Chapter 7

Summary

7.1 Contributions

I have discussed and implemented a distributed system based on the computational paradigm of data propagation in an attempt to provide distributed solutions to problems with high-dimensionality. Along with this system, DProp, I have constructed several sample applications to demonstrate the distributed functionality of this system. This architecture, which I call "distributed propagation", achieves useful computation within a weakly-connected network, making it comparable to existing distributed computing approaches that also assume such. I designed this system with the principles of Representational State Transfer, or REST, in mind, resulting in a system that is both robust to failure and simple enough to be easily implemented on multiple platforms.

The development of distributed propagation has also led me to outline and implement a provenance mechanism in distributed propagator systems. This mechanism takes advantage of the explicit modularity of propagator networks and provides for the simple modification of existing propagator networks to add support for provenance. My implementation of a library to support the creation of distributed propagator networks demonstrates this by implementing fundamentally identical APIs that allow construction of both traditional and propagator-aware distributed propagator networks.

Constraint	Benefit
Idempotence	(with associativity) no locking during cell initialization
	no filtering for duplicate updates during resynchronization
Associativity	(with idempotence) no locking during cell initialization
	don't need to save all updates for resynchronization
Commutativity	removes need for global synchronization (timestamps)
	removes timeliness constraint on communications
Monotonicity	removes complexity caused by deletion operations
	allows computation of results with partial knowledge

Table 7.1: The four constraints on the merge operation and their benefits.

I have also outlined several constraints on merge operations used in data propagation that allow for meaningful distributed computation by reducing complexity. These constraints and their corresponding benefits are outlined in Table 7.1. Finally, I have also developed a loose guideline for how data structures may be represented in propagator networks, either by including data structures in the cells themselves, or by treating cell URLs as de-facto memory pointers.

7.2 Challenges and Future Work

7.2.1 Data Structures and Propagators

It is not entirely clear that the URL-based propagator data structure design that I propose in Section 4.1 is useful for more general computation in propagator networks. It may be difficult to merge two data structures for which only pointers are known. If two pointers are assigned at the same time to different (shared) cells, there is no way to determine how to merge these otherwise opaque pointers. A sensible merge algorithm may involve looking at the contents of each pointer so as to unify cells that may have been assigned different UUIDs.

The provenance-aware cells proposed in Section 4.2.1 are not affected by the issue raised above. When a provenance-aware cell is created, all pointers are allocated to appropriate subcells as well. This means that any merge operation on the subcells of

a provenance-aware cell should never have to merge two distinct pointer values that are assigned to the same field; the merge operation may safely ignore the contents of the pointer field outside of ensuring that it remains assigned.

7.2.2 Non-monotonic Propagator Networks & Garbage Collection

In this thesis, I have assumed that the collection of all cells in a propagator network is monotonically increasing. There is no explicit "deletion" or "unregistration" mechanism available for a distributed cell. Once a cell has been created, it may not be deleted, nor may a host disclaim any continuing interest in it. This means that there is no mechanism for garbage collection in a distributed propagator network, and may result in an intractibly large propagator network as time progresses.

In practical systems implemented with distributed propagator networks, it may be necessary to support such non-monotonic actions as deletion or unregistration of a cell from a network. Doing so will allow for resources on a host to be allocated more efficiently. Rather than spending the effort to resynchronize or maintain a copy of a cell that will never be used again, a host may instead spend its resources on more relevant computation.

A related need is for a mechanism to allow propagator networks to eventually prune unreachable peers. This would allow networks to eventually stop spending resources in trying to synchronize a node that no longer exists, regardless of whether the node was to partake in a future unregistration mechanism to disclaim interest in the cell. An implementation of garbage collection within propagator networks will resolve both of these issues.

7.2.3 Scalability and Practicality Issues

Although I have tested distributed propagation successfully across a network, these tests have largely been limited in scope to only two or three peers. Larger distributed propagator networks remain untested, and it is unknown whether they will scale

practically; the amount of resynchronization communications may eventually flood any network as the propagator network grows large.

Cell Discovery

Although this thesis has considered cell discovery to be orthogonal to the problem of distributing propagator networks, a truly scalable distributed propagator network will need to be able to determine what cells exist and where they may be connected. A number of mechanisms may be used to resolve this issue, including a priori knowledge passed to the API as supposed here. Other mechanisms may include name server mechanisms like DNS [20, 21] or service-discovery mechanisms such as DNS-SD [9] or SSDP (part of the universal plug-and-play architecture (UPNP) [36, 37]).

Compound Propagators in Distributed Propagator Networks

As cells must be assigned an identity *prior* to allowing remote connections or attaching propagators to them, there is no mechanism to create a cell on a remote host and spawn a given propagator attached to it. This makes it difficult to permit any negotiation or delegation of tasks in a distributed application. This also prevents the creation of compound propagators in distributed propagator networks. Although we may instantiate *local* propagators when a cell receives data, we may not task remote hosts with a given propagator and cell dynamically.

Possible solutions to enable the creation of dynamic propagator networks might include treating propagators as first-class objects that may be stored in a cell. We might then be able to distribute propagator and cell descriptions to remote nodes so that they may be instantiated for further processing. Allowing arbitrary code execution does raise security concerns, however, so any mechanism for compound propagation within distributed propagator networks will need to be carefully crafted to eliminate issues that arise due to such.

7.2.4 Security

I have outlined several approaches to security in Sections 5.2 and 5.3, however, my implementation, DProp and its accompanying library, currently does not implement them. As such, the security implementations proposed in this paper must be more carefully studied to ensure that data remains secure and that effective access control is possible in this system. Improved encryption may be had by moving away from an HTTPS implementation should SSL renegotiation prove to be unworkable, for example.

7.3 Conclusion

Distributed propagation provides a useful alternate mechanism to distributed computation that does not rely on simply distributing the data across the system for identical processing. This will provide for a larger number of computations that may be distributed, and perhaps allow for a greater number of applications for distributed computing. As the power of individual computers begins to reach a plateau without making use of concurrent processing, it will become ever more important to construct systems to make use of a larger number of hosts and processors and to move away from sequential processing.