Reinforcement Learning in the Load Balancing Problem for the iFDAQ of the COMPASS Experiment at CERN

Ondřej Šubrt[1], Martin Bodlák[2], Matouš Jandek[1], Vladimír Jarý[1], Antonín Květoň[2], Josef Nový[1], Miroslav Virius[1] and Martin Zemko[1]

[1]*Czech Technical University in Prague, Prague, Czech Republic*

[2]*Charles University, Prague, Czech Republic*

Keywords: Data Acquisition System, Artificial Intelligence, Reinforcement Learning, Load Balancing, Optimization.

Abstract: Currently, modern experiments in high energy physics impose great demands on the reliability, efficiency, and data rate of Data Acquisition Systems (DAQ). The paper deals with the Load Balancing (LB) problem of the intelligent, FPGA-based Data Acquisition System (iFDAQ) of the COMPASS experiment at CERN and presents a methodology applied in finding optimal solution. Machine learning approaches, seen as a subfield of artificial intelligence, have become crucial for many well-known optimization problems in recent years. Therefore, algorithms based on machine learning are worth investigating with respect to the LB problem. Reinforcement learning (RL) represents a machine learning search technique using an agent interacting with an environment so as to maximize certain notion of cumulative reward. In terms of RL, the LB problem is considered as a multi-stage decision making problem. Thus, the RL proposal consists of a learning algorithm using an adaptive ε–greedy strategy and a policy retrieval algorithm building a comprehensive search framework. Finally, the performance of the proposed RL approach is examined on two LB test cases and compared with other LB solution methods.

1 INTRODUCTION

In 2014, the COMPASS (COmmon Muon Proton Apparatus for Structure and Spectroscopy) (Alexakhin et al., 2010) experiment at the Super Proton Synchrotron (SPS) at CERN commissioned a novel, intelligent, FPGA-based Data Acquisition System (iFDAQ) (Bodlak et al., 2016; Bodlak et al., 2014) in which event building is exclusively performed. Since subevents are assembled in the FPGA cards (multiplexers (MUXes)) from ingoing data streams, these data streams must be properly allocated to six MUXes (up to eight in full setup) in order for the load to be well-balanced in the system. Then complete events are assembled from subevents in a specialized FPGA card fulfilling the role of a switch. Finally, four (up to eight in full setup) read out engine computers equipped with spillbuffer cards readout assembled events and transfer them to CERN permanent storage (CASTOR) (CERN, 2019).

This paper is organized as follows. Firstly, the Load Balancing (LB) problem is given in Section 2.

Secondly, Section 3 gives a description of Reinforcement Learning (RL). In Subsection 3.1, the LB problem is presented as a multi-stage decision making problem. Subsection 3.2 defines the learning algorithm and Subsection 3.3 shows the policy retrieval algorithm. Both algorithms are combined in Subsection 3.4, finalizing the RL search framework.

Finally, the numerical results based on RL are shown in Section 4 and are compared with other LB solution methods.

2 LOAD BALANCING PROBLEM

For the iFDAQ, the most challenging task from the LB point of view is load balancing at the multiplexer (MUX) level. The optimization criterion is minimization of the difference between the output flows of the individual multiplexers. This minimization is achieved by remapping the connection of inputs to input ports of the multiplexers. Each input port establishes a connection between a data source (a detector or a data concentrator) and the MUX level. For the COMPASS experiment, it is necessary to consider flows varying from 0 B to 10 kB for each input port.

In Figure 1, a visualization of LB at the MUX

Šubrt, O., Bodlák, M., Jandek, M., Jarý, V., Květoň, A., Nový, J., Virius, M. and Zemko, M.
Reinforcement Learning in the Load Balancing Problem for the iFDAQ of the COMPASS Experiment at CERN.
DOI: 10.5220/0009035107340741
In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020) - Volume 2*, pages 734-741
ISBN: 978-989-758-395-7; ISSN: 2184-433X

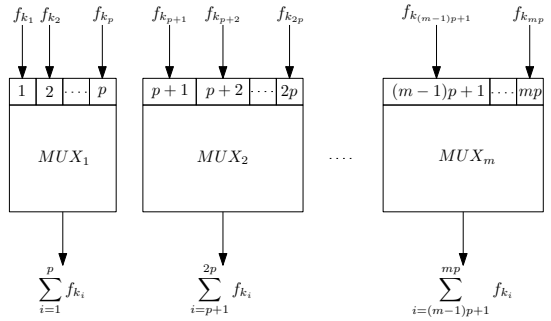

Figure 1: Visualization of LB at the MUX level.

level is given. There are m MUXes with p ingoing ports each. Moreover, $n \in \mathbb{N}$ flows $f_{k_1}, f_{k_2}, \ldots, f_{k_{mp}} \in \mathbb{N}_0$, where $n = m \cdot p$, are shown in the figure with indices $k_1, k_2, \ldots, k_{mp} \in \{i \mid 1 \leq i \leq n\} \wedge \forall i, j : k_i \neq k_j$.

Despite the fact that each flow varies from 0 B to 10 kB in the COMPASS experiment, the domain $\mathbb{N}_0$ is used. The motivation comes from a general approach to LB. Moreover, a flow with 0 B can be either a physical connected input port sending no data or an empty input port where no data source is connected to. In brief, there are always $n = m \cdot p$ flows regardless whether all ports are used or not.

2.1 PROBLEM FORMULATION

Firstly, this subsection deals with a proper definition of the LB problem.

