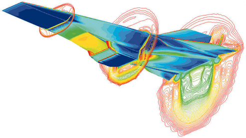
Cooperative Resource Management for Parallel and Distributed Systems

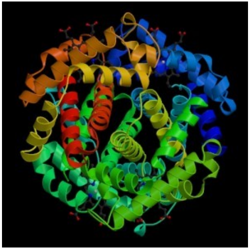
Cristian KLEIN

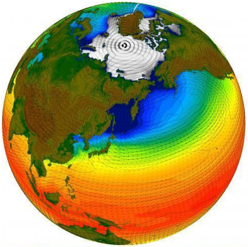
Avalon team, INRIA / LIP, ENS de Lyon, France
PhD Advisor: Christian PÉREZ

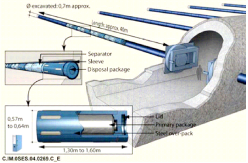
PhD Defence, Lyon, France
November 29, 2012

Computing needs are ever increasing . . .

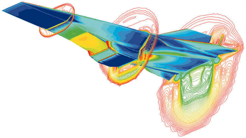


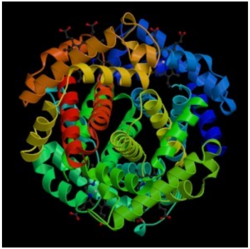


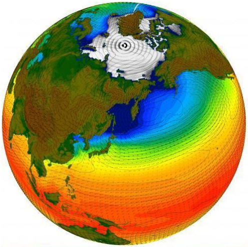


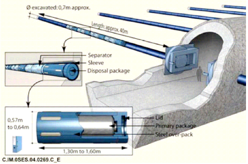


Computing needs are ever increasing . . .









Big computing and data requirements

Single-owner Computing Resources

Supercomputer

- Computers at the front-line
- Large-scale: 100,000 nodes; 1,500,000 cores
- **Complex** network topologies: torus, fat-tree
- **Heterogeneous** computing nodes
  - Blue Waters: CPU-only and CPU+GPU nodes
  - Curie: Fat, Hybrid, Thin nodes
- Top #1 (Titan): 188 M$ + 6 M$/yr

Single-owner Computing Resources

Supercomputer

- Computers at the front-line
- Large-scale: 100,000 nodes; 1,500,000 cores
- **Complex** network topologies: torus, fat-tree
- **Heterogeneous** computing nodes
  - Blue Waters: CPU-only and CPU+GPU nodes
  - Curie: Fat, Hybrid, Thin nodes
- Top #1 (Titan): 188 M\$ + 6 M\$/yr

Clusters

- Smaller scale
- Commodity hardware
- One cluster → nearly homogeneous
- Multiple cluster → **heterogeneous**

Multi-owner Computing Resources

Grid Computing

- Basically a multi-cluster system
- Geographically **dispersed**
- Owned by **multiple institutions**

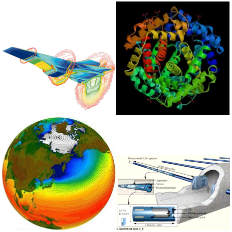
Cloud Computing

- **Renting** computing resource from a provider
- Amazon EC2 "supercomputer":
  - \$1060/hr for $1,250$ nodes ($10,000$ cores)

Sky Computing

- Renting from **multiple** providers

Running Applications on Computing Resources



Selecting Resources

- Take into account: heterogenity, centralized / distributed, price
- Goal: minimize completion time, cost, energy

Resource Management

Resource Management System (RMS)

- Multiplexes computing nodes among multiple users
- Aims at isolating them for security and **improved performance**

Current Practice

Dynamic allocations (à la Cloud)

[1] http://blog.cyclecomputing.com/2012/04/cyclecloud-50000-core-utility-supercomputing.html

[2] http://blog.cyclecomputing.com/2011/03/cyclecloud-4096-core-cluster.html

Current Practice

Dynamic allocations (à la Cloud)

- Clouds
  "The illusion of *infinite* computing resources available on demand"

[1] http://blog.cyclecomputing.com/2012/04/cyclecloud-50000-core-utility-supercomputing.html

[2] http://blog.cyclecomputing.com/2011/03/cyclecloud-4096-core-cluster.html

Current Practice

Dynamic allocations (à la Cloud)

- Clouds
  "The illusion of *infinite* computing resources available on demand"
  - Infinite? Actually up to 20 nodes

[1] http://blog.cyclecomputing.com/2012/04/cyclecloud-50000-core-utility-supercomputing.html

[2] http://blog.cyclecomputing.com/2011/03/cyclecloud-4096-core-cluster.html

Current Practice

Dynamic allocations (à la Cloud)

- Clouds
  "The illusion of *infinite* computing resources available on demand"
  - Infinite? Actually up to 20 nodes
  - "Supercomputer" of $5,674$ nodes (50,000 cores) spanning 7 Amazon EC2 regions[1]
  - **Out of capacity** errors[2]

[1] `http://blog.cyclecomputing.com/2012/04/cyclecloud-50000-core-utility-supercomputing.html`

[2] `http://blog.cyclecomputing.com/2011/03/cyclecloud-4096-core-cluster.html`

Current Practice

Dynamic allocations (à la Cloud)

- Clouds
  "The illusion of *infinite* computing resources available on demand"
  - Infinite? Actually up to 20 nodes
  - "Supercomputer" of $5,674$ nodes (50,000 cores) spanning 7 Amazon EC2 regions[1]
  - **Out of capacity** errors[2]

Static allocations (à la batch schedulers)

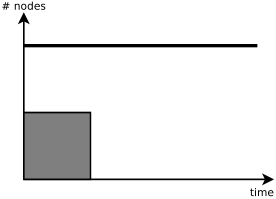
- a.k.a. rigid jobs (node-count times duration)
- **Misses** opportunities for improvement (next slide)

[1] `http://blog.cyclecomputing.com/2012/04/cyclecloud-50000-core-utility-supercomputing.html`

[2] `http://blog.cyclecomputing.com/2011/03/cyclecloud-4096-core-cluster.html`

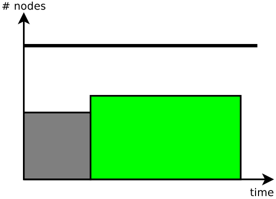
Application Properties by Resource Usage

Moldability



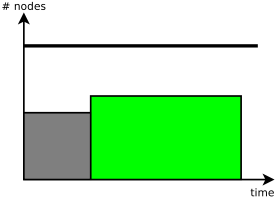
Application Properties by Resource Usage

