Towards a Generic Computation Model for Smart City Platforms

Subhav Pradhan, Abhishek Dubey, Sandeep Neema, and Aniruddha Gokhale
Institute for Software Integrated Systems, Department of EECS
Vanderbilt University, Nashville, TN, USA
Email:{subhav.m.pradhan, abhishek.dubey, sandeep.neema, a.gokhale}@vanderbilt.edu

***Abstract*—Smart emergency response systems, smart transportation systems, smart parking spaces are some examples of multi-domain smart city systems that require large-scale, open platforms for integration and execution. These platforms illustrate high degree of heterogeneity. In this paper, we focus on software heterogeneity arising from different types of applications. The source of variability among applications stems from (a) timing requirements, (b) rate and volume of data they interact with, and (c) behavior depending on whether they are stateful or stateless. These variations result in applications with different computation models. However, a smart city system can comprise multi-domain applications with different types and therefore computation models. As such, a key challenge that arises is that of integration; we require some mechanism to facilitate integration and interaction between applications that use different computation models. In this paper, we first identify computation models based on different application types. Second, we present a generic computation model and explain how it can map to previously identified computation models. Finally, we briefly describe how the generic computation model fits in our overall smart city platform architecture.**

***Index Terms*—Generic computation model, Dataflow graph, Smart city platform**

I. Introduction

Smart cities promise to enrich the lives of residents by providing better services while also empowering them to make efficient and informed decisions. Implementing smart cities requires large-scale platforms that facilitate collaboration between multi-domain systems, such as Electric Grid, Water Supply, Transportation Networks, Emergency Services, etc. We call these platforms *smart city platforms* and define them as loosely connected, multi-domain platforms that "virtualize" their heterogeneous resources to provide open platforms capable of simultaneously hosting multiple smart city systems.

In general, smart city systems are designed to collect data, process collected data, transmit data, and analyze data. In the context of smart city systems, data collection usually happens at the edge because that is where edge devices with sensors are deployed to monitor surrounding environments. Edge devices are getting more sophisticated as they have more computational resources, better battery life, and they are equipped with actuators allowing them to not only sense but interact with their surrounding environment. This aspect of smart city systems is cyber-physical in nature. Therefore, we need to view smart city platforms as Cyber-Physical Systems (CPS). Concepts similar to smart city platforms have been previously referred to as next generation CPS [1], large-scale CPS [2], and extensible CPS [3].

Unlike data collection, processing and analyzing data are resource intensive tasks that usually cannot be executed on resource-constrained edge nodes. In traditional large-scale CPS, this problem was solved using backend resources owned and maintained in-house [2]. This results in isolated islands of domain-specific platforms that incur significant construction and maintenance costs. Furthermore, integration across platforms becomes a very challenging task. Another approach to solving this issue is to take advantage of cloud computing technology resulting in a complex computing paradigm that involves deploying different kinds of applications on different kinds of resources. Resources can be provided by edge nodes, cloudlets and mobile clouds, private clouds, or public clouds. Applications deployed on edge nodes incur minimal, if any, network latency since they are close to the source of data. Whereas, applications deployed on public cloud resources incur significant network latency as they use remote computational resources that can be located far from their corresponding sources (e.g.,edge nodes with related sensors) and they require sensor data to be transmitted from different sources.

The possibility of different application types is a source of software heterogeneity in smart city platforms. Applications can be of different types due to varying (a) timing requirements, as they might need real-time, near real-time, or non real-time responsiveness, (b) rate and volume of data they interact with, and (c) behavior, as they can be stateful or stateless (*i.e.,* functional). Different application types can result in different computation and communication patterns. For example, a cyber-physical application that interacts with a physical environment via sensors and actuators requires as close to real-time responsiveness as possible since they need to react to real-time streams of sensor stimuli and control appropriate actuators [4]. This kind of application is usually implemented as a closed loop control application and deployed as close as possible to the target environment. In comparison, a long running, computation heavy, big-data application is usually implemented using some notion of a computation graph supported by existing dataflow engines such as Storm [5], Spark [6], or TensorFlow [7].

