Towards Decentralized Auto-Scaling Policies for Data Stream Processing Applications

Gabriele Russo Russo

Department of Civil Engineering and Computer Science Engineering
University of Rome Tor Vergata, Italy
russo.russo@ing.uniroma2.it

Abstract Data Stream Processing applications can process large data volumes in near real-time. In order to face varying workloads in a scalable and cost-effective manner, it is critical to adjust the application parallelism at run-time. We formulate the elasticity problem as a Markov Decision Process (MDP). As the MDP resolution requires full knowledge of the system dynamics, which is rarely available, we rely on model based Reinforcement Learning to improve the scaling policy at run-time. We show promising results even for a decentralized approach, compared to the optimal MDP solution.

Keywords: Data Stream Processing, Elasticity, Reinforcement Learning

1 Introduction

New emerging application scenarios (e.g., social analytics, fraud detection, Smart City) leverage Data Stream Processing (DSP) to process data streams in near real-time. A DSP application is usually represented as a directed acyclic graph (DAG), with data sources and operators as vertices, and streams as edges [5]. Each *operator* continuously receives data (e.g, *tuples*), applies a transformation, and generates new outgoing streams.

A commonly adopted DSP optimization is *data parallelism*, which consists of scaling-in/out the parallel instances of the operators, so that each processes a portion of the incoming data (at the cost of using more resources) [5]. Due to the unpredictable and variable rate at which the sources produce the streams, a key feature for DSP systems is the capability of *elastically* adjusting the parallelism at run-time. Most of the existing DSP frameworks allow to allocate more than one replica per operator, but their support for the run-time reconfiguration is quite limited, as regards both the mechanisms and the policies.

In this paper, we focus on the auto-scaling policies. We formalize the elasticity problem for a DSP application as a Markov Decision Process (MDP), presenting both centralized and decentralized formulations. Unfortunately, in practice the optimal MDP policy cannot be determined, because several system dynamics may be unknown. To cope with the model uncertainty, we rely on Reinforcement Learning (RL) approaches, which learn the optimal MDP policy on-line by

N. Herzberg, C. Hochreiner, O. Kopp, J. Lenhard (Eds.): 10^{th} ZEUS Workshop, ZEUS 2018, Dresden, Germany, 8-9 February 2018, published at http://ceur-ws.org/Vol-2072

interacting with the system. Specifically, we present a model based RL solution that leverages the partial knowledge of the model to speedup the learning process.

Elasticity for DSP is attracting many research efforts [1], with most approaches relying on heuristics to determine the scaling decisions. An optimization model that also considers the operator placement problem has been presented in [2], but it cannot be easily solved in a decentralized manner. Here we describe a simpler model, for which we can derive a decentralized formulation. The application of RL techniques to DSP elasticity is quite limited. Heinze et al. [4] propose a simple RL approach to control the system utilization, but they focus on infrastructure-level elasticity. Lombardi et al. [6] exploit RL in their elasticity framework as well, but the learning algorithm is only used for thresholds tuning. In [3] different RL algorithms have been compared for solving the elasticity problem for a single DSP operator in isolation, while in this work we consider whole applications.

In the rest of this paper, we first formulate the elasticity problem as an MDP in Sect. 2, presenting in Sect. 3 the RL based algorithm for learning the scaling policy; we evaluate the proposed solutions in Sect. 4, and conclude in Sect. 5.

2 Problem Formulation

In this paper, we consider the elasticity problem for a DSP application composed of N operators. Each operator is possibly replicated into a number of instances and, without lack of generality, we assume even distribution of the incoming data among the parallel instances. For each operator, an *Operator Manager* monitors the operator functionality, while an *Application Manager* supervises the whole application. The number of parallel instances used by each operator is adjusted either by its Operator Manager (*decentralized* adaptation) or by the Application Manager (*centralized* adaptation).

At each decision step, for each operator we can add an instance, terminate one, or keep the current parallelism. Following a scaling decision, the operator is subject to a reconfiguration process; as the integrity of the streams and the operator internal state must be preserved, the whole application is usually paused during the process, leading to *downtime* [2]. Our goal is to take reconfiguration decisions as to minimize a long-term cost function which accounts for the downtime and for the monetary cost to run the application. The latter comprises (i) the cost of the instances allocated for the next decision period, and (ii) a penalty in case of a Service Level Agreement (SLA) violation. In particular, we consider a constraint on the application response time[1], so that a penalty is paid every time the response time exceeds a given threshold.

