Big-Data Streaming Applications Scheduling with Online Learning and Concept Drift Detection

Karim Kanoun†, and Mihaela van der Schaar⋆
†Embedded Systems Laboratory (ESL), EPFL, Switzerland.
⋆University of California, Los Angeles (UCLA), U.S.A. email: karim.kanoun@epfl.ch, mihaela@ee.ucla.edu.

***Abstract*—Several techniques have been proposed to adapt Big-Data streaming applications to resource constraints. These techniques are mostly implemented at the application layer and make simplistic assumptions about the system resources and they are often agnostic to the system capabilities. Moreover, they often assume that the data streams characteristics and their processing needs are stationary, which is not true in practice. In fact, data streams are highly dynamic and may also experience concept drift, thereby requiring continuous online adaptation of the throughput and quality to each processing task. Hence, existing solutions for Big-Data streaming applications are often too conservative or too aggressive. To address these limitations, we propose an online energy-efficient scheduler which maximizes the QoS (i.e., throughput and output quality) of Big-Data streaming applications under energy and resources constraints. Our scheduler uses online adaptive reinforcement learning techniques and requires no offline information. Moreover, our scheduler is able to detect concept drifts and to smoothly adapt the scheduling strategy. Our experiments realized on a chain of tasks modeling real-life streaming application demonstrate that our scheduler is able to learn the scheduling policy and to adapt it such that it maximizes the targeted QoS given energy constraint as the Big-Data characteristics are dynamically changing.**

I. Introduction

Emerging Big-Data stream computing applications such as social media analysis, financial analysis and surveillance have stringent delay constraints. Moreover, they need to adapt to dynamically changing data stream characteristics and require dynamic data-driven topology graphs of tasks in order to efficiently process them [13], as well as high throughput efficiency which may require parallel data processing. For instance, stream mining applications [1], one of the main emerging Big-Data stream computing applications, are used to classify a high input of variable data stream and are in general modeled using a chain of classifiers and features-extraction tasks (e.g., Figure 1). Data is collected from multiple sources and is dynamically changing and multiple types of classifiers are applied on this data to extract relevant knowledge.

Even though, new memory hierarchies [8] have been proposed to increase the throughput of the execution of Big-Data stream applications, no existing many-core platform is able to effectively cope with the high-throughput and dynamic Big-Data. Numerous software solutions have been developed to increase the performance of such applications by controlling the throughput and/or the processing method. For instance, approaches based on load-shedding techniques [5][6] reduce the workload by selecting the percentage of data that will be processed while other approaches [1][2][3][4] control the processing method of the data streams to adapt to the given allocated resources. However, these approaches had very limited considerations to the dynamic characteristics of the data streams, which may experience *concept drift* [11] and thus require continuous adaptation. Approaches that rely on offline information are not able to adapt to these concept drifts online.

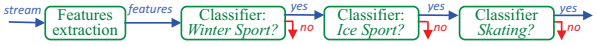

Fig. 1. Example of stream mining application for ice-skating detection [3]

To address these challenges, we propose an online energy-efficient scheduler that adopts reinforcement learning techniques to learn the environment dynamics in order to maximize the Quality of Service QoS (e.g., throughput and quality) of dynamic Big-Data streaming applications given energy constraints. The key contributions of this work are as follows: • We model the scheduling problem as a Stochastic Shortest Path problem (SSP) and propose a reinforcement learning algorithm to learn the environment dynamics to solve this problem even in the presence of concept drift. • The exploration phase of our reinforcement learning algorithm guarantees fast convergence to the targeted QoS • Our experiments demonstrate that our scheduler learns online the dynamics of the environment and provide full control of the quality, throughput and resource constraints without any offline information.

II. Environment Model

We model a streaming application (e.g., [7] [1]) as a chain $C =< \mathcal{N}, \mathcal{E} >$ of dependent tasks t_i with non-deterministic workload w_i and periodic deadline d_i. $\mathcal{N}$ is the node set containing all the tasks. $\mathcal{E}$ is the edge set, which models the dependencies among the tasks. Each node in the chain denotes a task t_i. e_i^{i-1} denotes that there is a directed edge from t_{i-1} to t_i indicating that task i depends on task $i-1$. We denote n as the number of tasks in the chain. The quality of each task t_i can be controlled with a parameter that we call q^i. The exact quality value for each chosen q^i depends on the type of data stream input. The workload w_i of tasks t_i depends on the type of data and the chosen q^i. In our environment, we consider the Big-Data model where the throughput can not reach 100%. Moreover during each time slot, unprocessed data are discarded. Existing solutions assume stationary data stream while, in our model, the data stream is dynamic and experiences the concept drift. Therefore, in Section IV, we vary the size of the stream and type of its data.

