A Multi-level Elasticity Framework for Distributed Data Stream Processing

Matteo Nardelli(✉), Gabriele Russo Russo, Valeria Cardellini, and Francesco Lo Presti

Department of Civil Engineering and Computer Science Engineering, University of Rome Tor Vergata, Rome, Italy
{nardelli,russo.russo,cardellini}@ing.uniroma2.it,
lopresti@info.uniroma2.it

Abstract. Data Stream Processing (DSP) applications should be capable to efficiently process high-velocity continuous data streams by elastically scaling the parallelism degree of their operators so to deal with high variability in the workload. Moreover, to efficiently use computing resources, modern DSP frameworks should seamlessly support infrastructure elasticity, which allows to exploit resources available on-demand in geo-distributed Cloud and Fog systems. In this paper we propose E2DF, a framework to autonomously control the multi-level elasticity of DSP applications and the underlying computing infrastructure. E2DF revolves around a hierarchical approach, with two control layers that work at different granularity and time scale. At the lower level, fully decentralized Operator and Region managers control the reconfiguration of distributed DSP operators and resources. At the higher level, centralized managers oversee the overall application and infrastructure adaptation. We have integrated the proposed solution into Apache Storm, relying on a previous extension we developed, and conducted an experimental evaluation. It shows that, even with simple control policies, E2DF can improve resource utilization without application performance degradation.

Keywords: Data Stream Processing · Elasticity · Hierarchical control

1 Introduction

Exploiting on-the-fly computation, Data Stream Processing (DSP) applications can elaborate unbounded data flows so to extract high-value information as soon as new data are available. A DSP application is represented as a directed (acyclic) graph, with data sources, operators, and final consumers as vertices, and streams as edges. Importantly, these applications are usually long running and often subject to strict latency requirements that should be met in face of variable and high data volumes to process. To deal with operator overloading, a commonly adopted stream processing optimization is data parallelism, which consists in scaling-out or scaling-in the number of parallel instances for the operators, so that each instance can process a subset of the incoming data flow in parallel [7].

To execute the application, its operators are deployed on computing resources, which host the operator instances. We consider the emerging environment, where distributed Cloud and Fog computing resources can be acquired and released on demand. Specifically, Fog computing enriches powerful but distant Cloud data centers with micro-data centers located at the network periphery, closer to the users/devices that produce and consume data. Therefore, the abundant presence of geo-distributed computing nodes can be exploited so to decentralize the application execution as well, thus reducing the application latency and the movement of high data volume. In this environment, DSP frameworks should be able to scale their applications, by changing the operators parallelism (*application elasticity*), as well as to accordingly provision computing resources (*infrastructure elasticity*) [1]. While the application elasticity allows to better distribute computing capacity among DSP operators, the infrastructure elasticity allows to avoid resource wastage while guaranteeing that enough computing capacity is available when needed.

In this paper, we present *Multi-level Elastic and Distributed DSP Framework* (E2DF), which extends our hierarchical architecture for application-level elasticity [2] so to introduce infrastructure management capabilities. In E2DF, the application control system and the infrastructure control system are organized according to the Monitor, Analyze, Plan and Execute (MAPE) architectural pattern for self-adaptive systems. Differently from existing works [10,12] that consider multi-level elasticity in a clustered environment, our solution is designed for a geo-distributed operating environment. To manage a high number of geo-distributed nodes in a scalable manner, our infrastructure and application control systems are realized through a two-level hierarchical pattern.

Our main contributions are as follows:

- we present the infrastructure control system of E2DF. It relies on a high-level MAPE-based *Infrastructure Manager* that coordinates the run-time adaptation of subordinated MAPE-based *Region Managers*, which locally control the elasticity of computing resources within a single micro-data center;
- we present simple control strategies for each component of E2DF, namely a *local* policy for the Region Managers, and a *global* policy for the Infrastructure Manager;
- we implement and evaluate E2DF on top of our extension [2,4] of Apache Storm. Our results are promising and show the effectiveness of the proposed E2DF framework, which allows to reduce the amount of used computing resources, while keeping an acceptable level of application performance.

