Dynamic Adaptation of Urgent Applications in the Edge-to-Cloud Continuum

Daniel Balouek, Hélène Coullon

Dynamic Adaptation of Urgent Applications in the Edge-to-Cloud Continuum

Daniel Balouek and Hélène Coullon

IMT Atlantique, INRIA, LS2N, UMR CNRS 6004, F-44307 Nantes, France
daniel.balouek@inria.fr
helene.coullon@imt-atlantique.fr

Abstract. The integration of Urgent Computing is essential in order to adhere to stringent time and quality constraints of emerging distributed applications, hence facilitating efficient decision-making processes in numerous fields. Adaptation of such applications to produce outcomes within the desired confidence range and defined time interval can be of great benefit, especially in distributed and heterogeneous execution contexts. This study provides a justification for the necessity of dynamic adaptation in applications that are time-sensitive. Furthermore, we present our viewpoint on time-sensitive applications and undertake a thorough analysis of the underlying principles and challenges that need to be resolved in order to accomplish this goal. This research aims to provide a comparative analysis of our suggested vision for adaptation in contrast to the existing literature. We provide a comprehensive explanation of the architectural framework that we plan to construct, and conclude with discussing some on-going challenges.

Keywords: Urgent Computing · Computing Continuum · Dynamic Adaptation · Decentralized System

1 Introduction

Recent years have witnessed an increase in global emergencies, from pandemics that claim lives and livelihoods to risks of flooding and food harvesting challenges brought on by climate change. Society at large is wrestling with these pressing challenges, which warrant for solutions guided by science [26]. Advanced information and communication technologies, such as IoT, Artificial Intelligence, big data analytics, and super computers (*e.g.,* Clouds, High Performance Computing), have been proven to be more and more helpful in predicting scenarios' outcomes early enough to prevent disastrous situations or lessen their harmful effects. During the COVID-19 pandemic, supercomputers and data systems have worked together with unprecedented efficiency, from understanding the structure of the SARS-CoV-2 virus to modeling its spread, from therapeutics to vaccines, from medical response to managing the virus's impacts [17, 3]. Similarly, the VESTEC European FET project fuses super computers and real-time data for urgent decision-making [11].

Urgent computing aims at facilitating scientific applications that leverage distributed data sources to support critical decision-making in a timely manner. Overall, urgent applications consist of the identification of events from distributed data sources and the triggering of appropriate reactions to accelerate response [24, 19]. The time scales of the underlying processes vary from days in case of social unrest events, to hours in flood protection systems, to minutes in ship safety, and to seconds in earthquakes or tsunamis [10]. The time available for decision support therefore varies in different application domains. An urgent application is typically modeled as a workflow of components exchanging discrete or continuous flows of data.

Low-latency connections and distributed servers, illustrated by the Edge-to-Cloud Continuum (Computing Continuum, or Continuum for short), are demonstrating new potential for urgent computing. Running applications on an aggregation of resources spanning the edge of the network, where data are generated, to the cloud and HPC resources at the network's core, is an opportunity for steering computations and numerical models to help reduce latency while increasing flexibility [7, 22]. For example, the Tactile Internet is expected to enable the matching of specific needs in one physical location with the best skill in another location. Building on 5G-like connectivity and haptic encoders, this technology will require an end-to-end latency below 1ms with minimal outage [30, 2]. Due to these requirements, the Computing Continuum is an excellent enabler for urgent applications due to its holistic approach that exploits end-users, resources, and services at the logical extreme of the network and throughout the data path [6] [9].

The realization of urgent applications presents specific requirements and constraints due to the nature and distribution of the data, the complexity of the models involved, the stringent error thresholds, and the strict time constraints. Identifying and mapping various user concerns (such as functional and non-functional requirements, restrictions for response time, solution quality, data resolution, cost, energy use, etc.) against a heterogeneous and dynamic environment warrant the design of adaptation mechanisms. Examples of such adaptations are choosing suitable workflow components and services, modifying application parameters, modifying the connections between components of the workflow, changing the required constraints when provisioning resources (*e.g.,* CPU, RAM, network, etc.) Such mechanisms must be able to address cost/benefit trade-offs of the application.

