Untying RMS from Application Scheduling

Cristian KLEIN, Christian PÉREZ

Avalon / GRAAL, INRIA / LIP, ENS de Lyon

COOP Meeting, 13-14 Octobre 2010, Toulouse
Introduction

Let Us Start with a Complex Application

- Computational ElectroMagnetics (CEM) Application
  - multi-cluster, moldable, non-evolving
  - Uses MPI for intra-cluster communication
  - Uses TCP for inter-cluster communication
- Huge mesh (number of tetrahedra) $\rightarrow$ launch on multiple clusters
- Devised its own resource selection algorithm
  - cluster computation power
  - inter-cluster network metrics (latency, bandwidth)
Complex Resource Selection

CEM Application (Dynamic Environment)

to-violette
to-pastel
so-sol
so-helios
so-azur
re-paradent
re-paramount
re-paraquad
re-paravent
or-netgdx
or-gdx1
or-gdx
na-griffon
na-grelon
ly-sagittaire
ly-capricorne
li-chinqchint
li-chuque
li-chtia
li-chicon
gr-genepi
bo-borderline
bo-bordereau
bo-bordeplage
bo-bordemer
**Introduction**

**HPC Resources**

- Managed by Resource Management Systems (RMSs)
- Give users exclusive access to resources
  - Submit rigid jobs (apps are executed as resources become available)
  - Advance reservation (apps have a fixed start time)
- User specifies resource requirements:
  - Resource Specification Language (RSL – e.g. JSDL)
  - RMS chooses resources

**Problem:** How to effectively choose resources for moldable, non-evolving applications with complex resource selection?
1 Introduction

2 Related Work

3 CooRM
   - Rationale
   - Architecture
   - Application-side Scheduling
   - RMS-side Scheduling

4 Experiments
   - Simulations
   - Validation

5 Conclusions
1. Introduction

2. Related Work

3. CooRM
   - Rationale
   - Architecture
   - Application-side Scheduling
   - RMS-side Scheduling

4. Experiments
   - Simulations
   - Validation

5. Conclusions
Resource Specification Languages (RSLs) 27 min

Globus’ RSL  host count, min. memory, wall-time, etc.

JSDL  range of hosts, single wall-time, etc.

OAR’s RSL  allow multiple moldable configurations
  • exhaustively enumerating all configurations is impractical

Gather information about the system
Make targeted resource requests
Submit rigid jobs
Redundant requests
Make advance reservation
Resource Specification Languages (RSLs) 27 min

Globus’ RSL  host count, min. memory, wall-time, etc.

JSDL  range of hosts, **single** wall-time, etc.

OAR’s RSL  allow multiple **moldable** configurations
  - exhaustively enumerating all configurations is impractical

Circumventing the RSLs’ Limitations

- Gather information about the system
- Make targeted resource requests
  - Submit rigid jobs
    - Redundant requests
  - Make advance reservation
Targeted Job Submissions

- Execution time is difficult to compute
  - an upper bound (called wall-time) is given
- Also called submit-time moldability
- Has been shown to be inefficient → wanted: schedule-time moldability
Targeted Job Submissions

- Execution time is difficult to compute
  - an upper bound (called *wall-time*) is given
- Also called *submit-time moldability*
- Has been shown to be inefficient → wanted: *schedule-time moldability*

Redundant Requests

- Submit multiple jobs, when one starts cancel the others
- Has been shown to be unfair and to worsen estimated start times
Execution time is difficult to compute

→ an upper bound (called *wall-time*) is given

Advance Reservation create *resource fragmentation*
1. Introduction

2. Related Work

3. CooRM
   - Rationale
   - Architecture
   - Application-side Scheduling
   - RMS-side Scheduling

4. Experiments
   - Simulations
   - Validation

5. Conclusions
At each event → a new schedule is computed
  ▶ event = new application submitted, application finishes

Resource requests are added one-by-one to a Gantt chart
At each event → a new schedule is computed
  - event = new application submitted, application finishes

Resource requests are added one-by-one to a Gantt chart
At each event → a new schedule is computed
  ▶ event = new application submitted, application finishes

Resource requests are added one-by-one to a Gantt chart
At each event → a new schedule is computed
  ▶ event = new application submitted, application finishes

Resource requests are added one-by-one to a Gantt chart
Rationale 21 min
How Batch Schedulers Work

- At each *event* → a new schedule is computed
  - *event* = new application submitted, application finishes
- Resource requests are added one-by-one to a Gantt chart
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)
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)
- Application decides what resources to *request*
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)
- Application decides what resources to request
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)
- Application decides what resources to request
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)
- Application decides what resources to request
- Handled outside the scheduling loop
  - Unresponsive applications cannot block the whole system
  - RMS will push a view to the applications
  - Applications send resource requests
Resource Model

