Distributed Programming and Consistency: Principles and Practice

Peter Alvaro, Neil Conway, Joseph M. Hellerstein
UC Berkeley

July 18, 2012

Length

3 hour tutorial.

Intended Audience

Intermediate. We are targeting the tutorial at both industrial attendees and academics. We assume that audience members will have some familiarity with distributed programming but we do not assume any knowledge of Bloom, Datalog, or related technologies.

Contact Information

Contact Person: Neil Conway.

Neil Conway: nrc@cs.berkeley.edu
Peter Alvaro: palvaro@cs.berkeley.edu
Joseph M. Hellerstein: hellerstein@cs.berkeley.edu

Speaker Biographies

Peter Alvaro is PhD Student at the University of California, Berkeley. His research interests lie at the intersection of databases, distributed systems and programming languages. His research is supported by an NSF graduate fellowship.

Neil Conway is a PhD Candidate at the University of California, Berkeley. His research interests include large-scale data management, distributed systems, and logic programming. His research is supported by an NSERC graduate fellowship.

Joseph M. Hellerstein is a Chancellor's Professor of Computer Science at the University of California, Berkeley, whose work focuses on data-centric systems and the way they drive computing. He is an ACM Fellow, an Alfred P. Sloan Research Fellow and the recipient of two ACM-SIGMOD "Test of Time" awards for his research. In 2010, Fortune Magazine included him in their list of 50 smartest people in technology, and MIT's Technology Review magazine included his Bloom language for cloud computing on their TR10 list of the 10 technologies "most likely to change our world". A past research lab director for Intel, Hellerstein maintains an active role in the high tech industry, currently serving on the technical advisory boards of a number of computing and Internet companies including EMC, SurveyMonkey, Platfora and Captricity.

Distributed Programming and Consistency: Principles and Practice

Peter Alvaro
UC Berkeley
palvaro@cs.berkeley.edu

Neil Conway
UC Berkeley
nrc@cs.berkeley.edu

Joseph M. Hellerstein
UC Berkeley
hellerstein@cs.berkeley.edu

1. INTRODUCTION

In recent years, distributed programming has become a topic of widespread interest among developers. However, writing reliable distributed programs remains stubbornly difficult. In addition to the inherent challenges of distribution—asynchrony, concurrency, and partial failure—many modern distributed systems operate at massive scale. Scalability concerns have in turn encouraged many developers to eschew strongly consistent distributed storage in favor of application-level consistency criteria [5, 13, 22], which has raised the degree of difficulty still further.

To cope with the challenges of distributed programming without the benefit of strong consistency, practitioners have developed rules of thumb, such as using commutative, associative, and idempotent operations when possible [13, 17] and employing application semantics to resolve divergent replica states [11]. However, until recently there was relatively little work on principled approaches to enable application-level consistency criteria without requiring global coordination.

In this tutorial, we will review recent research on principled approaches to eventually consistent programming [2, 9, 10, 14, 18, 19, 20], and connect this body of work to practical systems and design patterns used by practitioners. We will begin by discussing how *semilattices* can be used to reason about the convergence of replicated data values, following the "CRDT" framework recently proposed by Shapiro et al. [19, 20]. After presenting several examples of how lattices can be used to achieve consistency without coordination, we will then discuss how lattices can be composed using *monotone functions* to form more complex applications [10]. We will connect this work to the tradition of logic programming from the database community [1], and to our work on the *CALM Theorem*, which characterizes the need for distributed coordination to ensure consistency at application level [2, 4, 14, 16]. Finally, we will discuss several design options for supporting non-monotonic operations:

(a) introducing coordination at appropriate program locations identified by CALM analysis
(b) employing "weak coordination" as a background operation (e.g., for distributed garbage collection)
(c) tolerating and then correcting inconsistency using taint tracking and after-the-fact "apology" or compensation logic [12, 13, 15]

We will conclude by summarizing the state of the art and highlighting open problems and challenges in the field.

Throughout the tutorial, we will use *Bloom*, a language for distributed programming that we have developed at Berkeley [6]. We will demonstrate the concepts introduced in this tutorial by using Bloom to interactively develop well-known distributed systems infrastructure components, including a key-value store, quorum replication, a distributed lock manager with deadlock detection, and a distributed commit protocol. Using tools distributed with the Bloom runtime, we will show how developers can visualize the distributed behavior of their programs, reason about the need for coordination across software components, compose individual monotonic components into larger programs, and employ Bloom's built-in tools for systematic distributed testing [3].

The tutorial will be *interactive*, in that simple installation instructions for the Bloom runtime will be provided and attendees will be given the complete source code for all example programs. During the tutorial, we plan to use the Bloom runtime to execute example programs, use the the built-in Bloom analysis tools to understand program behavior, and iteratively refine programs as appropriate. Attendees will have the option to run the tools themselves, although participation int this manner will not be mandatory. Note that for time reasons we do not expect attendees to develop Bloom programs from scratch during the tutorial, although several "homework" assignments will be made available to extend the example programs we present.