This paper proposes a vision of how adaptation mechanisms can be natively handled when designing urgent applications across the Continuum. The remainder of the paper is organized as follows: Section 2 gives a motivating use-case for our vision with an Early Earthquake Warning (EEW) workflow, which combines data streams from geo-distributed seismometers and high-precision GPS stations to detect large ground motions; Section3 presents our vision of urgent applications through the modeling of both the development and management aspects; Section 4 presents elements of solutions and open challenges around our vision, and finally, Section 5 concludes this work.

2 Motivating use case

In this section, we introduce the Earthquake Early Warning use case that motivated this work. We also discuss the challenges associated with executing distributed applications on the Computing Continuum.

Earthquakes are among the most destructive natural disasters. Networks of distributed seismic instruments on various scales are used for earthquake detection. *Earthquake Early Warning* (EEW) systems provide earthquake alerts before the shaking damage of a seismic event reaches sensitive areas, giving governments and communities a time window of seconds to minutes to take protective actions.

EEW can be described as a *classification problem* in which high-frequency seismic data stream from multiple sensors are processed to infer classes indicating the magnitude of the seismic event in a timely manner. Related efforts include novel algorithms recently developed to locate earthquakes and to calculate their magnitudes using P- and S-wave energy [31]. Recently, ShakeAlert proposed detecting and disseminating EEW alerts using smartphones, relying on the fact that they have become ubiquitous to the public [28, 21].

EEW is a workflow typically composed of four components: (1) the *preprocessing* component that collects and cleans raw data obtained from sensors; (2) the *classification* component that relies on window-based processing to gather measurements streamed by each sensor, and performs classification on this streamed data; (3) the *prediction* component that filters and aggregates the data streams by regions using a bag-of-words representation to calculate the final prediction of eventual seismic events; (4) the *alert* component that is responsible for broadcasting alerts based on class predictions (normal activity, medium earthquake, or large earthquake). Some of these components may run continuously somewhere in the Continuum, but when initial seismic waves are detected, this complete workflow has to be deployed in a timely manner.

Traditionally, EEW is executed in a fully centralized fashion with data from sensors being sent to clouds (private or public) which is compatible with a static way of designing the application. However, this vision of EEW will not withstand potential destruction or unavailability of the infrastructure (fault tolerance), and thus is not reliable. Instead, we would like to leverage heterogeneous geographically distributed resources across the Continuum, which requires applications to adapt dynamically their codes, communication protocols, etc. according to the dynamically chosen hosting resource. For instance, if deployed within the Cloud, the *prediction* component can leverage heavy stream processing frameworks such as Flink, whereas if deployed at the Edge on small devices, low-level C libraries and MPI would be preferable.

In previous work [16, 8] we proposed moving part of the sensor data processing toward the Edge to speed up detection and enhance fault tolerance. Figure 1 illustrates this static Edge deployment. However, if we have shown that different versions of the application running on different parts of the Continuum may have different behaviors, we did not address the automatic and dynamic adaptation of the application.

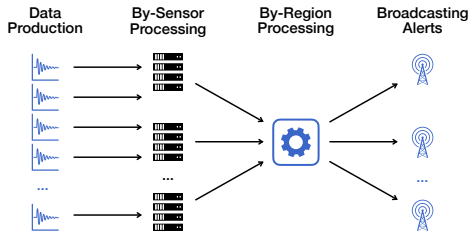

Fig. 1. An illustration of Distributed MultiSensor Earthquake Early Warning use-case (DMSEEW), previously proposed in [7]. Seismic sensors located in the Edge send measurements to gateways in the network which preprocess that data. Those preprocessed data are sent to cloud servers which complete that data processing and eventually broadcast earthquake alerts.

We use Figure 2 (extracted from [8]) to further illustrate the need for adaptation in this use-case. Figure 2 presents the latency resulting from processing one seismic event in a previous evaluation of our EEW implementation [8]. Using the *gros* cluster on the Grid'5000 experimental testbed, we deployed a prototype of the application under different network configurations: a 10Gb link between the Edge and Fog layers, and different links (2G, 3G, 10 Gb) between the Fog and the Cloud layers. *Edge* designates that each component is executed on a separated layer of the Continuum whereas *Cloud-only* means all components are executed on the Cloud. Of course, when running in the Edge or at the Cloud level, the components also have to be adapted (*e.g.,* libraries, frameworks, etc.).

