

A Spatial Computing Approach to Distributed Algorithms

Jacob Beal

Raytheon BBN Technologies
Cambridge, MA, USA
Email: jakebeal@bbn.com

Richard Schantz

Raytheon BBN Technologies
Cambridge, MA, USA
Email: schantz@bbn.com

Abstract—Creating distributed applications for large, decentralized networks is challenging for traditional programming approaches, posing a growing obstacle as the number and capabilities of networked devices continue to advance. In many applications, however, the network of devices is not itself of interest. Rather, we are interested in the relationships of the devices to their surroundings and their relative positions. We may thus instead write programs for the continuous space occupied by the devices, viewing the network as a discrete approximation of that space. This “amorphous medium” approach to spatial computing leads to algorithms based on manifold geometry, which are by their nature robust, adaptive, and scalable to vast numbers of devices. This paper brings together previous results into an overview of this programming approach and explains how the manifold geometry abstraction provides benefits in scalability, robustness, and adaptability.

Keywords-spatial computing; amorphous medium; distributed algorithms; Proto; self-*

I. INTRODUCTION

Distributed networked systems are a fundamental challenge in our current network-centric era. Centralized systems are often impractical due to considerations of speed, robustness, adaptivity, etc. Distributed systems have the potential to overcome all of these problems, but in order to do so many challenges must be addressed, such as coordination of decisions and adaptation to changes in the computation or in the network. The difficulty of overcoming these challenges means that current distributed systems generally address only a selected portion, and fall short of their overall potential. For example, in the domain of sensor networks, the DARPA SensIT project[1] performed collaborative tracking of moving vehicles, but network organization and recovery from failure were largely centralized or absent. The ExScal project[2], which followed it, demonstrated scaling to 1000 devices, but only by requiring a limiting hierarchical structure and precise deployment of all devices into a preconfigured pattern.

Many distributed systems, including sensor networks, multi-robot systems, and ad-hoc mobile networks, belong to the class of *spatial computers*—potentially large networks of devices where the cost of moving information between two devices is strongly related to the physical distance between them. To aid organization of distributed computation on such systems, we may model the system as a continuous space, rather than as a network. By programmatically manipulating regions of

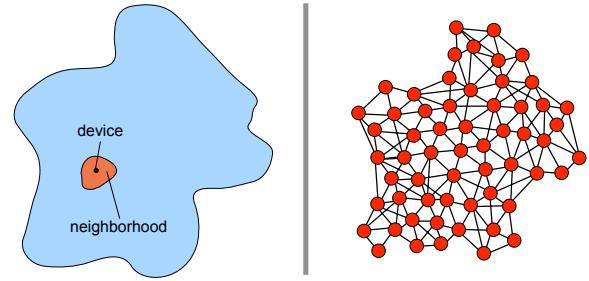
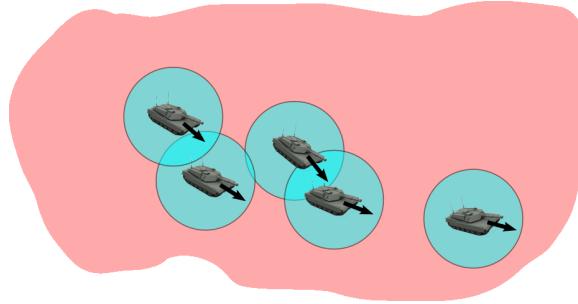


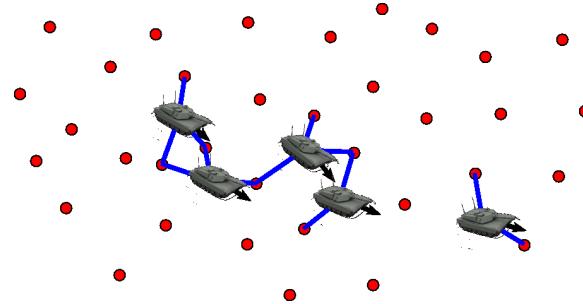
Fig. 1. An amorphous medium, highlighting a device and its neighborhood (left), and a network of devices that approximate the amorphous medium (right).