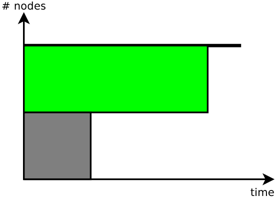
Moldability



Application Properties by Resource Usage

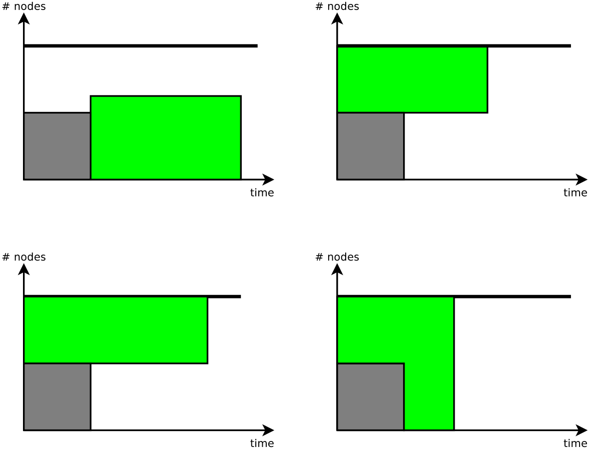
Moldability





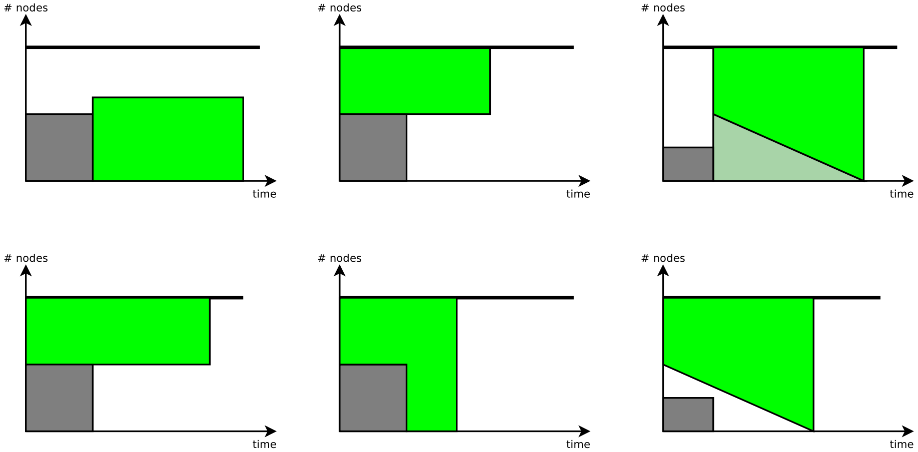
Application Properties by Resource Usage

Moldability | Malleability

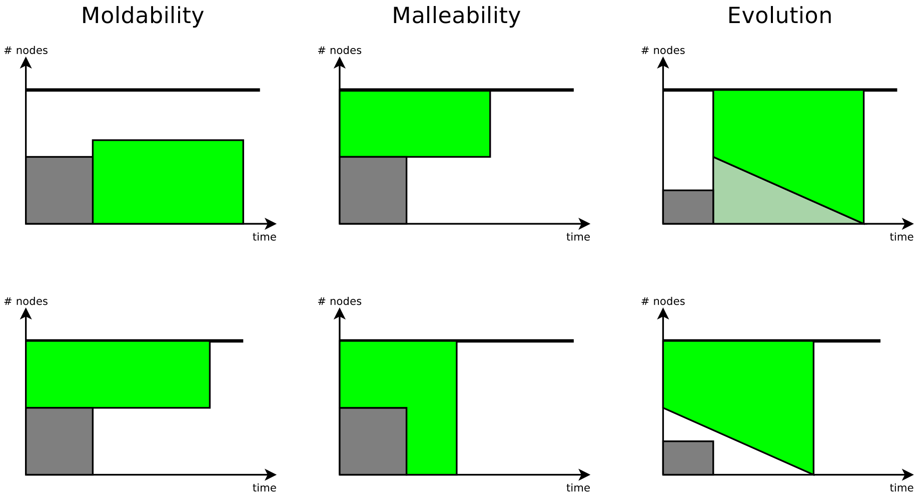


Application Properties by Resource Usage

Moldability	Malleability	Evolution



Application Properties by Resource Usage



Problem: Insufficiently supported in the state-of-the art.

Goal of the Thesis

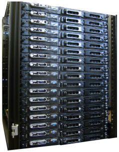
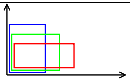
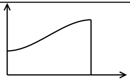
Improve resource management

- Resource utilization
- User-chosen criterion:
  - Application completion time
  - Energy consumption / Cost

How?

- Resource management architectures
- **Cooperates** with applications
- Support moldability, malleability, evolution
  - **without workarounds**
  - **reliably**
  - **efficiently**
- Focus is on **interaction**
  - **Re-use** proven **scheduling algorithms** as much as possible

Contributions: Resource Management Architectures

	 centralized	 distributed
 moldable	**CooRMv1** [1]	*dist*CooRM
 malleable/evolving	**CooRMv2** [2,3]	GridTLSE & DIET[4]

[1] C. Klein, C. Pérez, *An RMS Architecture for Efficiently Supporting Moldable Application*, HPCC, 2011

[2] C. Klein, C. Pérez, *Towards Scheduling Evolving Applications*, CGWS, 2011

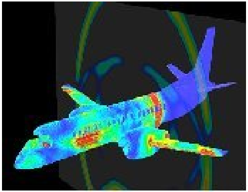
[3] C. Klein, C. Pérez, *An RMS for Non-predictably Evolving Applications*, Cluster, 2011

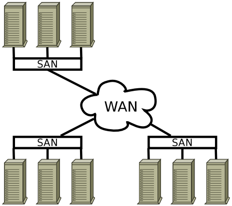
[4] F. Camillo, E. Caron, R. Guivarch, A. Hurault, C. Klein, C. Pérez, *Diet-ethic: Fair Scheduling of Optional Computations in GridRPC Middleware*, INRIA RR-7959, 2012

Computational Electromagnetics (CEM)

CEM Application

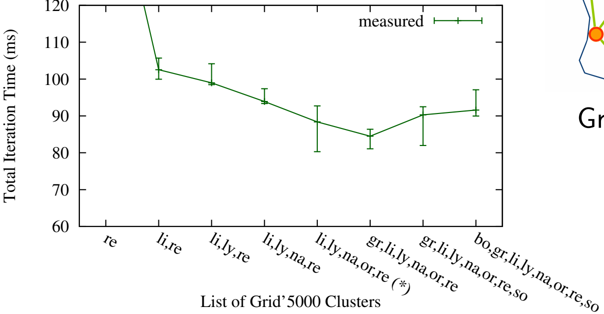
- Part of the ANR DiscoGrid project
- Antenna performance, electromagnetic compatibility ...
- Traditionally executed on a single cluster
- Huge mesh (number of tetrahedra) → launch on multiple clusters