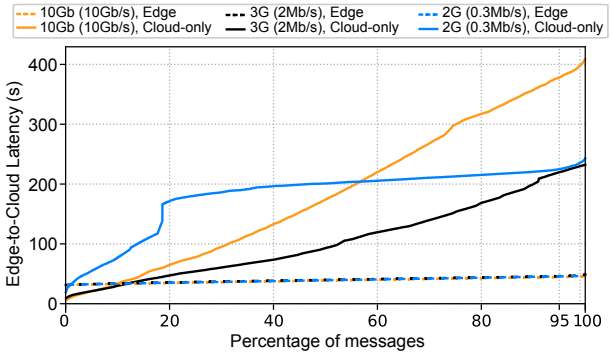

Fig. 2. Latency resulting from processing one seismic event under different applications and infrastructure scenarios, extracted from [8].

The performance of the application is influenced by two factors: the network and the bottlenecks resulting from the interaction between the different libraries (in the context of the current paper, *components*). The different values were obtained in separated experiments to understand the behavior of the application

under different conditions. Currently, there is no mechanism to switch from one configuration to another based on availability of the infrastructure and the events occurring in the system, but also there is a need to consider the cost and feasibility of the operations when such reconfiguration is needed, which motivates the need for adaptation mechanisms.

3 Our vision of Urgent Computing

First, we see the Continuum as an external infrastructure offered by a provider. We consider that we do not have to handle this Continuum but instead that the Continuum is offering APIs to ask for resources (*e.g.,* virtual or physical machines). In particular, we think that, in addition to the usual Cloud APIs that provides a type of resource as well as specific features (RAM, CPU, GPU etc.), one can also provision specific geographical locations in the Continuum.

Definition 1. *We model an urgent application as a pair* $(\mathcal{D}, \mathcal{M})$*, where* $\mathcal{D}$ *is specified by the application developers (functional and nonfunctional developers), and* $\mathcal{M}$ *is specified typically by DevOps engineers or system administrators to manage the application deployment and life-cycle (i.e., adaptation).*

This vision follows the one presented in [15], a survey on component-oriented reconfiguration.

3.1 Development part of an urgent application

Definition 2. *The development of an urgent application* $\mathcal{D}$ *is a tuple* (C, P, m, p_i, p_c) *with:*

- C *the set of software components available for the application, some of them being produced by the current developer, others being made available from the community;*
- P *the set of patterns or strategies of service assemblies;*
- m *a labeled finite state machine that indicates from which to which pattern in* P *the application may switch at runtime and under which conditions;*
- p_i *the initial pattern in* P*;*
- p_c *the current pattern in* P*.*

Definition 3. *A pattern* $p \in P$ *is defined as an assembly of components* (C_p, L_p) *where* $C_p \subseteq C$ *is the set of component instances in the pattern, and* L_p *is the set of connections or links between components.*

In urgent applications, the nature of the connections is typically data-driven, and thus the assembly of components can be considered as a dataflow or a workflow. However, we want to add configurations to the links between components, under the form of connectors in the CBSE literature [15], or service-mesh approaches [18, 25]. For example, we may want to redirect 50% of data to one component or another dynamically according to events. Hence, a link is configurable in our vision of urgent components.

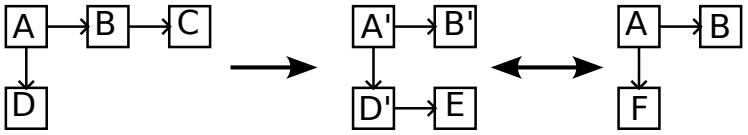

Fig. 3. An illustration of m, an object (state machine) modeling from which to which pattern the urgent application can switch. The pattern on the left is the initial pattern. In this example the initial pattern cannot be reached after a first switch.

Definition 4. *An urgent component $c \in C$ is defined as a tuple $(S_f, S_{nf}, V^c, C^c_{st})$, where S_f is the set of functional services of the urgent component, S_{nf} is the set of non-functional services of the component, V^c a set of variables to model the state, behavior or parameters of the component, and C^c_{st} is the set of constraints on variables.*

A component is associated with a set of variables V^c to model the component (state, behavior, parameters), thus representing either the static or dynamic knowledge (modeling) on each component. These variables can then be used within constraints C^c_{st} that should be satisfied on a set of variables. One typical constraint that we consider at the level of the component is the geographical location of the component.

