# AI-Linux

University of Glasgow | School of Engineering

Nathan Loretan - 2348246
MSc Computer System Engineering

Supervisor: Dr. Nicholas Bailey

## Implementing an Artificial Intelligence with the kernel of an Operating System

## Introduction

**What**: Modifying **Linux kernel** to implement an **Artificial Intelligence**
**Why**: Learning how to **distribute the resources** (CPU, memory)
**How**: Designing artificial agents
**Where**: Process scheduling – Process migration – Page reclaiming

## Learning

**Environment**: Partially observable, Stochastic, Dynamic, Episodic, Discrete, Multi-agent, Unknown

- The type of processes running is unknown
- The memory needed is unknown
- The number of CPU available is unknown
- The number of processes running not constant

**Approach**: Reinforcement Learning – *Markov Decision Process*
Q-Learning $Q(s,a)$ – *How good is it to do action $a$ in state $s$?*
Value Function Approx.– *Q based on weighted $\boldsymbol{w}$ features $\boldsymbol{x}$*

$$\hat{Q}(s,a) = \boldsymbol{x}(\boldsymbol{s})^T\boldsymbol{w} \qquad \boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha[R(s,a) + \gamma \max_{a'} Q(s',a') - \hat{Q}(s,a)]\frac{d\hat{Q}(s,a)}{d\boldsymbol{w}}$$

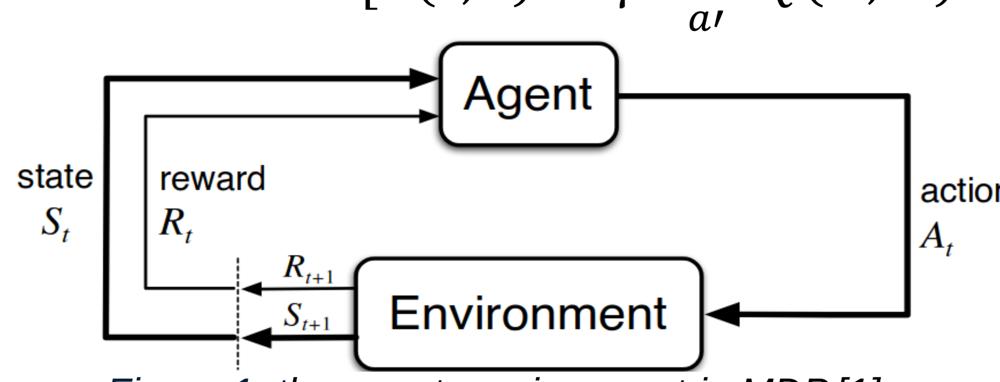



*Figure 1: the agent–environment in MDP.[1]*

## Design

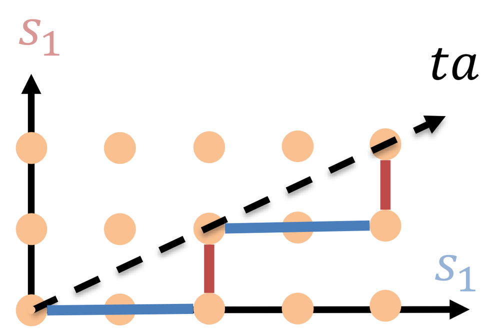
### Smartly Fair Scheduler – SFS

**Inspired by**: Real-time Scheduler with Reinforcement Learning [2]
**Idea**: Learn an **Utilization Target** (***ta***) – means perfect fairness

**Features $\boldsymbol{t_i}$** to estimate *ta:*
- processes' priority
- avg. nbr. of locks acquired
- avg. nbr. if times blocked
- avg. nbr. of times sleeping

| Process | State | Priority |
|---|---|---|
| P1 | s1 | 2 |
| P2 | s2 | 1 |

timeslice




**State**: $ith$ process's runtime $s_i$ with $s = \sum_i s_i$

**Q**: $\hat{Q}(s,a_i) = |s_i - ta_i \cdot s| - |s'_i - ta_i \cdot s'|$ with $ta_i = \dfrac{\boldsymbol{t_i}^T\boldsymbol{w}}{\sum_j \boldsymbol{t_j}^T\boldsymbol{w}}$

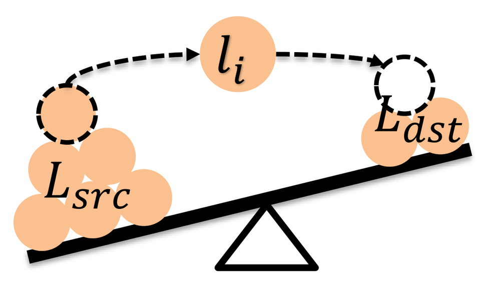
**Action**: $a = \operatorname{argmax}_{a_i} \hat{Q}(s,a_i)$ that schedules the $ith$ process

**Reward**: $R(s,a_i) = \sum_{j \neq i} |s_i p_j - s_j p_i| - \sum_{j \neq i} |s'_i p_j - s'_j p_i|$ with $p$ the priority