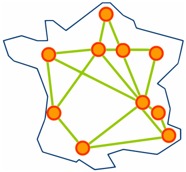




Efficient Execution of a Multi-cluster Moldable Applications

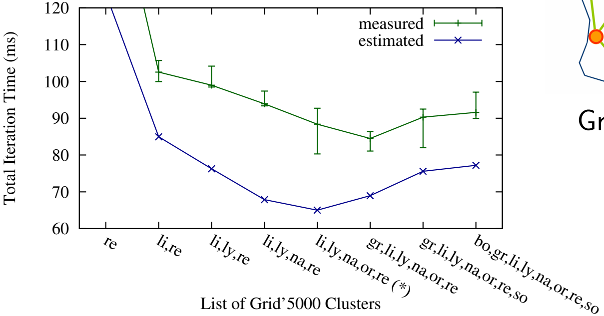
Performance of the CEM Application

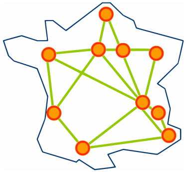




Grid'5000

Performance of the CEM Application



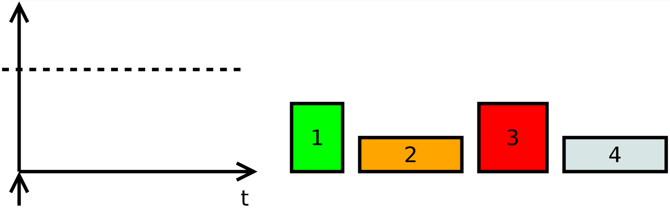


Grid'5000

- Devised a performance model
  - cluster computation power
  - inter-cluster network metrics (latency, bandwidth)
- Devised a **custom** resource selection **algorithm**

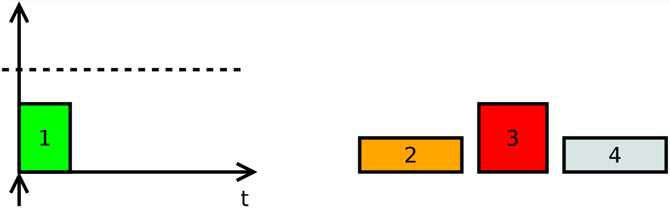
Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers



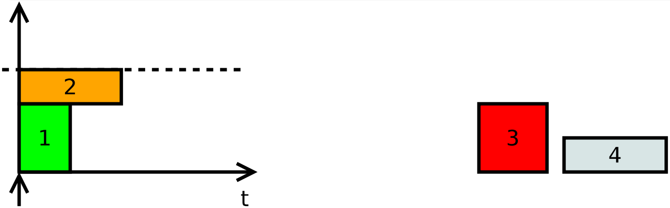
Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers



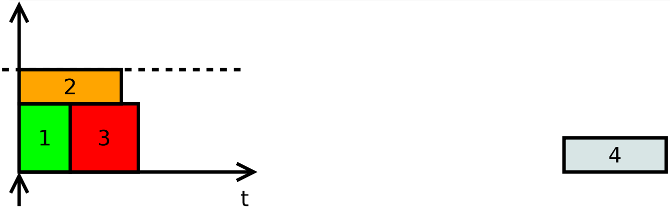
Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers



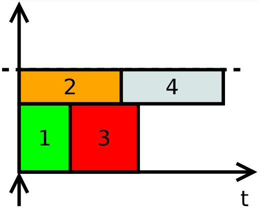
Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers



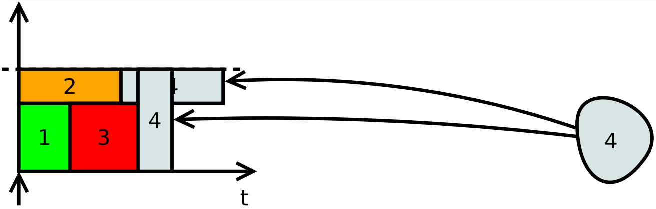
Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers



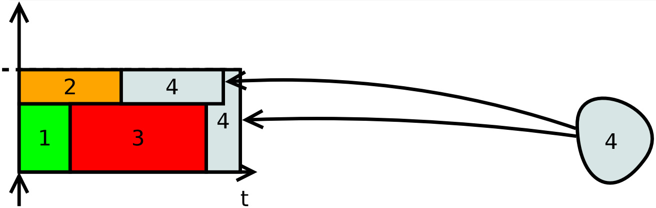
Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers



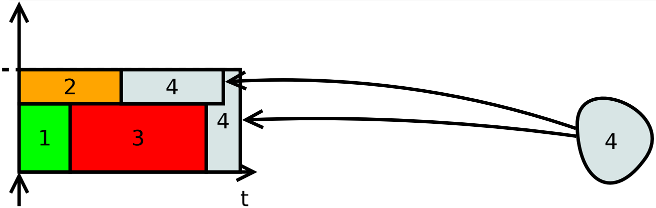
Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers



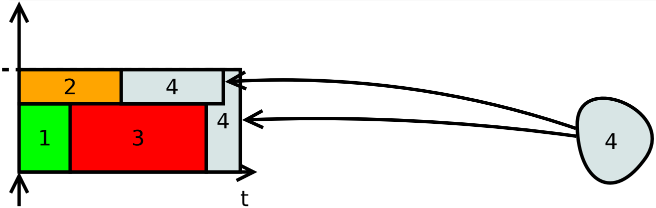
Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers
  - Workaround: re-implement RMS's scheduling algorithm



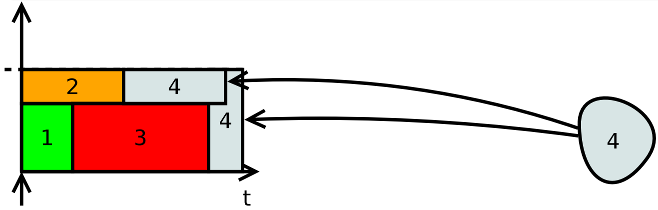
Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers
  - Workaround: re-implement RMS's scheduling algorithm
- **Limited** moldability: range of node-counts and a **single** duration
  - 8–16 nodes $\times$ 2 hours
  - e.g., SLURM



Towards Moldability

- No moldability (rigid jobs): fix node-count and duration
  - Most batch schedulers
  - Workaround: re-implement RMS's scheduling algorithm
- **Limited** moldability: range of node-counts and a **single** duration
  - 8–16 nodes × 2 hours
  - e.g., SLURM
- Moldable configurations: list of node-counts × durations
  - 8 nodes × 2 hours *OR* 16 nodes × 1 hour *OR* ...
  - e.g., OAR, Moab
  - **Impractical**: large number of configurations (next slide)



