CooRMv2: An RMS with Support for Non-predictably Evolving Applications

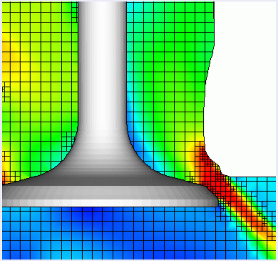
Cristian KLEIN, Christian PÉREZ

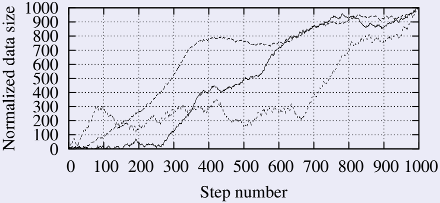
Avalon / GRAAL, INRIA / LIP, ENS de Lyon

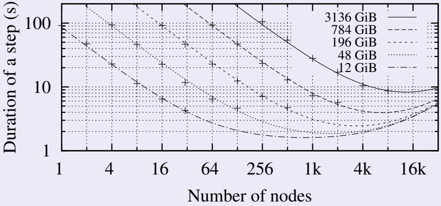
Scheduling Workshop, May 29–June 1, 2011, Aussois

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**





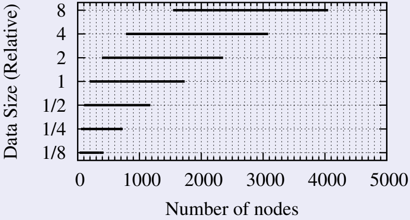


Goal: maintain a given target efficiency

Executing AMR applications on HPC resources (1/2)

Use **static** allocations (rigid jobs)

- E.g., cluster, supercomputing batch schedulers
- Evolution is not known in advance
  → User is forced to over-allocate
  → 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)

Executing AMR applications on HPC resources (2/2)

Use **dynamic** allocations

- **Malleable jobs**: RMS tells applications to grow/shrink

Executing AMR applications on HPC resources (2/2)

Use **dynamic** allocations

- **Malleable jobs**: RMS tells applications to grow/shrink
- Clouds
  "The illusion of *infinite* computing resources available on demand"

Executing AMR applications on HPC resources (2/2)

Use **dynamic** allocations

- **Malleable jobs**: RMS tells applications to grow/shrink
- Clouds
  "The illusion of *infinite* computing resources available on demand"
  - Infinite? Actually up to 20

Executing AMR applications on HPC resources (2/2)

Use **dynamic** allocations

- **Malleable jobs**: RMS tells applications to grow/shrink
- Clouds
  "The illusion of *infinite* computing resources available on demand"

  - Infinite? Actually up to 20
  - Even without this limit: "Out of capacity" errors

  → Application may run **out-of-memory**

Executing AMR applications on HPC resources (2/2)

Use **dynamic** allocations

- **Malleable jobs**: RMS tells applications to grow/shrink
- Clouds
  "The illusion of *infinite* computing resources available on demand"
  - Infinite? Actually up to 20
  - Even without this limit: "Out of capacity" errors

  → Application may run **out-of-memory**

- **Ideally**, RMS guarantees the availability of resources to an AMR application?

Problem

A Resource Management System (RMS) which allows non-predictably evolving applications

- To use resources efficiently
- Guarantee the availability of resources

1. Introduction

2. CooRMv2
   - Resource Requests
   - High-level Operations
   - Views
   - Scheduling Algorithm

3. Application Examples
   - Non-predictably Evolving: Adaptive Mesh Refinement
   - Malleable: Parameter-Sweep Application

4. Results

5. Conclusions

1. Introduction
2. CooRMv2
    - Resource Requests
    - High-level Operations
    - Views
    - Scheduling Algorithm
3. Application Examples
    - Non-predictably Evolving: Adaptive Mesh Refinement
    - Malleable: Parameter-Sweep Application
4. Results
5. Conclusions

Resource Requests

- Cluster ID, number of nodes, duration
- RMS chooses start time → node IDs are allocated to the application