Definition 1. *Let $m \in \mathbb{N}$ denote the number of MUXes with $p \in \mathbb{N}$ ingoing ports each, i.e., $n = m \cdot p \in \mathbb{N}$ ingoing ports in total and flows $f_1, f_2, \ldots, f_n \in \mathbb{N}_0$. Let $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ be subsets of indices and $F = \left\lceil \sum_{i=1}^{n} f_i / m \right\rceil$ be a theoretical average flow for one MUX. The Load Balancing (LB) problem is an optimization problem such that:*

To minimize

$$\sqrt{\sum_{i=1}^{m} \left(F - \sum_{j \in \mathcal{S}_i} f_j \right)^2}, \tag{1}$$

subject to the constraints

- *each flow must be allocated*

$$\bigcup_{i=1}^{m} \mathcal{S}_i = \{i \mid i \in 1, \ldots, n\} \tag{2}$$

- *each flow must be allocated at most once*

$$\mathcal{S}_i \cap \mathcal{S}_j = \emptyset \quad \forall i, j = 1, \ldots, m \wedge i \neq j \tag{3}$$

- *each MUX has p ports*

$$|\mathcal{S}_i| = p \quad \forall i = 1, \ldots, m \tag{4}$$

Secondly, a formulation of the index function is given being helpful for the RL algorithms description.

Definition 2. *Let $\mathcal{S} = \{a_{k_1}, a_{k_2}, \ldots, a_{k_n}\}$ be a set of elements with indices $k_1, k_2, \ldots, k_n \in \{i \mid 1 \leq i \leq n\} \wedge \forall i, j : k_i \neq k_j$. Function $\varphi(i, \mathcal{S})$ is the index function such that*

$$\varphi(i, \mathcal{S}) = k_i \quad \forall i = 1, \ldots, n. \tag{5}$$

3 REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a study of how animals and artificial systems can learn to optimize their behaviour in the face of rewards and punishments (Sutton and Barto, 1998; Szepesvari, 2010; Lapan, 2018). One way in which animals acquire complex behaviours is by learning to obtain rewards and to avoid punishments. During this learning process, the agent interacts with the environment. At each step of interaction, on observing or feeling the current state, an action is taken by the learner (agent). Depending on the goodness of the action at the particular situation, it is tried at the next stage when the same or similar situation arises (Powell, 2007; Borrelli et al., 2017; Busoniu et al., 2010). Finally, the best action at each state or the best policy is manipulated based on the observed rewards.

3.1 Load Balancing as a Multi-stage Decision Making Problem

In order to view the LB problem as a multi-stage decision making problem, the various stages of the problem are to be identified. Consider a system with $n \in \mathbb{N}$ flows $f_1, f_2, \ldots, f_n \in \mathbb{N}_0$ committed for allocation to $n = m \cdot p$ ports. Then the LB problem involves selecting $p \in \mathbb{N}$ flows to be allocated to the first MUX from flows $\mathcal{R}_1 = \{f_i \mid i \in \{1, \ldots, n\}\}$, i.e., determined by subset $\mathcal{S}_1$. For the second MUX, p flows are selected from flows $\mathcal{R}_2 = \{f_i \mid i \in \{1, \ldots, n\}\} \backslash \mathcal{S}_1\}$ and described by $\mathcal{S}_2$. The last, i.e., the m-th MUX is occupied by remaining p flows $\mathcal{R}_m = \{f_i \mid i \in \{1, \ldots, n\} \backslash \bigcup_{j=1}^{m-1} \mathcal{S}_j = \mathcal{S}_m\}$ and in fact, there is no selection procedure at all and the subset $\mathcal{S}_m$ is determined directly. In general, the i-th MUX selects p flows from flows $\mathcal{R}_i = \{f_j \mid j \in \{1, \ldots, n\} \backslash \bigcup_{k=1}^{i-1} \mathcal{S}_k\}$ and a subset $\mathcal{S}_i$ contains flow indices of the i-th MUX.

The problem statement follows. Initially, there are p flows to be allocated in the i-th MUX chosen from $n - (i-1)p$ flows. In this formulation, a flow to be

allocated at $stage_k$ is denoted as F_k^{A} and is based on an action a_k. In RL terminology, the action a_k corresponds to a flow allocation either $f_{\varphi(k,\mathcal{R}_i)}$ or 0 to the i-th MUX at $stage_k$, i.e.,

$$F_k^{\mathrm{A}} = \begin{cases} 0 & \text{if} \quad a_k = 0 \\ f_{\varphi(k,\mathcal{R}_i)} & \text{if} \quad a_k = 1. \end{cases} \tag{6}$$

Therefore, the action set $\mathcal{A}_k$ consists of either 2 possibilities (allocate $a_k = 1$ and do not allocate $a_k = 0$) or 1 possibility (allocate $a_k = 1$ or do not allocate $a_k = 0$) at $stage_k$. That is,

$$\mathcal{A}_k = \{a_k^{\min}, \ldots, a_k^{\max}\}, \tag{7}$$

$a_k^{\min}$ being the minimum possible action at $stage_k$ and $a_k^{\max}$ being the maximum possible action at $stage_k$. Values of $a_k^{\min}$ and $a_k^{\max}$ depend on the total flow which has already been allocated at the previous $k-1$ stages, the number of flows already allocated, a flow at the k-th stage and flows that can be allocated at the remaining $(n-(i-1)p)-k$ stages.

The initial state is denoted as $stage_1$. At $stage_1$, a decision is made on whether a flow $f_{\varphi(1,\mathcal{R}_i)}$ is allocated or not. This action is denoted as a_1 and corresponds to F_1^{A} allocation at $stage_1$.