III. Scheduling with Online Reinforcement Learning and Fast Concept Drift Detection

Figure 2 illustrates our algorithm. First, a state is observed then an initial action is selected based on either the scheduling

This work was supported in part by a Joint Research Grant for ESL-EPFL by CSEM, as well as by the ObeSense (no. 20NA21 143081) RTD project evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing. M. van der Schaar was supported by the US Air Force Office of Scientific Research under the DDDAS Program.

policy or a greedy method if the state is unknown. The initial action might then be adjusted during the exploration phase depending on the observed QoS and the environment dynamics. The scheduling policy is updated when a concept drift is detected. We explain this flow in this section.

A. SSP formulation: generating the scheduling policy

Unlike existing SSP formulation that are based on Markov Decision Process and where the state space is known and the state transition probabilities is stationary, we propose a solution where the state space is built online and the state transition probabilities are adapted to the dynamics of the environment. We define then a tuple, $(\mathcal{S}, \mathcal{A}, T, R)$ where $\mathcal{S}$ is the state space that is built online, $\mathcal{A}$ is the action space used to control the scheduler, T is the state transition probabilities distribution such that $T(S, a, S') = Pr(S_t = S' | S_{t-1} = S, a_{t-1} = a)$ is the probability for transitioning from state S to state S' after taking action a, where $\sum_{S' \in \mathcal{S}} T(S, a, S') = 1, \forall (S, a)$. T is learned online using the observations obtained from the environment after each action. The agent updates its belief distribution over $\mathcal{S}$ while interacting with the environment. We define $T(S^1, a, S^2) = \frac{g(S^1, a, S^2)}{\sum_{S' \in \mathcal{S}} g(S^1, a, S')}$ with $g(S^1, a, S^2)$ as the number of times the state S^2 has been observed after taking action a for state S^1. Before each update, we multiply previous observation by a coefficient $\epsilon < 1$ to give more weight to the recent observations. R is the reward function mapping $\mathcal{S}$x$\mathcal{A}$x$\mathcal{S}$ to a real number that represents the agent immediate reward for making action a when transitioning from state S to state S'. This is described later in this Section.

We define the local throughput $th^{i,t}_{local}$ as the amount of data processed by the task t_i divided by the size of its data stream input x_i while the global throughput $th^{i,t}_{global}$ is the amount of processed data since the data entered the chain.

We consider a time-slotted system in which the actions decisions are made every time slot. The scheduler interacts with the environment by means of actions $a^i_t = (th^{i,t}_{local}, q^i_t)$ where $th^{i,t}_{local}$ is the desired local throughput and q^i_t is the desired quality level for task t_i at time slot t. During each time slot, an action a^i_t is taken for each task t_i in the chain. In return, the scheduler observes from the environment $n+1$ states (i.e., one state between each two consecutive tasks plus one state at the beginning of the chain and one at the end of the chain). We define a state $S_t = (i_t, x^{i_t}_t, r^{i_t}_t, th^{i_t-1,t}_{global}, c^{i_t-1}_t)$ to link the current state of task t_{i_t} with the state of the platform and the data stream input by means of 5 state parameters presented in the following: the task index i_t to indicate to which task t_{i_t} belongs the observed state ($i_t = n$ if the state is provided after the execution of the last task in the chain), the size $x^{i_t}_t$ of the

