7

Harnessing the Computing Continuum for Programming Our World

Pete Beckman[1], Jack Dongarra[2,4], Nicola Ferrier[1], Geoffrey Fox[3], Terry Moore[2], Dan Reed[5], and Micah Beck[2]

[1] *Argonne National Laboratory, Lamont, IL, USA*
[2] *University of Tennessee, Knoxville, TN, USA*
[3] *Indiana University, Bloomington, IN, UA*
[4] *Oak Ridge National Laboratory, Oakridge, TN, USA*
[5] *University of Utah, Salt Lake City, UT, USA*

7.1 Introduction and Overview

The number of network-connected devices (sensors, actuators, instruments, computers, and data stores) now substantially exceeds the number of humans on this planet. This is a tipping point, and the societal and intellectual effects of this are not yet fully understood. Billions of things that sense, think, and act are now connected to a planet-spanning network of cloud and high-performance computing (HPC) centers that contain more computers than the entire Internet did just a few years ago. We are now critically dependent on this expanding network for our communications and social discourse; our food, health, and safety; our manufacturing, transportation, and logistics; and our creative and intellectual endeavors, including research and technical innovation. Despite our increasing dependence on this massive, interconnected system of systems in nearly every aspect of our social, political, economic, and cultural lives, we lack ways to analyze its emergent properties, specify its operating constraints, or coordinate its behavior.

Simply put, today we have the tools to instrument and embed intelligence in everything, and we are doing so at a prodigious pace. Although we are the globally distributed designers, builders, and users of this immense, multilayered

Dr. Beck is an associate professor at University of Tennessee, Knoxville. He is currently on detail to the National Science Foundation in the Office of Advanced Cyberinfrastructure. The work discussed herein was completed prior to his government service and does not reflect the views, conclusions, or opinions of the National Science Foundation or of the US government.

Fog Computing: Theory and Practice, First Edition.
Edited by Assad Abbas, Samee U. Khan, and Albert Y. Zomaya.

environment, we are not truly its masters. Each of us manages only some of the networks components, and we can neither predict its aggregate behavior nor easily specify our intensional goals in intuitive language. For all of us collectively, and each of us individually, this must change. Today, we program in the relatively small confines of a single node, defining individual device, instrument, and computing element behaviors, and we are regularly confounded by unanticipated outcomes and unexpected behaviors that result once this individual node/device is exposed to the network collective. As consumers, we want our Internet-capable environmental devices (e.g. thermostats, lighting, and entertainment preferences) to adapt seamlessly to our changing roles and expectations, regardless of location. And yet, rather than specifying the ends we seek, we must specify detailed behaviors for home, office, car, and transient locale. In environmental health, we build and deploy arrays of wireless environmental sensors and edge devices when our goal may really be to "reprioritize edge resources to search for mosquitoes, given a statistically significant change in seasonal temperature and humidity across the nearby river basin." In disaster planning, when satellites show hurricane formation, we manually redirect data streams and simulation software stacks, when our goal is really to "retarget advanced computing resources to predict storm surge levels along the eastern seaboard." In science, when the Laser Interferometer Gravitational-Wave Observatory (LIGO) detects a gravity wave, we scramble to reposition the global network of telescopes to capture multispectral data and launch simulations, when our true goal is to "identify and analyze correlated transient phenomena."

Whatever the desires of consumers, companies, governments, and scientific researchers may be, we continue to build this increasingly digital world with only ad hoc, experiential, and intuitive expectations for the efficacy of alternative design choices. More perniciously, once these choices have been made, modifying or reversing them is often impossible. In large measure, this is because two constraints – resource capabilities and desired outcomes – are convolved, artificially and unnecessarily, on two time scales – design and deployment. The first of these is at the time of design and construction; the second is during outcome. At either time, the resource components may change (e.g. due to availability or failure) or expectations may shift (e.g. due to addition of new instruments or new questions). Moreover, the lifetime of many computations is not minutes or hours, but often days, months, or years. As Figure 7.1 shows, these components vary dramatically in capabilities and numbers but in aggregate define a complex collective that we call the "computing continuum." While significant research and development exists at specific places along this continuum (i.e. focus on HPC, or cloud, or Internet of Things [IoT]), we seek to develop approaches that include the entire computing continuum as a collective whole. Just as early experimentalists who studied molecular behavior in isolation struggled for want of a predictive theory