Upon having made this decision, $stage_2$ is reached. The expression $(F_1^{\mathrm{T}} + F_1^{\mathrm{A}})$ represents the total flow which has already been allocated at the previous stages, i.e., $stage_1$. At $stage_2$, a decision a_2 is made on whether a flow $f_{\varphi(2,\mathcal{R}_i)}$ allocates or does not allocate. Generally at $stage_k$, a decision is made on whether a flow f_k is allocated or not. Finally, $stage_{n-(i-1)p}$ is reached and a decision $a_{n-(i-1)p}$ is made on whether a flow $f_{\varphi(n-(i-1)p,\mathcal{R}_i)}$ is allocated or not.

Each state at any $stage_k$ can be defined as a tuple (k, F_k^{T}) where k is the stage number and F_k^{T} is the total flow which has already been allocated at the previous $k-1$ stages.

Thus for $k = 1$, the state information is denoted as $(1, F_1^{\mathrm{T}})$ where F_1^{T} is equal to 0 since no decision concerning any flow allocation has been made so far. The algorithm for the LB problem selects one among the permissible set of actions and either allocates or does not allocate a flow $f_{\varphi(1,\mathcal{R}_i)}$ at $stage_1$ so that it reaches the next stage $k = 2$ with the total flow already allocated and $(n-(i-1)p)-1$ flows for an allocation decision. Transition from $(1, F_1^{\mathrm{T}})$ on performing an action $a_1 \in \mathcal{A}_1$ results in the next state reached as $(2, F_2^{\mathrm{T}})$, where

$$F_2^{\mathrm{T}} = F_1^{\mathrm{T}} + F_1^{\mathrm{A}}. \tag{8}$$

Generally at $stage_k$, from a state x_k on performing an action a_k reaches a state x_{k+1}, i.e., state transition is from (k, F_k^{T}) to $(k+1, F_{k+1}^{\mathrm{T}})$, where

$$F_{k+1}^{\mathrm{T}} = F_k^{\mathrm{T}} + F_k^{\mathrm{A}}. \tag{9}$$

This repeats until the last stage. Therefore, a state transition can be denoted as

$$x_{k+1} = f(x_k, a_k), \tag{10}$$

where $f(x_k, a_k)$ is the function of state transition defined by Equation 9.

Thus, the algorithm for the LB problem can be treated as one of finding an optimum mapping from the state space $\mathcal{X}$ to the action space $\mathcal{A}$. The algorithm design for the LB problem is finding or learning a good or optimal policy (flows allocation) which is the optimum allocation at each stage. Such allocation can be treated as elements of an optimum policy π^*. For finding the cost of allocation, it cumulates the costs at each of the $n-(i-1)p$ stages of the problem. These costs can be treated as a reward for performance of an action in the perspective of the LB problem. The cost of generation on following a policy π can be treated as a measure of goodness of that policy. The Q–learning technique is employed to cumulate costs and thus find out the optimum policy.

For updating the Q value associated with the different state–action pairs, one should cumulate the total reward at different stages of allocation. In the LB problem, the reward function $g(x_k, a_k, x_{k+1})$ can be chosen as $-F_k^{\mathrm{A}}$ at $stage_k$. The rewards are negative since Q–learning is considered as a minimization problem. In the RL terminology, the immediate reward is

$$r_k = g(x_k, a_k, x_{k+1}). \tag{11}$$

Since the aim is to allocate as large as possible a total flow, the estimated Q values of the state–action pair are modified at each step of learning as

$$\begin{aligned} Q^{s+1}(x_k, a_k) = Q^s(x_k, a_k) + \alpha[g(x_k, a_k, x_{k+1}) \\ + \gamma \min_{a' \in \mathcal{A}_{k+1}} Q^s(x_{k+1}, a') - Q^s(x_k, a_k)]. \end{aligned} \tag{12}$$

Here, α is the learning parameter and γ is the discount factor. When the system comes to the last stage of decision making, there is no need of accounting the future effects and then the estimate of Q value is updated using the equation

$$\begin{aligned} Q^{s+1}(x_k, a_k) = Q^s(x_k, a_k) + \alpha[g(x_k, a_k, x_{k+1}) \\ - Q^s(x_k, a_k)]. \end{aligned} \tag{13}$$

For finding an optimum policy, a learning algorithm is designed. It iterates through each of the $n-(i-1)p$ stages at each step of learning. As the learning steps are carried out a sufficient number of times, the estimated Q values of state–action pairs will approach the optimum so that the optimum policy $\pi^*(x)$ corresponding to any state x can be easily retrieved.

3.2 RL Algorithm for the LB Problem using ε–greedy Strategy

In the previous subsection, the LB problem is formulated as a multi-stage decision making problem. To find the best policy or the best action corresponding to each state, the RL technique is used. The solution consists of two phases, namely the learning phase and the policy retrieval phase.

To carry out the learning task, one issue concerns how to select an action from the action space. In this subsection, the ε–greedy strategy of exploring action space is used.