Resource Requests

- Cluster ID, number of nodes, duration
- RMS chooses start time → node IDs are allocated to the application
- Type
  - Non-preemptible (default in major RMSs)

Resource Requests

- Cluster ID, number of nodes, duration
- RMS chooses start time → node IDs are allocated to the application
- Type
  - Non-preemptible (default in major RMSs)
  - Preemptible (think OAR best-effort jobs)

Resource Requests

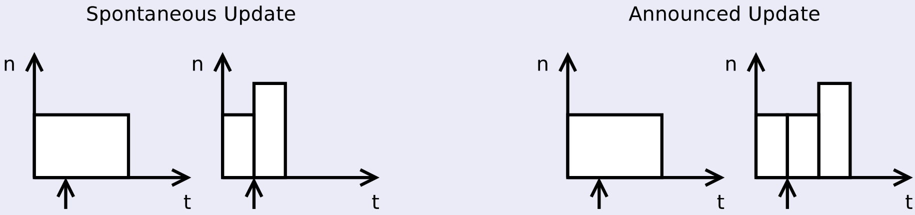
- Cluster ID, number of nodes, duration
- RMS chooses start time → node IDs are allocated to the application
- Type
  - Non-preemptible (default in major RMSs)
  - Preemptible (think OAR best-effort jobs)
  - Pre-allocation
    "I do not currently need these resources, but make sure I can get them immediately if I need them."

High-level Operations

Low-level Operations

- CooRMv2 defines simple, low-level operations on requests

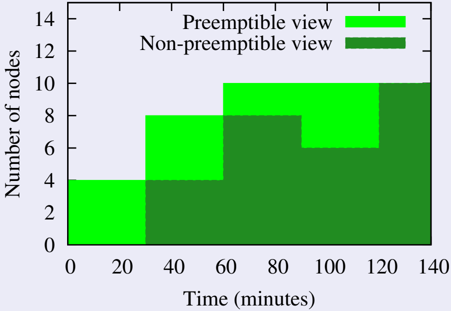
High-level Operations



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

Views

- Apps need to adapt their requests to the availability of the resources
- Each app is presented with two views: non-preemptible, preemptible
- Preemptible view informs when resources need to be preempted



Scheduling Algorithm

- Pre-allocations and non-preemptible requests
  - Conservative Back-Filling (CBF)
- Preemptible requests
  - equi-partitioning

1. Introduction
2. CooRMv2
   - Resource Requests
   - High-level Operations
   - Views
   - Scheduling Algorithm
3. Application Examples
   - Non-predictably Evolving: Adaptive Mesh Refinement
   - Malleable: Parameter-Sweep Application
4. Results
5. Conclusions

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 speed-up

Behaviour in CooRMv2

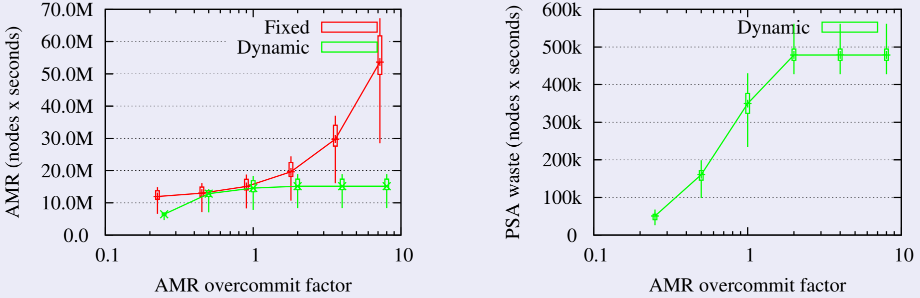
- Send **preemptible** requests
- Spawn tasks if resources are available
- Kill tasks if RMS asks to (increases **waste**)
- Stop tasks if will not be available (no waste)