This paper is organized as follows. We review related work in Sect. 2. In Sect. 3, we present the hierarchical distributed architecture of E2DF for the autonomous control of application and infrastructure elasticity. In Sect. 4, we present simple control policies for each component of E2DF. In Sect. 5, we evaluate the ability of E2DF to dynamically manage applications and computing resources. We conclude in Sect. 6.

2 Related Work

Run-time adaptation of DSP applications has attracted attention in recent years [1], mainly focusing on the application elasticity and the adaptation policies and mechanisms that support it. Some works, e.g., [6,8], exploit best-effort threshold-based policies based on the utilization of either the system nodes or the operator instances. Other works, e.g., [3,9,11,16], use more complex centralized policies to plan the scaling decisions. Heinze et al. [9] estimate latency spikes caused by operator reallocations through a model and use it to define a heuristic placement algorithm. Lohrmann et al. [11] propose a scaling strategy that enforces latency constraints by relying on a predictive queueing theory model. Stela [16] relies on throughput-based metric to identify the operators that need scaling. In [3] we formulate a centralized optimization problem for the run-time elasticity management of DSP applications that takes into account reconfiguration costs.

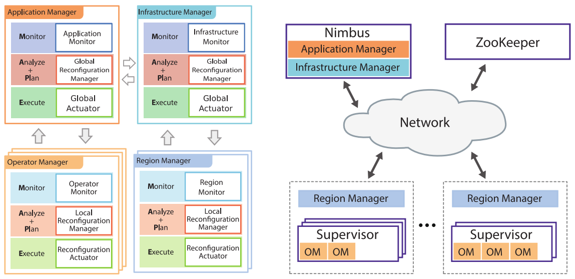
Current open-source DSP frameworks (e.g., Flink, Heron, Samza, Storm, Spark Streaming) manage the DSP application distribution, execution, and adaptation. However, as regards the application elasticity, most of them (except Heron and Spark Streaming) only support the manual scaling of operators, which can lead to sub-optimal application performance and operating costs. Dhalion, a framework on top of Heron, provides application elasticity by scaling out/in operators so to satisfy their throughput; Spark Streaming supports elastic scaling of the number of executors. As regards the infrastructure elasticity, the above frameworks can take advantage of the elasticity support of Cloud infrastructures [5]. However, in most cases the reconfiguration is enacted by restarting the DSP application, thus causing downtime and possible state loss. Moreover, elasticity decisions at the two different levels are, when available, independent and uncoordinated, which could led to sub-optimal adaptation.

Only few solutions explicitly consider the reconfiguration of DSP applications in Fog and Cloud geo-distributed environments. SpanEdge [14] uses Cloud and Fog data centers and follows a master-worker architecture implemented in Storm, but it does not support operator migrations. Firework [17] provides only elasticity of computing resources. Decentralized solutions for the elasticity of DSP applications do not suffer as their centralized counterpart from network latencies in geo-distributed environments. Among them, Mencagli [13] presents a game-theoretic strategy where the control logic is distributed on each operator. In [2] we propose a hierarchical distributed architecture for the autonomous control of elastic DSP applications and present distributed self-adaptation policies also based on reinforcement learning; in this paper, we extend that architecture to support elasticity also at the infrastructure level.

The works most closely related to our own have been presented in [10,12], which consider multi-level elasticity both at the application and infrastructure level. Liu et al. [10] propose a stepwise profiling framework that evaluates the efficiency of possible configurations of parallelism. Similarly to us, their goal is to avoid resource wastage; however, they do not propose auto-scaling policies. Lombardi et al. [12] consider at the same time the elasticity at the operator and resource levels, where scaling actions can be executed either in a reactive or proactive fashion, and implement their proposal in Storm. Differently from us, all these works are designed for a traditional clustered system and therefore could suffer from scalability issues in a geo-distributed environment. The E2DF framework we propose is a first step towards coordinated multi-level elasticity in geo-distributed Cloud and Fog systems.

