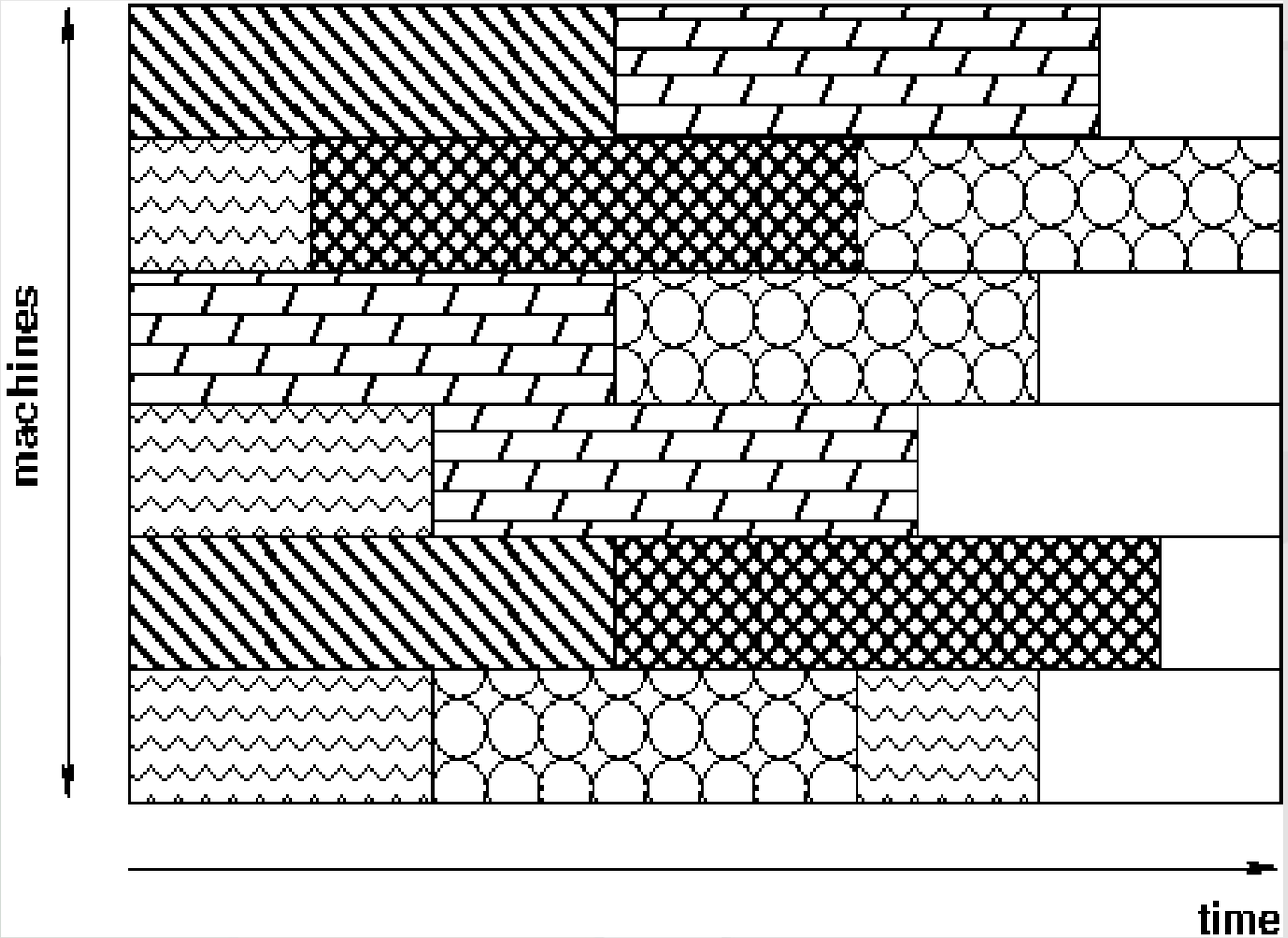
- Solve the remaining problem
  - Less tasks
  - Less money left
  - Track already-paid time left on machines
- If budget violation expected, get more machines with better price/performance ratio, and drop others
- If makespan violation expected, get more fast machines, and drop others
- If both budget 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 *tail phase:*
  - Replicate running tasks onto idle machines
- Which task (of the running ones) 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
- (works well, not shown for lack of time)

