Metaheuristic and Agent-based Scheduling in Heterogeneous Environments

Marc Eduard Frîncu, Flavia Micota, Daniela Zaharie, Dana Petcu
West University of Timisoara and e-Austria Research Institute, Romania

Jan 25th 2011
Outline

1. Introduction

2. Meta-heuristics for Scheduling Problems
   - Scheduling Meta-heuristics for Heterogeneous Environments
   - Switching Between Scheduling Policies

3. Multi-Agent Scheduling Platform
   - Platform Architecture
   - Platform Feedback Control Loops
   - Platform Tests

4. Conclusions
Heterogeneous Distributed Environments

- **heterogeneous** - different computational costs on different resources for every task
- **dynamic** - resource workload and network availability can change in time
- **unpredictable behavior** - no suitable model for predicting the future characteristics of the DS
Introduction

Heterogeneous Distributed Environments

- **heterogeneous** - different computational costs on different resources for every task
- **dynamic** - resource workload and network availability can change in time
- **Solution**: scheduling algorithms and load balancing techniques
- **unpredictable behavior** - no suitable model for predicting the future characteristics of the DS
- **Solution**: dynamic and adaptive scheduling; self-healing abilities; distributed decision (multi-agent systems)
## Introduction

### Heterogeneous Distributed Environments

- **heterogeneous** - different computational costs on different resources for every task
- **dynamic** - resource workload and network availability can change in time
- **Solution**: scheduling algorithms and load balancing techniques
- **unpredictable behavior** - no suitable model for predicting the future characteristics of the DS
- **Solution**: dynamic and adaptive scheduling; self-healing abilities; distributed decision (multi-agent systems)

*Scheduling algorithms* are central to ensuring the quality of the service
However . . .

Scheduling algorithms give different results on different platforms ⇒ need for a stable algorithm and/or an optimal policy selection strategy.

Switching between SAs is difficult ⇒ need for a new model for representing SA.

Resource Management Systems (RMS) must be fault-tolerant and multi-provider enabled ⇒ need to self-heal and scale.
However . . .

Scheduling algorithms give different results on different platforms ⇒ need for a stable algorithm and/or an optimal policy selection strategy

- **Solution**: SAs and some switching policies, however large execution times and small policy selection

Switching between SAs is difficult ⇒ need for a new model for representing SA

Resource Management Systems (RMS) must be fault-tolerant and multi-provider enabled ⇒ need to self-heal and scale
However . . .

<table>
<thead>
<tr>
<th>Scheduling algorithms give different results on different platforms ⇒ need for a stable algorithm and/or an optimal policy selection strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Solution:</strong> SAs and some switching policies, however large execution times and small policy selection</td>
</tr>
<tr>
<td><strong>Our approach:</strong> meta-heuristics for scheduling and for selecting the proper scheduling strategy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Switching between SAs is difficult ⇒ need for a new model for representing SA</th>
</tr>
</thead>
</table>

| Resource Management Systems (RMS) must be fault-tolerant and multi-provider enabled ⇒ need to self-heal and scale |
However...

Scheduling algorithms give different results on different platforms ⇒ need for a stable algorithm and/or an optimal policy selection strategy

- **Solution**: SAs and some switching policies, however large execution times and small policy selection
- **Our approach**: meta-heuristics for scheduling and for selecting the proper scheduling strategy

Switching between SAs is difficult ⇒ need for a new model for representing SA

- **Solution**: no such model

Resource Management Systems (RMS) must be fault-tolerant and multi-provider enabled ⇒ need to self-heal and scale
Scheduling algorithms give different results on different platforms ⇒ need for a stable algorithm and/or an optimal policy selection strategy

- **Solution**: > SAs and some switching policies, however large execution times and small policy selection
- **Our approach**: meta-heuristics for scheduling and for selecting the proper scheduling strategy

Switching between SAs is difficult ⇒ need for a new model for representing SA

- **Solution**: no such model
- **Our approach**: nature inspired formalism


Resource Management Systems (RMS) must be fault-tolerant and multi-provider enabled ⇒ need to to self-heal and scale
However . . .