Given that there can be applications with varying compu-

978-1-5090-1169-8/16/$31.00 ©2016 IEEE

tation patterns, realizing smart city platforms requires us to devise solutions that facilitate integration of and interaction between applications with varying computation models. In this paper, we present a generic computation model to address this problem. We first identify common computation patterns, and then we describe our generic computation model in detail, and finally show how this computation model fits within our smart city platform called Cyber-pHysical Application aRchItecture with Objective-based re-configuraTion (CHARIOT).

The rest of this paper is organized as follows: Section II briefly describes a smart city platform architecture; Section III presents different challenges; Section IV describes our contributions in detail; Section V compares our work with related literature; Finally, Section VI provides concluding remarks alluding to future work.

II. Smart City Platform Architecture

Realizing smart city platforms requires distributed architectures that allow us to view CPS from a collaborative perspective, where it is important to utilize advancement in other computing paradigms such as cloud computing to realize a complex and heterogeneous architecture. Below, we describe one such architecture by first presenting a resource model. Second, we present different types of applications that can be deployed on smart city platforms and we identify existing computation patterns that are used for these applications.

A. Resource Model

The physical computing infrastructure available to smart parking systems comprises *computation* and *communication* resources. Computation resources include hardware facilities required to execute computation tasks, wheres, communication resources represent facilitates required for interaction between tasks executing on different computation nodes. This includes communication bandwidth, latency, network topology, and available security measures.

In order to represent collection of above described resources, we introduce the concepts of *computing groups* (CG) and *computing group collections* (CGC). A CG is a collection of physical computing nodes that share a common communication network that can be thought of as a *subnet*. Similarly, a CGC is a collection of one or more computing groups. Any two CGs of a CGC can have a virtual link between them. Presence of a virtual link indicates that entities of the associated CGs can communicate with each other via their corresponding gateway nodes. Different CGs of a CGC can be classified into categories, such that each CG belongs to a category and multiple CGs can belong to the same category.

Given these abstract notions of resource collection, we can easily represent resources provided by a smart city platform as CGCs. To better describe these concepts, in Figure 1 we present a smart city platform that consists of a single CGC comprising four different CGs (a) smart home, (b) smart grid, (c) micro data center and cloudlets, and (d) public cloud. These CGs are divided into three categories described below:

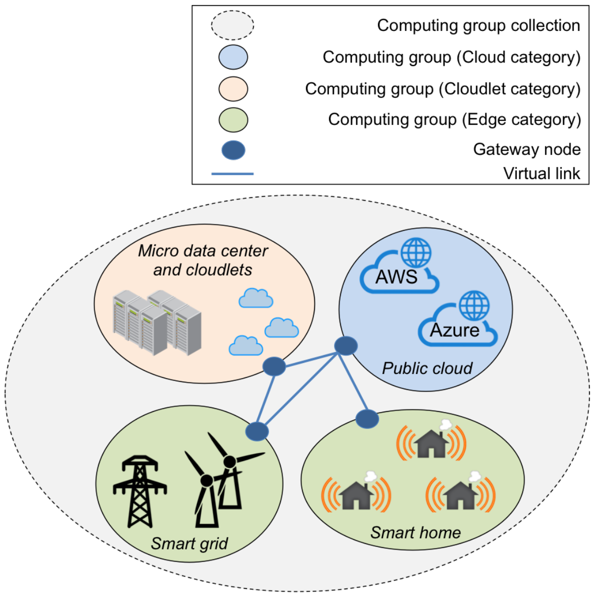
Computing group collection
Computing group (Cloud category)
Computing group (Cloudlet category)
Computing group (Edge category)
Gateway node
Virtual link
Micro data center and cloudlets
AWS
Azure
Public cloud
Smart grid
Smart home

Fig. 1: A smart city platform comprising a single computing group collection composed of four different computing groups of three different categories (a) edge category, (b) cloudlet category, and (c) cloud category.