Instead of choosing all actions several times, it makes sense to choose the actions which may be the best action. The greedy action is chosen with a probability $(1-\varepsilon)$ and one of the other actions with a probability ε. The greedy action a_k^{g} corresponds to the action with the best estimate of the Q value at $stage_k$, i.e.,

$$a_k^{\mathrm{g}} = \arg\min_{a \in \mathcal{A}_k} Q^s(a). \tag{14}$$

It may be noted that if $\varepsilon = 1$, the algorithm will select one of the actions with uniform probability and if $\varepsilon = 0$, the greedy action will be selected. Initially, the estimates $Q^s(a)$ may be far from their true value. However as $s \to \infty$, $Q^s(a) \to Q(a)$, the information contained in $Q^s(a)$ becomes increasingly exploitable. So in the ε–greedy algorithm, initially ε is chosen close to 1 and as s increases, ε is gradually reduced. Finally, proper balancing of exploration and exploitation of the action space ultimately reduces the number of trials needed to find the best action.

For solving this multi-stage problem using RL, the first step is fixing of state space $\mathcal{X}$ and action space $\mathcal{A}$ precisely. The whole concept for the i-th MUX is explained in a general way, where the number of flows is $n-(i-1)p$.

The fixing of state space $\mathcal{X}$ primarily depends on the number of flows and the possible values of the total flow in the i-th MUX (which in turn directly depends on the minimum and maximum values of each flow). Since there are $n-(i-1)p$ stages for solution of the problem, the state space is also divided into $n-(i-1)p$ subspaces. Thus, if there are $n-(i-1)p$ flows to be allocated, then

$$\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \ldots \cup \mathcal{X}_{n-(i-1)p}. \tag{15}$$

The allocation problem should go through $n-(i-1)p$ stages for making decision to allocate or not to allocate for each of the $n-(i-1)p$ flows. At any $stage_k$, the part of state space to be considered $\mathcal{X}_k$ consists of the different tuples having the stage number as k and the total flow already allocated varying from F_k^{Tmin} to F_k^{Tmax}, where F_k^{Tmin} is the minimum possible total flow already allocated and F_k^{Tmax} the maximum possible total flow already allocated at the previous $k-1$ stages. Thus,

$$\mathcal{X}_k = \{(k, F_k^{\mathrm{Tmin}}), \ldots, (k, F_k^{\mathrm{Tmax}})\}, \tag{16}$$

where F_k^{Tmin} is the minimum possible total flow already allocated at the previous $k-1$ stages, i.e.,

$$F_k^{\mathrm{Tmin}} = 0 \tag{17}$$

and F_k^{Tmax} is the maximum possible total flow already allocated at the previous $k-1$ stages, i.e.,

$$F_k^{\mathrm{Tmax}} = \sum_{j=1}^{k-1} f_{\varphi(j,\mathcal{R}_i)}. \tag{18}$$

At each step, the LB problem algorithm will select an action from the permissible set of actions and forward the system to one among the next permissible states. Therefore, the action set $\mathcal{A}_k$ is a dynamically varying one, depending on the flows already allocated at the previously considered stages. As the number of MUXes or number of ports in each MUX increases, the number of states in the state space increases. Thus, state space and action space are both discrete.

The action set $\mathcal{A}_k$ consists of either 2 possibilities (allocate $a_k = 1$ and do not allocate $a_k = 0$) or 1 possibility (allocate $a_k = 1$ or do not allocate $a_k = 0$) at $stage_k$. At the current state x_k, the action set $\mathcal{A}_k$ depends on the total flow already allocated F_k^{T}, the number of flows already allocated p^{A}, a flow at the k-th stage $f_{\varphi(k,\mathcal{R}_i)}$ and flows that can be allocated at the remaining $(n-(i-1)p)-k$ stages, i.e., $f_{\varphi(k+1,\mathcal{R}_i)}, \ldots, f_{\varphi(n-(i-1)p,\mathcal{R}_i)}$. Therefore, the action set $\mathcal{A}_k$ is dynamic in nature in the sense that it depends on the total flow already allocated up to that stage and also the number of flows already allocated at the previous $k-1$ stages. If F_k^{T} is the total flow already allocated, p^{A} is the number of flows already allocated, $f_{\varphi(k,\mathcal{R}_i)}$ is a flow at $stage_k$, the minimum value and the maximum value of action a_k are defined as

$$a_k^{\min} = \begin{cases} 0 \text{ if } p = p^{\mathrm{A}} \\ 0 \text{ if } F - F_k^{\mathrm{T}} < f_{\varphi(k,\mathcal{R}_i)} \\ 0 \text{ if } p - p^{\mathrm{A}} < (n-(i-1)p)-k \quad \wedge \\ \qquad F - F_k^{\mathrm{T}} < L_{p-p^{\mathrm{A}}-1,(n-(i-1)p)-k} \\ \qquad\qquad + f_{\varphi(k,\mathcal{R}_i)} \\ 0 \text{ if } p - p^{\mathrm{A}} < (n-(i-1)p)-k \quad \wedge \\ \qquad F - F_k^{\mathrm{T}} \geq L_{p-p^{\mathrm{A}},(n-(i-1)p)-k} \\ 1 \text{ otherwise,} \end{cases}$$

$$a_k^{\max} = \begin{cases} 0 \text{ if } p = p^{\mathrm{A}} \\ 0 \text{ if } F - F_k^{\mathrm{T}} < f_{\varphi(k,\mathcal{R}_i)} \\ 0 \text{ if } p - p^{\mathrm{A}} < (n-(i-1)p)-k \quad \wedge \\ \qquad F - F_k^{\mathrm{T}} < L_{p-p^{\mathrm{A}}-1,(n-(i-1)p)-k} \\ \qquad\qquad + f_{\varphi(k,\mathcal{R}_i)} \\ 1 \text{ otherwise,} \end{cases} \tag{19}$$

where $L_{u,v}$ denotes the sum of the u smallest flows at last v stages. Below, Equation 19 is discussed in more detail.