Number of Configurations

For a multi-cluster system:

- e.g., number of nodes on each cluster
- # configurations is large (exponential)

# clusters:	C
# nodes per clusters:	N
# configurations:	$(N+1)^{\mathbf{C}} - 1$

For a supercomputer:

- number of CPU nodes
- number of CPU+GPU nodes
- network topology
- # configurations is large (potentially exponential)

Number of Configurations

For a multi-cluster system:

- e.g., number of nodes on each cluster
- # configurations is large (exponential)

# clusters:	C
# nodes per clusters:	N
# configurations:	$(N+1)^{\mathbf{C}}-1$

For a supercomputer:

- number of CPU nodes
- number of CPU+GPU nodes
- network topology
- # configurations is large (potentially exponential)

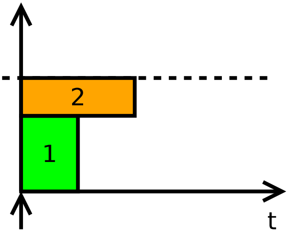
Problem

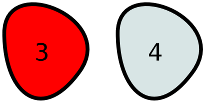
What interface should the RMS expose to allow moldable applications to effectively select resources?

Rationale

How CooRM Should Work

- Applications should take a more active role in the scheduling

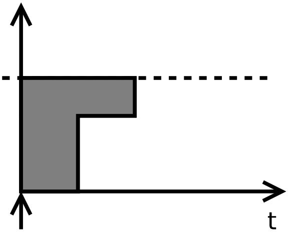


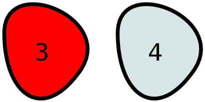


Rationale

How CooRM Should Work

- Applications should take a more active role in the scheduling
- RMS gives application the resource occupation (we call this a **view**)
  - No need to re-implement RMS's scheduling algorithm

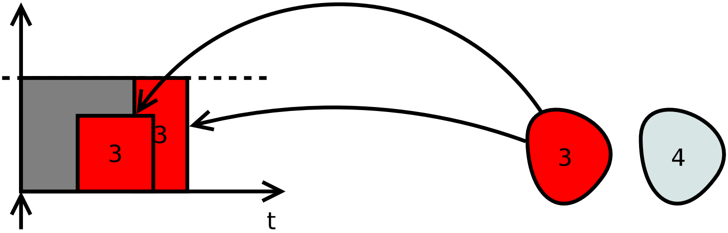




Rationale

How CooRM Should Work

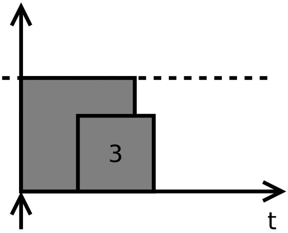
- Applications should take a more active role in the scheduling
- RMS gives application the resource occupation (we call this a **view**)
  - No need to re-implement RMS's scheduling algorithm
- Applications send a **resource requests**
  - Computed using custom resource selection algorithm

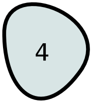


Rationale

How CooRM Should Work

- Applications should take a more active role in the scheduling
- RMS gives application the resource occupation (we call this a **view**)
  - No need to re-implement RMS's scheduling algorithm
- Applications send a **resource requests**
  - Computed using custom resource selection algorithm

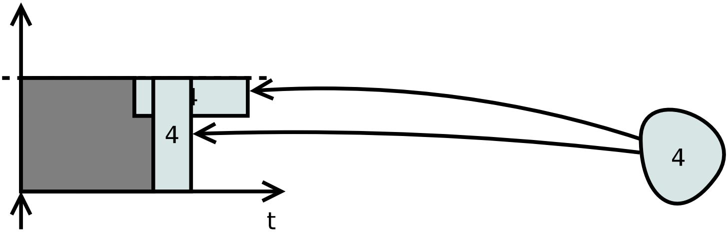




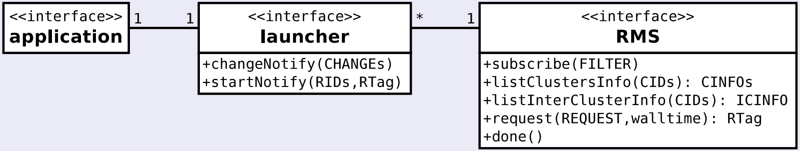
Rationale

How CooRM Should Work

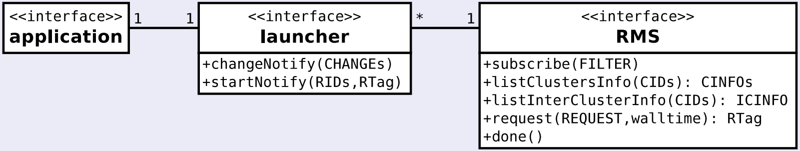
- Applications should take a more active role in the scheduling
- RMS gives application the resource occupation (we call this a **view**)
  - No need to re-implement RMS's scheduling algorithm
- Applications send a **resource requests**
  - Computed using custom resource selection algorithm

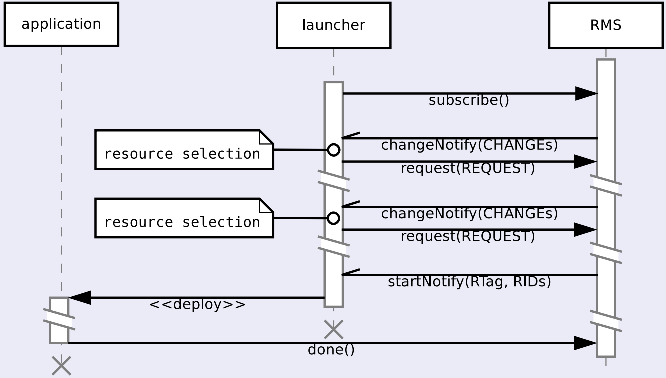


Architecture



Architecture





RMS-side Scheduling

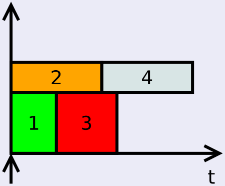
Main Responsibilities

- Compute views
- Compute start-times for requests
- Allocate node IDs

Example Implementation

- Based on Conservative Back-Filling (CBF)

Fair-start Delay and Ghosts



Initial schedule

RMS-side Scheduling

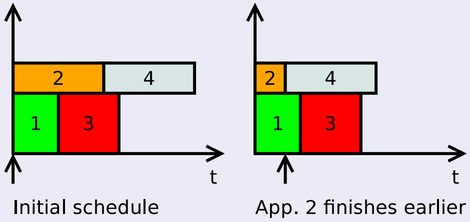
Main Responsibilities

- Compute views
- Compute start-times for requests
- Allocate node IDs

Example Implementation

- Based on Conservative Back-Filling (CBF)