1. Introduction
2. CooRMv2
   - Resource Requests
   - High-level Operations
   - Views
   - Scheduling Algorithm
3. Application Examples
   - Non-predictably Evolving: Adaptive Mesh Refinement
   - Malleable: Parameter-Sweep Application
4. Results
5. Conclusions

Scheduling with **Spontaneous** Updates

Experimental Setup

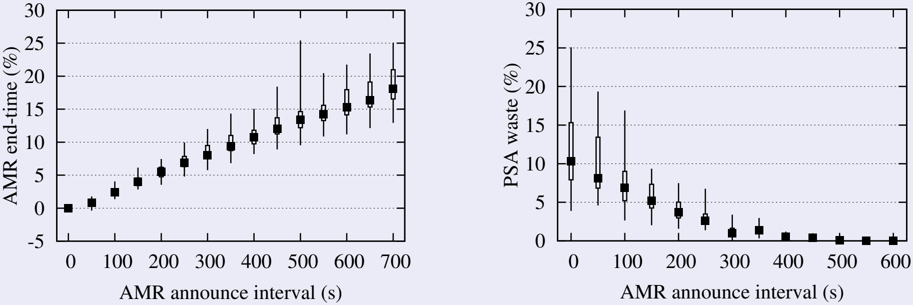
- Apps: 1xAMR (target eff. = 75%), 1xPSA (task duration = 600 s)
- Resources: number of nodes just enough to fit the AMR
- AMR uses fixed / dynamic allocations



Scheduling with **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*)



1. Introduction

2. CooRMv2
   - Resource Requests
   - High-level Operations
   - Views
   - Scheduling Algorithm

3. Application Examples
   - Non-predictably Evolving: Adaptive Mesh Refinement
   - Malleable: Parameter-Sweep Application

4. Results

5. Conclusions

Conclusions

CooRMv2

- A centralized RMS which supports
  - Evolving apps
  - Malleable apps
- Can be used to manage federation of clusters

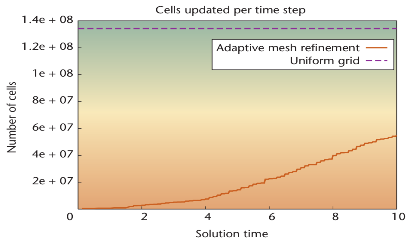
Perspectives

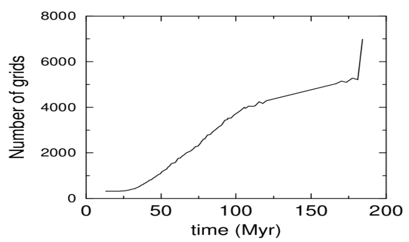
- What economic model?
  - Charge for unused pre-allocated resources?
  - Charge for frequency / size of updates?
  - Charge for quality / timeliness of updates?
- Non-homogeneous networks (e.g., torus topology)?

Backup Slides

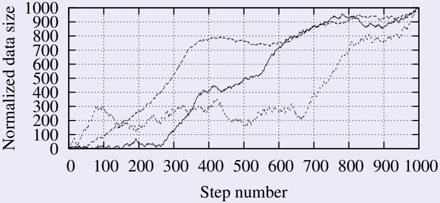
AMR Evolution

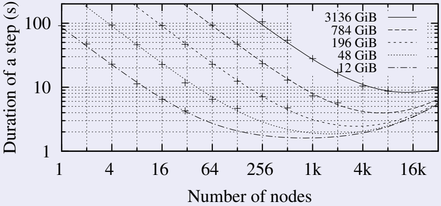
AMR Examples





AMR Model





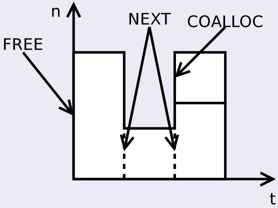
Principles — Request Relations

Request Relations

- dynamic applications → multiple requests
- + temporal constraints between requests

  relatedTo an existing request

  relatedHow FREE, NEXT, COALLOC
- request(), done()