The conditions $F - F_k^{\mathrm{T}} < f_{\varphi(k,\mathcal{R}_i)}$ and $p = p^{\mathrm{A}}$ are common for both $a_k^{\min}$ and $a_k^{\max}$. Clearly, if a flow $f_{\varphi(k,\mathcal{R}_i)}$ at $stage_k$ is greater than $F - F_k^{\mathrm{T}}$ (the rest what remains to allocate), it does not allow the flow $f_{\varphi(k,\mathcal{R}_i)}$ to be allocated at $stage_k$. Similarly, if the number of flows already allocated p^{A} is equal to the number of ports p, there is no free port available for a flow allocation. Therefore, both $a_k^{\min}$ and $a_k^{\max}$ are equal to 0 if the condition is true.

The second condition $p - p^{\mathrm{A}} < (n-(i-1)p) - k \wedge F - F_k^{\mathrm{T}} < L_{p-p^{\mathrm{A}}-1,(n-(i-1)p)-k} + f_{\varphi(k,\mathcal{R}_i)}$ is also common for both $a_k^{\min}$ and $a_k^{\max}$. The condition examines whether the flow $f_{\varphi(k,\mathcal{R}_i)}$ has to be necessarily allocated or not. The first part, $p - p^{\mathrm{A}} < (n-(i-1)p) - k$, checks whether there are enough flows at the remaining $(n-(i-1)p) - k$ stages to fill remaining free ports if the flow $f_{\varphi(k,\mathcal{R}_i)}$ is not to be allocated. The second part, $F - F_k^{\mathrm{T}} < L_{p-p^{\mathrm{A}}-1,(n-(i-1)p)-k} + f_{\varphi(k,\mathcal{R}_i)}$, checks if the flow $f_{\varphi(k,\mathcal{R}_i)}$ together with the sum of the $p - p^{\mathrm{A}} - 1$ smallest flows at the last $(n-(i-1)p) - k$ stages are higher than $F - F_k^{\mathrm{T}}$, ensuring that the total i-th MUX flow is less or equal than F. If both parts are satisfied, both $a_k^{\min}$ and $a_k^{\max}$ are equal to 0.

The last condition deals with $a_k^{\min}$ only. The condition $p - p^{\mathrm{A}} < (n-(i-1)p) - k \wedge F - F_k^{\mathrm{T}} \geq L_{p-p^{\mathrm{A}},(n-(i-1)p)-k}$ consists of two parts. The first part, $p - p^{\mathrm{A}} < (n-(i-1)p) - k$, checks again whether there are enough flows at the remaining $(n-(i-1)p) - k$ stages to fill remaining free ports if the flow $f_{\varphi(k,\mathcal{R}_i)}$ is not to be allocated. The second part, $F - F_k^{\mathrm{T}} \geq L_{p-p^{\mathrm{A}},(n-(i-1)p)-k}$, determines whether the sum of the $p - p^{\mathrm{A}}$ smallest flows at the last $(n-(i-1)p) - k$ stages is less than $F - F_k^{\mathrm{T}}$, ensuring that the total i-th MUX flow is less or equal than F. If both parts are satisfied, the flow $f_{\varphi(k,\mathcal{R}_i)}$ does not necessarily have to be allocated, since all LB problem conditions can be still satisfied at the remaining $(n-(i-1)p) - k$ stages.

Clearly, if none of the above-mentioned conditions is satisfied, both $a_k^{\min}$ and $a_k^{\max}$ are equal to 1 and the flow $f_{\varphi(k,\mathcal{R}_i)}$ has to be allocated in the i-th MUX.

To conclude, if $F - F_k^{\mathrm{T}} < f_{\varphi(k,\mathcal{R}_i)}$ or $p = p^{\mathrm{A}}$ is satisfied, the choice can only be made from one action – do not allocate $a_k = 0$. There is also no choice if $p - p^{\mathrm{A}} < (n-(i-1)p) - k \wedge F - F_k^{\mathrm{T}} < L_{p-p^{\mathrm{A}}-1,(n-(i-1)p)-k} + f_{\varphi(k,\mathcal{R}_i)}$ is true, since it leads to only one possible action again – do not allocate $a_k = 0$.

The situation when two actions are feasible (allocate $a_k = 1$ and do not allocate $a_k = 0$) depends on the condition $p - p^{\mathrm{A}} < (n-(i-1)p) - k \wedge F - F_k^{\mathrm{T}} \geq L_{p-p^{\mathrm{A}},(n-(i-1)p)-k}$. The decision to allocate the flow $f_{\varphi(k,\mathcal{R}_i)}$ can be seen as a substitution of the $(p-p^{\mathrm{A}})$-th smallest flow at last $(n-(i-1)p) - k$ stages for the flow $f_{\varphi(k,\mathcal{R}_i)}$.

All other situations lead to both $a_k^{\min}$ and $a_k^{\max}$ being equal to 1 and the only one possible decision is to allocate the flow $f_{\varphi(k,\mathcal{R}_i)}$ in the i-th MUX.