Fair-start Delay and Ghosts



RMS-side Scheduling

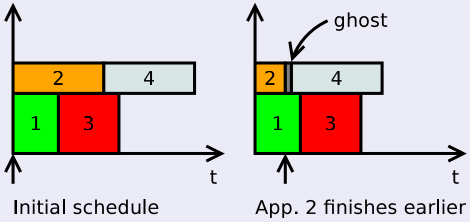
Main Responsibilities

- Compute views
- Compute start-times for requests
- Allocate node IDs

Example Implementation

- Based on Conservative Back-Filling (CBF)

Fair-start Delay and Ghosts



RMS-side Scheduling

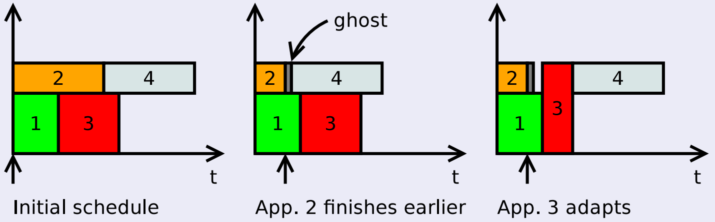
Main Responsibilities

- Compute views
- Compute start-times for requests
- Allocate node IDs

Example Implementation

- Based on Conservative Back-Filling (CBF)

Fair-start Delay and Ghosts



Application-side Scheduling

CEM Application

$$f_R : R_i \mapsto R_s, d$$

Input Resources (list of CIDs, n_H)

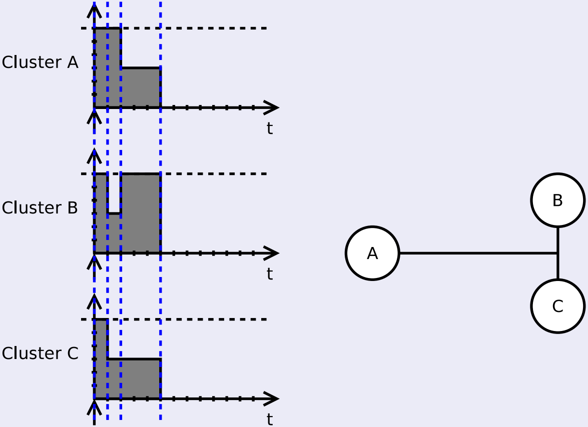
Selected Resources

execution time

Application-side Scheduling

CEM Application

$$f_R : R_i \mapsto R_s, d$$



Linear complexity

$< 1\,\mathrm{ms}$

Experimental Setup

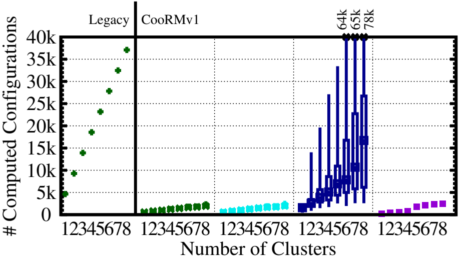
Resource Model

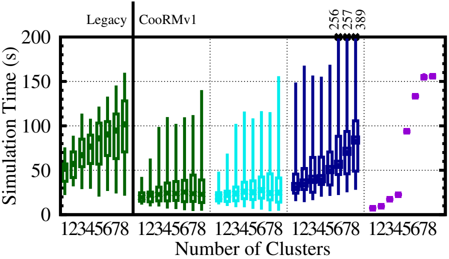
- n_C clusters, each having 128 hosts
- Cluster i is considered $10\%, 20\%, \ldots$ faster than cluster 1
- WAN: $5\,\mathrm{ms}$ same city, $10\,\mathrm{ms}$ same country, $50\,\mathrm{ms}$ otherwise

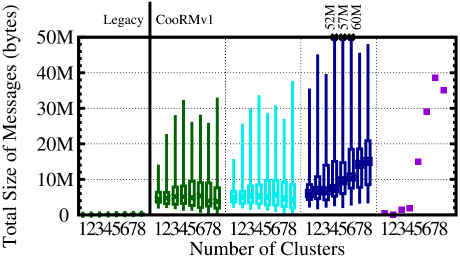
Application Model

- 200 application arriving at $1\,\mathrm{app}/\,\mathrm{second}$
- Mixture of
  - rigid, single-cluster moldable
    - consecutive jobs from `LLNL-Atlas-2006-1.1-cln`
    - 80% rigid jobs (as in traces)
    - 20% single-cluster moldable jobs (using Amdahl's law)
  - multi-cluster moldable applications (CEM)

Simulation Results







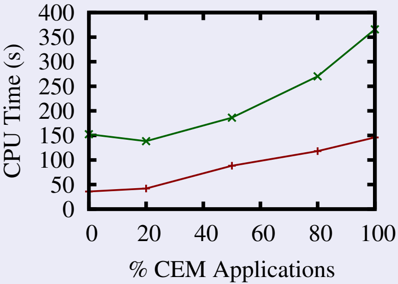
0% CEM, 0.5% CEM, 50% CEM, 100% CEM

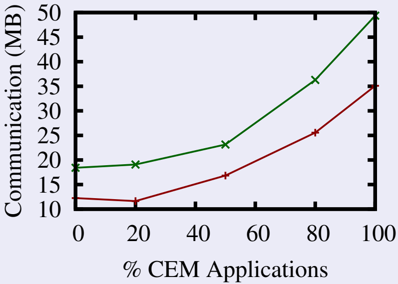
$<$ 200 KB/application

CooRMv1 Implementation

- $2,300$ SLOC of Python code
- Prototype implementation using CORBA (omniORBpy)
- CPU-time vs. simulation time
- TCP payload vs. size of messages

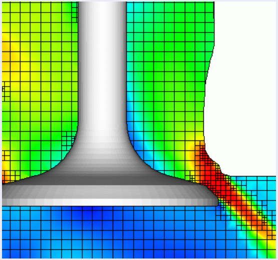
Simulations vs. Real Experiments

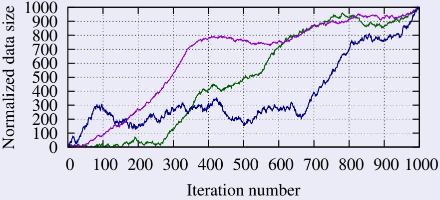


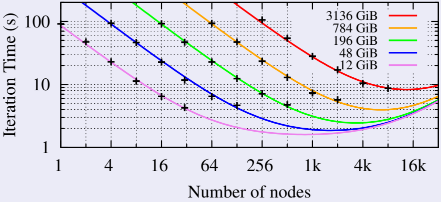


Adaptive Mesh Refinement Applications (AMR)