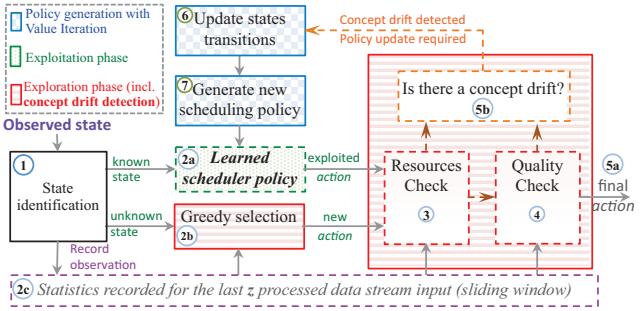

Fig. 2. Global overview of our full proposed solution

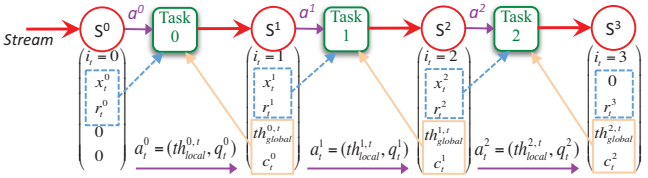

Fig. 3. Integration of the states and the actions in a tasks chain

data stream input of t_{i_t}, the remaining available resources $r^{i_t}_t$ after allocating resources for t_j with $0 \leq j < i_t$, the global throughput $th^{i_t-1,t}_{global}$ observed at task t_{i_t-1} and finally a quality metric $c^{i_t-1}_t$ measured on its previous tasks t_{i_t-1} in the chain. If $i_t = 0$, then $th^{i_t-1,t}_{global} = 0$ and $c^{i_t-1}_t = 0$.

The key point allowing the modelization of the problem as stochastic shortest path problem is that at any time slot t, the state parameters of a state belonging to task i integrate the results of the actions made on all previous tasks in the chain in previous time slots. The states form then a linked list and the actions model the edges. Figure 3 illustrates how the states and the actions are integrated in a chain of three tasks.

Finally, we define the reward function of our SSP model:

$$R_{\alpha,\beta,\theta}(S, S', a) = P_\alpha(S', S, a) + QT_{\beta,\theta}(S', S, a), \quad (1)$$

with $P_\alpha(S', S, a)$ as the power gain metric used to set constraints on the resources usage and $QT_{\beta,\theta}(S', S, a)$ as quality-throughput metric used to quantify the throughput and output quality. Thus, we define for the last observed state in the chain

$$P_\alpha(S^n_t, S^{n-1}_{t-1}, a^n_{t-1}) = \alpha . r^n_t - \frac{r^n_t . r^n_t}{2}, \quad (2)$$

with α as the percentage of desired unused resources. α represents then the remaining available resources that can be exploited for energy saving purposes either by switching the unused cores off or by duplicating the tasks to unused cores and applying DVFS. Figure 4.a illustrates a simple example of the variation of the resources usage with respect to the throughput for different tasks workloads. If the chain includes task 5, then it will be the bottleneck of the global throughput of the chain. A reduction of 20% in the resources usage will then cost at most 5% of the global throughput. Figure 4.b illustrates the reward that quantifies the target QoS for $\alpha = 20\%$ (for the clarity of the figure, we multiplied the reward values by 0.2).

We add an additional reward $QT(S^{i_t+1}_{t+1}, S^{i_t}_t, a)$ related to the throughput and the output quality to drive the learning algorithm through the different tasks in the chain. We define

$$QT_{\beta,\theta}(S^{i_{t-1}+1}_t, S^{i_{t-1}}_{t-1}, a^{i_{t-1}}_{t-1}) = \beta . th^{i_{t-1},t} + \theta . c^{i_{t-1},t}, \quad (3)$$

with β and θ as the weight given to the throughput and quality respectively ($0 < \beta < \theta$) and such that $QT_{\beta,\theta}(S^{i_{t-1}+1}_t, S^{i_{t-1}}_{t-1}, a^{i_{t-1}}_{t-1}) << P_\alpha(\alpha)$. Therefore, the priority is first given to the average unused resources then the output quality and finally the throughput.

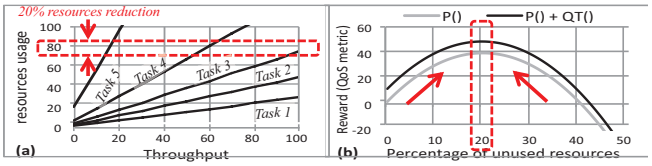

Fig. 4. (a) Variation of the resources usage with respect to the throughput. (b) Adapting the Reward to the targeted QoS and resources savings (example with a savings of 20% of resources)