3 System Architecture

The MAPE loop represents a well-know architectural pattern to organize the autonomous control of a software system, where four components (Monitor, Analyze, Plan, and Execute) are responsible for the primary functions of self-adaptation. When the controlled system is geo-distributed, as in Fog computing, a centralized MAPE loop, where analysis and planning are carried on by a single component, may suffer from scalability issues. As described in [15], different patterns to decentralize the MAPE components have been used in practice. Among them, the hierarchical control pattern is of particular interest. It revolves around the idea of a layered architecture, where each layer works at a different level of abstraction. In this pattern, multiple MAPE control loops work with time scales and concerns separation. Lower levels operate on a shorter time scale and deal with local adaptation. Exploiting a broader view on the system, higher levels steer the overall adaptation by providing guidelines to the lower levels.



(a) E2DF hierarchical MAPE loops (b) Extended Storm architecture

Fig. 1. System architecture

Multi-level Elastic and Distributed DSP Framework (E2DF) includes two management systems that are organized according to a two-level hierarchical pattern: the Application Control System, which adapts the DSP operators deployment, and the Infrastructure Control System, which realizes resource elasticity. Figure 1a illustrates the conceptual architecture of E2DF, highlighting the hierarchy of the multiple MAPE loops and the system components in charge of the MAPE loop phases. The Infrastructure Control System includes a centralized *Infrastructure Manager* (IM), which cooperates with multiple decentralized

Region Managers (RM). Similarly, the Application Control System comprises a centralized *Application Manager* (AM) and decentralized *Operator Managers* (OM). Besides controlling the applications and computing resources in E2DF, the IM and AM can interact to adapt their behavior at run-time. Specifically, the IM can expose different views of the computing resources upon which the AM can run the application. In such a way, the IM can dynamically partition resources among applications. Differently from the approaches where the infrastructure is adapted without considering the application needs, the IM-AM interaction enables to realize cross-level optimizations. For example, when the IM detects that computing resources are underutilized, it can propose the AM to consolidate the managed applications on a reduced number of resources. Similarly, the AM can prevent the IM from terminating underutilized nodes when the latter execute critical DSP operators.

Infrastructure Control System. The *Region Manager* (RM) realizes the lower level MAPE loop of the Infrastructure Control System. It is a distributed entity that oversees resource elasticity within a single region (i.e., data center, micro-data center). To this end, it monitors the computing nodes used by E2DF within the region through the *Resource Monitor*. Then, through the *Local Reconfiguration Manager*, it analyzes the monitored data and determines if new resources should be acquired or leased ones should be released. When the RM determines that some adaptation should occur, it issues an adaptation request to the higher layer.

At the higher level, the *Infrastructure Manager* (IM) coordinates the resource adaptation among the different computing regions through a global MAPE loop. By means of the *Infrastructure Monitor* it collects aggregated monitoring data from the different available regions. Then, through the *Global Reconfiguration Manager*, it analyzes the monitored data and the reconfiguration requests received by the multiple RMs, and decides which reconfigurations should be granted. For example, the Global Reconfiguration Manager can decide that it is more convenient to acquire resources from a specific region, so it will inhibit scaling operations proposed for other regions. According to its internal policy, the Global Reconfiguration Manager can interact with the AM and adapt its behavior accordingly. For example, it may suggest the AM to consolidate the managed DSP operators on fewer computing nodes (the AM can accordingly accept or deny the request). Using the *Global Actuator*, the IM communicates its reconfiguration decisions to each RM, which can, finally, scale the computing infrastructure by means of the their local *Reconfiguration Actuators.*

Application Control System. The Application Control System manages the run-time adaptation of a DSP application. Similarly to the Infrastructure Control System, it implements a hierarchical MAPE loop where an Application Manager oversees subordinate Operator Managers. At the lower level, the *Operator Manager* (OM) controls the reconfiguration of a single DSP operator and proposes reconfiguration requests to the higher level. At the higher level, the *Application Manager* (AM) is the centralized entity that coordinates the adaptation request aiming to obtain good overall DSP application performance. We refer the reader to [2] for further details.