In order to keep the system model simple, we consider a deployment scenario with (i) homogeneous computing resources on which the operator instances are executed, and (ii) negligible communication latency between them. We defer to future work the extension of the model for a more realistic distributed setting.

[1] We define the response time as the maximal source-sink total processing latency over the application DAG.

System Model In the considered system, reconfiguration decisions are taken periodically. Therefore, we consider a slotted time system with fixed-length intervals of length Δt, with the i-th time slot corresponding to the time interval $[i\Delta t, (i+1)\Delta t]$. We denote by $k_{op,i} \in [1, K_{max}]$ the number of parallel instances at the beginning of slot i for operator op, and by $\lambda_{op,i}$ its average input rate measured during the previous slot. Additionally, we use Λ_i to denote the overall application input rate (i.e., the total data sources emission rate). At the beginning of slot i, a decision a_i is made on whether reconfiguring each operator.

We first consider a centralized model, in which the reconfiguration decisions are taken by the Application Manager; then, at the end of the section, we consider the case in which the responsibility of making scaling decisions is decentralized, and each Operator Manager acts as an independent agent. In both the cases, we formalize the resulting problem as a discrete-time Markov Decision Process (MDP).

Centralized Elasticity Problem An MDP is defined by a 5-tuple $\langle \mathcal{S}, \mathcal{A}, p, c, \gamma \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}(s)$ a finite set of actions for each state s, $p(s'|s,a)$ are the state transition probabilities, $c(s,a)$ is the cost when action a is executed in state s, and $\gamma \in [0,1]$ is a future cost discounting factor.

We define the state of the system at time i as $s_i = (\Lambda_i, k_{1,i}, k_{2,i}, \ldots, k_{N,i})$. For the sake of analysis, we discretize the arrival rate Λ_i by assuming that $\Lambda_i \in \{0, \bar{\Lambda}, \ldots, L\bar{\Lambda}\}$ where $\bar{\Lambda}$ is a suitable quantum. For each state s, the action set is $\mathcal{A}(s) = \mathcal{A}_1(s) \times \cdots \times \mathcal{A}_N(s)$, where, for each operator op, $\mathcal{A}_{op}(s) = \{+1, -1, 0\}$ (except for the boundary cases with minimum or maximum replication).

System state transitions occur as a consequence of scaling decisions and arrival rate variations. It is easy to realize that the system dynamic comprises a stochastic component due to the exogenous rate variation, and a deterministic component due to the fact that, given action a and the current number of instances, we can readily determine the next number of instances. An example of a system state transition is illustrated in Fig. 1.

To each state pair we associate a cost $c(s,a)$ that captures the cost of operating the system in state s and carrying out action a, including:

1. the *resource* cost $c_{res}(s,a)$, required for running $(k_{op} + a_{op})$ instances for each operator op, assuming a fixed cost per instance;
2. the reconfiguration cost $c_{rcf}(a)$, which accounts for the application downtime, assuming a constant reconfiguration penalty;
3. the SLA violation cost $c_{SLA}(s,a)$, which captures the penalty incurred whenever the response time $T(s,a)$ violates the threshold T_{SLA}.

We define the cost function $c(s,a)$ as the weighted sum of the normalized terms:

$$c(s,a) = w_{res} \frac{\sum_{o=1}^{N} k_{op} + a_{op}}{N K_{max}} + w_{rcf} \mathbb{1}_{\{\exists o : a_o \neq 0\}} + w_{SLA} \mathbb{1}_{\{T(s,a) > T_{SLA}\}} \qquad (1)$$

where w_{res}, w_{rcf} and w_{SLA}, $w_{res} + w_{rcf} + w_{SLA} = 1$, are non negative weights.

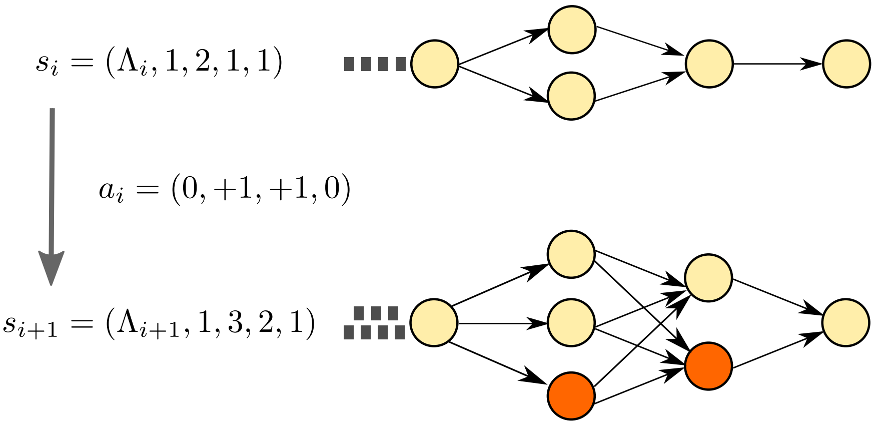

$s_i = (\Lambda_i, 1, 2, 1, 1)$
$a_i = (0, +1, +1, 0)$
$s_{i+1} = (\Lambda_{i+1}, 1, 3, 2, 1)$