A non-functional service is a piece of software responsible for features not related to the functional aspect of the component, for instance how the component communicates (*i.e.,* communication protocol), handles privacy, or energy etc. Services are deployed (instantiated) from types known in advance, and thus one service can be switched to another if compatible. At the level of one component, this catalog of services should be quite small. It is offered by the developer. Both the elements of S_f and S_{nf} are considered as configurable. Functional and non-functional services will typically change their configuration according to the constraints imposed by the infrastructure. For instance, if placed in an IoT device, the urgent component communicates through the LoRa protocol and optimizes the energy consumption rather than the performance. As a consequence the non-functional service responsible for communications may be modified, and the version of the functional service may need to be a downgraded version.

Each service is also associated with a set of variables V^s modeling their parameters, state, behavior, etc., and these variables can then be used within constraints C^s_{st} that should be satisfied on a set of variables. For example, we could have a constraint on the amount of memory required by a service.

Definition 5. *A service is defined as a tuple (t, V^s, C^s_{st}), where t is the type of the service, V^s is the set of variables of the service, and C^s_{st} is the set of constraints to satisfy for the service.*

3.2 Management part of an urgent application

The management $\mathcal{M}$ of an urgent application is a complex task responsible for deploying and then adapting the application dynamically at run-time accord-

ing to events. However, because of scalability and fault tolerance issues when considering urgent cases, the management cannot be considered as a centralized process.

We model the management of the application as a set of controllers C_t, one around each component in C involved in the application. Each controller is responsible for an autonomic infinite loop MAPE-K [20] for its own component: monitoring the component and its environment (through sensors, infrastructure, or application instrumentation); deciding which new configuration should be targeted according to the monitoring knowledge (inference through constraints, by applying rules and policies); planning an adaptation program to move from the current to the targeted configuration (writing a program to modify the state of the component [13, 14]); and execute this plan (*i.e.,* program). Controllers, by being local to one component, have the possibility of adapting locally if isolated from the rest of the application, but in a normal context neighbor controllers can collaborate to take better decisions [4].

It is assumed that each controller component operates on its own unique time scale, independent from the others. For example, monitoring can be an ongoing process, whereas analysis of the data that has been acquired most likely occurs only on occasion. Both processes are used to analyze the information that was collected. The planning is done on a consistent basis or in reaction to changes, and the executions are done as a direct result of the planning.

Definition 6. *The management of an urgent application $\mathcal{M}$ is a tuple (C_t, P, m, p_i, p_c) where C_t is the set of controllers, P, m, p_i, and p_c respectively the same set of patterns, state machine, initial and current patterns as in $\mathcal{D}$.*

Definition 7. *Two controllers are neighbors if their components are connected in the current pattern (graph, assembly) $p \in P$ of the urgent application. The neighbors of a controller are denoted $neighbors(c_t)$.*

Definition 8. *A controller $c_t \in C_t$ is a piece of software responsible for running an autonomic infinite loop MAPE-K [20] for its own component, denoted $component(c_t)$.*

We denote by X^{c_t} the local projection of any piece of the above knowledge in $(\mathcal{D}, \mathcal{M})$ to a controller $c_t \in C_t$. A local projection contains the information that relates to the local component that is controlled, and the neighbor components in the current pattern $p \in P$.

Definition 9. *The local knowledge (K) of each controller c_t is composed of a subpart of the application model $(\mathcal{D}, \mathcal{M})$ as follows:*

- *in $\mathcal{D}$, the local projection of the set of components C^{c_t};*
- *in $\mathcal{M}$, the local projection of the set of controllers $C_t^{c_t}$;*
- *for both $\mathcal{D}$ and $\mathcal{M}$, the local projection of the set of patterns P^{c_t}, and the initial and current patterns $p_i^{c_t}$ and $p_c^{c_t}$;*
- *for both $\mathcal{D}$ and $\mathcal{M}$, the labeled state machine m.*

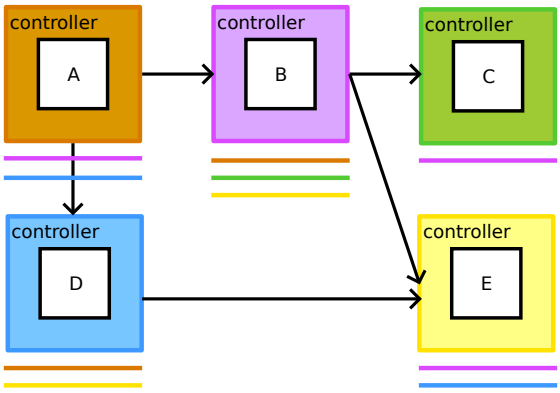

Fig. 4. An illustration of a pattern with five components and their associated controllers (one color per controller). The colored lines under components represent the local projection of the controller associated with the color, in other words the knowledge of each controller. For instance, the controller of the component B (purple) has knowledge of components A, C and E represented by purple lines.