geometric space rather than the individual devices, many of the details of the program and its adaptability are made implicit. Doing so allows a programmer to work more directly on the end goal of the application of interest, rather than trying to build it from the bottom up in terms of individual devices exchanging bits. Not only does this potentially simplify the programming problem, but since many types of adaptability in these networks can be derived from the spatial abstraction, we can build distributed algorithms that, by the very nature of their construction, address many of the problems that are often neglected because they are too hard or less well understood, and wind up making prototype distributed systems un-deployable in practice. This continuous space approach has been developed incrementally across a number of previous papers[3], [4], [5], [6], [7], [8], [9], [10].

This paper presents an overview of the programming approach and explains how the manifold geometry abstraction produces benefits in scalability, robustness, and adaptability. We begin with a brief review of spatial computing and the *amorphous medium* abstraction. We then give an example, drawn from the domain of sensor networks, of how complex applications can be formulated in terms of geometric computations over a manifold. This geometric formulation creates implicit program adaptability at two levels: local adaptation through maintenance of the continuous/discrete relationship and long-range adaptation through the conformance of manifold geometric operations to network structure. Finally, we discuss open problems and likely benefits for practical distributed applications.



(a) Continuous Model



(b) Discrete Model

Fig. 2. Continuous and discrete models of a unit of tanks moving through a network of sensor devices (red) that can detect (blue) nearby tanks.

II. SPATIAL COMPUTING AND THE AMORPHOUS MEDIUM

The key insight enabling a continuous spatial approach is the recognition that there are many systems where the focus is best placed not on the devices that make up the system, but rather on the space through which the devices are distributed. Sensor networks are a prototypical example: e.g., the point of a target-tracking network is to monitor the movement of entities through an area. The fact that this involves observations made by and at particular devices is only of interest so far as it contributes toward that goal. Multi-robot systems and ad-hoc mobile networking are good examples as well: e.g., the point of a robot coverage algorithm is to examine all points in a space of interest, and the point of an ad-hoc routing algorithm is to move information across space to where it is needed.

If devices only communicate directly over short distances (e.g. to nearby neighbors over low-power wireless communication) then the aggregate structure of the communication network forms a discrete approximation of the structure of the space of interest. Combining these two observations allows us to view the network using an abstraction that we call the *amorphous medium*[3]. An amorphous medium is a Riemannian manifold¹ with a computational device at every point, where every device knows the recent past state of all other devices in a local neighborhood (Figure 1). A network of locally communicating devices can thus be viewed as a discrete approximation of an amorphous medium, with each device representing a small region of nearby space and messages sent between nearby devices implementing the information flow through neighborhoods.

With carefully chosen computational primitives and a means of combining those primitives, such as provided by Proto[4], [6], [7], [10], it is possible to maintain a tight relationship between an abstract computation specified for a Riemannian manifold and an actual computation being carried out on a real network that is distributed through that space, so that a program written to execute over continuous space can be

¹A manifold is a mathematical object that looks like Euclidean space locally, but globally may be different. For example, the surface of the Earth is a 2-dimensional manifold: locally it looks flat, but if you keep going in a straight line, you will return to your starting location. A Riemannian manifold also guarantees the availability of other key geometric building blocks such as angle, distance, area and volume, curvature, and gradients (generalized derivatives).

approximated on real devices. The problem of building a distributed system utilizing this abstraction is thus factored into three components:

- 1) Application code built in terms of geometric computations and information flows on regions of a manifold.
- 2) Algorithms that map from any combination of geometric computations and information flows into a robust, distributed implementation on the neighborhood interactions of an amorphous medium.
- 3) A virtual machine for approximating the neighborhood interactions of an amorphous medium on a network of communicating devices.

III. EXAMPLE GEOMETRIC APPLICATION: TRACKING A UNIT OF TANKS

In this section, we will walk through an example of how a complex distributed application can be formulated geometrically. Traditionally, distributed algorithms are almost always designed and analyzed in terms of the local network interactions of individual devices—Lynch’s classic textbook[11] is a good representative example. As a result, it can be hard to conceive of how to re-formulate applications for a continuous geometric world.