Algorithm 1: Learning algorithm for the LB problem of the i-th MUX using ε–greedy strategy.

load flows $\mathcal{R}_i$, average flow F
set the learning parameter α, the discount factor γ and the greedy factor ε
set the maximum iteration $s_{\max}$
set Q values to zeros
for $i = 1 \rightarrow n-(i-1)p$ **do** ▷ stages initialization
    $\mathcal{X}_i$ = initStage(i)
    for all $x_k \in \mathcal{X}_i$ **do** ▷ states initialization
        $a_k^{\min}$ = getMinAction() ▷ see Eq. 19
        $a_k^{\max}$ = getMaxAction() ▷ see Eq. 19
        setStatePermissibleActions($x_k, a_k^{\min}, a_k^{\max}$)
    end for
end for
for $s = 1 \rightarrow s_{\max}$ **do** ▷ learning phase
    $F_1^{\mathrm{T}} = 0, p^{\mathrm{A}} = 0$
    for $k = 1 \rightarrow n-(i-1)p$ **do**
        x_k = getCurrentState(k, F_k^{T})
        $\mathcal{A}_k$ = getActions($k, x_k, F, p^{\mathrm{A}}$) ▷ see Eq. 19
        a_k = getGreedyAction($Q, \mathcal{A}_k, \varepsilon$)
        $p^{\mathrm{A}} = p^{\mathrm{A}} + a_k$
        $F_{k+1}^{\mathrm{T}} = F_k^{\mathrm{T}} + f_{\varphi(k,\mathcal{R}_i)} \cdot a_k$
        if $k < n-(i-1)p$ **then**
            Q = updateQ(Q, x_k, a_k) ▷ see Eq. 12
        else
            Q = updateQ(Q, x_k, a_k) ▷ see Eq. 13
        end if
    end for
    set the greedy factor $\varepsilon = 1 - s/s_{\max}$
end for

The learning procedure can now be summarized, see Algorithm 1. Initially, the total flow already allocated is set to 0 at $stage_1$. Then an action is performed that either allocates or not the flow at $stage_1$ and then it proceeds to the next stage ($k = 2$) with the total flow already allocated. This proceeds until all the $n-(i-1)p$ flows are either allocated or not. At each state transition step, the estimated Q value of the state–action pair is updated using Equation 12.

As the learning process reaches the last stage, a flow at $stage_{n-(i-1)p}$ is either allocated or not. Then

the Q value is updated using Equation 13. The transition process is repeated a sufficient number of times (iterations) and each time the allocation process goes through all the $n-(i-1)p$ stages.

Algorithm 2: Policy retrieval algorithm for the LB problem of the i-th MUX using RL.

load flows $\mathcal{R}_i$, average flow F, Q values
$F_1^{\mathrm{T}} = 0, p^{\mathrm{A}} = 0$
$\mathcal{S}_i = \{\}$
for $k = 1 \rightarrow n-(i-1)p$ **do** ▷ retrieval phase
  x_k = getCurrentState(k, F_k^{T})
  $\mathcal{A}_k$ = getActions$(k, x_k, F, p^{\mathrm{A}})$ ▷ see Eq. 19
  $a_k^{\mathrm{g}} = \arg\min_{a \in \mathcal{A}_k} Q(x_k, a)$
  if $a_k^{\mathrm{g}} = 1$ **then**
    $\mathcal{S}_i = \mathcal{S}_i \cup \varphi(k, \mathcal{R}_i)$
  end if
  $p^{\mathrm{A}} = p^{\mathrm{A}} + a_k^{\mathrm{g}}$
  $F_{k+1}^{\mathrm{T}} = F_k^{\mathrm{T}} + f_{\varphi(k,\mathcal{R}_i)} \cdot a_k^{\mathrm{g}}$
end for

3.3 Policy Retrieval

After the learning phase is done, the retrieval phase begins. The system is learnt and the learnt values are stored in a lookup table. The retrieval phase accesses the lookup table in order to retrieve the required results. The learnt values in the lookup table can be used unless the parameters of the system change. In this case, the system must be learnt again by triggering a new run of learning phase.

As the learning proceeds, and the Q values of state-action pairs are updated, Q^s approaches Q^*. Next, the optimum Q^s values are used to obtain the optimum allocation. The retrieval algorithm is summarized in Algorithm 2. At $stage_1$, $F_1^{\mathrm{T}} = 0$ is initialized and thus, the state of the system is $(1, F_1^{\mathrm{T}})$. The algorithm finds the greedy action at this stage as a_1^{g} which is the best allocation F_1^{A} for $stage_1$. The retrieval algorithm reaches the next state as $(2, F_2^{\mathrm{T}})$ where $F_2^{\mathrm{T}} = F_1^{\mathrm{T}} + a_1^{\mathrm{g}} \cdot f_{\varphi(1,\mathcal{R}_i)}$ and finds the greedy action corresponding to $stage_2$ as a_2^{g}. This proceeds up to $stage_{n-(i-1)p}$. Finally, a set of actions (allocations) $a_1^{\mathrm{g}}, a_2^{\mathrm{g}}, \ldots, a_{n-(i-1)p}^{\mathrm{g}}$ is obtained, which is the optimum allocation $F_1^{\mathrm{A}}, F_2^{\mathrm{A}}, \ldots, F_{n-(i-1)p}^{\mathrm{A}}$ for the i-th MUX. Thus, it builds up the subset $\mathcal{S}_i = \{\varphi(j, \mathcal{R}_i) \mid F_j^{\mathrm{A}} \neq 0 \quad \forall j = 1, \ldots, n-(i-1)p\}$.

3.4 Complete Algorithm

Finally, an algorithm considering all $m \in \mathbb{N}$ MUXes is proposed, see Algorithm 3. The idea is to learn and retrieve $\mathcal{S}_1$ for the first MUX, then reduce a set of flows $\mathcal{R}_1$ by the allocated flows and get a set of flows $\mathcal{R}_2$ for the second MUX, etc. Generally, the i-th MUX is learnt, $\mathcal{S}_i$ is retrieved and $\mathcal{R}_i$ is reduced by $\mathcal{S}_i$ giving $\mathcal{R}_{i+1}$ for the $(i+1)$-th MUX. Regarding the last MUX, the m-th MUX, a set $\mathcal{R}_m$ determines directly $\mathcal{S}_m$.