The Computing Continuum

IoT/Edge — Fog — HPC/Cloud

Size	Nano	Micro	Milli	Server	Fog	Campus	Facility
Example	Adafruit Trinket	Particle.io Boron	Array of Things	Linux Box	Colocated Blades	1000, node cluster	Datacenter
Memory	0.5K	256K	8GB	32GB	256G	32TB	16PB
Network	BLE	WiFi/LTE	WiFi/LTE	1 GigE	10GigE	40GigE	N*100GigE
Cost	$5	$30	$600	$3K	$50K	$2M	$1000M

Count = 10^9 Size = 10^1 — Count = 10^1 Size = 10^9

Figure 7.1 The Computing Continuum: Cyberinfrastructure that spans every scale. Components vary from small, inexpensive devices with limited computer resources (IoT) to modestly priced servers with mid-range resources to expensive high performance computers with extensive compute, storage, and network capabilities. This range of capabilities, cost, and numbers forms a continuum.

of gases and materials properties, so we struggle in the absence of a specification methodology and predictive understanding of this new computing continuum. We need a conceptual breakthrough for continuum programming that elevates specifications from components and behaviors to systems and objectives.

This paper outlines a vision for how best to harness the continuum of interconnected sensors, actuators, instruments, and computing systems, from small numbers of very large devices to large numbers of very small devices. Our hypothesis is that only via a continuum perspective can we intentionally specify desired continuum actions and effectively manage outcomes and systemic properties – adaptability and homeostasis, temporal constraints and deadlines – and elevate the discourse from device programming to intellectual goals and outcomes.

7.2 Research Philosophy

As the deployment of intelligent network-connected devices accelerates, so does the urgency with which this research area must be addressed. Industry analysts have begun predicting that "the edge will eat the cloud" [1]. The challenge, however, is not that one form will supplant another, but that we lack a programming and execution model that is inclusive and capable of harnessing the entire computing continuum to program our new intelligent world. Thus, development of a framework for harnessing the computing continuum would catalyze new consumer services, business processes, social services, and scientific discovery.

We believe programming the continuum is not only possible, but realizable, though breakthroughs in the concepts and abstractions are needed for its coordination and management. Our common models of computation assume

enumerated resources and predictable computing capabilities. However, in the continuum, the capabilities and numbers of components change dramatically over time, and aggregate functions can be long running – lasting months or years. In such contexts, intelligent, adaptive, closed-loop operations are de rigueur.

To be adopted, any broadly applicable continuum framework must be domain independent and exploit and interconnect extant, lower-level programming models and software systems. It is neither possible nor practical to obviate extant tools and techniques. Thus, the challenge lies in developing new annotations and composable abstractions for continuum computation and data movement, where none currently exist.

Table 7.1 highlights a range of scientific examples where the fabric of computing spans many orders of magnitude, and complex, open, and closed-loop behavior is required [2]. For example, ecologists should be able to link ecosystem monitoring with cloud-based simulations. Edge analysis routines, written to process telescope data, could be connected with HPC work flows that retrain machine learning algorithms automatically; the results of this machine learning could then be used to reposition observational assets based on the detection of new phenomenon. Similarly, materials scientists could write programs linking live instrument analysis at the edge with predictive models running on large-scale computing systems to detect experimental configuration errors.

Table 7.1 Exemplar continuum computing science applications.

Project	Description
Array of Things Urban Science Instrument	Environmental sensors, computer vision, deep learning inference, triggered weather, and traffic computations
Atmospheric Radiation Measurement Climate Research Facility	Software-controlled radar, user-provided sensors, data archive, climate models
Large Synoptic Survey Telescope	Transient phenomena detection and multispectral image correlation
National Ecological Observatory Network	Field-deployed instrumented towers and sensor arrays, sentinel measurements, specimen collection protocols, remote sensing capabilities, natural history archives
Precision Weather Forecasting and Sustainable Agriculture	Inexpensive environmental sensors and citizen science drive customized simulation models for microclimate weather forecasts and aquifer depletion reduction
Intelligent Manufacturing	Sensor measurement, modeling (digital twin), analysis, and control

Quite clearly, however, the merit of advances in continuum programming is not limited to science but will be a catalyst for exploration and advancement in almost every human endeavor.