Figure 1: Example of a state transition in the centralized MDP model. At time i, the application input rate is $\mathit{\Lambda}_i$ and the components run a single instance, except for the second one which run two. The ApplicationManager picks action $(0, +1, +1, 0)$ at time i, thus adding an instance of the second and the third operator. The resulting parallelism degree of the operators at time $i + 1$ is, respectively, 1, 3, 2, and 1. The input rate at time $i + 1$ is $\mathit{\Lambda}_{i+1}$, which obviously does not depend on a_i.

Decentralized Elasticity Problem In the decentralized adaptation scenario, we assume that each Operator Manager independently acts on its associated operator, having only a local view of the system. We again rely on MDP to formalize the cost minimization problem for each agent (i.e., the Operator Managers). Omitting the reference to the specific operator, we define the state at time i as the pair $s_i = (\lambda_i, k_i)$, where λ_i is discretized using a suitable quantum for each operator. The action set is simply $\mathcal{A}(s) = \{+1, -1, 0\}$ (except for the boundary cases with minimum or maximum replication).

Because the agents have not a global view of the application, they can only optimize local metrics, and thus we have to formulate a new local cost function $c'(s, a)$. We replace the SLA violation penalty with one based on the operator utilization $U(s, a)$ and a target utilization $\bar{U}$. We get:

$$c'(s, a) = w_{res} \frac{k + a}{K_{max}} + w_{rcf} \mathbb{1}_{\{a \neq 0\}} + w_{util} \mathbb{1}_{\{U(s,a) > \bar{U}\}} \tag{2}$$

where w_{res}, w_{rcf} and w_{util}, $w_{res} + w_{rcf} + w_{util} = 1$, are non negative weights.

3 Learning an Optimal Policy

A policy is a function π that associates each state s with the action a to choose. For a given policy π, let $V^{\pi}(s)$ be the value function, i.e., the expected infinite-horizon discounted cost starting from s. It is also convenient to define the action-value function $Q^\pi : \mathcal{S} \times A \rightarrow \Re$ which is the expected discounted cost achieved by taking action a in state s and then following the policy π:

$$Q^\pi(s,a) = c(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^\pi(s'),\ \forall s \in \mathcal{S} \tag{3}$$

It is easy to realize that the value function V and the Q function are closely related in that $V^\pi(s') = \min_{a \in A(s)} Q^\pi(s',a),\ \forall s \in \mathcal{S}$. More importantly, the knowledge of the Q function is fundamental in that it directly provides the associated policy: for a given function Q, the corresponding policy is $\pi(s) = \arg\min_{a \in A(s)} Q(s,a),\ \forall s \in \mathcal{S}$. We search for the optimal MDP policy π^*, which satisfies the Bellman optimality equation:

$$V^{\pi^*}(s) = \min_{a \in \mathcal{A}(s)} \left\{ c(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V^{\pi^*}(s') \right\},\ \forall s \in \mathcal{S} \tag{4}$$

In the ideal situation, we have full knowledge of the system, and we can directly compute π^* using the *Value Iteration* algorithm [7]. In more realistic cases, we have only partial knowledge of the underlying system model (e.g., the workload distribution is usually unknown). We can resort to Reinforcement Learning (RL) approaches, which are characterized by the basic principle of learning the optimal policy by direct interaction with the system. In particular, we consider a model based RL algorithm that, at each time step, improves its estimates of the unknown system parameters, and performs an iteration of the Value Iteration algorithm (see Algorithm 1). Simpler model-free RL algorithms like *Q-learning* have been shown to achieve bad performance even on smaller tasks [3].