Evaluation Platform

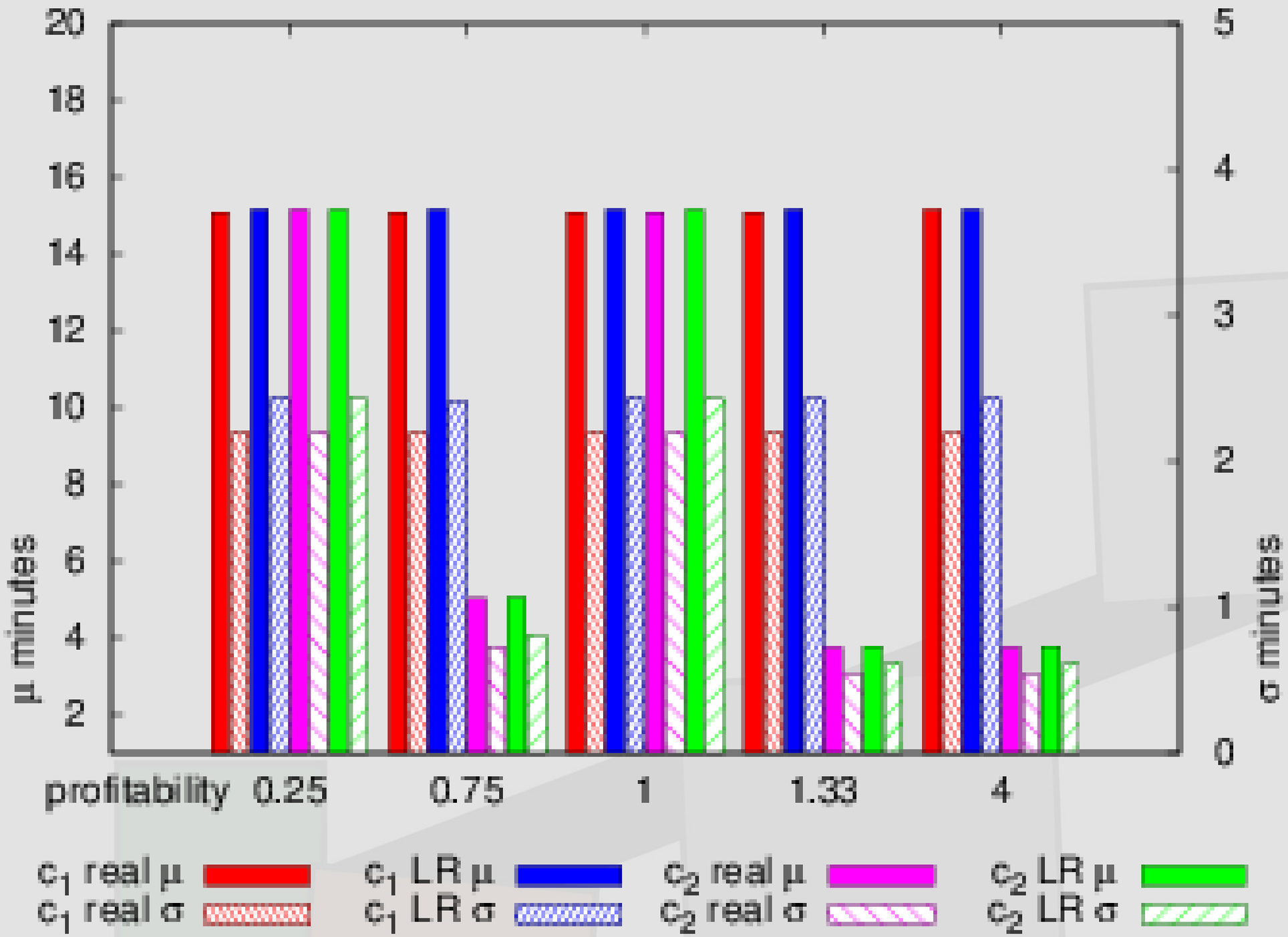
- **DAS-3 multi-cluster system**
- **Emulate 2 clusters (clouds) of 32 machines each**
- **Machine allocation by job submission via SGE**
  - (without competing users)
- **Bag of 1000 tasks with predefined runtimes**
  - Normal distribution mean = 15min, stddev = 2.27 min
  - [Iosup et al., HPDC 2008] show that bags typically have some normal distribution
- **Task "execution" by sleep(runtime)**
- **Fast/slow machines emulated by linearly modifying the sleep time**

Profitability (experiment setup)