Scheduling algorithms give different results on different platforms ⇒ need for a stable algorithm and/or an optimal policy selection strategy

- **Solution:** > SAs and some switching policies, however large execution times and small policy selection
- **Our approach:** meta-heuristics for scheduling and for selecting the proper scheduling strategy

Switching between SAs is difficult ⇒ need for a new model for representing SA

- **Solution:** no such model
- **Our approach:** nature inspired formalism


Resource Management Systems (RMS) must be fault-tolerant and multi-provider enabled ⇒ need to self-heal and scale

- **Solutions:** > RMS for cluster and single provider, < multi-provider and self-healing RMS
Scheduling algorithms give different results on different platforms ⇒ need for a stable algorithm and/or an optimal policy selection strategy

- **Solution**: > SAs and some switching policies, however large execution times and small policy selection
- **Our approach**: meta-heuristics for scheduling and for selecting the proper scheduling strategy

Switching between SAs is difficult ⇒ need for a new model for representing SA

- **Solution**: no such model
- **Our approach**: nature inspired formalism

---


Resource Management Systems (RMS) must be fault-tolerant and multi-provider enabled ⇒ need to to self-heal and scale

- **Solutions**: > RMS for cluster and single provider, < multi-provider and self-healing RMS
- **Our approach**: self-healing scheduling platform
Population-based Meta-heuristic

- uses a population of schedules (random + heuristic initial schedules)
- each schedule in the population is perturbed using an heuristic (MinQL, DMECT, hybrid move/swap) leading to several variants: pMinQL, pDMECT, SPS
- the surviving elements are selected by using tournament selection
- the iterative process stops when no significant improvement in the quality of schedule (e.g. makespan) was noticed

SimplePopulationScheduler

1. Generate the set of initial schedules:
2. $S \leftarrow \{S_1, \ldots, S_N\}$
3. while (the stopping condition is false) do
4. for $i = 1, N$ do
5. $S'_i \leftarrow \text{perturb}(S_i)$
6. end for
7. $S \leftarrow \text{select}(S, \{S'_1, \ldots, S'_N\})$
8. end while

Experiments:
Simulated testing environment

The population-based heuristics lead to significantly better schedules that non-population one but also to higher computational costs in estimating the optimal schedule

Population-based Meta-heuristic

- uses a population of schedules (random + heuristic initial schedules)
- each schedule in the population is perturbed using an heuristic (MinQL, DMECT, hybrid move/swap) leading to several variants: pMinQL, pDMECT, SPS
- the surviving elements are selected by using tournament selection
- the iterative process stops when no significant improvement in the quality of schedule (e.g. makespan) was noticed

SimplePopulationScheduler

1: Generate the set of initial schedules:
2: \( S \leftarrow \{S_1, \ldots, S_N\} \)
3: while (the stopping condition is false) do
4:   for \( i = 1, N \) do
5:     \( S'_i \leftarrow \text{perturb}(S_i) \)
6:   end for
7:   \( S \leftarrow \text{select}(S, \{S'_1, \ldots, S'_N\}) \)
8: end while

Experiments:

- Simulated testing environment
- The population-based heuristics lead to significantly better schedules than non-population one but also to higher computational costs in estimating the optimal schedule

Improvement with respect to the MinMin schedule:

\[
\frac{(\text{makespan}(\text{MinMin}) - \text{makespan}(\text{Heuristic})) \times 100}{\text{makespan}(\text{MinMin})}
\]

- the population based variants of MinQL and DMECT leads to better results but with the price of a longer scheduling time
- the simple SPS algorithm (with non-iterated perturbation operator) ensures a good compromise quality vs. scheduling time
Overview

Changes in system (resources/network/tasks) require a constant adaptation of the SA in order to minimize the makespan.

Existing solutions include: switching algorithms, best policy selection strategies, ...
Overview

Changes in system (resources/network/tasks) require a constant adaptation of the SA in order to minimize the makespan.

Existing solutions include: switching algorithms, best policy selection strategies, . . .

