Reinforcement learning for grid resource allocation

J. Perez, B. Kégl, C. Germain-Renaud

Université Paris-Sud, LRI, TAO Team

October 19, 2007

Outline

Outline

- Motivations
- State of the art
- Formalism
- Grid implementation and first results
- Perspectives

Grid scheduling

Soft real-time systems

- At arrival, jobs receives some **utility** as a **function of its completion time**.
- **Scheduling objective**: Maximizing the total **Utility Accrual** of the system.

Non-adaptive algorithm of scheduling soft real-time systems

- Based on the idea that the future is **unpredictable**.
- **Greedy strategy**: Scheduling as many high utility jobs as early as possible[1]

[1] E.D. Jensen, 1985, **A time driven scheduling model for real-time operating systems**

Efficient scheduling policy for grid infrastructures

Design issues

- **High-level scheduling goals** formulation
- **Fair share** modeling

Technical issues

- Fault tolerance
- Arrival rate **fluctuations**
- Partial perception of the environment

Outline

State of the art (1/2)

Model-based approaches

- Simple to sophisticated **queuing models** to predict the performance of the ressources[2].
- Based on richness of information obtained through online measurement[3].

Model-free approaches

- Build a **relationship** between state of the environment (e.g. *grid*), available actions, (e.g. *jobs to schedule*) and expected long term reward, (e.g.*utility*) [4].
- **Reinforcement learning formalism**.

---

[2] R.Doyle and al, 2003,**Model-based resource provisioning in a web service utility**

[3] D. Vengerov, 2005,**A reinforcement learning framework for utility-based scheduling in resource-constrained systems**

[4] G. Tesauro, 2005, **Model-Based and Model-Free Approaches to Autonomic Resource Allocation**

State of the art (2/2)

Propositions for grid scheduling

- **Goal** : Maximizing the **productivity**, the average utility of completed jobs per unit of time.
- **Means**: Approximation of a **value function** that gives **expected long-term productivity** of each machine as a function of its current state.
- **Scheduling decisions**: Modifying the existing state of each machine with the goal of **increasing** its value function.

Outline

Motivations

State of the art

Formalism

Grid implementation and first results

Perspectives

Markov Decision Process (1/3)

Definition

- Set of states, S
- Set of actions, A
- Transition probabilities, $P^a_{ss'} = P\{s_{t+1} = s' | s_t = s, a_t = a\}$
- Reward function, $R^a_{ss'} : S \times A(s) \times S^+ \rightarrow \mathbb{R}$

Goal

Finding **a stationary policy** $\pi : S \rightarrow A$ that maximizes the **long-term sum of rewards**.

Markov Decision Process (2/3)

Key idea

- Use of **value function** $V^\pi(s)$ to organize and structure the search for good policy.

$$
\begin{aligned}
V^\pi(s) &= E_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... | s_t = s] \\
&= E_\pi[r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a] \\
&= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]
\end{aligned}
$$

Markov Decision Process (3/3)

If the environment's dynamic (**P** and **R**) is **known**

- V is a system of $|S|$ equations.
- In principle, its solution is a straightforward computation, but **tedious**.
- In practice, **iterative** methods are most suitable: **Dynamic Programming**

If the environment's dynamic (**P** and **R**) is **unknown**

- V is learnt by *interactions*: **Reinforcement Learning**

Grid scheduling problem

Grid state representation

- Avg utility expected for currently running jobs
- Remaining times before any running job is completed
- Number of currently idle CPUs
- Workload of the queue

Scheduling action

- Waiting jobs placed in the queues
  - Expected utility
  - Ressource requirements
  - Execution time

Utility reward

- Sum of unit-utility function

Reinforcement Learning paradigm (1/4)

Description

- Collection of algorithms that can be used to compute optimal policy **without any knowledge of the environment** :
  - Transition probabilities : $P^a_{ss'} = P\{s_{t+1} = s' | s_t = s, a = a_t\}$
  - Reward function : $R^a_{ss'} = S \times A(s) \times S^+ \rightarrow \mathbb{R}$
- Necessary when the number of states/actions is **too large** for an exhaustive listing.
- Necessity of making a **trade off** between **exploration/exploitation** during optimal policy search.