- Mesh is dynamically refined / coarsened as required by numerical precision
  - Memory requirements increase / decrease
  - Amount of parallelism increases / decreases
- Generally **evolves non-predictably**







End-user's Goal: maintain a given target efficiency

Problem and Goal

Problem

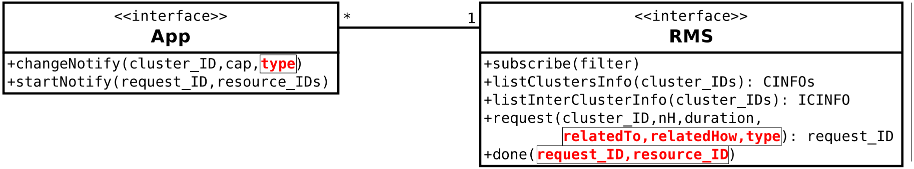
- Static allocations → **inefficient** resource utilisation
- Dynamic allocations (à la Cloud) → **out of capacity**
- Malleable jobs (KOALA, ReShape, Faucets . . .)
  → no guarantees ⇒ **application may crash**
  → **difficult to target custom objective**

Goal

An RMS which allows non-predictably evolving applications

- To use resources **efficiently**
- **Guarantee** the availability of resources

CooRMv2: Additions to CooRMv1



Overview

- Resource requests types
- Request relations
- Preemptible views

Resource Requests

- Number of nodes, duration
- RMS chooses start time $\rightarrow$ node IDs are allocated to the application

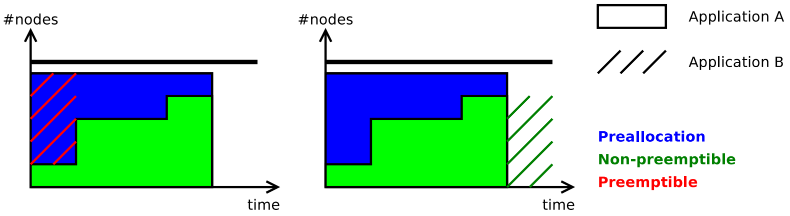
Resource Requests

- Number of nodes, duration
- RMS chooses start time $\rightarrow$ node IDs are allocated to the application
- Type
  - Non-preemptible (default in major RMSs, i.e., are not taken away)

Resource Requests

- Number of nodes, duration
- RMS chooses start time → node IDs are allocated to the application
- Type
  - Non-preemptible (default in major RMSs, i.e., are not taken away)
  - Preemptible (i.e., can be taken away at any time)

Resource Requests



- Number of nodes, duration
- RMS chooses start time → node IDs are allocated to the application
- Type
  - Non-preemptible (default in major RMSs, i.e., are not taken away)
  - Preemptible (i.e., can be taken away at any time)
  - Preallocation
    "I do not currently need these resources, but make sure I can get them immediately if I need them."

Request Relations

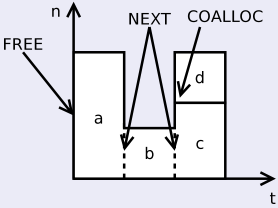
Request Relations

- Dynamic applications $\rightarrow$ multiple requests + temporal constraints

  relatedTo an existing request

  relatedHow FREE, NEXT, COALLOC

- Two methods: request(), done()



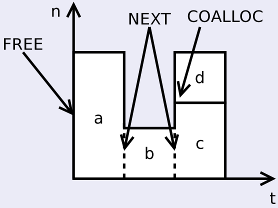
Request Relations

Request Relations

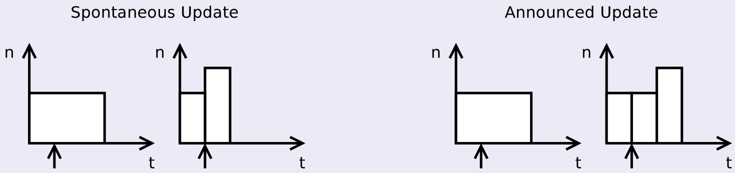
- Dynamic applications →
  multiple requests + temporal constraints

  relatedTo an existing request
  relatedHow FREE, NEXT, COALLOC

- Two methods: request(), done()



High-level Operations

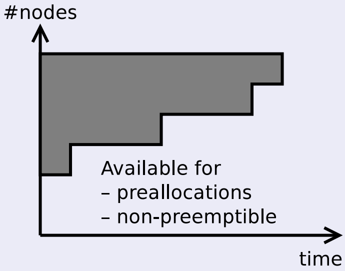


- An update is guaranteed to succeed only inside a pre-allocation

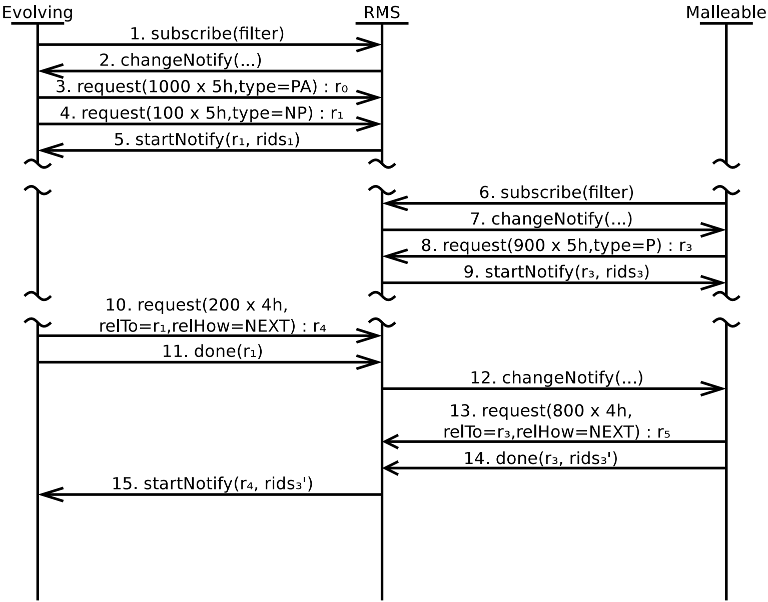
Views

Views

- Apps need to adapt their requests to the availability of the resources
- **Preemptible view** informs when resources need to be preempted



Example Interaction



CooRMv2 RMS Implementation

Overview

- Compute views
- Compute start times for each requests
- Start requests and allocate resources

Main Idea of the Scheduling Algorithm

- Pre-allocations and non-preemptible requests
  - Conservative Back-Filling (CBF)
- Preemptible requests
  - Equi-partitioning
  - Allow unused partitions to be filled by other applications

Non-predictably Evolving: Adaptive Mesh Refinement

Application Model

- Application knows its speed-up model
- Cannot predict its data evolution
- **Aim:** maintain a given target efficiency