The local projections of a pattern can be seen as a subgraph of the overall component assembly that contains as vertices

$$component(c_t) \cup neighbors(component(c_t)) \tag{1}$$

As always in distributed systems, we assume that the knowledge shared by controllers is consistent, which is a challenging issue to address. However, to facilitate the global consistency in our modeling, we use a local projection on the local component and its neighbors. The labeled state machine m, which represents the rules under which an urgent application may change its pattern, is the only piece of knowledge that is centralized in this vision. This state machine is given by the developer. However, as will be discussed in the challenges of this paper, it is possible that m change through time, in which case a consistent vision of m is difficult to guarantee.

A controller can adapt many aspects related to an urgent component such as: (1) change the constraints of the component related to the Continuum C^c_{st}; (2) change the versions of the services within the component while respecting the composability (compatibility of output/input data) (S_f, S_nf); (3) change the configuration of the services of a component (t, V^s, C^s_{st}); (4) change the current pattern p_c to another one by following m, thus changing the topology between components and involved components; (5) change the configuration of the link between components; and (6) spawn, or ask for the destruction of a neighbor component. The "urgent" nature of the domain makes adaptation sensitive to time, which will be discussed in the challenges.

4 Elements of solution and open scientific challenges

As detailed in the previous section, each controller of a component handles a local autonomic loop MAPE-K. However, the loops cannot be strictly local and requires to collaborate with their neighbors so that the overall application does not deviate from a stable state. In particular, the (A), (P) and (E) steps require collaborations between controllers by exchanging some elements of their local knowledge (K) and by reaching a consensus in their decisions [4]. For instance, if deciding locally to apply a switch of pattern, the decision should spread across all the controllers and should be compatible for all controllers. We give some elements of solutions in the rest of this section.

(K)nowledge - In our vision the first element to study is how to concretely model the programming and management of urgent applications. We think we can build our vision on three domains of the literature. First, we think that leveraging modern *workflow engines* based on the FaaS paradigm (Function-as-a-Service), such as Argo workflows [1], or other engines adapted to heterogeneous workflows [12], is an interesting starting point to model urgent applications. Second, this vision of an urgent application as a workflow of components will have to be coupled with the *service-oriented* and *microservices-oriented* trend (*e.g.,* REST, gRPC, GraphQL APIs). In particular, in our vision each component of the workflow is divided into services. Third, we think we will have to model the variability of components and services composing urgent applications with the help of *feature models*. Feature models are used in software engineering to capture the various configurations that can be selected or enabled in a software system. A feature model consists of a set of features, relationships between features, and constraints on their combinations. Features represent distinct functionalities, or behaviors that can be included or excluded in the software system. Coupling these three domains is not trivial and requires a software-engineering-oriented study. For example, coupling feature models with workflows has been studied in the past [1], as well as coupling feature models and micro-services [29], but the three of them together is a new challenge.

(A)nalysis - The second element to study is how to decide which local configuration to target according to internal and external events and to some degree of knowledge from neighboring controllers. In particular, if all controllers take a local decision, will the decisions be compatible , and how to make sure of it? We think that the three domains could be combined to handle this aspect. First, we think that distributed algorithms such as consensus, gossip, and leader election algorithms [23, 4] could be helpful to reach a convergence between controllers in their decisions. Second, we think that auto-stabilizing versions of these algorithms ccould be important to avoid too many communications but still guarantee the convergence (stabilization) of the overall system [5]. Third,

[1] https://argoproj.github.io/argo-workflows/

we think that because of the huge variability when adapting urgent application (large search space), and because of the urgent nature of the decisions, machine learning could be of interest. In particular, perhaps reinforcement learning and federated learning could be studied so that decisions become better through time and so that learning models are trained in a distributed fashion among the controllers. We also think that a learning approach could be supplemented with more traditional approaches to solve satisfaction problems such as constraint solvers or SAT/first-order logic solvers to keep a certain level of explainability in the decision.