Consider an example from the domain of sensor networks: tracking a unit of moving tanks. We wish to track these tanks using a network of sensor devices, where each sensor device can detect the relative position of any tank within 50 meters of the device. Let us describe how this can be done, highlighting in bold face the elementary geometric operations of the computation.

Assume that the sensor network programmer wants to consider tanks to be acting as part of the same unit whenever are close to one another (e.g., within 200 meters) and are moving together (e.g. their most recent motions were within 10 km/hr speed and 30 degrees direction), and wants to report the size and aggregate movements of each detected unit. For example, Figure 2 shows a single unit of 5 tanks moving at 60 km/hr on a bearing of 120 degrees.

A tank can be detected by multiple sensors, and each sensor can detect multiple tanks, so a good place to start is aggregating detections into tanks: each device runs a computation for each tank it detects, and computations **share information with**

neighboring computations only if they are about the same tank. This can be determined for each neighbor of a device by comparing the relative position of the local detection with the sum of the **neighbor's current value** for its detection and the **neighbor's relative position**.

Each tank's speed and bearing can be estimated geometrically by tracking what devices are entering and exiting its detection region. The speed can be estimated by comparing the **area** of the region of detection to that of the regions of devices that have joined or left the region of detection over a short time interval of length dt . The bearing can be estimated by taking a distributed **surface integral** of relative tank position over the devices that have just entered the region of detection and subtracting a **surface integral** of relative tank position over the devices that have just exited the region of detection, then normalizing the resulting vector.

To cluster tanks into units, we begin by selecting, for each tank, a region of **every point within 100 meters**: this guarantees that if two tanks are within 200 meters of one another (and therefore close enough to be in the same unit) there will be at least one point where their regions overlap. These regions can be constructed by **measuring the distance** to the tank and testing whether the distance is less than 100 meters, producing an **indicator field**: a function mapping each point in space to a boolean, true for points in the region and false for points not in the region. For those devices that cannot detect the tank directly, distance can be estimated by applying the triangle inequality: the **minimum over all neighbors** of the sum of a **neighbor's current value** for the estimate and the **range to the neighbor**.²

Devices that know the current speed and bearing estimates for a tank can supply them to others in the 100-meter region by taking the **gradient** of the distance estimates. This produces a **vector field** indicating which direction the estimates should **flow from the source**. Given recent speed and bearing estimates, points where two **regions intersect** can compare them and determine whether the two tanks are part of the same unit. Finally, the number of tanks in a unit and its aggregate velocity can be computed with **surface integrals** over the **union of regions** for tanks in the same unit.

We can thus see how a distributed algorithm might be formulated in terms of geometric computations. These computations in turn rely on maintaining the Riemannian manifold abstraction over the network. More sophisticated applications, for example, incorporating reliability estimates from sensors or accounting for sharp turns in the unit definition, can be implemented with more sophisticated geometric computations.

IV. BENEFITS OF GEOMETRIC SPECIFICATIONS

In this section, we explain how using a manifold geometry abstraction provides benefits in scalability, robustness, and adaptability. Specifically, we detail two ways of exploiting the inherently spatial nature of domains like sensor networks.

²A more sophisticated geometric computation (e.g. [12], [13]) is necessary if we want the estimate to adapt smoothly to changes in the source region or the structure of the manifold.

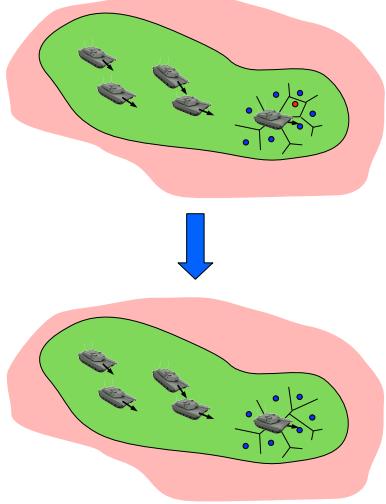


Fig. 3. Local adaptivity through maintenance of the continuous/discrete relationship: when an individual device fails (red dot), others in the vicinity (blue dots) split up responsibility for the space it previously approximated, coarsening algorithm execution.