- **Edge category**: In this category nodes are resource-constraint and they are usually deployed in or close to the target physical environment. Further, edge nodes are generally equipped with sensors and actuators to monitor and control their physical environment. As such, edge nodes are ideal for deploying cyber-physical applications that need to interact with the physical environment in real-time using available sensors and/or actuators. Smart home and smart grid are examples of this category.
- **Cloudlets category**: This category comprises nodes with more computing and storage resources than edge nodes. Unlike public clouds with data centers located in different geographic regions, cloudlets comprise of resources located in specific regions of interest resulting in less communication latency between these resources and edge nodes [8]. Connectivity between edge nodes and cloudlets can span from best-effort, internet-type connectivity over low-latency Local Area Networks (LANs) to real-time industrial Ethernet or field busses. As such, these resources can be used to host (soft) real-time applications.
- **Cloud category**: In this category nodes are highly resourceful and resources can be scaled up or down as modern cloud infrastructures support on-demand elasticity. Data centers that provide these resources are usually located far from edge nodes, as such, the communication latency between these resources and edge nodes are comparatively higher than that between cloudlets and edge nodes. Therefore, this category is well-suited to host long-running computation intensive tasks that do not require real-time responsiveness. Example of this category are various public cloud service providers.

B. Application Types and Existing Computation Patterns

Smart city platforms can hosts different applications. In some cases, these applications are deployed on resources of same computing group, while in other cases, applications are deployed on resources of different computing groups. For the former, consider a smart home application that uses temperature monitors to check indoor temperature and control available thermostats accordingly. For the latter, consider a smart grid application, where different sensor applications are deployed on resources of edge category and short-term analysis and planning applications are deployed on resources of cloudlet category. Finally, a long running evaluation and design application can also be part of the smart grid application; this application will be resource intensive, and therefore, is well-suited to run on resources of cloud category.

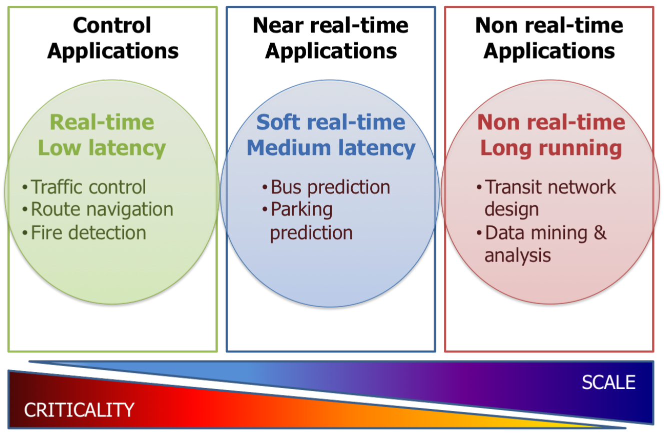

Fig. 2: Different types of applications based on resource requirement, timing requirement, criticality, and scale.

Applications can be of different kinds. One way of differentiating applications is by determining their properties using key characteristics such as resource requirement, timing requirement, criticality, and scale. As shown in Figure 2, we use these characteristics to introduce three different application types (a) control applications, (b) near real-time applications, and (c) non real-time applications. Control applications are those applications that are cyber-physical in nature and by default are critical and require real-time responsiveness. These are low latency applications. Fire detection application in smart homes, and traffic control application in smart transportation are some examples of control applications.

Near real-time applications are those applications that rely on best-effort responsiveness. These applications are medium latency applications that aren't very critical, therefore, these applications do not have strong real-time requirement. Short-term traffic analysis and prediction application, and smart home energy consumption analysis and prediction application are some examples of near real-time applications. Finally, non real-time applications are those applications that are not critical and do not require real-time responsiveness. However, these applications are usually large-scale, long running, computation intensive tasks.

Depending on the above-described application types, corresponding applications can have varying computation patterns resulting in challenges described in Section III. Below we identify relevant exiting computation patterns/models for smart city platforms:

- **Time-driven component assembly**: This pattern involves designing applications as composition of interacting components [9] based on an underlying component model that focuses on assuring domain requirements, which includes non-functional properties such as timeliness. Examples of such component models have been presented in [10]. In this pattern, the computation logic itself is executed either periodically, depending on available timers, or reactively depending on external events, such as message arrival at a component's port. This pattern can be used by control applications.
- **Batch processing**: In this pattern, computations are performed in *batches*, *i.e.*, a collection of non-interactive jobs are executed all at once. This pattern becomes useful in scenarios where high volume data acquisition happens over a period of time. Once data is acquired and stored, it can be processed in batches to obtain computation results. MapReduce [11] is a popular batch processing paradigm. Spark [6] framework, at its core, also supports batch processing, but it also has support for stream processing. TensorFlow [7] is a machine learning related data flow engine that uses batch processing concepts. This pattern can be used by non real-time applications as it cannot support real-time data processing.
- **Stream processing**: This pattern involves processing data in real-time or near/soft real-time. Unlike batch processing, real-time data acquired is not stored for future processing in this pattern. Rather, real-time stream of data is continually fed as input, which is then processed to generate output. What this means is that computations always happen on real-time data as it flows through a system. Storm [5] is an example of a popular stream processing framework. The Spark framework can also be used for stream processing. S4 [12] is another solution that can be used for stream processing. This pattern can be used by near real-time applications.

III. Challenges

Smart city platforms can host different smart city systems simultaneously. These systems are multi-domain; as such, they can be composed of different types of applications. Therefore, a smart city platform requires some mechanism that facilitates collaboration between applications of different types. In order to devise any such mechanism, we need to be able to address the following challenges:

- We need to be able to design and develop applications without having to worry about differences in their type. To resolve this challenge we need an abstraction that allows us to view all applications as the same so that we can model their compositions and interactions. This also allows us to better reason about a system as a whole.

This is the challenge we are addressing in this paper by designing an abstract and generic computation model that can be used to construct systems with varying application types and computation patterns.

- We need to be able to manage heterogeneous applications. Regardless of what an application's type is, we require some mechanism that is capable of initial deployment and any runtime reconfiguration required for it to be resilient. Resilience implies the ability to either tolerate failures or recover from them quickly without impacting top-level functionalities. Our initial work towards resolving this challenge is presented in [13], [14].
- Since smart city platforms are cyber-physical in nature, we need to be able to reason about Quality-of-Service (QoS) of different applications. Also, we need to make sure that the desired QoS of different applications are met throughout their lifetime. An important QoS parameter is deadline; it is of utmost importance to make sure applications with real-time requirements meet their deadline.

IV. Contributions

In this section, we present our current effort towards achieving a generic computation model that fits into CHARIOT, our existing platform for smart city systems. First, we describe our computation model in detail. Second, we show how our computation model fits into the existing CHARIOT architecture.

A. CHARIOT Computation Model

A multi-domain smart city system can comprise applications with varying types resulting in a need for some mechanism to facilitate composition and interaction between the computation patterns identified in Section II-B. To generalize the different computation patterns, the CHARIOT component model represents distributed computations to be a computation graph that results in a data flow network. This approach aligns well with existing data processing engines – such as Storm, Spark, S4, and TensorFlow – as these technologies also rely on some form of a computation graph. For example, in the case of Storm, *topologies* are used to define computation graphs that comprise *spouts*, which represent streams of data sources, and *bolts*, which represents data processing. Edges between nodes of a topology represent data flow. Similarly, in the case of Spark, an application is first divided into jobs, which are then broken down to computation graphs composed of *task stages*. Edges between task stages represents a data flow. S4's computation graphs are composed of nodes called *Processing Elements* and edges called *Streams*. Finally, TensorFlow's computation graphs are composed of nodes, which can be *ops* (operational) or *source ops*, and edges between these nodes are represented by the concept of *tensors*, which are typed multi-dimensional arrays.

In order for a generic computation pattern based on data flow graphs to work, we need to make sure that it is something that we can use for edge applications as well. Traditionally, these real-time, edge applications have been designed using time-driven component assembly II-B, but they can be designed using a data flow graph approach as well; nodes can represent computations, whereas, edges can represent events or dataflow. An added advantage of this approach is that the computation model can easily support a generic reactive model whereby events can be separated from related computations. This is not necessarily the case with existing component models that have tight coupling between events (specially external message related event) and corresponding computation logic. This is mainly because existing computation models are designed to work with a specific communication middleware.