Algorithm 1 RL based Elastic Control Algorithm

```
Initialize the action-value function $Q$
loop
    choose an action $a_i$ (based on current estimates of Q)
    observe the next state $s_{i+1}$ and the incurred cost $c_i$
    update the unknown system parameters estimates
    for all $s \in \mathcal{S}$ do
        for all $a \in A(s)$ do
            $Q_i(s,a) \leftarrow \hat{c}_i(s,a) + \gamma \sum_{s' \in \mathcal{S}} \hat{p}(s'|s,a) \min_{a' \in A(s')} Q_{i-1}(s',a')$
        end for
    end for
end loop
```

We first consider the case in which the operator response time model is known, and let the algorithm learn the state transition probabilities.. In order to estimate $p(s'|s,a)$, it suffices to estimate the input rate transition probabilities

$P[\lambda_{i+1} = \lambda' | \lambda_i = \lambda]$[2], since the dynamics related to the number of instances are known and deterministic. Hereafter, since λ takes value in a discrete set, we will write $P_{j,j'} = P[\lambda_{i+1} = j'\bar{\lambda} | \lambda_i = j\bar{\lambda}]$, $j, j' \in \{0, \ldots, L\}$ for short. Let $n_{i,jj'}$ be the number of times the arrival rate changes from state $j\bar{\lambda}$ to $j'\bar{\lambda}$, in the interval $\{1, \ldots, i\}$, $j, j' \in \{1, \ldots, L\}$. At time i the transition probabilities estimates are

$$\widehat{P_{j,j'}} = \frac{n_{i,jj'}}{\sum_{l=0}^{L} n_{i,jl}} \tag{5}$$

If we remove the assumption on the known response time model, we have to estimate the cost $c(s, a)$ as well, because we cannot predict the SLA/utilization violation any more. So, we split $c(s, a)$ and $c'(s, a)$, respectively defined in (1) and (2), into known and unknown terms: the known term $c_k(s, a)$ accounts for the reconfiguration cost and the resources cost, whereas the unknown cost $c_u(s, a)$ represents the SLA (or utilization) violation penalty. We use a simple exponential weighted average for estimating the unknown cost:

$$\hat{c}_{u,i}(s, a) \leftarrow (1 - \alpha)\hat{c}_{u,i-1}(s, a) + \alpha c_{u,i} \tag{6}$$

where $c_i, u = w_{SLA}$ (or w_{util}) if a violation occured a time i and 0 otherwise.

As regards the complexity of the algorithm, the size of the state-action space is critical, since each learning iteration requires $O(|\mathcal{S}|^2 |\mathcal{A}|^2)$ operations. We observe that in the centralized model $|\mathcal{S}|$ and $|\mathcal{A}|$ grow exponentially with the number of operators N, whereas they are not influenced by N in the decentralized model.

4 Evaluation

We evaluate by simulation the presented models, and compare the policies learned through RL to the MDP optimal one. In order to explicitly solve the MDP, we need a state transition probability matrix, which is not available in practical scenarios. Thus, for evaluation, we consider a dataset made available by Chris Whong[3] that contains information about taxis activity, and extract a state transition probability applying (5). We then evaluate the proposed solutions on a new workload, generated according to those probabilities.

For simplicity, we consider a *pipeline* application, composed of a data source and up to 4 operators. Each operator runs at most $K_{max} = 5$ instances, each behaving as a M/D/1 queue with service rate μ_{op}. For evaluation, we consider a scenario with slightly different service rates, and set $\mu_1 = 3.7$, $\mu_2 = \mu_4 = 3.3$, and $\mu_3 = 2.7$ tuple/s. Because of space limitation, we defer the evaluation of real world topologies to future work. We consider $\Delta t = 1$ min, and aggregate the events in the dataset over one minute windows. We assume $\Lambda_i = \lambda_{o,i}, \forall o$, discretized with $\bar{\lambda} = 20$ tuple/min. For the cost function, we set $w_{sla} = w_{util} = w_{rcf} = 0.4$, $w_{res} = 0.2$, $T_{SLA} = 650$ ms, and $\bar{U} \in \{0.6, 0.7, 0.8\}$. As regards the learning

[2] To simplify notation, we simply use λ to denote the input rate. In the centralized model, we use the same estimates for the total application input rate Λ.

[3] http://chriswhong.com/open-data/foil_nyc_taxi/

Table 1: Results for a 3-operators application. The "+" sign denotes the knowledge of the response time model.