(P)lanning and the (E)xecution - In order to tackle many controllers, each with its own autonomic loop, we need languages to define adaptation actions (*e.g.,* changing the pattern, changing internal behaviors of components or services, changing the links between components, etc.), and an associated engine to run these actions in a decentralized fashion. Indeed, if each controller takes its own decision, controllers are not independent because components are connected together, thus requiring some coordination when applying changes. For example, if updating one component, thus requiring it be interrupted, the components that depend on it have to adapt temporarily their own behavior.

In the literature planning and executing an adaptation is often referred as a reconfiguration problem: a reconfiguration language offers programming support to structure and expresses the actions to perform to change the system state; a reconfiguration engine executes and coordinates actions [14, 13]; and a reconfiguration inference system automatically generates the set of actions to apply (the plan) [27] according to the current and target configurations as inputs. However, existing solutions are mainly centralized in the literature [15, 4].

Moreover, the usual reconfiguration systems in the literature do not tackle the reconfiguration of the links between components of the system [15]. For this reason, we think that concepts of service-mesh [25, 18] could be of interest to couple with the usual reconfiguration solutions.

5 Conclusion

Urgent applications are of significant interest in the literature due to the increase of global challenges. The availability of data and pervasive computing promise potential for supporting timely decision-making across the Computing Continuum. In this paper, we presented a vision for modeling urgent applications, with a particular focus on adaptation aspects across heterogeneous and geo-distributed resources. Building on an earthquake early warning motivating use-case, we propose an approach based on component-based reconfiguration. We expect the realization of this vision to impact cyber-infrastructures and similar shared sensors platform in a way that will manage time-sensitive applications and associated requirements with few assumptions during the design of applications.

References

1. Acher, M., Collet, P., Lahire, P., France, R.B.: Managing Variability in Workflow with Feature Model Composition Operators. In: International Conference on Software Composition 2010 (2010)
2. Aijaz, A., Dohler, M., Aghvami, A.H., Friderikos, V., Frodigh, M.: Realizing the Tactile Internet: Haptic Communications over Next Generation 5G Cellular Networks. IEEE Wireless Communications (2017)
3. Alashhab, Z.R., Anbar, M., Singh, M.M., Leau, Y.B., Al-Sai, Z.A., Alhayja'a, S.A.: Impact of coronavirus pandemic crisis on technologies and cloud computing applications. Journal of Electronic Science and Technology **19**(1), 100059 (2021)
4. Alidra, A., Bruneliere, H., Coullon, H., Ledoux, T., Prud'Homme, C., Lejeune, J., Sens, P., Sopena, J., Rivalan, J.: SeMaFoR - Self-Management of Fog Resources with Collaborative Decentralized Controllers. In: SEAMS 2023 - IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (2023). https://doi.org/10.1109/SEAMS59076.2023.00014
5. Altisen, K., Devismes, S., Dubois, S., Petit, F.: Introduction to Distributed Self-Stabilizing Algorithms (2019). https://doi.org/10.2200/S00908ED1V01Y201903DCT015
6. Balouek-Thomert, D., Renart, E.G., Zamani, A.R., Simonet, A., Parashar, M.: Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows. The International Journal of High Performance Computing Applications (2019)
7. Balouek-Thomert, D., Rodero, I., Parashar, M.: Harnessing the computing continuum for urgent science. ACM SIGMETRICS Performance Review (2020)
8. Balouek-Thomert, D., Silva, P., Fauvel, K., Costan, A., Antoniu, G., Parashar, M.: Mdsc: modelling distributed stream processing across the edge-to-cloud continuum. In: Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion. pp. 1–6 (2021)
9. Beckman, P., Dongarra, J., Ferrier, N., Fox, G., Moore, T., Reed, D., Beck, M.: Harnessing the computing continuum for programming our world (2019)
10. Boukhanovsky, A.V., Krzhizhanovskaya, V.V., Bubak, M.: Urgent computing for decision support in critical situations. Future Generation Computer Systems (2018)
11. Brown, N., Nash, R., Gibb, G., Prodan, B., Kontak, M., Olshevsky, V., Der Chien, W.: The role of interactive super-computing in using hpc for urgent decision making. In: High Performance Computing: ISC High Performance 2019 International Workshops, Frankfurt, Germany, June 16-20, 2019, Revised Selected Papers 34. pp. 528–540. Springer (2019)
12. Cadorel, E., Coullon, H., Menaud, J.M.: Handling heterogeneous workflows in the cloud while enhancing optimizations and performance. In: 2022 IEEE 15th International Conference on Cloud Computing (CLOUD) (2022). https://doi.org/10.1109/CLOUD55607.2022.00021
13. Chardet, M., Coullon, H., Pérez, C.: Predictable Efficiency for Reconfiguration of Service-Oriented Systems with Concerto. In: CCGrid 2020 : 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (2020). https://doi.org/10.1109/CCGrid49817.2020.00-59
14. Chardet, M., Coullon, H., Robillard, S.: Toward Safe and Efficient Reconfiguration with Concerto. Science of Computer Programming (2021). https://doi.org/10.1016/j.scico.2020.102582