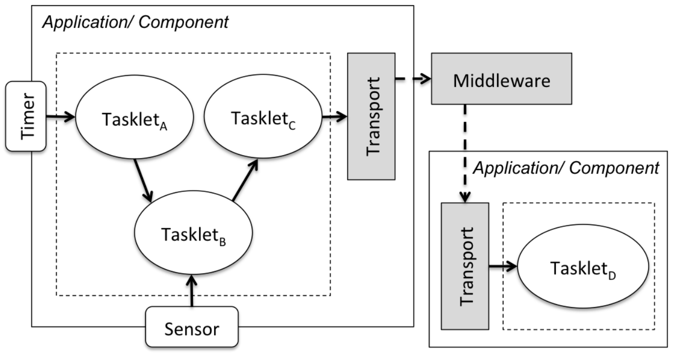

Fig. 3: Overview of the CHARIOT computation model.

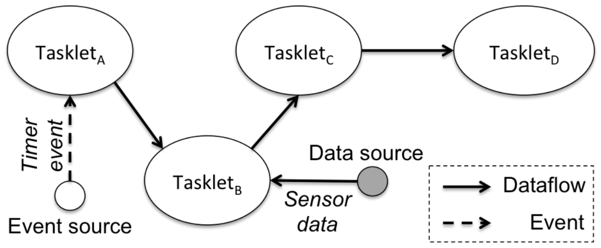

Fig. 4: Computation graph for applications in Figure 3.

Figure 3 presents an overview of the CHARIOT computation model. Each application is composed of one or more components, where each component can contain one or more *tasklets*. In Figure 3, we assume a scenario where applications always contain a single component, as such, an application becomes synonymous to a software component. Tasklets represent computation nodes and edges between tasklets represent dataflow. Tasklets can be both stateful or stateless (functional). Depending on how a computation graph is deployed, a dataflow between tasklets deployed on different physical nodes require some mechanism that facilitates remote communication. For this very purpose, there exists different communication middleware. In order to support various middleware, CHARIOT implements *transport* objects. A transport object is specific to a particular middleware but all transport objects expose standard interfaces that can be used by tasklets to send and/or receive messages. This approach yields a middleware agnostic solution, which at its very core relies on generic data types [3].

A tasklet can also have edges that connect it to an event source and/or data source. An example of an event source is a

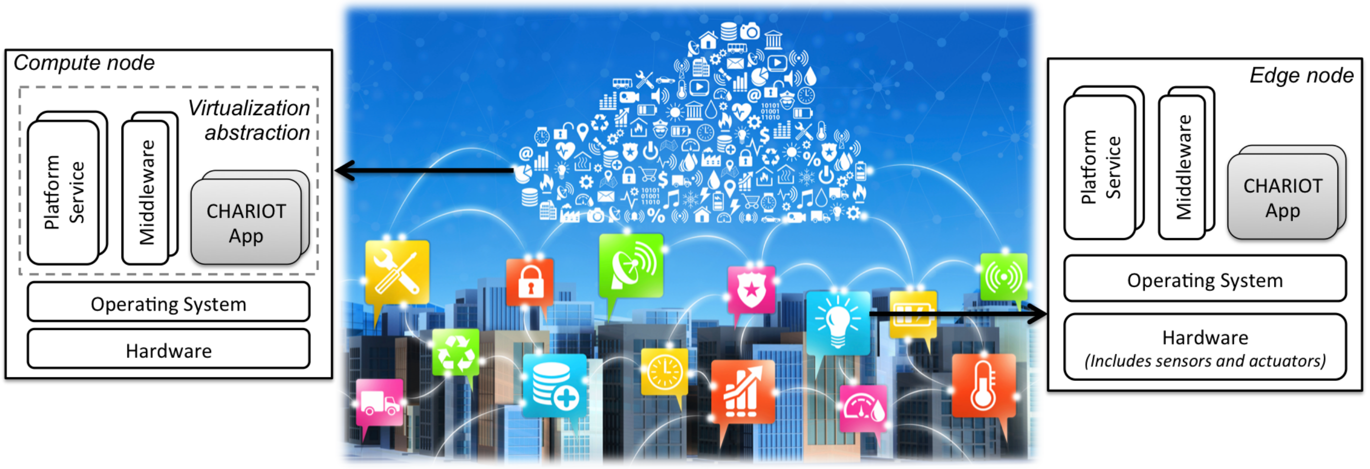

Fig. 5: CHARIOT architecture overview.

timer, which periodically fires a timer event, or an application lifecycle manager, which fires lifecycle events (application start, pause, etc.). Sensors are an example of data source. Data files stored in a filesystem can be another data source. Figure 4 presents a computation graph corresponding to Figure 3.