7.3 A Goal-oriented Approach to Programming the Computing Continuum

As Figure 7.1 shows, along this continuum the product of device count and device size is roughly constant. At either extreme, the scale is large, the resources are geographically distributed, their availability varies over time, and they frequently span multiple control domains. Thus, the computing continuum future should integrate two primary activities with multiple subsidiary goals: (1) developing a goal-oriented annotation and high-level programming model that specifies desired outcomes for the aggregate, rather than a collection, of component behaviors; (2) building mapping tools, a run-time system, and an execution model for managing continuum resources as an abstract machine that also monitors behavior and triggers remappings when necessary.

In one sense, this approach maps loosely to a classic view of computation: a program, a run-time system, and an abstract machine. However, this traditional motif for computation evolved from our experiences directly specifying the actions of hardware under our control. Even the word "program" evokes the notion that we provide commands via a sequence of steps.

However, in the computing continuum, we cannot uniformly command all components to do our bidding – the building blocks are too diverse, the scale is too large, and the component owners and operators are sometimes unknown. Hence, we must begin by describing the goal.

7.3.1 A Motivating Continuum Example

To highlight the breakthrough we envision, consider a realistic scenario from the south Chicago neighborhood of Chatham, which suffers from the highest level of home flooding and associated insurance claims in the city. Chicago's infrastructure is a "combined system," where stormwater and raw sewage share the same underground structures, and when rainwater exceeds capacity, sewage enters homes, endangering health in addition to ruining personal property. Local residents are eager to see sensors, which we actually have deployed in Chatham, linked with weather models to provide warnings for impending danger.

The science/public policy problem can be succinctly stated as: "Eight hours before anticipated rainfall […] predict underground infrastructure responses and trigger intelligent reactions and warnings if greater than 5% of homes flood or if

traffic capacity falls below 70% of normal; then monitor and dynamically adapt urban controls to reduce harm." Decomposing this example, we can sketch the components and control flow, as below.

Eight hours before anticipated rainfall. On an HPC system, periodically run focused weather forecasts. As the risk of rainfall increases, run the forecast more frequently and with a finer resolution.

Predict underground infrastructure responses. Three cloud/HPC models are coupled to predict the response of the storm infrastructure: (1) a regional floodplain model predicts waterway inputs to the neighborhood; (2) a computational hydrology model predicts the absorption of incoming water by soil and infrastructure; and (3) a model of the underground infrastructure predicts when capacity is exceeded. Inputs for the linked models come from other parts of the continuum: (1) live measurements from soil moisture monitors, sump pumps, basement water, and sewer levels provide immediate hyper-local data; and (2) Array of Things (AoT) nodes use intelligent libraries and edge computing resources to analyze camera images of rising floodwater in rivers, on streets, and in rain gardens and detention ponds [3].

Trigger intelligent reactions. Based on a computational sewer model and learned responses to historical actuator settings (e.g. the behavior of pumps and valves under stress), reconfigure local fog/edge components to autonomously respond resiliently, adjusting flow rates and motors within the disrupted network environment.

Trigger public warnings if greater than 5% of homes flood or if traffic capacity falls below 70% of normal. Using cloud/HPC models, calculate the impact of the potential flooding and alert city officials and residents as needed. Furthermore, as new flood data become available from home and citizen science sensors, use image data from consumer cameras and sensors to push computer vision algorithms along optimized intelligent libraries and new learning models to edge resources to identify and then predict when bridges, underpasses, or quickly flowing water will disable vehicles.

Monitor and dynamically adapt urban controls to reduce harm. As the flood danger increases, reprioritize available edge and fog resources to detect, predict, and report on events like rising waters, stalled vehicles, and water erupting from manholes and drains.

In this example, realizing end-to-end data capture, modeling, and analysis, and timely response (closed loop) requires systemic coordination between sensors (edge), behavior detection (fog), models (cloud/HPC), and actuators (edge). Notice that it would be necessary to push new computational elements from the cloud to edge/fog resources during the storm – the continuum is bidirectional, and code is dynamic. Traditional computational work-flow systems like Pegasus

[4] and Kepler [5] provide mechanisms for triggering data movement and computation. However, our goal is much broader, as we intend to program the continuum so that new algorithms and deep learning models can be pushed to appropriate locations (i.e. edge, fog, cloud, and/or HPC computing resources) using a simple lambda (function as a service) [6, 7] abstraction. Ideally, both the annotation and the high-level mapping would be portable to other contexts, spanning cities – Portland to Barcelona – and rural areas, even when the edge hardware and cloud providers differ significantly. For rural and agricultural areas, the challenges shift to dam spillway management, agricultural runoff and water supply quality assessment, and community evacuation.