15. Coullon, H., Henrio, L., Loulergue, F., Robillard, S.: Component-based distributed software reconfiguration: A verification-oriented survey. ACM Comput. Surv. (2023). https://doi.org/10.1145/3595376, just Accepted
16. Fauvel, K., al.: A Distributed Multi-Sensor Machine Learning Approach to Earthquake Early Warning. In: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (2020)
17. Friji, H., Hamadi, R., Ghazzai, H., Besbes, H., Massoud, Y.: A generalized mechanistic model for assessing and forecasting the spread of the covid-19 pandemic. Ieee Access **9**, 13266–13285 (2021)
18. Ganguli, M., Ranganath, S., Ravisundar, S., Layek, A., Ilangovan, D., Verplanke, E.: Challenges and opportunities in performance benchmarking of service mesh for the edge. In: 2021 IEEE International Conference on Edge Computing (EDGE) (2021). https://doi.org/10.1109/EDGE53862.2021.00020
19. Gibb, G., Nash, R., Brown, N., Prodan, B.: The technologies required for fusing hpc and real-time data to support urgent computing. In: 2019 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC). pp. 24–34. IEEE (2019)
20. Kephart, J., Chess, D.: The vision of autonomic computing. Computer (2003)
21. Kohler, M.D., Smith, D.E., Andrews, J., Chung, A.I., Hartog, R., Henson, I., Given, D.D., de Groot, R., Guiwits, S.: Earthquake Early Warning ShakeAlert 2.0: Public Rollout. Seismological Research Letters **91**(3), 1763–1775 (04 2020). https://doi.org/10.1785/0220190245, https://doi.org/10.1785/0220190245
22. Kumar, R., Baughman, M., Chard, R., Li, Z., Babuji, Y., Foster, I., Chard, K.: Coding the computing continuum: Fluid function execution in heterogeneous computing environments. In: 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 66–75. IEEE (2021)
23. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. (1982)
24. Leong, S.H., Kranzlmüller, D.: Towards a general definition of urgent computing. Procedia Computer Science **51**, 2337 – 2346 (2015), international Conference On Computational Science, ICCS 2015
25. Li, W., Lemieux, Y., Gao, J., Zhao, Z., Han, Y.: Service mesh: Challenges, state of the art, and future research opportunities. In: 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE) (2019)
26. Peleg, K., Bodas, M., Hertelendy, A.J., Kirsch, T.D.: The covid-19 pandemic challenge to the all-hazards approach for disaster planning. International Journal of Disaster Risk Reduction **55**, 102103 (2021)
27. Robillard, S., Coullon, H.: SMT-Based Planning Synthesis for Distributed System Reconfigurations. In: FASE 2022 : 25th International Conference on Fundamental Approaches to Software Engineering (2022). https://doi.org/10.1007/978-3-030-99429-7_15
28. Rochford, K., Strauss, J.A., Kong, Q., Allen, R.M.: Myshake: Using human-centered design methods to promote engagement in a smartphone-based global seismic network. Frontiers in Earth Science **6**, 237 (2018)
29. Sousa, G., Rudametkin, W., Duchien, L.: Automated setup of multi-cloud environments for microservices applications. In: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD) (2016)
30. Van Den Berg, D., Glans, R., De Koning, D., Kuipers, F.A., Lugtenburg, J., Polachan, K., Venkata, P.T., Singh, C., Turkovic, B., Van Wijk, B.: Challenges in Haptic Communications Over the Tactile Internet. IEEE Access (2017)
31. min W., Y., liang T., T.: A virtual sub-network approach to earthquake early warning. Bull. Seism. Soc. Am pp. 2008–2018 (2002)