The above-described computation model is generic and can be easily mapped to existing computation patterns. In the case of the time-driven component assembly pattern, existing component business logic can be written in terms of tasklets, event sources can be used to capture timer events, data sources can be used to capture various sensors, and any actuation related logic can also be written in terms of tasklets. Furthermore, CHARIOT transport object provides an excellent mechanism to enforce clean separation-of-concerns between computation and communication logic, which results in middleware agnostic applications.

Similarly, in the case of batch processing and stream processing patterns, the CHARIOT computation model readily maps to existing dataflow engines such as Spark, Storm, S4, and TensorFlow. Tasklets can be mapped to computation nodes of each engine, data source can be mapped to different implementation of source nodes, and edges will always represent dataflow. Event sources, however, do not easily map to these dataflow engines. But we can argue that event sources are mainly relevant only for time-driven component assembly pattern where periodic timers are used. Any middleware used by these dataflow engines can be supported in CHARIOT via transport objects. For example, a transport object can be implemented for Kafka [15] so that it can be used with Storm.

B. Overall CHARIOT Architecture

The CHARIOT computation model is part of the CHARIOT platform. Figure 5 shows how this computation model fits into the overall CHARIOT architecture. As shown in the Figure, there are two kinds of nodes (a) edge node, and (b) compute node. Edge nodes are same as that of the smart city platform presented in Section II-A. Compute nodes are resourceful nodes that, unlike edge nodes, are not equipped with sensors and/or actuator devices.

Both kinds of nodes can host platform services; they are long running system services, such as application management service, failure monitoring service, runtime reconfiguration management service, etc. Also, as shown in Figure 5, both edge and compute nodes can host multiple middleware and applications (CHARIOT App). Applications, as described before, are composed of one or more tasklets and use transport object to interact with available middleware solutions (see Figure 3).

V. Related Work

Our work presented in this paper can be compared to existing dataflow engines [5], [6], [12]. All of these existing dataflow engines use some form of a computation graph comprising computation nodes and dataflow edges. These engines are designed for batch-processing and/or stream-processing high volumes of data in resource intensive nodes. As such they might not be well-suited for resource-constrained edge nodes. Furthermore, these engines, unlike our solution, do not have flexible architectures to use different middleware solution for communication. Again, this might not be crucial if these engines are used in isolation but interaction with edge devices requires possible support for other middleware as well. Finally, another advantage of our approach is that our computation model can be easily mapped to any of the aforementioned dataflow engines.

TensorFlow [7] is another dataflow engine, but it is mainly directed towards machine learning. Real-time data processing is not supported, as it is mainly a batch processing engine. As with other dataflow engines, our computation model can be mapped to TensorFlow. This capability will be useful as machine learning can play a critical role in large-scale data processing and quality modeling.

FIWARE [16] is a an open platform that integrates the cloud computing paradigm with the concept of Generic Enablers (GE), allowing GEs to interact with the cloud. GEs are heterogeneous architectural components that comprise a FIWARE platform. The main idea here is to support a variety of GEs that can be reused for different purposes by different applications. FIWARE has been previously used to build smart city systems. For example, in [17] the authors present a FIWARE infrastructure for smart home application. There is no specific computation model per say in FIRWARE but by implementing appropriate GEs and adapters, FIWARE could possibly be used to integrate applications with varying computation patterns.

VI. Conclusions and Future Work

Smart city platforms are open and extensible cyber-physical platforms that are required for execution and integration of different smart city applications. These are city-scale distributed platforms that will more than likely consist of physical resources with distributed ownership. This will result in highly heterogeneous platforms. This paper specifically focuses on heterogeneity arising from varying application types and associated computation models. Realizing smart city systems requires interaction and collaboration between different types of applications, which necessitates interaction between associated component models. To solve this issue, in this paper, we present the CHARIOT computation model, which is a generic computation model based on dataflow graphs.