Realizing and implementing this continuum programming model requires balancing conflicting constraints and translating the high-level specification into a form suitable for execution on a unifying abstract machine model. In turn, the abstract machine must implement the mapping of specification demands to end-to-end resources.

7.3.2 Goal-oriented Annotations for Intensional Specification

This approach to programming the continuum stems from the fluid nature of the underlying resources, especially those at the edge. If every program component and its behavior were static, an imperative behavioral specification could be mapped directly to resources, assuming the resources themselves were static. For static cases, and when developers wish to implement all dynamic management, one could expose the resource-demand graph specifications and control mechanisms for direct use. However, this is rarely possible or practical, as the following examples illustrate:

When weather models predict roadways will degrade, adjust traffic signaling and preferred routing based on local conditions to optimize safety for both pedestrians and vehicles.

When waterway sensors detect increased phosphates, use edge device sensor data and satellite image analysis as inputs to simulations that can predict harmful algae blooms.

After significant seismic activity, reprioritize edge computation to detect smoke, distress calls, and natural gas leaks. Based on air quality and local weather, predict location of the source(s).

Simply put, the programming equivalent to source routing, where the packet originator completely determines the route, is rarely possible in the continuum.

For dynamic cases – the majority – the community must devise an approach that is more declarative and constraint based. It must succinctly describe the aim and enable efficient mapping (and remapping) to disjointed, heterogeneous,

shifting resources that behave more like independent agents than a single, cohesive machine. This is analogous to packet routing, which describes where the data should be sent but not how it should be sent.

Attacking this problem requires a two-pronged approach. First, one must devise languages for describing the resources of the continuum, including intelligent libraries, sensors, instruments, and cloud services. Tightly coupled with this, our abstract machine and runtime system will keep historical metrics of performance, interconnection bandwidth, and computational capacity that can be used for building execution graphs. Such work could build on the World Wide Web Consortium's (W3C's) specifications for the Semantic Web [8], including a resource description framework (RDF), web ontology language (OWL), and a query language for RDF (SPARQL). Although the Semantic Web has been slow to evolve, several well-developed technologies for describing resources, data, and ontologies have been deployed.

This first prong is insufficient to fully harness the continuum, for one must also specify goals and desired outcomes. Thus, we must also link the exciting and intense resurgence in autonomous agent research, fueled by advances in machine learning, to build goal-based specifications that can be mapped to resources and computation. The foundations for this field were built decades ago [9–11]. Today, we see successful work in a wide range of fields – from goal-based, human-machine teaming to flocks of autonomous drones.

By fusing the research from these domains with our novel work on edge computing, intelligent libraries, data logistics, and HPC, we believe the community could revolutionize the computing continuum.

7.3.3 A Mapping and Run-time System for the Computing Continuum

Realizing the continuum programming model requires translation of high-level specifications into a form suitable for execution on underlying resources. Cleanly separating the intensional specification from the execution strategies is key to managing temporally varying application demands and shifting resource availability and capability, which is a defining element of the continuum. To accomplish this, we would need to translate goal-oriented application specifications into an annotated resource demand graph and a set of constraints. In turn, continuum resources will be represented by an annotated resource capabilities graph with its own set of constraints (an abstract machine). This abstract machine must instantiate specification demands on continuum resources. An intelligent run-time system is then responsible for mapping and adaptively remapping the resource demands to continuum capabilities. As Figure 7.2 illustrates, these elements define the programming model for the computing continuum.

With this backdrop, consider our motivating example once again: "Eight hours before anticipated rainfall, predict underground infrastructure responses

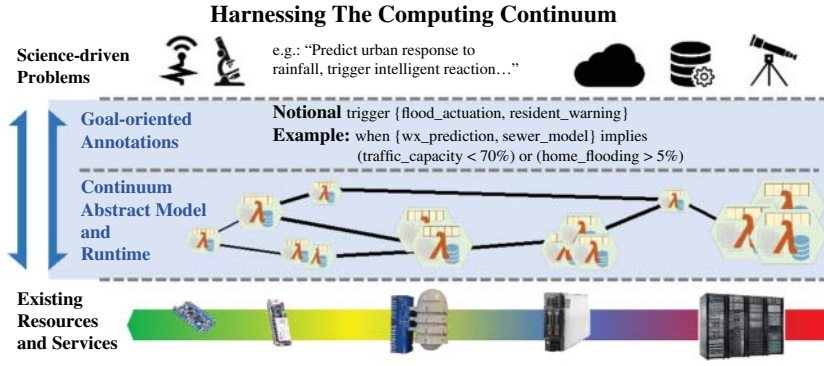

Figure 7.2 Continuum Computing Research Areas: A pictorial depiction of the computer science areas that require research to successfully program the computing continuum. To address problems, such as the sciences examples given in Table 7.1, using existing resources and services, we need an abstract programming model with goal-oriented annotations, along with a run-time system and an execution model. (*See color plate section for the color representation of this figure*)