Algorithm 3: The complete LB problem algorithm considering m MUXes using RL.

load flows $\mathcal{R}_1$
for $i = 1 \rightarrow m-1$ **do** ▷ get LB of the i-th MUX
  $F = \left\lceil \sum_{j=1}^{n-(i-1)p} f_{\varphi(j,\mathcal{R}_i)} / (m-i+1) \right\rceil$
  Q = learningPhase$(\mathcal{R}_i, F)$ ▷ see Alg. 1
  $\mathcal{S}_i$ = retrievalPhase$(\mathcal{R}_i, F, Q)$ ▷ see Alg. 2
  $\mathcal{R}_{i+1} = \{f_j \mid j \in \{1, \ldots, n\} \setminus \bigcup_{k=1}^{i} \mathcal{S}_k\}$
end for
$\mathcal{S}_m = \{\varphi(j, \mathcal{R}_m) \mid \forall j = 1, \ldots, p\}$

4 NUMERICAL RESULTS

The RL approach has been implemented in C++ and Matlab (R2018a, 64-bit) on a personal computer equipped with Intel(R) Core(TM) i7-8750H CPU (@2.20 GHz, 6 Cores, 12 Threads, 9M Cache, Turbo Boost up to 4.10 GHz) and 16 GB RAM (DDR4, 2 666 MHz) memory. The RL approach is examined on two test cases and numerical results are compared with other methods of solving the LB problem.

The results are investigated with respect to the error and computation time. The error is defined as the objective function of the LB problem, see Equation 1.

4.1 Test Case 1

The Test Case 1 (TC1) consists of $m = 6$ MUXes with $p = 15$ ingoing ports each. It corresponds to the iFDAQ setup used in the Run 2016, 2017 and 2018. It considers $n = m \cdot p = 6 \cdot 15 = 90$ flows with values randomly generated in the range from 0 B to 10 kB.

The proposed RL algorithm is strongly stochastic and hence, it produces a unique solution in every execution. The parameters used for the RL algorithm to solve the TC1 are given in Table 1. The quality of the results depends on $s_{\max}$ directly controlling how well state–action pairs are learnt represented by Q values. The $s_{\max}$ setup requires experience of an executor.

In Table 2, the results produced in C++ and Matlab for the TC1 using RL in each execution are stated.

Table 1: Parameters used for the RL algorithm.

Parameter	s_{max}	α	γ	ε
Value	100,000	0.1	1	1

Table 2: The TC1 results using the RL approach in each execution.

Ex.	C++		Matlab	
	Error	t [ms]	Error	t [ms]
1	10.44	32,404	10.44	117,253
2	10.44	32,515	10.44	117,252
3	10.44	32,761	10.44	117,106
4	10.44	32,961	10.44	119,307
5	10.44	33,357	10.44	119,113
6	10.44	34,818	10.44	117,690
7	10.44	35,409	10.44	118,236
8	10.44	34,544	10.44	118,427
9	10.44	35,130	10.44	118,242
10	10.44	36,324	10.82	117,918

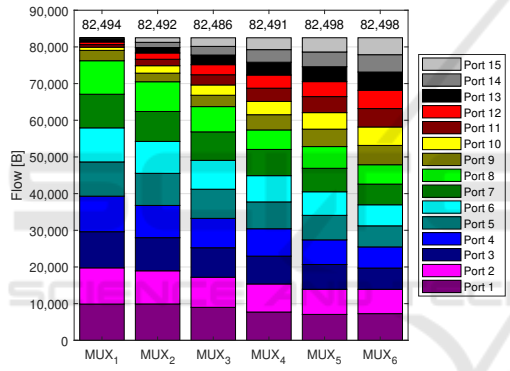

Figure 2: The same best TC1 flow allocation based on RL produced in C++ (Execution 1) and Matlab (Execution 3).

Table 3: Comparison of LB solution methods based on the best flow allocation for the TC1.

Met.	C++		Matlab	
	Error	t [ms]	Error	t [ms]
GH	243.89	47	243.89	1
ILP	21.19	17,127	23.81	1,465
RL	10.44	32,404	10.44	117,106

The error is equal approximately to 10.44 giving a solution close to the global optimum in each execution. In Figure 2, the same best TC1 flow allocation produced in C++ (Execution 1) and Matlab (Execution 3) is shown. Moreover, a comparison of the lowest total flow allocation and highest total flow allocation for the respective MUXes might be performed. The third MUX has the lowest total flow allocation with value of 82,486 B and on the other hand, the fifth and sixth MUX have the highest total flow allocation with a value of 82,498 B each. The ratio of the flows is $82,486/82,498 \approx 99.99\%$.

Table 4: The TC2 results using the RL approach in each execution.

Ex.	C++		Matlab	
	Error	t [ms]	Error	t [ms]
1	11.23	59,566	11.66	206,358
2	7.07	61,716	5.29	203,017
3	11.66	63,979	6.48	203,287
4	11.31	66,219	11.22	204,427
5	11.31	74,745	6.32	207,476
6	6.48	77,605	11.22	208,410
7	7.87	79,134	2.45	204,824
8	11.66	80,781	6.16	205,160
9	2.83	75,882	11.49	204,299
10	11.23	76,138	11.14	204,525

In order to calculate an optimal solution, Matlab consumes several times more computational time than a version implemented in C++. The high computational time for Matlab comes from a type of operation required in an optimization process. In RL, the main mathematical operations are performed on submatices of matrices, e.g., sum of elements, minimum or maximum from elements of a submatrix. In C++, pointers are a very efficient way how to implement such mathematical operations. Since pointers are absent in Matlab, a submatrix must be always copied to perform a particular mathematical operation.