Integration of E2DF in Storm. We have implemented the proposed E2DF architecture in EDF [2], our extension of Apache Storm. EDF, by relying on Distributed Storm [4], enhances the official Storm release by introducing an infrastructure-level and application-level monitoring system and by supporting run-time stateful operator scaling and migration (i.e., it enables the application elasticity while preserving its integrity). Due to space limitations, we omit a detailed description of EDF and Distributed Storm and refer the reader to [2,4]. As represented in Fig. 1b, we introduce the E2DF components into the Storm architecture. More precisely, the AM, IM, and OM are implemented within existing Storm components, whereas the RM constitutes a new component to be deployed in every region along with Storm Supervisors.

The IM runs within Nimbus, i.e., Storm's master node. As soon as it is created, it runs its MAPE control loop and waits for requests by the RMs and AMs. The RMs are statically defined, one per region; each RM executes its local policy and operates autonomously with one another. To acquire and release resources, the Reconfiguration Actuator of the RM can be implemented to manage virtual machines or software containers. In our current implementation, it uses software containers, managed through Docker. In such a way, each Storm worker node runs within a container that can be quickly spawn and terminated at run-time. The RM first retrieves monitoring information about CPU utilization of the computing resources used by Distributed Storm within the region. Then, it uses the local policy to determine whether a resource scaling operation should be performed, and possibly forwards the request to the IM. Should the reconfiguration be performed, the Reconfiguration Actuator of the RM scales the computing resources using the Docker APIs.

When a new application is submitted to Storm, Nimbus creates one AM and multiple OMs (one per operator). While the AM runs in Nimbus, the OMs are assigned to the available worker nodes by the Storm scheduler. As soon as the AM is created, it determines the initial application placement on the set of worker nodes. At run-time, Nimbus executes periodically the AM, which analyzes the monitored application response time, acquired from Distributed Storm, and collects the reconfiguration requests coming from the decentralized OMs. Then, the global policy is executed so to coordinate and grant the reconfiguration actions. To enact the deployment changes, the Global Actuator of the AM relies on the `rebalance` command of Storm and on the stateful migration mechanisms of Distributed Storm, which allow to preserve the operators internal state while reconfiguring. Each OM collects information about the managed operator (e.g., resource usage) and relies on its local policy to identify beneficial reconfigurations and to propose them to the AM. Should a reconfiguration be performed, the OM Reconfiguration Actuator adapts the operator deployment (e.g., by changing its replication degree), while preserving the operator internal state.

4 Multi-level Elasticity Policy

The proposed two-layered architecture identifies different macro-components (i.e., AM-OM, IM-RM) that cooperate to adapt the deployment of DSP applications and infrastructures at run-time. The E2DF architecture is general enough to not limit the specific internal policies and goals for these components. By properly selecting the internal policy for each component, the proposed solution can address the needs of different execution contexts, thus encompassing applications with different requirements, infrastructures with different kind of computing resources, and different user preferences. For example, the planning components can be either activated periodically or on event-basis, can rely on optimization problem formulation or heuristics that minimize the application response time, maximize its availability, or a combination thereof.

Since the control components (i.e., AM, OM, IM, RM) work at different abstraction layers, we need two-layered control policies as well. Specifically, we will consider *local policies*, associated with RM and OM, which are concerned with low-level adaptation actions and exploit a fine grained view on a subset of the controlled entities (i.e., the replicas of a single operator and the computing resources in a given region, respectively). The local policy does not directly enact planned adaptation actions, which instead are communicated to the higher level components, i.e., AM and IM. These components are each equipped with a *global policy* that works at the granularity of the whole application/infrastructure. On the basis of the overall monitored performance and the application performance requirements (e.g., coming from a SLA), the global policies identify the most effective reconfigurations proposed by the decentralized agents, providing an implicit coordination mechanism among the independent local policies.