Behaviour in CooRMv2

- Sends one **pre-allocation**
  - Simulation parameter: **overcommitFactor**
- Sends **non-preemptible** requests inside the pre-allocation

Malleable: Parameter-Sweep Application

Application Model

- Infinite number of single-node tasks
- All tasks have the same **duration** (known in advance)
- **Aim:** maximize throughput

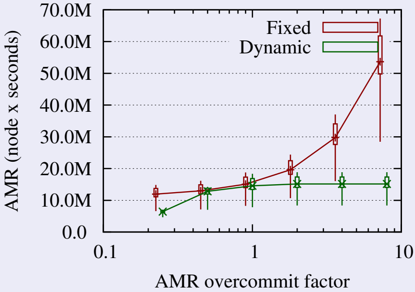
Behaviour in CooRMv2

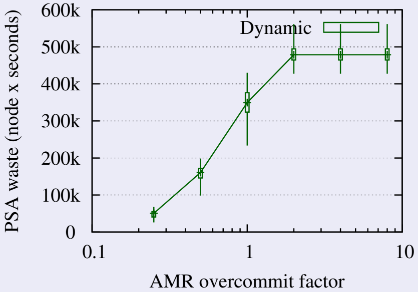
- Send **preemptible** requests
- Spawn tasks if resources are available
- Kill tasks if RMS asks to (increases **waste**)
- Wait for task completion, if informed in due time (no **waste**)

Scheduling with **Spontaneous** Updates

Experimental Setup

- Apps: 1xAMR (target eff. = 75%), 1xPSA (task duration = 600 s)
- Resources: number of nodes just enough to fit the AMR
- AMR uses fixed / dynamic allocations

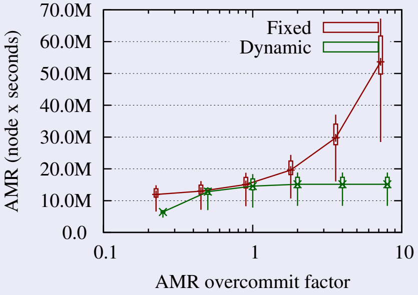


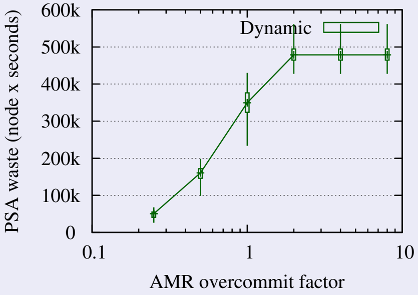


Scheduling with **Spontaneous** Updates

Experimental Setup

- Apps: 1xAMR (target eff. = 75%), 1xPSA (task duration = 600 s)
- Resources: number of nodes just enough to fit the AMR
- AMR uses fixed / dynamic allocations



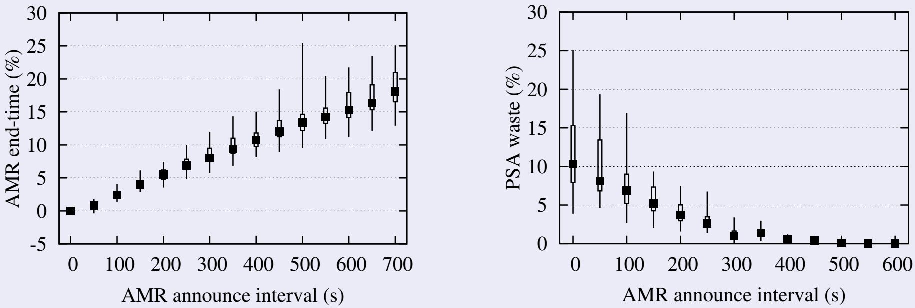


Wasted 40 M against 500 k (node×seconds)

Scheduling with **Announced** Updates

Experimental Setup

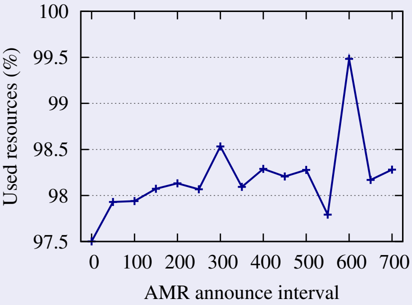
- Apps: 1xAMR (target eff. = 75%), 1xPSA (task duration = 600 s)
- Resources: number of nodes just enough to fit the AMR
- AMR uses announced updates (*announce interval*)

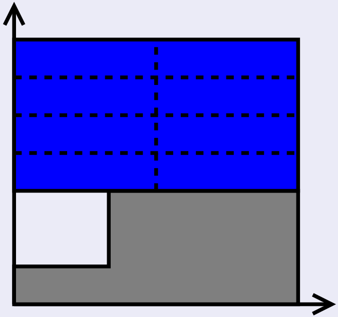


Announced Updates: Nice Resource "Filling"

Experimental Setup

- 1xAMR application
- PSA_1: task duration $= 600\,\mathrm{s}$

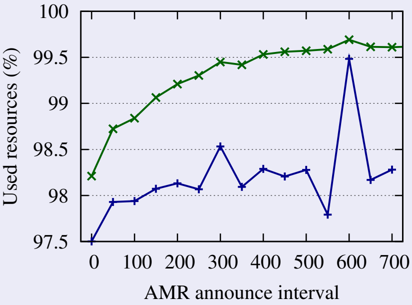


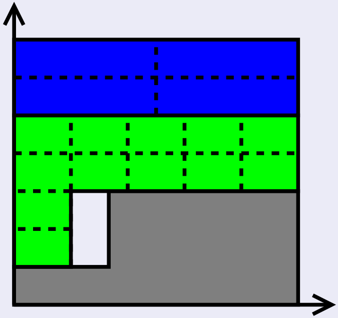


Announced Updates: Nice Resource "Filling"

Experimental Setup

- 1xAMR application
- PSA_1: task duration = 600 s
- PSA_2: task duration = 60 s

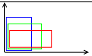
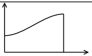




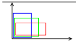
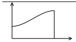
Conclusions

Goal: Improve resource management

- Proposing resource management architectures
- Promote **collaboration** with applications

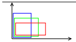
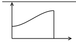
	 centralized	 distributed
 moldable	CooRMv1	*dist*CooRM
 malleable/evolving	CooRMv2	GridTLSE & DIET

CooRMv1 (1/2)

		
	**CooRMv1**	*dist*CooRM
	CooRMv2	GridTLSE & DIET

- Resource Management Architecture
- Efficiently support **moldable** applications
- Number of configurations is significantly reduced (10^3 vs. 10^{17})
- New cases become **practical**
- Validated through simulation and prototype implementation
- Studied time needed for applications to adapt

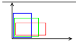
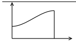
CooRMv1 (2/2)