Scaling Policy	Avg. Cost	SLA Violations	Reconf.	Avg. Instances
Centralized MDP	0.163	8903	13882	10.95
Centralized RL$^+$	0.164	9505	13684	10.94
Centralized RL	0.167	15681	14579	10.79
Decentralized ($\bar{U} = 0.6$)	0.178	3639	30104	11.46
Decentralized ($\bar{U} = 0.7$)	0.173	17111	30404	10.30
Decentralized ($\bar{U} = 0.8$)	0.205	79670	29681	9.15

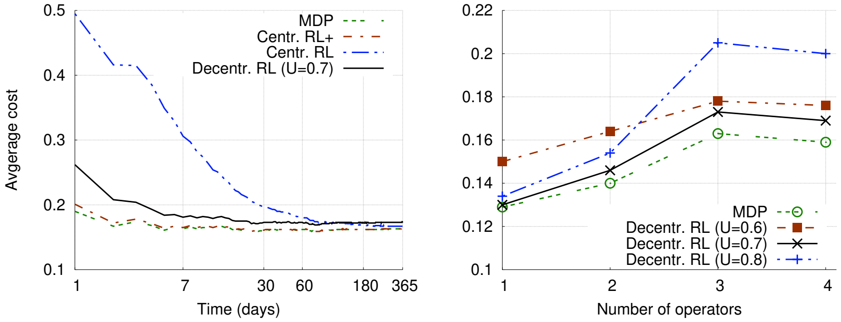
algorithm, we set $\gamma = 0.99$, and $\alpha = 0.1$. We compare the results obtained by solving the MDP to those achieved by the centralized RL algorithm (with and without the known response time model) and by the decentralized solution.

In Table 1 we report the results for a 3-operators topology. As expected, the minimum average cost is achieved solving the MDP; interestingly, the centralized RL solution incurs almost negligible performance degradation, and the gap with the decentralized approach is not significant as well. However, we note that the performance of the decentralized solution depends on the target utilization $\bar{U}$, which has still to be set manually in our work. Setting a too high (or too low) value results in a different trade-off between SLA violations and used instances, with negative effects on the overall cost as well. The decentralized solution shows a higher number of reconfigurations, due to the lack of coordination between the agents. As illustrated in Fig. 2a, the convergence velocity of the different solutions is similar, except for the centralized RL algorithm. In absence of the response time model, the algorithm is indeed significantly slower to learn than the other solutions. When the response time model is known, the algorithm converges much faster, despite the large state-action space.

We also compare the decentralized approach to the MDP varying the number of operators in the application. As shown in Fig. 2b, the cost gap between the two solutions slightly increases as the application gets more complex. However, we observe that the decentralized algorithm has not scalability issues as the number of operators increases, while solving a centralized problem gets easily impractical.

5 Conclusion

In this paper we have formalized the elasticity problem for DSP applications as a Markov Decision Process, and proposed a Reinforcement Learning based solution to cope with the limited knowledge of the system model. Our numerical evaluation shows promising results even for a fully decentralized solution which, leveraging the available knowledge about the system, does not suffer from the extremely slow convergence of model-free RL algorithms. In practical scenarios, we could also combine the proposed solution with a simple threshold-based policy



(a) (b)

Figure 2: Average cost during one simulated year for a 3-operators application (a), and for different number of operators (b). The "+" sign denotes the knowledge of the response time model.

to be used at the beginning, while the agents learn a good policy to be adopted in the following.

For future work, our goal is twofold. We plan to improve the decentralized learning algorithm exploring RL techniques specifically targeted to multi-agent systems. At the same time, we will extend the model to cope with a more complex and realistic scenario, considering, e.g., resource heterogeneity and network latency in distributed deployments.

References

1. Assuncao, M.D., da Silva Veith, A., Buyya, R.: Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. Journal of Network and Computer Applications 103, 1 – 17 (2018)
2. Cardellini, V., Lo Presti, F., Nardelli, M., Russo Russo, G.: Optimal operator deployment and replication for elastic distributed data stream processing. Concurrency and Computation: Practice and Experience (2017), `http://dx.doi.org/10.1002/cpe.4334`
3. Cardellini, V., Lo Presti, F., Nardelli, M., Russo Russo, G.: Auto-scaling in data stream processing applications: A model based reinforcement learning approach. In: Proc. of InfQ '17 (in conjunction with VALUETOOLS '17) (2018, to appear)
4. Heinze, T., Pappalardo, V., Jerzak, Z., Fetzer, C.: Auto-scaling techniques for elastic data stream processing. In: Proc. IEEE ICDEW '14. pp. 296–302 (2014)
5. Hirzel, M., Soulé, R., Schneider, S., Gedik, B., Grimm, R.: A catalog of stream processing optimizations. ACM Comput. Surv. 46(4) (Mar 2014)
6. Lombardi, F., Aniello, L., Bonomi, S., Querzoni, L.: Elastic symbiotic scaling of operators and resources in stream processing systems. IEEE Transactions on Parallel and Distributed Systems PP(99), 1–1 (2017)
7. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press, Cambridge (1998)