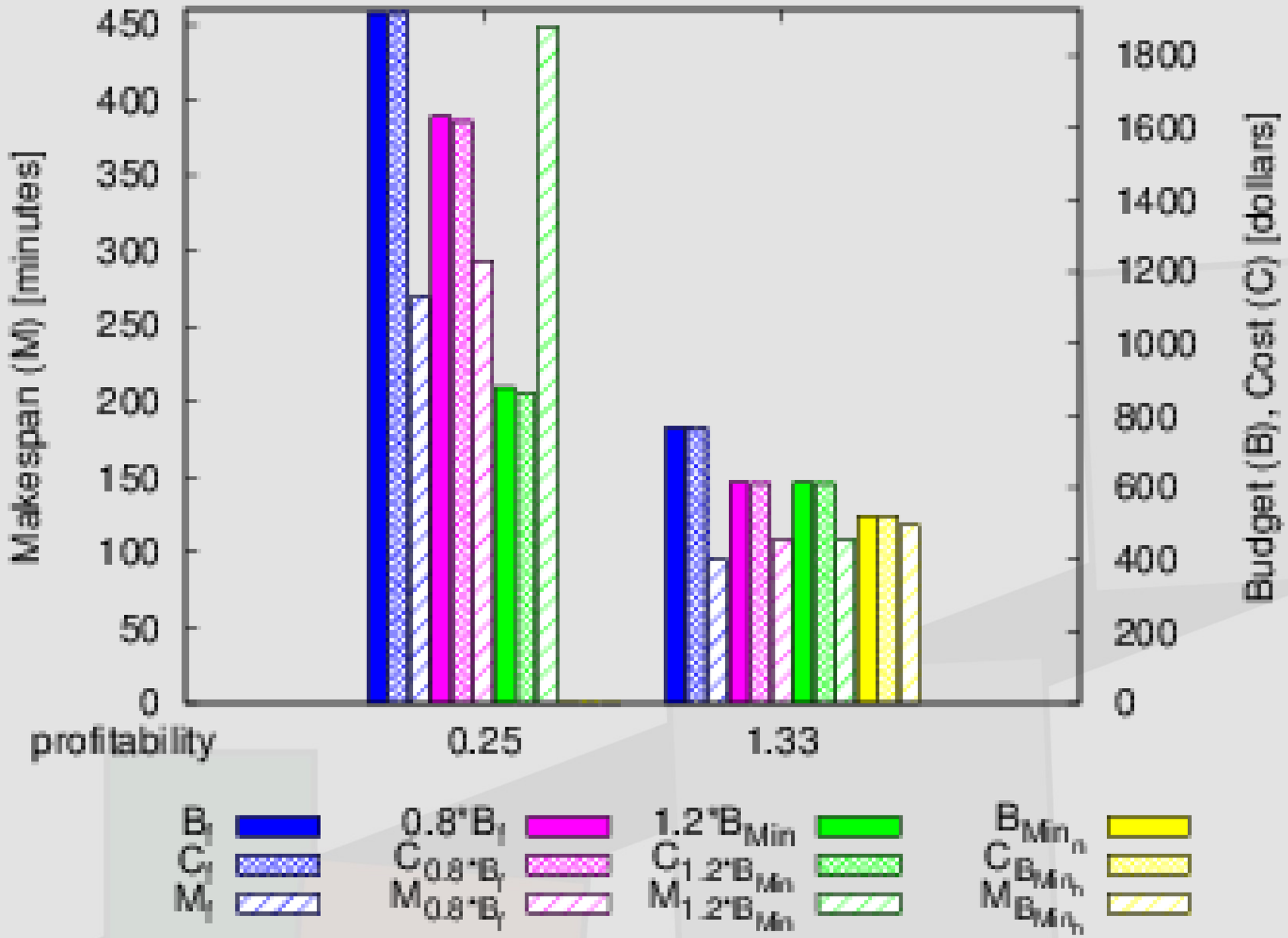
- Cluster 1 with normalized speed and cost
- Cluster 2 variable
- Design space for BaTS is profitability of cluster 2 w.r.t. cluster 1

	cluster 1		cluster 2		average	
profitability	speed	cost	speed	cost	speed	cost
0.25	1	1	1	4	1	2.5
0.75	1	1	3	4	2	2.5
1	1	1	1	1	1	1
1.33	1	1	4	3	2.5	2
4	1	1	4	1	2.5	1

Quality of Estimation (linear regression)



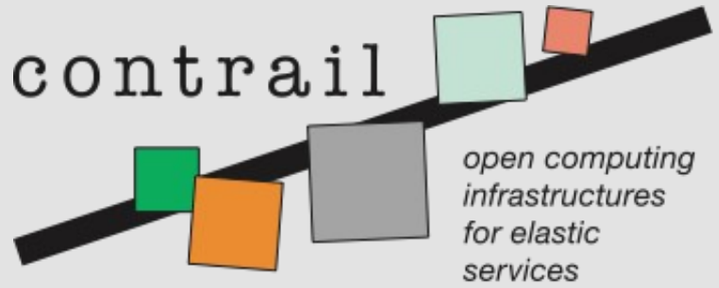
Quality of Schedules



Conlusions

- Bags of Tasks are an important class of applications, well suited for computing on clouds
- Choosing the right cloud offering(s) is tough
- **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:
    - Deal with costs for file I/O
    - Handle fluctuating node performance
    - Support workflows (tasks with dependencies)

Questions?