- **Local Adaptation:** maintenance of the continuous/discrete relationship allows programs written using physical units to automatically adapt to local changes in the network that approximates the space.
- **Long-Range Adaptation:** the conformance of manifold geometric operations to network structure allows programs written using geometric computations and information flow to automatically adapt to gross changes in the structure of the space or its network approximation.

A. Physical Units Enable Local Adaptation

When a domain is inherently spatial, many of the computations that we wish to express in that domain are more natural to express in physical units such as meters, seconds, density, curvature, etc. The continuous space abstraction allows programs to be expressed directly in these units, rather than units specific to a particular network, such as hops, devices, and rounds of communication. A distributed virtual machine[7] is then responsible for making a best-effort translation between physical units and the current values of hops, neighbors, rounds, etc. for each device in its interactions with others nearby, and also for reporting the current quality of these approximations, so that an application can determine when its requirements cannot be met.

As a result, a continuous space program can be written agnostic as to which devices represent which portions of space and how they maintain connectivity, and also agnostic to the rates of program execution on devices[5]. If the virtual machine that implements the continuous/discrete relationship is written to adapt to changes in the set of other devices it communicates with, then every physical unit computation on neighborhoods will inherit this adaptability as well.

For example, if a device fails, its neighbors need to stop using information from the dead device and to split up

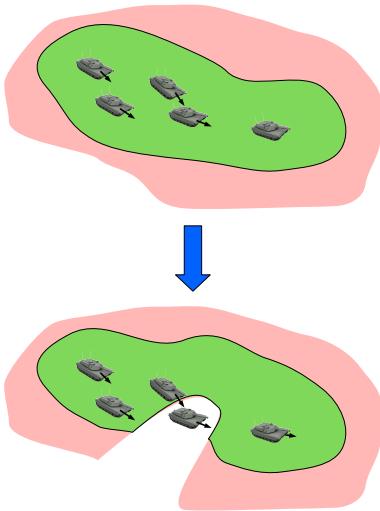


Fig. 4. Because manifold geometry operations conform to the structure of the space, a geometric program adapts automatically to large-scale changes in network structure.

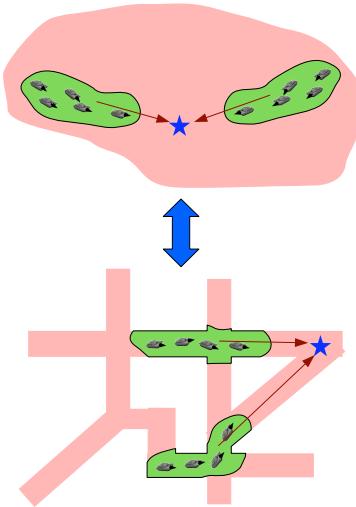


Fig. 5. The adaptivity of manifold geometric operations can allow a program to adapt to qualitatively different execution environments, such as computing the expected rendezvous of two units in an open field versus city streets.

responsibility for the space that the dead device was previously responsible for, making the approximated computation coarser (Figure 3). A program based on physical units does not need to know how this happens, just to be able to measure the local density of devices (which can also be expressed in physical units) and change its behavior only if the computation has become too coarse to meet its high-level goals. Likewise, adding new devices simply increases the resolution of the discrete approximation, automatically improving the quality of a computation so long as there are not so many neighbors that they begin to interfere with one another's communications.

The virtual machine may even be able to automatically adapt to problems with too few or too many neighbors. If there are too few neighbors, the device may be able to increase the range at which it communicates (e.g. by adjusting the power of a radio transmitter). If there are too many neighbors, the device might decrease the range of communication, or might also decrease the frequency of communication. A program written in physical units will automatically adapt to these changes as well: all that has happened is that the values have changed for variables in its computation such as speed of information flow, distance to neighbors, and age of information.