We claim that the CHARIOT computation model is generic because (a) it can be used to model time-driven component assemblies for real-time applications, and (b) it can be mapped to the computation model of existing dataflow engines related to both batch and stream processing patterns. Therefore, our computation model covers all three different application types that we think are relevant for smart city systems. In this paper, we also show how the CHARIOT computation model fits into the overall middleware-agnostic CHARIOT platform.

This paper mainly describes our ongoing research. We are striving to achieve the following research goals in near future:

- Modifying existing CHARIOT Domain Specific Language (DSL) [3] to allow modeling of applications using CHARIOT computation model.
- Design, develop, deploy and evaluate end-to-end smart city systems to verify the CHARIOT platform architecture.

Acknowledgments

This work is sponsored in part by a research grant from Siemens Corporate Technology, AFOSR DDDAS program under contract FA9550-13-1-0227, and NSF EAGER program under award CNS-1528799. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Siemens Corporate Technology, AFOSR, and NSF.

References

[1] G. Karsai, D. Balasubramanian, A. Dubey, and W. Otte, “Distributed and managed: Research challenges and opportunities of the next generation cyber-physical systems,” in *17th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2014, Reno, NV, USA, June 10-12, 2014*, 2014, pp. 1–8.

[2] D. C. Schmidt, J. White, and C. D. Gill, “Elastic infrastructure to support computing clouds for large-scale cyber-physical systems,” in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on*. IEEE, 2014, pp. 56–63.

[3] S. M. Pradhan, A. Dubey, A. Gokhale, and M. Lehofer, “Chariot: A domain specific language for extensible cyber-physical systems,” in *Proceedings of the 15th Workshop on Domain-Specific Modeling (To be published)*. ACM, 2015, pp. 9–16.

[4] E. Lee *et al.*, “Cyber physical systems: Design challenges,” in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*. IEEE, 2008, pp. 363–369.

[5] Apache Software Foundation, “Apache Storm,” http://storm.apache.org/.

[6] ——, “Apache Spark,” http://spark.apache.org/.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, “Tensorflow: Large-scale machine learning on heterogeneous systems, 2015,” *Software available from tensorflow. org*.

[8] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for vm-based cloudlets in mobile computing,” *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[9] G. T. Heineman and B. T. Councill, *Component-Based Software Engineering: Putting the Pieces Together*. Reading, Massachusetts: Addison-Wesley, 2001.

[10] W. R. Otte, A. Dubey, S. Pradhan, P. Patil, A. Gokhale, G. Karsai, and J. Willemsen, “F6COM: A Component Model for Resource-Constrained and Dynamic Space-Based Computing Environment,” in *Proceedings of the 16th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC ’13)*, Paderborn, Germany, Jun. 2013.

[11] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[12] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, “S4: Distributed stream computing platform,” in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. IEEE, 2010, pp. 170–177.

[13] S. Pradhan, W. R. Otte, A. Dubey, A. Gokhale, and G. Karsai, “Key Considerations for a Resilient and Autonomous Deployment and Configuration Infrastructure for Cyber-Physical Systems,” in *Proceedings of the 11th IEEE International Conference and Workshops on the Engineering of Autonomic and Autonomous Systems (EASe ’14)*, Laurel, MD, USA, Sept 2014.

[14] S. Pradhan, W. Otte, A. Dubey, C. Szabo, A. Gokhale, and G. Karsai, “Towards a self-adaptive deployment and configuration infrastructure for cyber-physical systems,” *ISIS*, vol. 14, p. 102, 2014.

[15] J. Kreps, N. Narkhede, J. Rao *et al.*, “Kafka: A distributed messaging system for log processing.” NetDB, 2011.

[16] T. Usländer, A. J. Berre, C. Granell, D. Havlik, J. Lorenzo, Z. Sabeur, and S. Modafferi, “The future internet enablement of the environment information space,” in *Environmental Software Systems. Fostering Information Sharing*. Springer, 2013, pp. 109–120.

[17] A. Bellabas, F. Ramparany, and M. Arndt, “Fiware infrastructure for smart home applications,” in *Evolving Ambient Intelligence*. Springer, 2013, pp. 308–312.