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

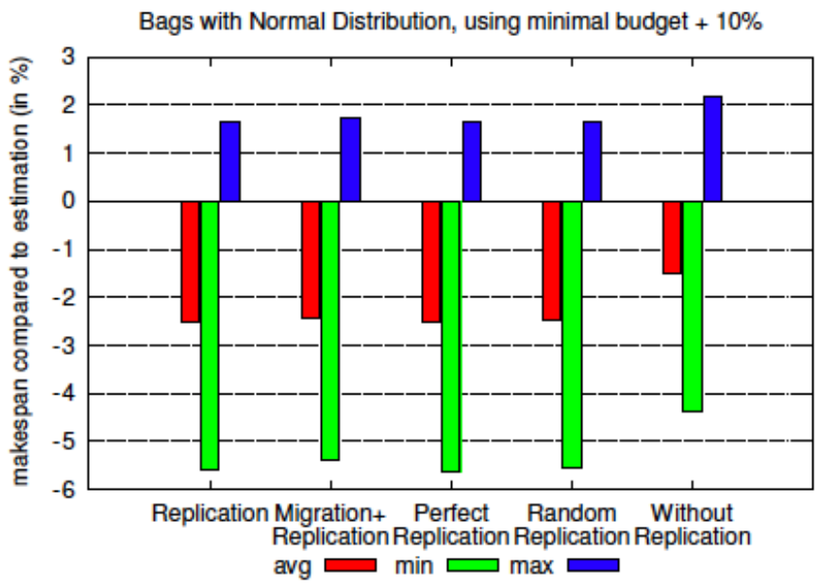
EU contribution: 8,3 million euro

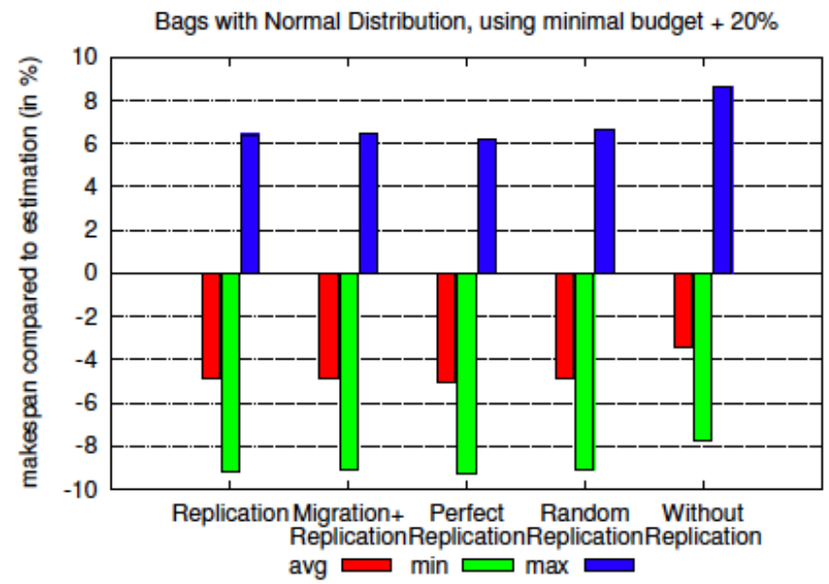
Execution: From 2010-10-01 till 2013-09-30