B. Manifolds Enable Long-Range Adaptation

Where physical units provide adaptability in local interactions, geometric computations on a manifold provide adaptability to gross changes in the structure of the space of interest, its approximation by the network, or the goals of a program. Geometric computations on a manifold are warped to fit its shape, so when a computation is run on a manifold approximated by a network, large-scale changes in the structure of the space or the network automatically change the results of the computation to suit the new structure of the network (Figure 4). Note that a large-scale change of the network could involve any number of nodes—for example,

breaking a connection between two regions could involve the failure of massive numbers of nodes or a single critical link, and represent the same change of structure for the space being approximated. Likewise, scaling up an algorithm to larger networks can be represented by either increasing the volume of approximated space or increasing the quality of approximation.

The Riemannian manifold representing a network is implied by its neighborhoods: once we have modeled local interactions as neighborhoods of continuous space, the regions of space represented by individual devices are stitched together along neighborhood relations to form a manifold that represents the gross structure of the network. Geometric constructions on a Riemannian manifold (e.g. finding a bisector or the shortest path between points) work much like ordinary Euclidean constructions, except that a construction is warped by the shape of the manifold and the distance metric implied by its neighborhoods. This means that geometric computations will be able to automatically adapt to different circumstances.

Consider, for example, the tracking tanks in an open field versus through city streets. If communication is short-range and blocked by buildings (e.g. low-power, high-frequency wireless), then the structure of the network approximates that of the space: a large open region for the field and an interconnected web for city streets (Figure 5). This means that distance estimates in the open field will be as-the-crow-flies, while distance estimates in the city will be along the streets, and the computation described above will automatically adapt such that tanks that are near one another in absolute terms but distant in terms of their ability to move through the streets will not be considered part of the same units.

The toolkit of geometric constructions on Riemannian manifolds is slightly different than for Euclidean space, but many Euclidean constructions do translate easily. For example, the distance between two locations in a Riemannian manifold can be quickly and efficiently calculated by applying the

triangle inequality, and this computation is straightforward to distribute (though making it adapt efficiently as the network changes is more challenging[12], [13]). Most importantly, if the distributed versions of the elementary toolkit of geometric constructions can be made to adapt efficiently to changes in manifold structure, then any complex program constructed from them will adapt as well.

Just as with physical units, this is not a panacea, but separates small changes that can be handled automatically from big ones that a system designer needs to address carefully. For example, a frequent challenge in networking is determining when a failure really matters: for example, if there is a lot of connectivity near a communications path, a cheap local repair process can fix a broken connection, but if a critical device fails an entirely new route may need to be constructed. Viewing a network as a Riemannian manifold makes such a distinction easy and natural, since a failure in a well-connected region does not change the structure of the manifold abstraction, but the failure of a critical device does. Moreover, measuring geometric properties of the manifold in physical units could be a useful predictor of when a communications path is endangered—for example, integrating density on lines perpendicular to a communication path gives a good approximation of the underlying connectivity of the network in that area, and therefore the fragility of the communications path.

We thus see that when there is a strong relationship between a physical space and the structure of a networked computation, using physical units and manifold geometry calculations allows a one-time investment in infrastructure to harden a system against a wide variety of failures and changes in the network or its environment.

V. CONTINUING CHALLENGES AND POTENTIAL IMPACT

We have presented an overview of the continuous-space approach to distributed algorithm programming. We have seen that complex distributed algorithms can be specified geometrically, using the amorphous medium abstraction. This can provide implicit scalability, robustness, and adaptability, both locally through maintenance of the continuous/discrete relationship, and over long ranges by the conformance of manifold geometric operations to network structure.

