Two-Level, Self-Verifying Data

IRIS Student Workshop
November 7, 2004

Patrick Eaton, Hakim Weatherspoon, and John Kubiatowicz
University of California, Berkeley
Outline

• Self-verifying data overview
• P2P Application survey
  – Design patterns
  – Consequences
• Two-level, Self-verifying data
  – Advantages
  – Implementation
• Conclusion
Self-Verifying Data Overview

• Allows peers to validate integrity of data
  – Function of name and data

• Fundamental to P2P systems
Hash-Verified Data

- Named by a secure hash of data
Key-verified Data

- Named by public key
  - Key verifies signature over data
P2P Application Survey

• Study how apps use self-verifying data
  – CFS, Ivy, OceanStore, PAST, Venti

• Emerged a set of design patterns
Design Patterns
Consequences of Usage Model

**Benefits**
- Conceptually simple
- Matches client’s view of data

**Costs**
- High query load on infrastructure
- High per-object storage overhead
- High cost to maintain data

**Root cause:**
- All components must access data at same granularity
Solution: Two-Level Naming

- Allow components to address data at different granularities.
General Approach

- **Extents** are containers for data blocks
- Peers *pack* blocks into extents
- **Map** temporary names to verifiable names
Design Issues for Extents

- Export extent format to infrastructure
- Naming and lookup is two-level process
- Maintain verifiability while packing data
Status and Conclusions

- Self-verifying data is fundamental
- Many apps have same usage model
  - Imposes high cost on system
- Extents allow variable granularity access
  - Address data more efficiently
- Have built a extent toolkit for apps
  - Hope to deploy apps soon!
Two-Level, Self-Verifying Data

Patrick Eaton, Hakim Weatherspoon, and John Kubiatowicz
University of California, Berkeley