The goal is to find the optimal policy that maximizes the expected reward given the constructed belief from observed states and actions taken previously. The total discounted average reward can be expressed as

$$\bar{R}_{\alpha,\beta,\theta} = \max_{S_t,a_t,\forall t\in N} E[\sum_{t=0}^{n} \gamma^t R_{\alpha,\beta,\theta}(S_t,a_t)], \quad (4)$$

where $\gamma \in [0,1)$ is the discount factor, and the expectation is over the sequences of states $\{S_t \mid t \in N\}$. The problem of maximizing the discounted average reward can be then mapped to the dynamic programming equation

$$V^*_{\alpha,\beta,\theta}(S) = \max_a [R_{\alpha,\beta,\theta}(S,a) + \gamma \sum_{S'} p(S'|S,a) V^*_{\alpha,\beta,\theta}(S')]. \quad (5)$$

We use the value iteration algorithm [10], which has been already proposed for solving SSP problems. Unlike existing approaches in non-related scheduling problems where the algorithm was applied with stationary transition probabilities and pre-known set of states, in our paper Equation 5 uses a continuously updated non-stationary state transition probabilities given recent observations and applied only on explored states. The complexity of the algorithm is $\mathcal{O}(n.|\mathcal{A}|.|\mathcal{S}|^2)$. n is the number of tasks and refers to the horizon.

B. Exploration phase with concept drift detection

A scheduling policy generated with the value iteration algorithm can get stuck in local maxima when states with higher rewards remain unexplored. Moreover, dynamic data stream input requires a continuous adaptation of the scheduling policy. To address these two problems, we use a *sliding window* sw to record real time statistics related to the execution of the application during the last x time slots.

At each time slot, the scheduler receives a state for each task t_{i_t} in the chain. If a state is observed for the first time, we want to maximize the quality and the throughput but without interfering with the actions made on the next states in the chain. Thus, the selected quality q^i_t and throughput $th^{i_t}_{local}$ can consume at most $r^{i_t} - r^{i_t+1}$ resources. An optimized action is selected based on the input size of the data x_{i_t}, available resources $(r^{i_t} - r^{i_t+1})$ and the estimated workload for each quality measured in the sliding window.

An initial action is then selected either through the generated scheduling policy (i.e., exploitation phase) or through the greedy method. Next, we run our exploration and concept drift detection module which exploits the statistic collected from the sliding window to check if the initial actions need to be adjusted to explore states with higher rewards. An update of the state transition probabilities and the scheduling policy is requested only when the number of adjustment l^v_{adj} or unexplored states l^v_{unexp} accumulated during the last v time slots reaches a certain threshold that we call $l^{v,max}_{adj}$ and $l^{v,max}_{unexp}$.

Each variable with the subscript sw indicates that the variable is a normalized average value measured with the real-time statistics of the *sliding window*. We define then $th^{i,sw}_{global}$ as the sw global throughput of task t_i, $th^{avg,sw}_{global} = \frac{1}{n}.\sum_{i=1}^{n} th^{i,sw}_{global}$ as the average sw global throughput among all the tasks and r^{sw}_n as the sw available resources after scheduling all the tasks. We use these parameters to drive the exploration phase to the states with higher rewards. Figure 5 describes how the exploration phase is driven. The curve in Figure 5 shows the reward function when $\alpha = 30\%$ (i.e., requesting 30% of resources off for energy saving purposes). We identify 3 main regions in the curve. An execution in region A suggests that the explored states are not optimal and make the scheduling policy over-utilize the resources. The throughput is then reduced with respect to the resource constraints (i.e., $\alpha - r^{sw}_n$) in order to explore the states generating the reward curves of region B where it is maximized, $th^0_{local} = th^{avg,sw}_{global} - (\alpha - r^{sw}_n).\frac{th^{avg,sw}_{global}}{100 - r^{sw}_n}$. Then, an execution in region C with a throughput (or quality) under 100% suggests that the explored states make the generated scheduling policy under-uses the available resources. The throughput is then increased with respect to the available resources (i.e., $r^{sw}_n - \alpha$) to explore the states that generate the reward curves of region B, $th^0_{local} = th^0_{local} + (r^{sw}_n - \alpha) * \frac{th^{avg,sw}_{global}}{100 - r^{sw}_n}$. Finally an execution in region B (or in region C with 100% throughput and maximum quality) suggests that sates with maximum rewards have been explored and the execution is now fully driven by the generated scheduling policy. The system is then stabilized. The ζ parameter showed in Figure 5 controls how close we want the scheduling policy to be to the highest possible reward. It also controls the sensitivity of the exploration to the observed variation. Once the throughput actions are adjusted to explore states with higher rewards, the algorithm continues with a quality check. Depending on the targeted quality, the algorithm decides whether to keep the chosen action quality value or adjust it to explore a state with a higher reward. Finally, in the single chain task model, only the control of the local throughput of the first task in the chain is required as the waste of resources is minimized when the data is not discarded after the first stage. Therefore, th_{local} of tasks, positioned after the first task, are set to the maximum.