However, the computational time for both C++ and Matlab is quite high, resulting in an exclusion of the RL algorithm as a real-time LB solver. On the other hand, the error is quite small. Therefore, the RL approach can be considered for a long-term LB setup, where no frequent changes in the flows are expected.

The proposed RL algorithm is compared with other LB solution methods – Greedy Heuristic (GH) and Integer Linear Programming (ILP) – in Table 3.

4.2 Test Case 2

The Test Case 2 (TC2) consists of $m = 8$ MUXes with $p = 15$ ingoing ports each and thus, it corresponds to the iFDAQ full setup. However, the iFDAQ full setup has never been in operation for the COMPASS experiment since it was not required by any physics program. It considers $n = m \cdot p = 8 \cdot 15 = 120$ flows with values randomly generated in the range from 0 B to 10 kB. The parameters used for the RL algorithm to solve the TC2 are the same as for TC1, see Table 1.

In Table 4, the results produced in C++ and Matlab for the TC2 using RL in each execution are stated.

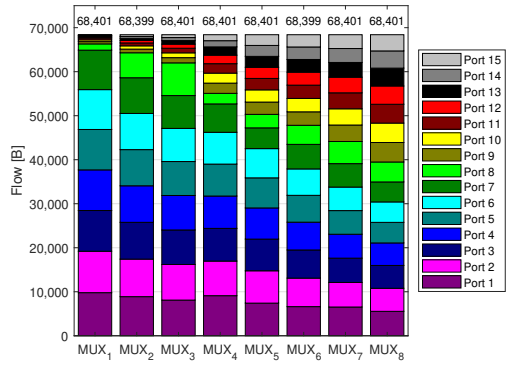

Figure 3: The best TC2 flow allocation based on RL produced in C++ corresponding to Execution 9.

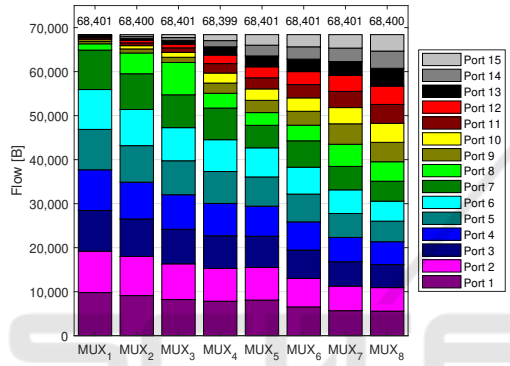

Figure 4: The best TC2 flow allocation based on RL produced in Matlab corresponding to Execution 7.

The best TC2 flow allocation based on RL produced in C++ corresponding to Execution 9 is given in Figure 3 and produced in Matlab corresponding to Execution 7 is given in Figure 4. In almost each execution, a unique solution is retrieved giving very precise LB with the small error. Finally, the TC2 results produced by RL are compared with results acquired by GH and ILP in Table 5.

5 CONCLUSION

The paper has introduced the LB problem of the iFDAQ of the COMPASS experiment at CERN. RL refers to a kind of machine learning method in which an agent receives a delayed reward in the next time step to evaluate its previous action.

The high computational time and the low error cause the proposed RL approach to be more fitting for a long-term LB setup, where no frequent changes in the flows are expected. In addition, the RL approach might lead to a quite high RAM memory consumption during execution to store values of each state, since the problems can be quite complex.

Thus, a question of real-time LB is still open and requires further investigation.

Table 5: Comparison of LB solution methods based on the best flow allocation for the TC2.

Met.	C++		Matlab	
	Error	t [ms]	Error	t [ms]
GH	224.22	63	224.22	1
ILP	194.43	49,927	134.61	95,251
RL	2.83	75,882	2.45	204,824

ACKNOWLEDGEMENTS

This research has been supported by OP VVV, Research Center for Informatics, CZ.02.1.01/0.0/0.0/16_019/0000765.

REFERENCES

Alexakhin, V. Y. et al. (2010). *COMPASS-II Proposal*. The COMPASS Collaboration. CERN-SPSC-2010-014, SPSC-P-340.

Bodlak, M. et al. (2014). FPGA based data acquisition system for COMPASS experiment. *Journal of Physics: Conference Series*, 513(1):012029.

Bodlak, M. et al. (2016). Development of new data acquisition system for COMPASS experiment. *Nuclear and Particle Physics Proceedings*, 273(Supplement C):976–981. 37th International Conference on High Energy Physics (ICHEP).

Borrelli, F., Bemporad, A., and Morari, M. (2017). *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, Cambridge, UK, first edition.

Busoniu, L., Babuska, R., Schutter, B. D., and Ernst, D. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Boca Raton, Florida.

CERN (2019). CASTOR — CERN Advanced Storage manager.

Lapan, M. (2018). *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing, Birmingham, UK. ISBN 1788834240.

Powell, W. B. (2007). *Approximate Dynamic Programming*. Willey-Interscience, New York. ISBN 0470171553.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA. ISBN 0262193981.

Szepesvari, C. (2010). *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, San Rafael, California. ISBN 1608454924.