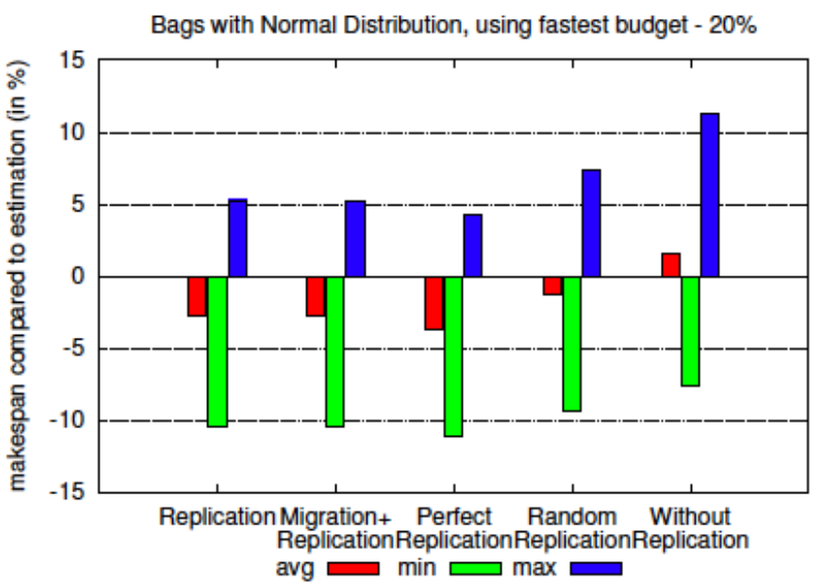
Duration: 36 months

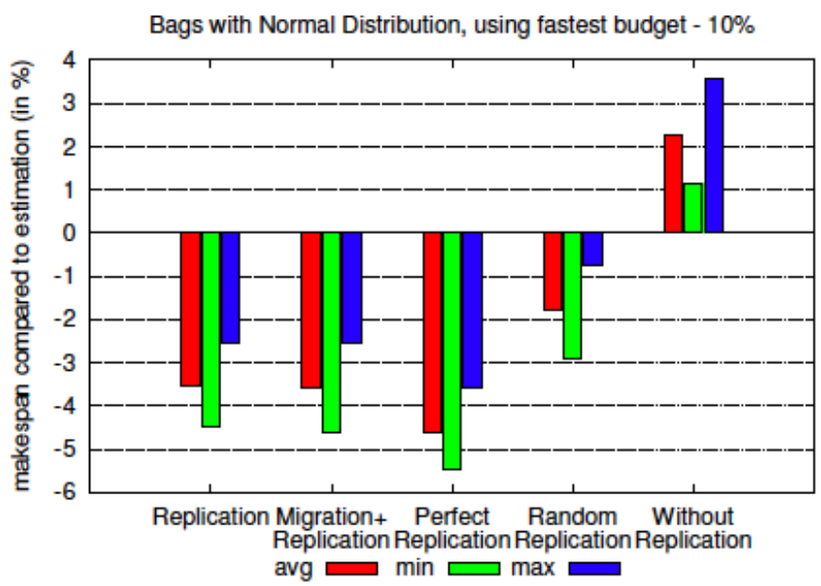
Contract type: Collaborative project (generic)

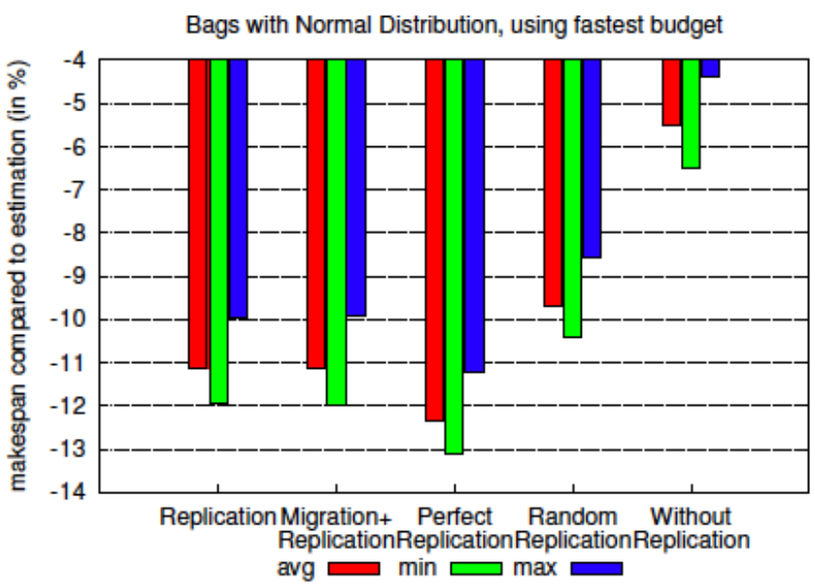
Tail Phase Optimization











Adding a “cushion”

- When planning, BaTS estimates the total unused time in the final ATU
  - Assuming each task has average completion time
- If tasks are running into the unused time, BaTS adds extra machines/time to the schedule
- Still no hard guarantees for meeting budget/makespan
  - We may always be unlucky with a heavy outlier towards the end
  - Improvement by separate tail phase