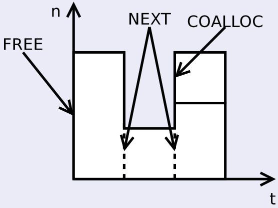
Principles — Request Relations

Request Relations

- dynamic applications → multiple requests
- + temporal constraints between requests

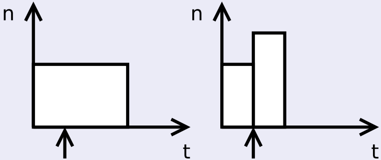
  relatedTo an existing request

  relatedHow FREE, NEXT, COALLOC
- request(), done()

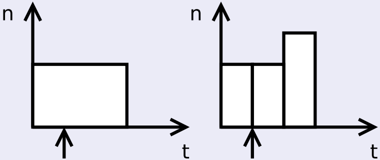


High-level Operations

Spontaneous Update

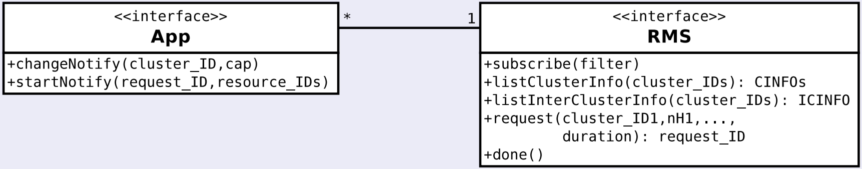


Announced Update



Architecture

CooRM



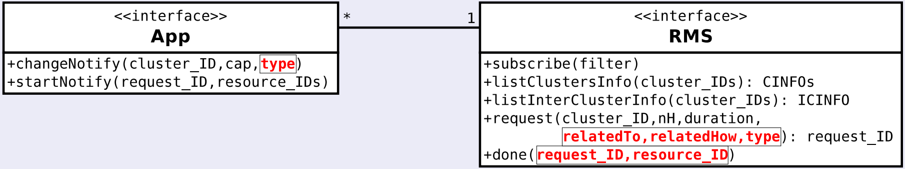
<<interface>>
App

+changeNotify(cluster_ID,cap)
+startNotify(request_ID,resource_IDs)

<<interface>>
RMS

+subscribe(filter)
+listClusterInfo(cluster_IDs): CINFOs
+listInterClusterInfo(cluster_IDs): ICINFO
+request(cluster_ID1,nH1,...,
duration): request_ID
+done()

CooRMv2



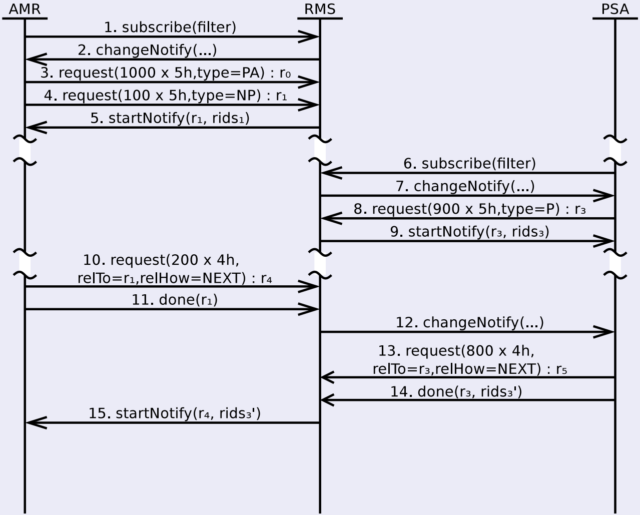
<<interface>>
App

+changeNotify(cluster_ID,cap,type)
+startNotify(request_ID,resource_IDs)

<<interface>>
RMS

+subscribe(filter)
+listClustersInfo(cluster_IDs): CINFOs
+listInterClusterInfo(cluster_IDs): ICINFO
+request(cluster_ID,nH,duration,
relatedTo,relatedHow,type): request_ID
+done(request_ID,resource_ID)

Interaction



AMR

RMS

PSA

1. subscribe(filter)

2. changeNotify(...)