and trigger intelligent reactions and public health warnings if highway traffic capacity is significantly impacted or if more than 5% of homes flood, then monitor and adapt controls during the storm." This high-level specification results in a resource demand graph of (1) an HPC-based mesoscale weather simulation, (2) water level sensors, (3) roadway flooding detection via image analysis, and (4) public health models and warning and evacuation models. Constraints include computationally intense models, geographic sensor dispersion, and real-time adaptation and actuator control.

Similarly, the resource capabilities graph for the sensor set, edge devices, actuators, and cloud or HPC resources would include annotations that specify: (1) performance characteristics such as node interconnection bandwidth and connectivity, storage capacity, and computation speed; (2ii) programmability (i.e. fixed function device or "over the air" programmable); (3) multiplicity (i.e. an estimate of the number of instances, recognizing these vary over time); (4) control span (i.e. single or shared function and ownership); and (v) domain-specific constraints (e.g. geographic location, power limitations, or maximum usage frequency).

The mapping function would then instantiate environmental monitoring by tapping data streams from a statistical sample of the available water sensors, reprioritize flood image analysis on AoT sensors and fog devices, and then launch a weather model parameterized by a terrain model with a real-time constraint on prediction cycles. Because these resource demands may conflict with or sometimes exceed resource availability, and the resources themselves may shift over time (e.g. due to sensor loss or replacement), any mapping is necessarily imprecise. Limited cloud or HPC availability might force a reduction

in forecast accuracy and redeployment of alternative library versions to meet deadline constraints. Thus, the execution system must monitor the efficacy of each mapping and adapt accordingly.

It is also necessary to explore several techniques for mapping annotated resource demand graphs, mapping constraints to resource capability graphs, and learning and adaptively remapping these elements as demands and resources shift. As shown in Figure 7.3, these techniques include, but need not be limited to: (1) the intensional, goal-oriented program specification, translated to an annotated resource demand graph via autonomous agents and machine learning; (2) a resource registry that contains a time-varying list of available resources, attributes, and constraints; (3) optimized, multiversion libraries suitable for deployment on continuum resources, spanning computation-limited sensors, sophisticated edge devices, HPC systems, and cloud resources; (4) temporal "fuzzy logic" [12, 13] for qualitative constraint specification – an approach we used successfully in real-time adaptive control of parallel scientific applications [14, 15]; (5) machine learning [16] and auto-tuning that exploit behavioral data and temporal fuzzy logic constraint specifications to map and remap resource demands to extant resources; and (6) very low overhead, distributed behavioral monitoring tools [17], a behavioral repository, and data sharing via Message Queuing Telemetry Transport (MQTT) [18, 19].

7.3.4 Building Blocks and Enabling Technologies

There are many potential building blocks and enabling technologies for building the continuum. Several of these have been developed by the authors.

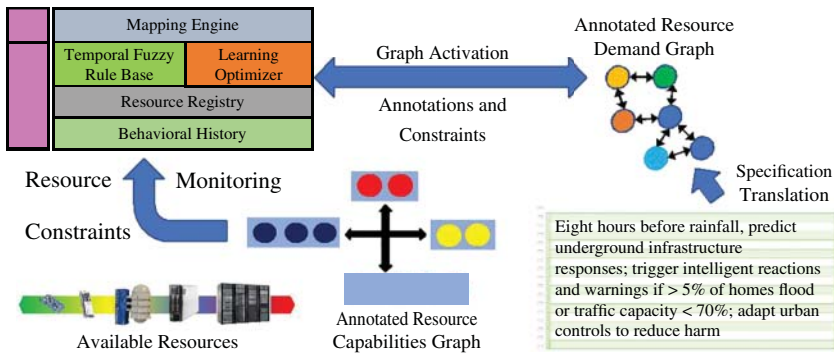

Figure 7.3 Continuum mapping and execution: Research is needed to explore techniques for mapping goal-oriented specifications to the available resources. This graphic shows a possible technique that translates the goal specification into an annotated resource demand graph that is used by the mapping engine, along with resource capability, availability and constraint information, multiversion libraries, and behavioral history.