As a proof-of-concept of the proposed architecture, we present simple heuristic elasticity policies whose overall adaptation goal is to preserve the application performance in face of varying workloads, avoiding computing resources wastage.

4.1 Infrastructure Control Policy

The Infrastructure Control System manages the computing resources (e.g., containers, VM) allocated for the execution of DSP applications. As a proof-of-concept policy, we consider a simple threshold-based approach, which is the most commonly used one in Cloud auto-scaling systems [5]. The *local policy* executed by the RM in each region r considers: (i) C_n, the *capacity* of each node n, defined as the maximum number of application operators' instances it can host (e.g., proportional to the number of CPU cores); (ii) A_n, the number of operators instances currently assigned to each node; and (iii) U_n, the CPU utilization of each node.[1] For each region r, we consider a target capacity C_r, which should always be available for deploying new operator replicas. Hence, we require that $\sum_{n \in Nodes(r)} (C_n - A_n) \geq C_r$. Whenever this constraint is violated,

[1] The policy can be easily extended to consider other load metrics (e.g., related to memory or network bandwidth utilization).

the RM proposes to add one or more computing resources to satisfy the capacity requirement. For simplicity, we assume that, if the RM can pick different kinds of resources, it will choose the one with minimum capacity C_n, in order to have a fine-grained control over the resource allocation. On the other hand, when the available capacity in the region exceeds the minimum required amount, the RM searches for computing resources to turn off. Specifically, the RM searches for nodes that do not host application operators' instances, and, if any, issues a request to the IM for terminating them.

Moreover, the local policy searches for nodes that host one or more application operators' instances, but seem to be under-loaded. Specifically, the RM looks for nodes whose CPU utilization does not exceed a predefined threshold $\bar{U}_{low,r}$ (i.e., $U_n \leq \bar{U}_{low,r}$). The replicas running on those nodes might be easily migrated elsewhere, in order to consolidate the active computing nodes. The RM issues a request to the IM for freeing and terminating these nodes.

Finally, the local policy communicates to the IM its proposed actions. The IM *global policy* can accept/reject adaptation requests based on functional and non-functional requirements. For the sake of simplicity, to evaluate the proposed framework, we rely on a simple global policy, which accepts all the actions proposed by the RM, except for those requiring the termination of a computing resource currently occupied by one or more DSP applications, which require special attention. When the RM proposes to turn off such a node, the IM will in turn issue a request to the involved AMs for migrating their operators to different nodes. The IM also removes the node from the list of available resources, to avoid that other operators are assigned to it by any AM. After a configurable time interval, if the AM has not moved away the operators from the under-loaded node, the scale-in procedure is canceled and the node considered available again.

4.2 Application Control Policy

Relying on a local policy executed by the OMs, and on a global policy executed by the AM, the Application Control System manages the DSP applications elasticity and placement. The OM *local policy* implements the Analyze and Plan phases of the decentralized MAPE loop, which controls the execution of a single DSP operator. Running on a decentralized component, this policy has only a local view of the system, which consists of the status (i.e., resource utilization) of each operator replica and of a restricted suitable set of computing nodes (i.e., located in the same region). By analyzing this information, the policy can plan a reconfiguration of the operator deployment, by changing the number of its replicas. We adopt a simple threshold-based policy for planning scaling actions [2]. Let us denote by S_α the resource utilization of replica α, which measures the fraction of CPU time used by α. When the utilization of α exceeds a usage threshold $S_{\text{s-out}} \in [0, 1]$ (i.e., $S_\alpha > S_{\text{s-out}}$), the OM proposes to add a new replica. The new replica is allocated on the least utilized computing resource within the same region of the other operator replicas. Conversely, the OM proposes a scale-in operation, which removes one of the running n replicas, when the sum of their utilization divided by $n - 1$ is significantly below the usage threshold, i.e., when

$\sum_{\alpha=1}^{n} S_\alpha/(n-1) < cS_{\text{s-out}}$, being $c < 1$. The replica to be removed is randomly chosen between the two replicas with the highest utilization. Similarly to the RM local policy, the OM proposes reconfiguration actions to the high-level AM, which can accept or reject them, based on its global policy.