Our solution: use (un)supervised based on task/resource information

**Unsupervised case: Fuzzy C-Mean Clustering**

![Bar chart](image)

Figure: Unsupervised case: gain with regard to the Best Selection strategy of several scheduling heuristics

Matching percentage: $63.93 \pm 10.59\%$ (h=0) and $74.81 \pm 6.58\%$ (h=42)
Switching Between Scheduling Policies

Supervised case: MultiLayer Perceptron neural network

Table: Confusion matrix for the case of using a $h = 42$ training set with 8 hidden layer elements and a learning rate of 0.3

<table>
<thead>
<tr>
<th></th>
<th>a = MinQL</th>
<th>b = Max-Min</th>
<th>c = Sufferage</th>
<th>d = Min-Min</th>
<th>e = DMECT</th>
<th>f = DMECT2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>45</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>127</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td>10</td>
<td>23</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>22</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

Matching percentage: between 71.94% to 80.85% ($h=0$) and between 60.65% and 82.24% ($h=42$)
Multi-Agent Self-Healing Scheduling Platform

Distributed heterogeneous systems need to be: reliable, efficient, secure & accessible.

Possible solution

- MAS could provide a solution as they are: **distributed, cooperative, “intelligent”** & autonomous
- Feedback Control Loops (FCL) could provide the recovery facilities as they:
  - allow self-* by following the MAPE loop
  - can recover from expected and unexpected failures
  - can be easily integrated with MAS
Multi-Agent Self-Healing Scheduling Platform

Distributed heterogeneous systems need to be: reliable, efficient, secure & accessible.

Possible solution

- MAS could provide a solution as they are: distributed, cooperative, “intelligent” & autonomous
- Feedback Control Loops (FCL) could provide the recovery facilities as they:
  - allow self-* by following the MAPE loop
  - can recover from expected and unexpected failures
  - can be easily integrated with MAS
Distributed heterogeneous systems need to be: reliable, efficient, secure & accessible.

**Possible solution**

- **MAS could provide a solution as they are:** distributed, cooperative, “intelligent” & autonomous
- **Feedback Control Loops (FCL) could provide the recovery facilities as they:**
  - allow self-* by following the MAPE loop
  - can recover from expected and unexpected failures
  - can be easily integrated with MAS

Building a functional platform which is fully distributed in terms of communication, storage, scheduling and negotiation solutions and also self healing is still an open technical and research challenge
Multi-Agent Self-Healing Scheduling Platform

Distributed heterogeneous systems need to be: reliable, efficient, secure & accessible.

**Possible solution**

- MAS could provide a solution as they are: distributed, cooperative, “intelligent” & autonomous
- Feedback Control Loops (FCL) could provide the recovery facilities as they:
  - allow self-* by following the MAPE loop
  - can recover from expected and unexpected failures
  - can be easily integrated with MAS

Building a functional platform which is fully distributed in terms of communication, storage, scheduling and negotiation solutions and also self healing is still an open technical and research challenge

**Proposed solution overview**

Adaptive inter-provider MAS scheduling platform:

- **modular agents** ⇒ distributed and custom built agents;
- **two agent types**: scheduling & self-healing;
- agents are designed as intelligent FCL;
- distributed storage & communication: HDFS, HBase, RabbitMQ.
Platform Architecture

Platform Overview - Architecture

Figure: Self-adaptive MAS scheduling platform: high level architecture

**Platform Feedback Control Loops**

**Platform Overview - FCLs**

![Feedback Loop Diagram for Task Rescheduling Process](image1)

**Figure:** The feedback loop for the task rescheduling process

![Feedback Loop Diagram for Module Recovery Process](image2)

**Figure:** The feedback loop for the module recovery process
Platform Tests

Platform Recovery Tests

Figure: Recovery time vs. number of failed modules in the platform
The meta-heuristics and machine learning proved to be viable approaches in designing schedulers flexible enough to deal with the particularities of heterogeneous computational environments.

Online dynamic selection of the “best” scheduling policy can be used to improve the schedule when dynamic environments are considered.

Self-healing was applied to both platform recovery and rescheduling.
Thank you

Any Questions?