7.3.4.1 The Array of Things (AoT)

AoT [3] is an NSF-sponsored project at the University of Chicago, Northwestern University, and Argonne National Laboratory focused on supporting urban sciences. AoT is deploying 500 sensor nodes in Chicago, with more than 100 already operational. The core AoT hardware/software platform, Waggle [20, 21], is also being used by a diverse community of scientists for things like exploring the hydrology of a pristine prairie owned by the Nature Conservancy and for studying pollen and asthma in Chattanooga, Tennessee. These sophisticated sensors as well as their lightweight, battery-powered, thumb-sized cousins, support edge computation – the ability to directly analyze the data stream in situ. In addition to air-quality sensors, the nodes support edge computing for in-situ computer vision, machine learning, and audio analysis. For example, the number of pedestrians using a crosswalk can be calculated by GPU-based computer vision algorithms using a combination of Kalman filters and deep learning. The computed results are pushed into the cloud.

7.3.4.2 Iowa Quantified (IQ)

A complementary project, when Reed was at the University of Iowa, has been designing inexpensive battery and solar-powered wireless sensors with multiple wireless protocols (WiFi, LoRaWAN [22]) that are integrated with cloud and data analytics services. Configurable sensors span a wide range of environmental data, including temperature, humidity, gases, and particulate matter. This Iowa Quantified (IQ) project has deployed multidepth soil sensors to enable understanding of moisture dynamics and agricultural productivity [23], and other projects are being planned around microscale weather forecasting and aquifer depletion minimization for precision agriculture.

7.3.4.3 Intelligent, Multiversion Libraries

Across the continuum, the efficient application of scientific computing techniques requires specialized knowledge in numerical analysis, architecture, and programming models that many working researchers do not have the time, energy, or inclination to acquire. With good reason, scientists expect their computing tools to serve them and not the other way around. Unfortunately, the highly interdisciplinary problems of programming the continuum, using more and more realistic simulations on increasingly complex computing platforms, will only exacerbate the problem. To address this, we believe a two-pronged strategy is needed that leverages expertise in developing optimized, automatically tuned libraries that can be deployed dynamically, based on the goal-based annotation and mapping described above. At a low level, the community needs operations that closely mimic the building blocks of the Basic Linear Algebra Subroutines (BLAS) [24] but perform well across the continuum. Studying different approaches to optimization

(e.g. mixed-precision algorithms and empirical auto-tuning for different platforms on the continuum) will enable us to provide the intelligent libraries needed to meet the performance constraints of the target application scenarios. At a high level, defining a common name space for different types of operations and services (e.g. "compress," "merge," and "SVD") will enable us to move the locus of computation, in a seamless fashion, between the cloud and the edge. To integrate the bidirectional workflow across the variety of actors, the run-time system should match the resource requirements of the named components of a prototype library with the resources that are actually available when the work flow executes.

7.3.4.4 Data Flow Execution for Big Data

Currently, we are exploring the Twister2 environment [25, 26], which is an early prototype of some of the ideas expressed above. Twister2 has a modular implementation with separate components summarized in Table 7.2, which include the key features of existing "big data" programming environments and have been designed as a toolkit so that one can pick and choose capabilities from different sources. Further, all components have been redesigned as necessary to obtain high performance. Twister2 contains three separate communication packages: (1) one aimed at classic parallel programming, (2) Twister:Net [27] aimed at distributed execution, and (3) publish-subscribe messaging as seen in Apache Storm and Heron to connect edge to cloud.

The first of these communication environments addresses problems in parallel programming that are well known from MPI, and implements bulk synchronous processing; it extends MPI by adding some collectives required for solving big data problems, including topic modeling and graph analytics [28–30]. However, the leading programming environments Spark, Storm, Heron, and Flink that target big data offer as default, a different communication model built around data flow. Twister2 has implemented this as a separate communication system, Twister:Net, which offers a user friendly data application programming interface (API) rather than a messaging API. Twister:Net automatically breaks data bundles into messages and, if necessary, uses disk storage for large volume transfers. A second and more critical feature is that the communication system supports distinct source and target tasks; therefore, Twister:Net can be used in a fully distributed environment, as seen in edge-cloud environments. Twister:Net also implements a rich set of collectives such as keyed reduction [31] where all the famous MPI and MapReduce operations (gather, scatter, shuffle, merge, join, etc.) can be considered as particular versions of "keyed collectives." Here we define a collective operation as a combined communication and computation (as in reduced) operation that involves some variant of all to all linkage of the source and target tasks. The final communication subsystem in Twister2 also supports different source and target

Table 7.2 Twister2 components and status.