3. request(1000 x 5h,type=PA) : r_0

4. request(100 x 5h,type=NP) : r_1

5. startNotify(r_1, rids$_1$)

6. subscribe(filter)

7. changeNotify(...)

8. request(900 x 5h,type=P) : r_3

9. startNotify(r_3, rids$_3$)

10. request(200 x 4h,
relTo=r_1,relHow=NEXT) : r_4

11. done(r_1)

12. changeNotify(...)

13. request(800 x 4h,
relTo=r_3,relHow=NEXT) : r_5

14. done(r_3, rids$_3$')

15. startNotify(r_4, rids$_3$')

RMS Implementation

Main Responsibilities

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

Main Idea of the Scheduling Algorithm

- Applications are ordered according to arrival time
- Pre-allocated resources cannot be pre-allocated by next applications
- Preemptible resources are shared equally

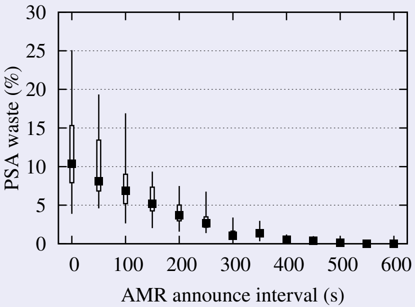
AMR Pre-announcements

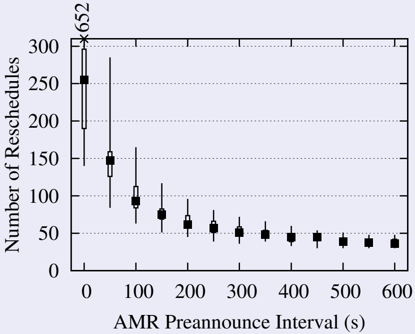
Experimental Setup

- launched at $t = 0$: 1xAMR application, 1xPSA application
- PSA: task duration $= 600\,\mathrm{s}$
- AMR: "pre-announces" changes (*pre-announce interval*)
  - Done either to be nice to other apps
  - Basically, the AMR application makes an UPDATE every interval

AMR Pre-announcements (cont.)

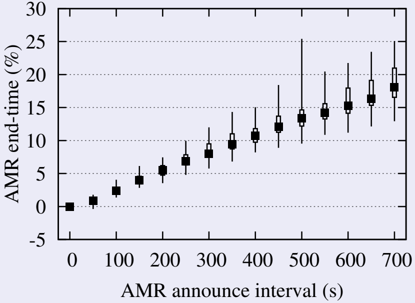
Pros

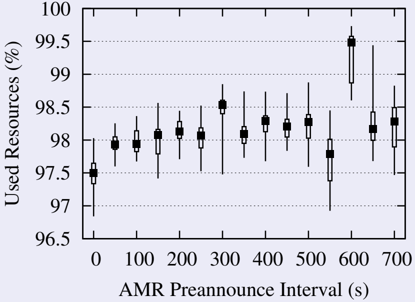




AMR Pre-announcements (cont.)

Cons

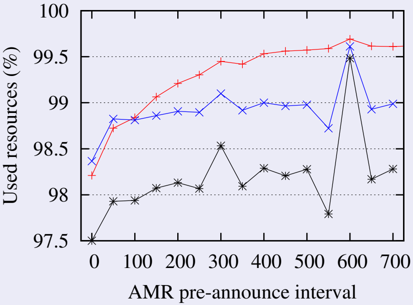




Nice Resource “Filling”

Experimental Setup

- launched at $t = 0$: 1xAMR application, 2xPSA application
- PSA_1: task duration $= 600\,\mathrm{s}$, PSA_2: task duration $= 60\,\mathrm{s}$



1xPSA

2xPSA

2xPSA (strict equi-partitioning)