Reinforcement Learning paradigm (2/4)

Off-policy learning algorithms

- Learning an optimal policy by **exploring/sampling the environment** using a **behaviour policy**.
- Useful in **small** and easy-to-sample environments with strong **stationarity hypothesis**.

On-policy learning algorithms

- Learning a **unique** policy that explores/samples the environment while being improved.
- Useful for environment dynamics (**P** and **R**) **without strong stationarity hypothesis**.
- Necessary if the training period is subject to **minimal QoS** and the policy must be **directly usable**.

Reinforcement Learning paradigm (3/4)

Key ideas

- Use of **action-value function** $Q^\pi(s,a)$ to organize and structure the search for good policy.

$$
\begin{aligned}
Q^\pi(s,a) &= E_\pi\{R_t | s_t = s, a_t = a\} \\
&= E_\pi\{\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\}
\end{aligned}
$$

Reinforcement Learning paradigm (4/4)

SARSA: On-policy Temporal-Difference Control Learning

Require: $Q(s, a) \leftarrow$ *arbitrarily*

```
repeat
    Initialize s
    Choose action a in s using policy derived from Q
    repeat
        Take action a, observe r, s'
        Choose a' from A(s') using policy derived from Q
        Q(s,a) ← Q(s,a) + α[r + γQ(s',a') − Q(s,a)]
        s ← s'; a ← a'
    until s is terminal
until forever
```

Time utility function (1/2)

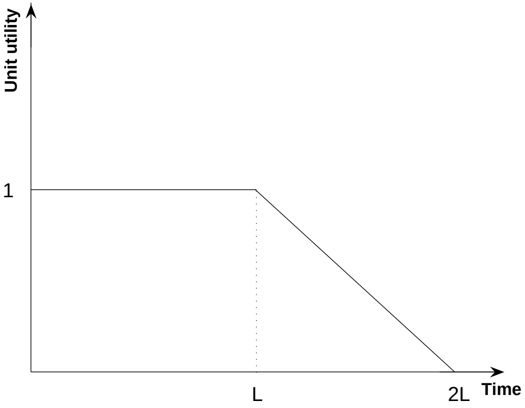

Figure: Example of Time Utility function used by the jobs, where L is the ideal execution time.

Time utility function (2/2)

Modulation factors

- Initial utility value, $\in [0, 1]$
- Ideal Execution Time, L
- Descent form and associated coefs. :
  - **Linear**
  - **Negative Exponential**

Outline

Simulation plateform (1/2)

Experimentation environment

- Developed in Matlab
- Multi-CPUs
- Simple/Multi Queues
- Job description, Utility functions
- Performance/Utility measures

Baseline algorithms

- Earliest Deadline First
- First In First Out
- Random

Simulation plateform (2/2)

Policy learning algorithms

- SARSA: on-policy temporal-difference learning

Generalization methods

- K-Nearest Neighboors
- Neural Networks
- Gaussian Process regression

First results

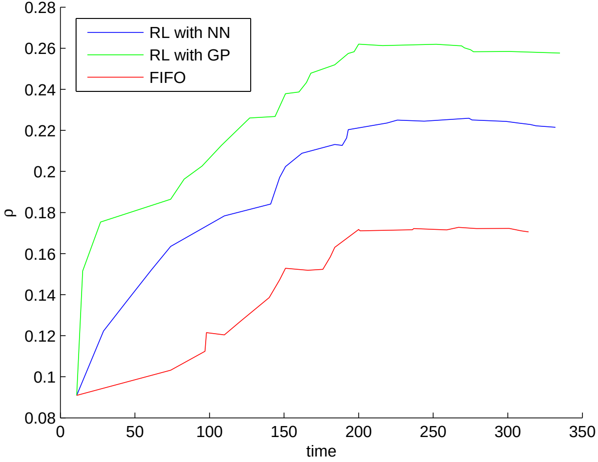

Figure: Avg reward for 100 jobs on 20 CPUs.

Outline

Perspectives

Grid perspectives

- Improving grid state description
- Implementation in a grid infrastructure

Learning perspectives

- New generalization algorithms:
  - Deep Belief Network
  - Echo State Machine
- Multi-objective reinforcement learning
- Distributed reinforcement learning