Component	Area	Current implementation	Future implementation
Connected Data Flow	Internal fine-grain data flow or external data flow (workflow)	Dynamic internal data flows	Ongoing and add coarse-grain external data flow
High Level API's	Distributed Data Set, SQL, Python, Scala, Graph	Tsets (Twister2 implementation of RDD and Streamlets), Java	Data flow optimizations, SQL, Python, Scala, Graph
Task System	Task migration	Not started	Streaming job task migrations
	Streaming	Streaming execution of task graph	FaaS Function as a Service
	Task execution	Process, threads	More executors
	Task scheduling	Dynamic scheduling, static scheduling; pluggable Scheduling algorithms	More algorithms
	Task graph	Static graph, dynamic graph generation	Cyclic graphs for iteration as in Timely
Communication	Data flow communication DFW	Twister:Net. MPI Based or TCP. Batch and streaming operations	Integrate to other big data systems, Integrate with RDMA
	BSP communication	Conventional MPI, Harp with extra collectives	Native MPI Integration
Job Submission	Job submission (dynamic/static) Resource allocation	Plugins for Slurm, Mesos, Kubernetes, Aurora, Nomad	Yarn, Marathon
Data Access	Static (batch) data	File systems, including HDFS	NoSQL, SQL
	Streaming data	Kafka connector for Pub-Sub Communication, Storm API	RabbitMQ, ActiveMQ

tasks, but uses publish-subscribe messaging. It is more suitable than Twister:Net for unreliable links, such as those found in edge-to-cloud communication.

Twister2 also has a carefully designed data flow execution model that supports linking intelligent nodes; this feature enables support for fault tolerance (i.e. as in the creation of resilient distributed datasets [RDDs] used in Apache Spark) and allows wrapping nodes with rich tools such as learning systems. Further, Twister2

recognizes that some coarse-grain data flow nodes (e.g. those seen in job work flows) are not performance sensitive, but other finer-grain nodes need low overhead implementations with, in particular, use of streaming data without the overhead associated with reading and writing of intermediate files. Note currently if you use Apache Storm or Spark to link distributed subsystems – data center to data center or edge-to-fog to data center – different jobs in each subsystem must be linked. Twister2 offers the possibility of invoking a single data flow across all parts of a distributed system.

7.4 Summary

Modern society is now critically dependent on a global and pervasive network of intelligent devices, both large and small, that are themselves connected to a planet-spanning network of cloud and HPC centers. Despite this dependence, analyzing this burgeoning network's emergent properties and coordinating its integrated behavior is not straightforward. This paper outlines an ambitious plan to elevate programming, coordination, and control of the continuum from ad hoc component assembly to intensional specification and intelligent, homeostatic control.

References

1 Bittman, T. (2017). *Maverick* Research, The edge will eat the cloud*. Gartner Research, Online https://www.gartner.com/doc/3806165/maverick-research-edge-eat-cloud.
2 Stuart Anderson, Ewa Deelman, Manish Parashar et al., Report from the NSF Large Facilities Cyberinfrastructure Workshop, September 2017. http://facilitiesci.org.
3 Array of Things, March 2019. www.arrayofthings.org.
4 Deelman, E., Vahi, K., Rynge, M. et al. (2016). Pegasus in the cloud: science automation through workflow technologies. *IEEE Internet Computing* 20 (1): 70–76.
5 Altintas, I., Berkley, C., Jaeger, E. et al. (2004). Kepler: an extensible system for design and execution of scientific workflows. In: *Proceedings of 16th International Conference on Scientific and Statistical Database Management*, 423–424. IEEE.
6 Apache OpenWhisk, 2018 (accessed April 2018). https://openwhisk.apache.org.
7 Amazon Web Services, Lambda features, 2018 (accessed April 2018). https://aws.amazon.com/lambda/features.

8 Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American* 284 (5): 34–43.

9 Bradshaw, J.M., Johnson, M., Uszok, A. et al. (2004). Kaos policy management for semantic web services. *IEEE Intelligent Systems* 19 (7): 32–41.

10 Kagal, L., Finin, T., and Joshi, A. (2003). A policy language for a pervasive computing environment. In: *Proceedings Policy 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 63–74.