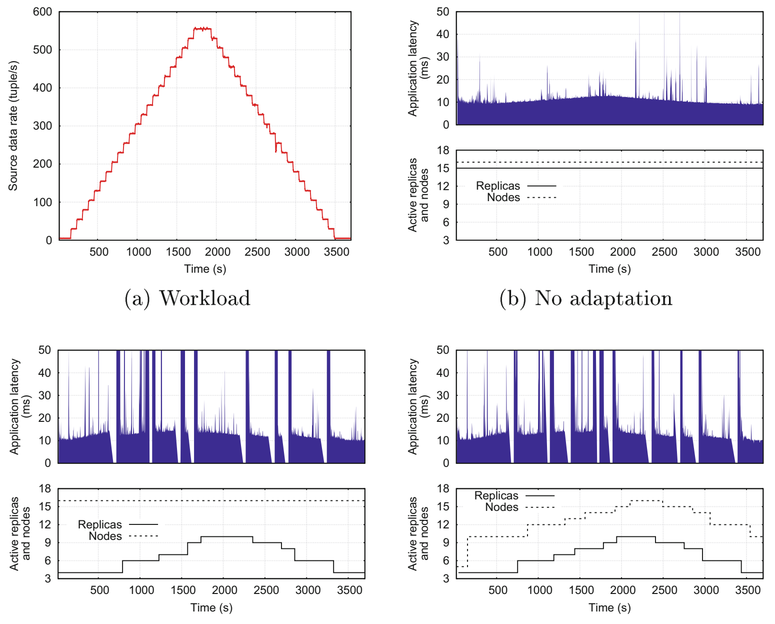
The AM determines the initial application deployment. To this end, it uses a placement policy that assigns the operators on an initial number of R_{init} computing nodes, aiming to balance the number of operator per node. To perform run-time adaptation, the AM adopts a *global policy* that implements the Analyze and Plan steps of the centralized MAPE loop. Its main goal is to coordinate the actions of the decentralized OMs, so to satisfy the DSP application performance requirements, while minimizing the allocated resources (or their cost). In particular, it monitors the application response time and analyzes its behavior, possibly by comparing it against a user-defined target performance. It can leverage this information to decide whether, e.g., a higher parallelism could be beneficial for the application, or the resource usage costs should be reduced. To this end, the policy determines which reconfiguration plans, proposed by the decentralized OMs, should be accepted. For the sake of simplicity, in this work we consider a very simple global policy, which only rejects reconfigurations when they try to acquire an already used resource (e.g., just assigned to another operator). More sophisticated approaches (e.g., based on a token bucket to limit the number of performed reconfigurations) can be proposed as well [2].

5 Evaluation

We evaluate the ability of E2DF to realize the multi-level elasticity. To more easily investigate the proposed architecture, we equip E2DF with the proposed proof-of-concept policies, and consider a single deployment region, where Storm worker nodes are allocated as Docker containers. Each container allows the worker node to run on a single CPU core, for no more than 50% of the time. Each worker node has capacity $C_n = 1$, thus can host a single operator instance. The Docker containers are executed on a single host machine, equipped with an Intel i7-4710HQ CPU and 16 GB of RAM.

As a reference application, we consider a simple *Word Count* topology, defined as a sequence of a source and 3 operators. The *datasource* emits random sentences at a variable rate; the *split* emits a tuple for each word in the received sentences; the *counter* traces how many times each word has appeared; the final *consumer* publishes statistics to a RabbitMQ queue. Specifically, as shown in Fig. 2a, the source emits data at a rate that grows from 5 to 550 tuples/s and then decreases back to the initial value.