There are many continuing challenges in the further development of the continuous-space programming approach. Amongst the most pressing are:

- Theory of computation and universality on continuous space-time surfaces, extending the work in [9].
- Quantitative prediction of the relationship between the discrete network and the quality with which a computation is approximated. This has been computed for special cases[14], and is uncomputable in general, but should be computable for a wide class of useful algorithms.
- Loosely constrained mobile devices can be modeled with a gaseous model, described in [8]. This model needs to be better formalized and also to be extended to liquid, solid, and mixed-state approximations for devices with more constraints on their motion.

- Expansion of the library of fast, self-stabilizing algorithms that implement manifold geometric operations, after the fashion of [12].
- Continued improvement of Proto and its free software implementation[15].

The future of distributed systems will be driven by smaller, cheaper, and more capable devices. The continued rapid pace of hardware improvements, combined with rapid advances in MEMS, micro-fluidics, and micro-scale sensing, means that the bulky, expensive, and fragile devices of the past will be replaced by ever-smaller, cheaper, and harder devices in every domain. As we have seen, the continuous space abstraction approach has the potential to make these problems significantly simpler to address and sustain under widely varying conditions. The impact on distributed systems may be nothing short of revolutionary: orders of magnitude increase in the number of devices that can practically participate in a computation, simple dynamic configuration and system maintenance, seamless integration of new devices and order of magnitude decrease in the cost of device failure or movement, and a greatly strengthened ability to predict algorithm behavior.

REFERENCES

- [1] G. Mitchell, J. Mazurek, K. Theriault, and P. Manghwani, “Sensoft: Development of a collaborative sensor network,” in *Distributed Sensor Networks*, S. S. Iyengar and R. R. Brooks, Eds. Chapman & Hall, 2004.
- [2] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, Z. Hongwei, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, M. Nesterenko, R. Shah, S. Kulkami, M. Aramugam, L. Wang, M. Gouda, Y. Choi, D. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker, “Exscal: Elements of an extreme scale wireless sensor network,” in *11th IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications*, 2005, pp. 102–108.
- [3] J. Beal, “Programming an amorphous computational medium,” in *Unconventional Programming Paradigms Int'l Workshop*, September 2004.
- [4] J. Beal and J. Bachrach, “Infrastructure for engineered emergence in sensor/actuator networks,” *IEEE Intelligent Systems*, pp. 10–19, March/April 2006.
- [5] J. Bachrach and J. Beal, “Programming a sensor network as an amorphous medium,” in *Distributed Computing in Sensor Systems (DCOSS) 2006 Poster*, June 2006.
- [6] J. Bachrach, J. Beal, and T. Fujiwara, “Continuous space-time semantics allow adaptive program execution,” in *IEEE SASO 2007*, July 2007.
- [7] J. Bachrach and J. Beal, “Building spatial computers,” MIT, Tech. Rep. MIT-CSAIL-TR-2007-017, March 2007.
- [8] J. Bachrach, J. Beal, and J. McLurkin, “Composable continuous space programs for robotic swarms,” *Neural Computing and Applications*, vol. 19, no. 6, pp. 825–847, 2010.
- [9] J. Beal, “A basis set of operators for space-time computations,” in *Spatial Computing Workshop*, 2010.
- [10] J. Beal and J. Bachrach, “Programming manifolds,” in *Computing Media and Languages for Space-Oriented Computation*, Dagstuhl Seminar Proceedings, A. DeHon, J.-L. Giavitto, and F. Gruau, Eds., no. 06361. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.
- [11] N. Lynch, *Distributed Algorithms*. Morgan Kaufman, 1996.
- [12] J. Beal, J. Bachrach, D. Vickery, and M. Tobenkin, “Fast self-healing gradients,” in *ACM Symposium on Applied Computing*, March 2008.
- [13] J. Beal, “Flexible self-healing gradients,” in *ACM Symposium on Applied Computing*, March 2009.
- [14] J. Bachrach, J. Beal, J. Horowitz, and D. Qumsiyeh, “Empirical characterization of discretization error in gradient-based algorithms,” in *IEEE Int'l Conf. on Self-Adaptive and Self-Organizing Systems*, October 2008.
- [15] “MIT Proto,” software available at <http://proto.bbn.com/>, Retrieved November 22, 2010.