2. OBJECTIVES AND OUTCOMES

1. Discuss the particular challenges and benefits associated with programming over systems that provide only eventual consistency [11, 21, 22], rather than the stronger guarantees provided by traditional distributed transactional systems. Summarize the motivations behind the current trend away from widespread use of strong consistency protocols.

2. Introduce and compare recent research proposals to simplify distributed programming without strong consistency.

3. Focus on the CRDT and Bloom/CALM approaches to distributed consistency, and show how lattices and monotone functions can be used to build provably consistent implementations of well-known distributed infrastructure.

4. Discuss the CALM Theorem, which sheds light on *why* certain programs require distributed coordination and why others do not. Characterize the situations in which monotone programs are not sufficient, and discuss several ways in which non-monotonic operations can be supported.

5. Introduce the Bloom programming language and related tools for visualization, debugging, and systematic testing of distributed programs.

6. Highlight open problems and research challenges in the area of principled eventual consistency.

3. OUTLINE

- The current landscape of distributed programming
  - CAP and partition tolerance [8, 7]
  - Design patterns for working with eventual consistency
- CRDTs: lattices for convergent replicated values
  - Example CRDTs: sets, counters, graphs
- Composing lattices using Bloom^L
  - Monotone functions and homomorphisms
  - CALM Theorem
  - Bloom and Bloom^L
    - Syntax, semantics
    - Visualization and debugging tools
  - Example programs: shopping cart, simple key-value store, vector clocks, quorum replication
- Dealing with non-monotonicity
  - Strong coordination: two-phase commit or Paxos in Bloom
    - Automatic coordination synthesis
  - Weak coordination: applying lattices to distributed garbage collection
  - Apologies and compensation logic [12, 13, 15]
- Complete example system: key-value store
  - Version 1: non-monotonic updates
  - Version 2: monotonic updates using a `map` lattice, quorum replication
  - Version 3: monotonic updates using lattices to implement version vectors (per Dynamo [11]).
  - Version 4: distributed locking, multi-key atomic updates
- Distributed testing
  - BloomUnit
  - Applying CALM to enhance systematic testing
- Summary and conclusion
  - State of the art in principled eventual consistency
  - Open research problems

4. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] P. Alvaro, N. Conway, J. M. Hellerstein, and W. R. Marczak. Consistency Analysis in Bloom: a CALM and Collected Approach. In *CIDR*, 2011.

[3] P. Alvaro, A. Hutchinson, N. Conway, W. R. Marczak, and J. M. Hellerstein. BloomUnit: Declarative Testing for Distributed Programs. In *DBTest*, 2012.

[4] T. J. Ameloot et al. Relational transducers for declarative networking. In *PODS*, 2011.

[5] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, 2009.

[6] Bloom programming language. `http://www.bloom-lang.org`.

[7] E. A. Brewer. Towards robust distributed systems. In *PODC*, 2000.

[8] E. A. Brewer. CAP Twelve Years Later: How the "Rules" Have Changed. *IEEE Computer*, 45:23–29, 2012.

[9] S. Burckhardt, M. Fähndrich, D. Leijen, and B. P. Wood. Cloud Types for Eventual Consistency. In *ECOOP*, 2012.

[10] N. Conway, W. R. Marczak, P. Alvaro, J. M. Hellerstein, and D. Maier. Logic and Lattices for Distributed Programming. Technical Report UCB/EECS-2012-167, EECS Department, University of California, Berkeley, June 2012.

[11] G. DeCandia et al. Dynamo: Amazon's highly available key-value store. In *SOSP*, 2007.

[12] H. Garcia-Molina and K. Salem. Sagas. In *SIGMOD*, 1987.

[13] P. Helland and D. Campbell. Building on quicksand. In *CIDR*, 2009.

[14] J. M. Hellerstein. The Declarative Imperative: Experiences and Conjectures in Distributed Logic. *SIGMOD Record*, 39(1):5–19, 2010.

[15] H. F. Korth, E. Levy, and A. Silberschatz. A formal approach to recovery by compensating transactions. In *VLDB*, 1990.

[16] W. R. Marczak, P. Alvaro, N. Conway, and J. M. Hellerstein. Confluence analysis for distributed

programs: A model-theoretic approach. In *Datalog 2.0*, 2012. To appear.

[17] D. Pritchett. BASE: An Acid Alternative. *ACM Queue*, 6(3):48–55, May 2008.

[18] H.-G. Roh, M. Jeon, J.-S. Kim, and J. Lee. Replicated abstract data types: Building blocks for collaborative applications. *Journal of Parallel and Distributed Computing*, 71(3):354–368, Mar. 2011.

[19] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. A comprehensive study of convergent and commutative replicated data types. Technical Report RR-7506, INRIA, 2011.

[20] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In *13th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2011.

[21] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *SOSP*, 1995.

[22] W. Vogels. Eventually Consistent. *CACM*, 52(1):40–44, 2009.