### Smart Balance – SB

**Inspired by**: Operating System scheduling on Heterogeneous Core systems [3]
**Idea**: Learn how to estimate the **Load** (***L***) of a CPU

**Features $\boldsymbol{t_i}$** to estimate *L*:
- avg. delta execution
- avg. nbr. of times scheduled
- avg. nbr. of times blocked $avg_b$
- avg. nbr. of completion $avg_c$




**State**: state of $ith$ process $\boldsymbol{t_i}$ and $yth$ CPU $\boldsymbol{c_y} = \sum_i \boldsymbol{t_i}$ with $s = \{c_1, \dots, c_k\}$

**Q**: $\hat{Q}(s,a_i) = \begin{cases} L_{dst} & \textit{keep it} \\ L_{src} + l_i & \textit{move it} \end{cases}$ with $l_i = \boldsymbol{t_i}^T\boldsymbol{w}$ and $L_y = \sum_i l_i$

**Action**: $a = \operatorname{argmax}_{a_i} \hat{Q}(s,a_i)$ that keeps the $ith$ process or moves it

**Reward**: $R(s,a_i) = (avg'_c - avg'_b) - (avg_c - avg_b)$

## Smart Page Frame Reclaiming Algorithm – SPFRA

**Idea:** learn how to estimate the **utility** of a page

**Features** to estimate page's ***utility:***
- nbr. of pages to reclaim
- time since the last access
- avg. time between each access to a page $avg_t$

(first two: for page queue $\boldsymbol{q}$; last two: for page $\boldsymbol{p_i}$)

**State**: state of $ith$ page $\boldsymbol{p_i}$ and queue $\boldsymbol{q}$ with $\boldsymbol{p} = \sum_i \boldsymbol{p_i}$ and $s = \{\boldsymbol{p}, \boldsymbol{q}\}$

**Q**: $\hat{Q}(s,a_i) = \begin{cases} \boldsymbol{p}^T\boldsymbol{w_p} - \boldsymbol{q}^T\boldsymbol{w_q} & \textit{keep it} \\ (\boldsymbol{p} - \boldsymbol{p_i})^T\boldsymbol{w_p} - \boldsymbol{q}^T\boldsymbol{w_q} & \textit{swap it out} \end{cases}$

**Action**: $a = \operatorname{argmax}_{a_i} \hat{Q}(s,a_i)$ that swaps the $ith$ page out or keeps it

**Reward**: $R(s,a_i) = \begin{cases} avg_t - t & \textit{keep it} \\ -(avg_t - t) & \textit{swap it out} \end{cases}$ with $t$ time since reclaiming

## Tests and Results

**Focus**: Behavior of the agents – Can they learn ?
**OS**: Debian (generated with Debootstrap)
**Virtualizer**: Qemu
**Test cases**:
- CPU-Bound processes
- I/O-Bound processes
- Processes blocking each others

Only the process scheduler and process migration are tested.