CID: A
CINFO:
- 6 hosts
- 2 CPUs / host
- 3.5 Mflops / host
- Gigabit Ethernet

ICINFO:
- 6 ms
- 1 Gbps

CID: B
CINFO:
- 24 hosts
- 4 CPUs / host
- 7 Mflops / host
- Infiniband

REQUEST: list of (CID, nH) + wall-time
Architecture

<<interface>>

application

<<interface>>

launcher
+changeNotify(CHANGEs)
+startNotify(RIDs,RTag)

<<interface>>

RMS
+subscribe(FILTER)
+listClustersInfo(CIDs): CINFOs
+listInterClusterInfo(CIDs): ICINFO
+request(REQUEST,walltime): RTag
+done()
Application-side Scheduling

CEM Application

\[ f_R : R_i \leftrightarrow R_s, d \]

Input Resources (list of CIDs, \( n_H \)) \( \leftrightarrow \) Selected Resources

\( \text{wall-time} \)
Application-side Scheduling

CEM Application

\[ f_R : R_i \mapsto R_s, d \]
Application-side Scheduling

CEM Application

\[ f_R : R_i \leftrightarrow R_s, d \]
Application-side Scheduling

CEM Application

\[ f_R : R_i \leftrightarrow R_s, d \]

Cluster A

Cluster B

Cluster C

\[ f_R : R_i \leftrightarrow R_s, d \]

A

B

C

Graph showing the application-side scheduling with clusters and time axis.
Application-side Scheduling

CEM Application

\[ f_R : R_i \leftrightarrow R_s, d \]
Application-side Scheduling

CEM Application

\[ f_R : R_i \leftrightarrow R_s, d \]
Application-side Scheduling

CEM Application

\[ f_R : R_i \leftrightarrow R_s, d \]

Best
Request: C/3, for 6h
End-time: now + 7h
Application-side Scheduling

CEM Application

\[ f_R : R_i \leftrightarrow R_s, d \]

Best
Request: C/3, for 6h
End-time: now + 7h
Application-side Scheduling

CEM Application

\[ f_R : R_i \leftrightarrow R_s, d \]

Best
Request: C/3, for 6h
End-time: now + 7h

Cluster A

Cluster B

Cluster C
Application-side Scheduling

CEM Application

\[ f_R : R_i \mapsto R_s, d \]

Cluster A

Cluster B

Cluster C

Best
Request: C/3, for 6h
End-time: now + 7h

request(C/3, 6h)
Application-side Scheduling

CEM Application

\[ f_R : R_i \leftrightarrow R_s, d \]

Previous Best
Request: C/3, for 6h
End-time: now + 7h
Application-side Scheduling

CEM Application

\[ f_R : R_i \mapsto R_s, d \]

**Previous Best**
Request: C/3, for 6h
End-time: now + 7h
Application-side Scheduling

CEM Application

\[ f_R : R_i \mapsto R_s, d \]

Cluster A

Cluster B

Cluster C

Previous Best
Request: C/3, for 6h
End-time: now + 7h

\[ \text{Previous Best} \]

\[ \text{Request: C/3, for 6h} \]

\[ \text{End-time: now + 7h} \]
Application-side Scheduling

CEM Application

\[ f_R : R_i \mapsto R_s, d \]

Cluster A

Previous Best
Request: C/3, for 6h
End-time: now + 7h

Best
Request: C/6, for 3h
End-time: now + 5h

request(C/6, 3h)

Cluster B

Cluster C

A

B

C

Untying RMS from Application Scheduling

COOP, Toulouse
RMS-side Scheduling (1/2)

Fair-start Delay and Ghosts

Initial schedule
RMS-side Scheduling (1/2)

Fair-start Delay and Ghosts

Application States inside the RMS

Rescheduling done only after re-scheduling timer expires

Cristian KLEIN (INRIA)

Untying RMS from Application Scheduling
RMS-side Scheduling (1/2)

Fair-start Delay and Ghosts

**Application States inside the RMS**

Rescheduling done only after re-scheduling timer expires

Cristian KLEIN (INRIA)

Untying RMS from Application Scheduling

COOP, Toulouse
RMS-side Scheduling (1/2)

Fair-start Delay and Ghosts

Application States inside the RMS

Rescheduling done only after re-scheduling timer expires
RMS-side Scheduling (1/2)

Fair-start Delay and Ghosts

Initial schedule

Job 2 finishes early

Job 3 adapts

Application States inside the RMS

- Rescheduling done only after re-scheduling timer expires
RMS-side Scheduling (2/2)

Scheduling Algorithm

1. Check whether there are applications which sent done and mark them as ghosts.

2. Check whether there are ghosts which have expired and mark them as finished.

3. Generate the initial last view by adding the Started applications and Ghosts to it.

4. For all RequestSent and Waiting applications, taken in the order of their submission time:
   1. Set the application’s view to the last view;
   2. Compute the application’s estimated start time, i.e. the first hole in the last view, where its request would fit for the specified wall-time;
   3. Add its request to the last view;
   4. Mark the application as Waiting.