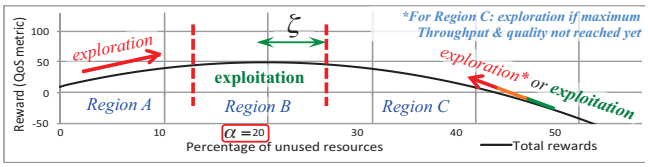

Fig. 5. Driving the exploration phase to states with higher rewards

A variation on the input size, the type of data (i.e., its impact on the required workload and the quality) or the energy saving goals (i..e, α) will drive the execution out of region B (or will decrease the throughput or the quality in region C). This variation will be instantly detected and explored by our tracking sliding window mechanism. Our solution continuously drives the exploration phase to the states that generate the maximum rewards. A limit on l^v_{adj} and l^v_{unexp} triggers a concept drift detection. The state transition probabilities are then updated and the policy is regenerated.

IV. Experimental Section

We developed both our scheduler and environment in C. We use the RL-Glue framework [12] to connect the scheduler to the environment. Figures 6.a and 6b illustrate the simulated dynamics of the environment where the stream data rate and the type of data are both variables. The type of data can refer for instance to the size of the image or to the source the data is coming from. As showed in the figure, with these variations, we simulate 6 different concepts in the environment. Finally, we model a streaming application with a chain of 4 tasks and 5 quality levels [7]. Each task has a different workload that varies with the selected quality and the type of input data. The overall quality of the chain varies with the type of input data.

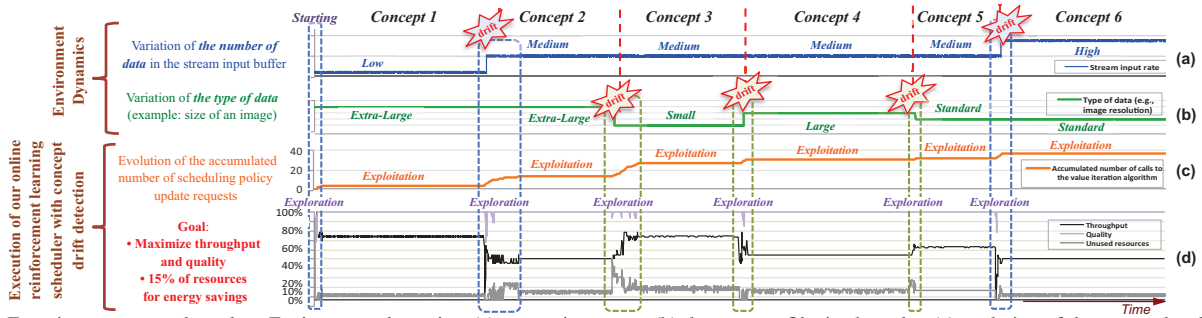

Fig. 6. Experiment setup and results - Environment dynamics: (a) stream input rate. (b) data type. - Obtained results: (c) evolution of the accumulated number of calls to the value iteration algorithm. (d) observed quality, throughput and available resources that can be used for energy savings

In this experiment we target 15% of resources to be used for energy savings. Therefore, in our algorithm, we set a constraint on the resources with $\alpha = 15$ and $\zeta = 5$. We also set $\lambda = 0.94$ and $\epsilon = 0.55$ for the forgetting coefficients of the sliding window and the state transition probabilities respectively and $l_{adj}^{v,max} = 20$ and $l_{unexp}^{v,max} = 10$ to control the frequency of the scheduling policy update. The discretization steps of the global throughput and local throughput are 4%. The input size, quality and available resources have a discretization step of 10%. The sliding window uses continuous values that are received directly from the environment.

Figure 6.d illustrates the obtained results. The figure shows that each time a drift occurs in the environment, it has a direct impact on either the allocated resources or the quality which are the two metrics that are continuously tracked and analyzed by our exploration phase. Figure 6.c, showing the accumulated number of scheduling policy updates, illustrates how the algorithm switches to the exploration phase during a concept drift. Once the constraints on the resources and the quality are satisfied (Figure 6.d), the algorithm switches back to the exploitation phase where the scheduling policy is not required to be updated anymore (Figure 6.c). In the following, we describe the exploration phases observed in Figure 6.d.