		
	**CooRMv1**	*dist*CooRM
	CooRMv2	GridTLSE & DIET

Integration with CooRMv1

- Implemented by N. Toukourou, Engineer, INRIA
- **Results**: Easier to launch computation schemas on computing resources

CooRMv1 (2/2)

		
	**CooRMv1**	*dist*CooRM
	CooRMv2	GridTLSE & DIET

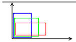
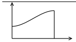
Integration with CooRMv1

- Implemented by N. Toukourou, Engineer, INRIA
- **Results**: Easier to launch computation schemas on computing resources

Custom Scheduling Algorithm for High-Level Waste Simulator

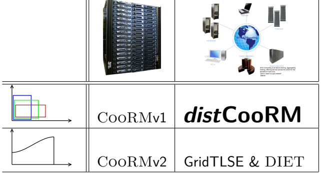
- Co-advised V. Lanore, ex M2 Student, ENS de Lyon
- Scheduling multi-level applications
- **Results**: Response-time improved (accepted ComPAS'13)

CooRMv2

		
	CooRMv1	*dist*CooRM
	**CooRMv2**	GridTLSE & DIET

- Extension of CooRMv1
- Efficiently deal with **evolving/malleable** applications
- Effective resource usage improved up to **3.6 times**
- Validated through simulations
- Prototype implementation is available

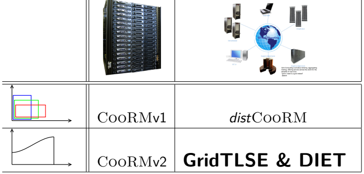
Other Contributions



distCooRM

- Collaboration with Y. Radenac, Myriads, INRIA
- Distributed version of CooRMv1
- **Results**: Shows good scalability (for a limited number of applications)

Other Contributions



distCooRM

- Collaboration with Y. Radenac, Myriads, INRIA
- Distributed version of CooRMv1
- **Results**: Shows good scalability (for a limited number of applications)

Optional Computation Support

- Collaboration with F. Camillo, R. Guivarch, A. Hurault, IRIT
- Grid-TLSE+DIET Use-case: Multiple Threshold Pivoting
- Architecture for efficiently dealing with optional computation
- **Results**: Improves user satisfaction and fairness (submitted CCGrid'13)
- **Transfer**: DIET patches submitted upstream

Perspectives

Short-term

- Integrate CooRMv1/v2 in existing batch schedulers
  - OAR, SLURM
- Validation with other applications
  - Cost, energy

Long-term

- **Topology** inside a supercomputer/cluster
  - Allow pre-launch topology optimization
- Economic model (à la Cloud)
  - Charge for pre-allocation?
  - Bonus for timely updates?
- *dist*CooRM
  - Improve scalability (add a pre-selection phase)
  - Add malleable / evolving support

Backup Slides

5 Related Work

- NIST Cloud Definition
- SLA Definition

6 CooRMv1

- Fairness

7 CooRMv2

- AMR Evolution Examples

NIST Cloud Definition

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

- Essential characteristics
  - on-demand self-service
  - broad network access
  - resource pooling
  - rapid elasticity
  - measured service
- Service models: SaaS, PaaS, IaaS
- Deployment models: private, community, public, hybrid

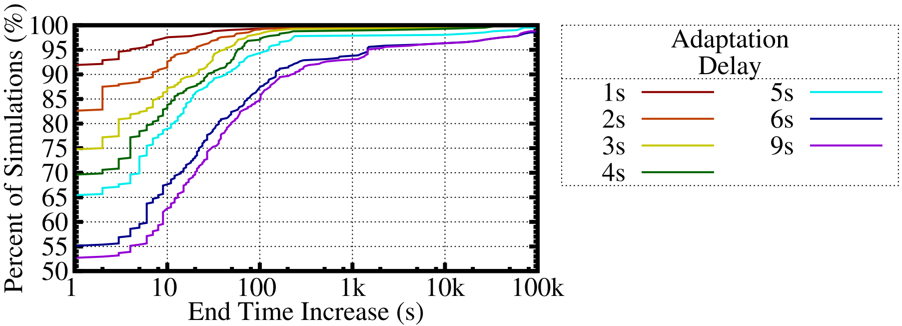
SLA@SOI Definition

- Machine-readable contract between a customer and a provider
- Guarantee that what you ask for, you get
- Allow you to verify provisioning
- Notify violations and define appropriate automated actions/penalties

Fairness

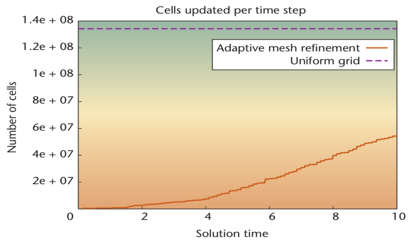
Simulation Setup

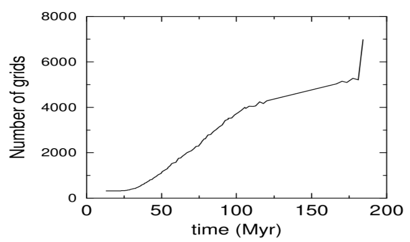
- Fair-start Delay: 5 seconds
- 1 x complex-moldable applications (CEM)
  - Simulated applications with lengthy resource selection
  - Added *adaptation delay*



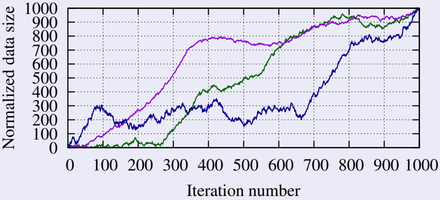
AMR Evolution

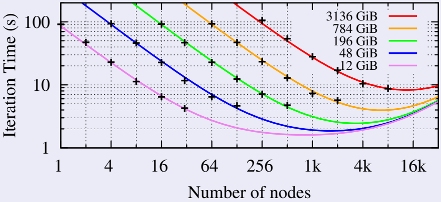
AMR Examples





AMR Model

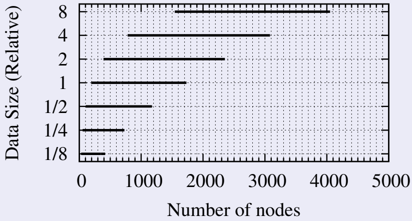




Executing AMR applications on HPC resources

Use **static** allocations (rigid jobs)

- E.g., cluster, supercomputing batch schedulers
- Evolution is not known in advance
  $\rightarrow$ User is forced to over-allocate
  $\rightarrow$ Inefficient resource usage
- Example: target efficiency 75% ($\pm 10\%$)



- **Ideally**, unused resources should be filled by other applications
  - Needs support from the Resource Management System (RMS)