11 Hexmoor, H., Castelfranchi, C., and Falcone, R. (2003). *Agent Autonomy: Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer.

12 Zadeh, L.A. (1986). Commonsense reasoning based on fuzzy logic. In: *Proceedings of the 18th Conference on Winter Simulation, WSC '86*, 445–447. New York, NY: ACM.

13 Frigeri, A., Pasquale, L., and Spoletini, P. (2014). Fuzzy time in linear temporal logic. *ACM Transactions on Computational Logic* 15: 4, pp. 30:1–30:22.

14 Droegemeier, K.K., Gannon, D., Reed, D. et al. (2005). Service-oriented environments for dynamically interacting with mesoscale weather. *Computing in Science and Engineering* 7 (6): 12–29.

15 Ramakrishnan, L. and Reed, D.A. (2008). Performability modeling for scheduling and fault tolerance strategies for scientific workflows. In: *Proceedings of the 17th International Symposium on High Performance Distributed Computing, HPDC '08*, 23–34. New York, NY: ACM.

16 Collins, A., Fensch, C., and Leather, H. (2012). Masif: machine learning guided auto-tuning of parallel skeletons. In: *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques, PACT '12*, 437–438. New York, NY: ACM.

17 Gamblin, T., de Supinski, B.R., Schulz, M. et al. (2010). Clustering performance data efficiently at massive scales. In: *Proceedings of the 24th ACM International Conference on Supercomputing, ICS '10*, 243–252. New York, NY: ACM.

18 Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta (eds.), MQTT Version 5.0, OASIS Committee Specification 01, December 2017. http://docs.oasisopen.org/mqtt/mqtt/v5.0/cs01/mqtt-v5-0-cs01.html.

19 Manzoni, P., Hernandez-Orallo, E., Calafate, C.T., and Cano, J.-C. (2017). A proposal for a publish/subscribe, disruption tolerant content island for fog computing. In: *Proceedings of the 3rd Workshop on Experiences with the Design and Implementation of Smart Objects, SMARTOBJECTS '17*, 47–52. New York, NY: ACM.

20 Waggle Project, March 2019. http://wa8.gl.

21 Beckman, P., Sankaran, R., Catlett, C. et al. (2016). Waggle: an open sensor platform for edge computing. In: *2016 IEEE Sensors*, 1, 3. IEEE.

22 LoRaWAN Wireless Specification, 2018. https://lora-alliance.org.

23 Lee, K.-P., Kuhl, S.J., Bockholt, H.J. et al. (2018). A cloud-based scientific gateway for internet of things data analytics. In: *Proceedings of the Practice and Experience in Advanced Research Computing, July 22–26*.

24 Dongarra, J.J., Du Cruz, J., Hammarling, S., and Duff, I.S. (1990). Algorithm 679: a set of level 3 basic linear algebra subprograms: model implementation and test programs. *ACM Transactions on Mathematical Software* 16 (1): 18–28.

25 Ekanayake, J., Li, H., Zhang, B. et al. (2010). Twister: A runtime for iterative MapReduce. In: *Proceedings of the First International Workshop on MapReduce and Its Applications of ACM HPDC 2010 conference, June 20–25, 2010*. ACM.

26 Supun Kamburugamuve, Kannan Govindarajan, Pulasthi Wickramasinghe et al., Twister2: design of a big data toolkit, Concurrency and Computation, EXAMPI 2017 workshop at SC17 conference, 2019.

27 Kamburugamuve, S., Wickramasinghe, P., Govindarajan, K. et al. Twister:Net – communication library for big data processing. In: *HPC and cloud environments, in Proceedings of Cloud 2018 Conference*. IEEE.

28 Qiu, J. Harp-DAAL for high-performance big data computing. In: *The Parallel Universe*, No. 32, 31–39. https://software.intel.com/sites/default/files/parallel-universe-issue-32.pdf.

29 Harp-DAAL high-performance data analytics framework website. https://github.com/DSC-SPIDAL/harp, https://dsc-spidal.github.io/harp/docs/harpdaal/harpdaal. Accessed September 2018.

30 Chen, L., Peng, B., Zhang, B. et al. (2017). Benchmarking Harp-DAAL: High performance hadoop on KNL clusters. In: *IEEE Cloud 2017 Conference*. IEEE.

31 Fox, G. (2019). Big data overview for Twister2 tutorial. In: *BigDat2019. 5th International Winter School on Big Data*. Cambridge, UK, January 7–11.