To show the potentialities of E2DF, we evaluate three different execution scenarios. In the baseline one, neither the infrastructure nor the application parallelism is adapted at run-time. We provisioned both the infrastructure and the application so to handle the peak load; we run 16 worker nodes and 15 total operator replicas for the reference application (namely, 2 replicas for *split*, 6 for *counter* and 6 for *consumer*). As a second scenario, we consider the case where the worker nodes are statically provisioned, but the operators parallelism can be adapted at run-time (i.e., only application-level elasticity). Finally, in the third scenario, we evaluate E2DF with all the self-adaptation features enabled, exploiting infrastructure-level elasticity as well. For the experiments, we let the IM and RM run once per minute, and the AM and OM twice per minute. For the RM policy, we set $C_r = R_{init} = 5$, and $\bar{U}_{low,r} = 0.1$. As regards the OM policy, we set $S_{\text{s-out}} = 0.7$, and $c = 0.75$.



(a) Workload (b) No adaptation

(c) E2DF with IM disabled (d) E2DF

Fig. 2. Application latency and allocated resources with different self-adaptation capabilities (b–d), when the application is subject to a linearly growing and decreasing input rate (a).

Results. Figure 2 reports the application latency, the application parallelism, and the number of active worker nodes throughout our experiments. Figure 2b illustrates the baseline scenario, when both the worker nodes and the operator replicas are statically provisioned. In this case, the average application latency throughout the experiment is 11.4 ms. Such a configuration is likely to waste resources by using the same computational power in face of different levels of incoming load. Our second experiment confirms this observation: indeed, the Application Control System, starting with a single replica per operator, adapts the number of operator replicas used by the DSP application at run-time (see Fig. 2c). Application elasticity allows to use, on average, 55% less replicas during the experiment, with only limited performance degradation: the average application latency in this setting is 19.3 ms.[2] Nonetheless, in this experiment we are likely to still over-provision the infrastructure resources, keeping 16 active worker nodes all the time.

The third scenario allows to evaluate the benefits of exploiting the full self-adapting capabilities of E2DF, i.e., when it can adapt the application and the infrastructure. The application performance is almost identical to that observed in the previous experiment, while the number of active worker nodes (and so the resource usage cost, in a real scenario) is reduced on average by 24%. As shown in Fig. 2d, the number of running worker nodes is readily adjusted as the application acquires or releases worker slots. These results demonstrate that our simple policies are effective in limiting the resource wastage, at the same time avoiding significant performance degradation.

6 Conclusions

In this paper, we presented Multi-level Elastic and Distributed DSP Framework (E2DF), a hierarchical approach for controlling DSP application elasticity and infrastructure elasticity. Designed according to the decentralized MAPE control pattern, our solution relies on a two layered approach with separation of concerns and time scale between layers. At the lower level, distributed components control the adaptation of DSP operators and computing resources within a deployment region. At the higher level, a per-application manager oversees and coordinates the DSP application adaptation, while a global IM supervises the management of computing resources across different regions. We prototyped the proposed solution within Distributed Storm and proposed proof-of-concept policies to evaluate the benefits of the proposed hierarchical and distributed architecture. The results show that our simple yet effective policies allow to significantly reduce resource wastage with respect to statically provisioned applications and infrastructures.

As future work, we will further investigate the presented hierarchical approach. We plan to design more complex decentralized policies, considering different (stringent and possibly conflicting) optimization objectives, and a larger set of constraints (e.g., related to network bandwidth). We will also investigate the interaction between the application-level and infrastructure-level elasticity. In particular, we will study the multi-agent optimization problem that arises from the interaction of the ACS and ICS, recurring to techniques specifically targeted to this class of systems (e.g., Multi-Agent Reinforcement Learning).