The first modeled concept drift scenario is illustrated with the significant variation in the size of data (concepts 2 and 6). For instance, in concept 2 there is an increase in the size of the data. New states are then observed. The algorithm tries first to maximize the throughput of the first task. However, There are no remaining resources available to assign to the next tasks in the chain which explains the very low value of the observed throughput. Then the algorithm increases the quality and lowers the desired throughput of the first task until the unused resources are in $[10\%, 20\%]$ (i.e., $\alpha \pm \zeta$) and the quality is maximized.

The second scenario is illustrated with the variation of the type of data (concepts 3, 4 and 5). For instance in concept 3, the type of data requires a lower computation resources compared to concept 2. This is illustrated in Figure 6.d with the sudden increase in the available resources. This variation is detected by our exploration module which instantly increases the throughput until the constraint on the resources is satisfied again or if the throughput reaches 100%. Similar to this concept, in concept 4 the new type of data requires a higher workload, the constraint on the resources is not satisfied anymore ($\leq 10\%$) the exploration algorithm lower the throughput until the constraint on the resources is satisfied.

Finally, in the exploitation phase of each concept, the available resource curve slightly oscillates because of the small variation that we also model in the stream input size during each concept. As expected, these oscillations of the unused resources are between 10% and 20% due to the curve of our reward function which has a maximum value at 15% and the ζ value that we have set to 5.

V. Conclusion

We have proposed a novel online energy-efficient scheduler that adopts reinforcement learning techniques to learn the environment dynamics in order to maximize the QoS of dynamic Big-Data streaming applications given resource usage constraints. We modeled the scheduling problem as a Stochastic Shortest Path problem (SSP) and proposed a reinforcement learning algorithm to learn the environment dynamics and to solve this problem even in the presence of concept drift. Our experiments demonstrated that our scheduler, without having access to any offline information, is able to learn the scheduling policy and to adapt it such that it maximizes the targeted QoS given resources constraints as the Big-Data characteristics are dynamically changing.

References

[1] R. Ducasse, *et al.*, "Adaptive Topologic Optimization for Large-Scale Stream Mining," in *IEEE JSTSP*, vol. 4, no. 3, pp. 620–636, June 2010.

[2] F. Fu, *et al.*, "Configuring competing classifier chains in distributed stream mining systems," in *IEEE JSTSP*, vol. 1, no. 4, pp. 548–563, December 2007.

[3] H. Park, *et al.*, "Foresighted tree configuring games in resource constrained distributed stream mining systems," in Proc. *ICASSP*, 2009.

[4] D.S. Turaga, *et al.*, "Resource Management for Networked Classifiers in Distributed Stream Mining Systems," in Proc. *ICDM*, 2006.

[5] N. Tatbul, *et al.*, "Load shedding in a data stream manager," in Proc. *VLDB*, 2003.

[6] Y. Chi, *et al.*, "Loadstar: Load shedding in data stream mining," in Proc. *VLDB*, 2005.

[7] J. Xu, *et al.*, "Learning optimal classifier chains for real-time Big-Data mining," in Proc. *Annual Allerton Conference*, 2013.

[8] K. Kanoun, *et al.*, "Low Power and Scalable Many-Core Architecture for Big-Data Stream Computing," in Proc. *ISVLSI*, 2014.

[9] D.P. Bertsekas, *et al.*, "An analysis of stochastic shortest path problems," in *Mathematics of Operations Research*, vol. 16, pp. 580–595, August 1991.

[10] M. L. Puterman, "Markov Decision Processes: Discrete Stochastic Dynamic Programming," John Wiley and Sons, New York, NY, 1994.

[11] J. Gama, *et al.*, "A survey on concept drift adaptation," *ACM Computing Surveys*, 2014.

[12] B. Tanner, *et al.*, "RL-Glue: Language-Independent Software for Reinforcement-Learning Experiments," in *JMLR*, vol. 10, pp. 2133–2136, September 2009.

[13] C. Ballard, *et al.*, " IBM InfoSphere Streams. Harnessing data in motion," IBM Redbooks 2010.

[14] J. G. Koomey, *et al.*, "Estimating Total Power Consumption by Servers in the U.S. and the World," Stanford Univ. Press, 2007.