Cloud Programming:
From Doom and Gloom to BOOM and Bloom

Peter Alvaro, Neil Conway

Faculty Recs: Joseph M. Hellerstein, Rastislav Bodik
Collaborators: Tyson Condie, Bill Marczak, Rusty Sears
UC Berkeley
Writing reliable, scalable distributed software remains extremely difficult
Three Hardware Trends

1. Cloud Computing
2. Powerful, heterogeneous mobile clients
3. Many-Core
Implications

1. Nearly every non-trivial program will be physically distributed
2. Increasingly heterogeneous clients, unpredictable cloud environments
3. Distributed programming will no longer be confined to highly-trained experts
The Anatomy of a Distributed Program

• In a typical distributed program, we see:
  – Communication, messaging, serialization
  – Event handling
  – Concurrency, coordination
  – Explicit fault tolerance, ad-hoc error handling

• What are we looking for?
  – Correctness (safety, liveness, ...)
  – Conformance to specification
  – High-level performance properties; behavior under network edge-cases
Data-Centric Programming

• **Goal:** Fundamentally raise the *level of abstraction* for distributed programming
• MapReduce: data-centric batch programming
  – Programmers apply *transformations* to *data sets*
• Can we apply a data-centric approach to distributed programming in general?
Bloom and BOOM

1. **Bloom**: A high-level, data-centric language designed for distributed computing

2. **BOOM**: Berkeley Orders of Magnitude
   - OOM bigger systems in OOM less code
   - Use Bloom to build real distributed systems
Agenda: Foundation

• Begin with a precise formal semantics
  – Datalog w/ negation, state update, and non-determinism
• Include primitives for distributed computation
• Enable formal methods for distributed programming
  – Model checking, theorem proving, ...
Agenda: Engineering

1. Efficient, low-latency dataflow engine (C4)
2. Network-oriented continuous program optimization
   – Automatically co-locate code and data
   – Adapt to current network and client conditions
   – Optimize for both power and performance
   – Leverage formal semantics: how does distribution change program behavior?
Agenda: Language Design

• How to expose these concepts to developers?
  – What are the right developer abstractions for common distributed programs?

• **Bloom** language design goals:
  1. Familiar syntax (list comprehensions, callbacks)
  2. Integration with imperative languages
  3. Modularity, encapsulation, and composition
Agenda: Validation

• How do we know that we’re solving real problems?
  – Build real systems
• Initial work: **BOOM Analytics**
  – Hadoop + HDFS in distributed logic
• **Goal:** Use Bloom to build a complete cloud computing stack
  – Google in 10KLOC?
Thank you!