5. Send changeNotify messages to applications whose view has changed because of the previous steps.
Conclusions So Far

What We Have

- CooRM – untying application scheduling from the RMS
- RMS schedules applications based on their current resource request and updates their view
- Applications receive a view and updated their resource requests
- System converges (i.e. messages eventually get consumed)

Solved Issues

- Flexibility: Applications can implement their own scheduling
- Authoritativeness: RMS keeps control over the resources

Open Issues

- Scalability: Are CPU / Network usage acceptable?
- Fairness: How to choose the fair-start parameter?
1. Introduction

2. Related Work

3. CooRM
   - Rationale
   - Architecture
   - Application-side Scheduling
   - RMS-side Scheduling

4. Experiments
   - Simulations
   - Validation

5. Conclusions
Simulations 11 min

- **Scalability**
  - CooRM vs. OAR (only simple applications)
    - Single-cluster
    - Multi-cluster
  - CooRM only (with the CEM application)
    - Multi-cluster

- **Fairness**
  - Unfairness induced by insufficient fair-start delay
  - Resource waste
Simulation Setup

Resource Model
- \( n_C \) clusters, each having 128 hosts
- Cluster \( i \) is considered 10\%, 20\%, ..., faster than cluster 1
- WAN: 5 ms same city, 10 ms same country, 50 ms otherwise

Application Model
- \( p_{cem} \) complex-moldable (CEM) applications
- \( 1 - p_{cem} \) simple applications
  - First 200 jobs from LLNL-Atlas-2006-1.1-cln
  - \( 1 - p_{mo} \) rigid jobs (as in traces)
  - \( p_{mo} \) simple-moldable jobs (use Amdahl’s law)
- Application arrival rate: \( 1/s \)
## Scalability (1/4) – Setup

### Initial Parameters
- Re-scheduling timer: 1 second
- Fair-start delay: 5 seconds

### Metrics
- Unique configurations
- Amount of communication
- Simulation time
Scalability (2/4) – CooRM vs. OAR

Simpler Applications / Single Cluster ($p_{cem} = 0$, $p_{mo} = \ldots$, $n_C = 1$)
Scalability (3/4) – CooRM vs. OAR

Simpler Applications / Multi-Cluster \((p_{cem} = 0, p_{mo} = 0.2, n_C = \ldots)\)
Scalability (4/4) – CEM on CooRM

Complex Applications / Multi-Cluster \( (p_{cem} = \ldots, p_{mo} = 0.2, n_C = \ldots) \)
Fairness (1/2)

Simulation Setup

- Fair-start Delay: 5 seconds
- To simulated applications with lengthy resource selection
  - Added *adaptation delay*

![Graph of CEM End Delaying (Average)](image1)

![Graph of CEM End Delaying (Maximum)](image2)
Fairness (2/2)

Simulation Setup

- Increased fair-start delay

![Graph showing resource waste vs. fair-start delay for different CEM levels]
Validation with a Prototype

- Prototype implementation using CORBA (omniORBpy)
- CPU-time vs. simulation time
- TCP payload vs. size of messages

**CooRMsIm vs. CooRMi**

![Graphs showing CPU time and communication (MB) vs. percentage of CEM applications for CooRMsIm and CooRMi.](attachment:graphs.png)
1 Introduction

2 Related Work

3 CooRM
   - Rationale
   - Architecture
   - Application-side Scheduling
   - RMS-side Scheduling

4 Experiments
   - Simulations
   - Validation

5 Conclusions
Conclusions

For COoRM

- Support complex, moldable applications on complex resources
- Experiments show that the system behaves well
- Future work
  - Support malleable / evolving applications (e.g. workflows)
  - De-centralize COoRM
Conclusions

For CooRM

- Support complex, moldable applications on complex resources
- Experiments show that the system behaves well
- Future work
  - Support malleable / evolving applications (e.g. workflows)
  - De-centralize CooRM

For COOP

- How to integrate CooRM with RMSs
  - XtreemOS, SPADES (ex-DIET), OAR
- How application can take advantage of CooRM
Ideas for COOP

Resource Managers

- XtreemOS
  - How to implement CooRM on XtreemOS?

Programming Model

- Grid-TLSE + CooRM
- EDF + CooRM
  - Discuss application needs
  - Is this RMS interface enough for resource selection?