[2] We can observe evident spikes in the measured application latency after each reconfiguration; they are due to the *pause-and-resume* stateful reconfiguration protocol adopted by Distributed Storm. Therefore, we compute our statistics excluding the first 2 min after each reconfiguration.

References

1. de Assunção, M.D., da Silva Veith, A., Buyya, R.: Distributed data stream processing and edge computing: a survey on resource elasticity and future directions. J. Netw. Comput. Appl. **103**, 1–17 (2018)
2. Cardellini, V., Lo Presti, F., Nardelli, M., Russo Russo, G.: Decentralized self-adaptation for elastic data stream processing. Future Gener. Comput. Syst. **87**, 171–185 (2018)
3. Cardellini, V., Lo Presti, F., Nardelli, M., Russo Russo, G.: Optimal operator deployment and replication for elastic distributed data stream processing. Concurr. Comput. Pract. Exp. **30**(9), e4334 (2018)
4. Cardellini, V., Grassi, V., Lo Presti, F., Nardelli, M.: Distributed QoS-aware scheduling in storm. In: Proceedings of ACM DEBS 2015, pp. 344–347 (2015)
5. Chen, T., Bahsoon, R., Yao, X.: A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems. ACM Comput. Surv. **51**, 61 (2018)
6. Fernandez, R.C., Migliavacca, M., Kalyvianaki, E., Pietzuch, P.: Integrating scale out and fault tolerance in stream processing using operator state management. In: Proceedings of ACM SIGMOD 2013, pp. 725–736 (2013)
7. Gedik, B., Schneider, S., Hirzel, M., Wu, K.L.: Elastic scaling for data stream processing. IEEE Trans. Parallel Distrib. Syst. **25**(6), 1447–1463 (2014)
8. Gulisano, V., Jiménez-Peris, R., Patiño Martínez, M., Soriente, C., Valduriez, P.: StreamCloud: an elastic and scalable data streaming system. IEEE Trans. Parallel Distrib. Syst. **23**(12), 2351–2365 (2012)
9. Heinze, T., Roediger, L., Meister, A., Ji, Y., et al.: Online parameter optimization for elastic data stream processing. In: Proceedings of ACM SoCC 2015, pp. 276–287 (2015)
10. Liu, X., Dastjerdi, A.V., Calheiros, R.N., Qu, C., Buyya, R.: A stepwise auto-profiling method for performance optimization of streaming applications. ACM Trans. Auton. Adapt. Syst. **12**(4), 24 (2018)
11. Lohrmann, B., Janacik, P., Kao, O.: Elastic stream processing with latency guarantees. In: Proceedings of IEEE ICDCS 2015, pp. 399–410 (2015)
12. Lombardi, F., Aniello, L., Bonomi, S., Querzoni, L.: Elastic symbiotic scaling of operators and resources in stream processing systems. IEEE Trans. Parallel Distrib. Syst. **29**(3), 572–585 (2018)
13. Mencagli, G.: A game-theoretic approach for elastic distributed data stream processing. ACM Trans. Auton. Adapt. Syst. **11**(2), 13 (2016)
14. Sajjad, H.P., Danniswara, K., Al-Shishtawy, A., Vlassov, V.: SpanEdge: towards unifying stream processing over central and near-the-edge data centers. In: Proceedings of 2016 IEEE/ACM Symposium on Edge Computing, pp. 168–178 (2016)
15. Weyns, D., et al.: On patterns for decentralized control in self-adaptive systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II. LNCS, vol. 7475, pp. 76–107. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_4
16. Xu, L., Peng, B., Gupta, I.: Stela: enabling stream processing systems to scale-in and scale-out on-demand. In: Proceedings of IEEE IC2E 2016, pp. 22–31 (2016)
17. Zhang, Q., Zhang, Q., Shi, W., Zhong, H.: Firework: data processing and sharing for hybrid cloud-edge analytics. IEEE Trans. Parallel Distrib. Syst. **29**(9), 2004–2017 (2018)