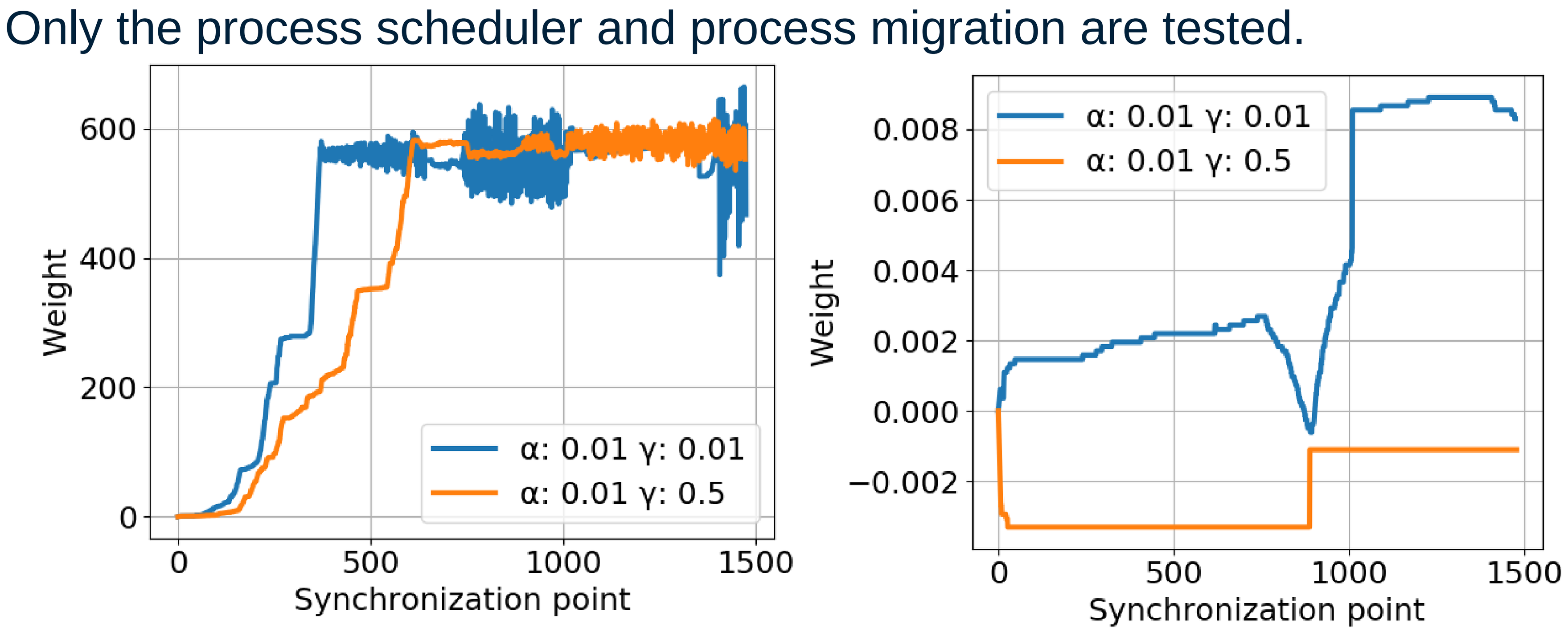



*Figure 2: SFS with CPU-Bound processes, priority and avg. nbr. of times sleeping weights*

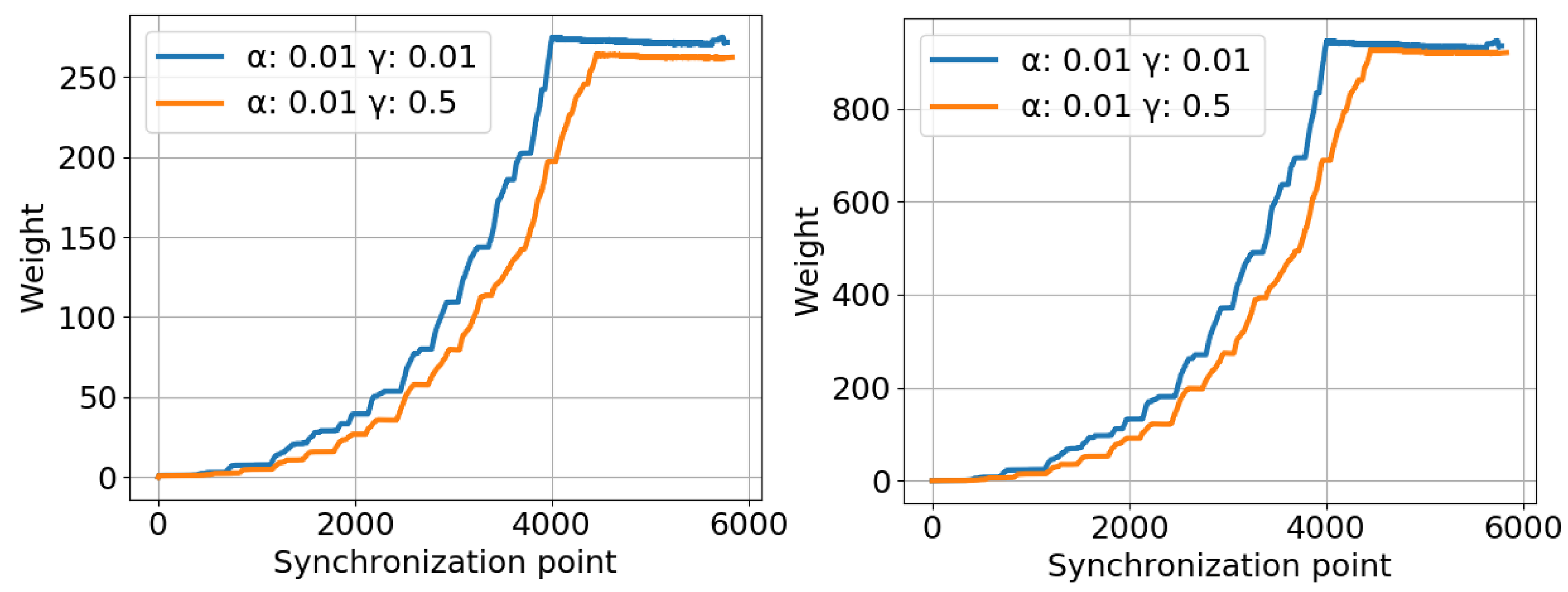



*Figure 3: SFS with I/O-Bound processes, priority and avg. nbr. of times sleeping weights*

## Conclusion

**Results**:
- The agents can learn regarding the different test cases
- Some difficulties in choosing learning and discount factors

**Further work**:
- Testing SPFRA
- Evaluating the performances
- Optimizing the code
- Designing agents for other parts of a kernel

The **final idea** on which is based this project is to **create a complete adaptable operating system** regarding its environment where **each part is implemented with an agent** that work together with the others.

## References

[1] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction.2nd Edition. MIT Press, 2017
[2] Christopher Gill Robert Glaubius Terry Tidwell and William D. Smart. "Real-Time Scheduling via Reinforcement Learning".
[3] David Vengerov Alexandra Fedorova and Daniel Doucette. "Operating system scheduling on heterogeneous core systems".