A Novel Data Placement Model for Highly-Available Storage Systems

Rama, Microsoft Research

joint work with

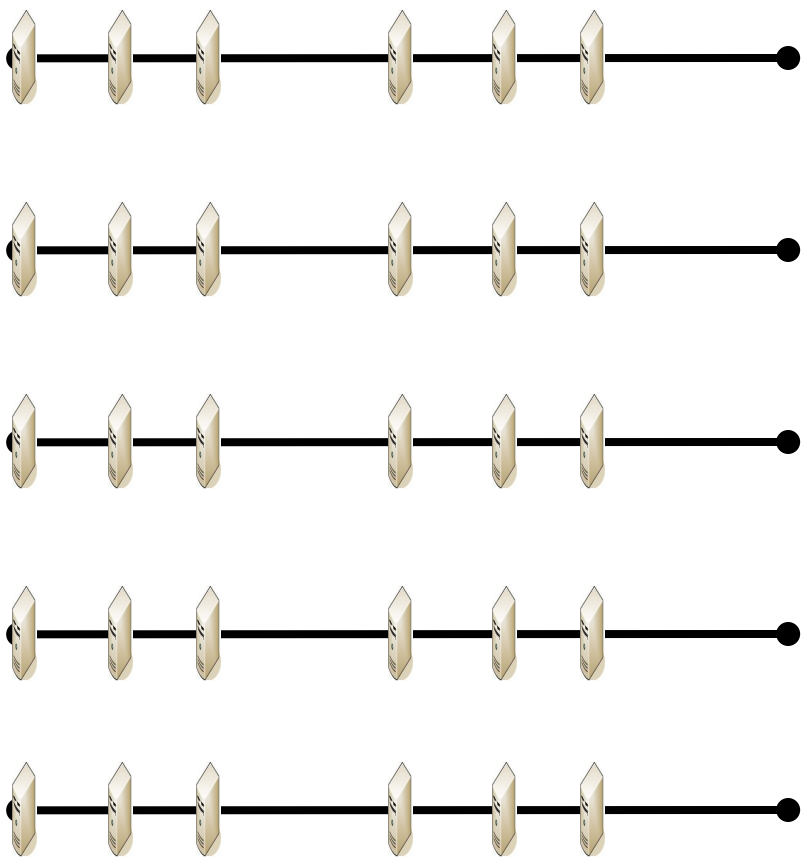
John MacCormick, Nick Murphy, Kunal Talwar,
Udi Wieder, Junfeng Yang, and Lidong Zhou

Introduction

- Kinesis:
  - Framework for placing data and replicas in a data-center storage system

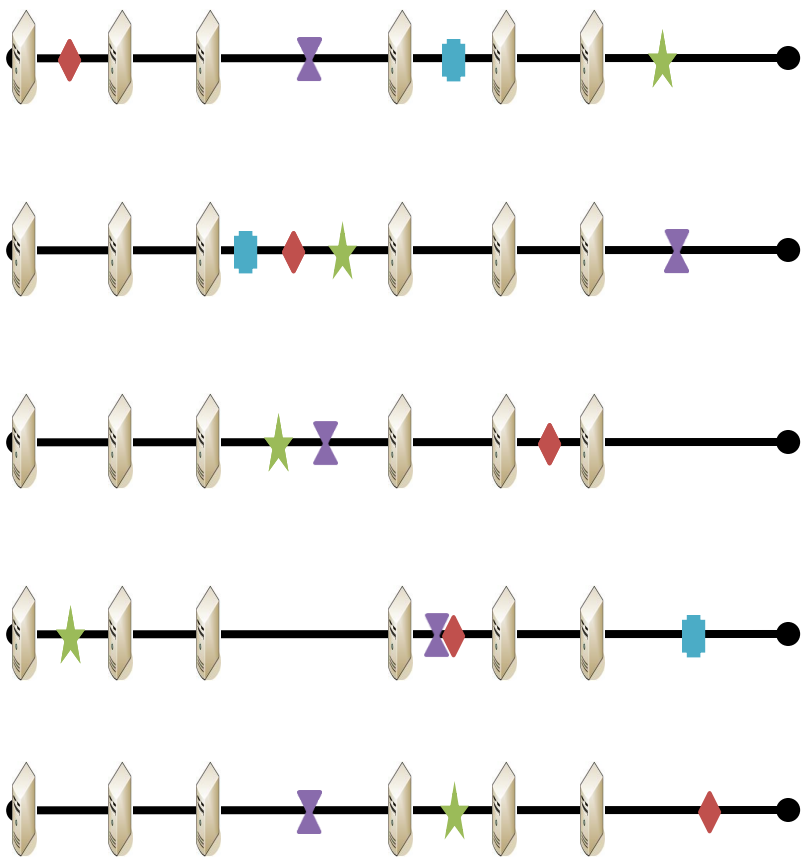
- Three Design Principles:
  - Structured organization
  - Freedom of choice
  - Scattered placement

Kinesis Design => Structured Organization



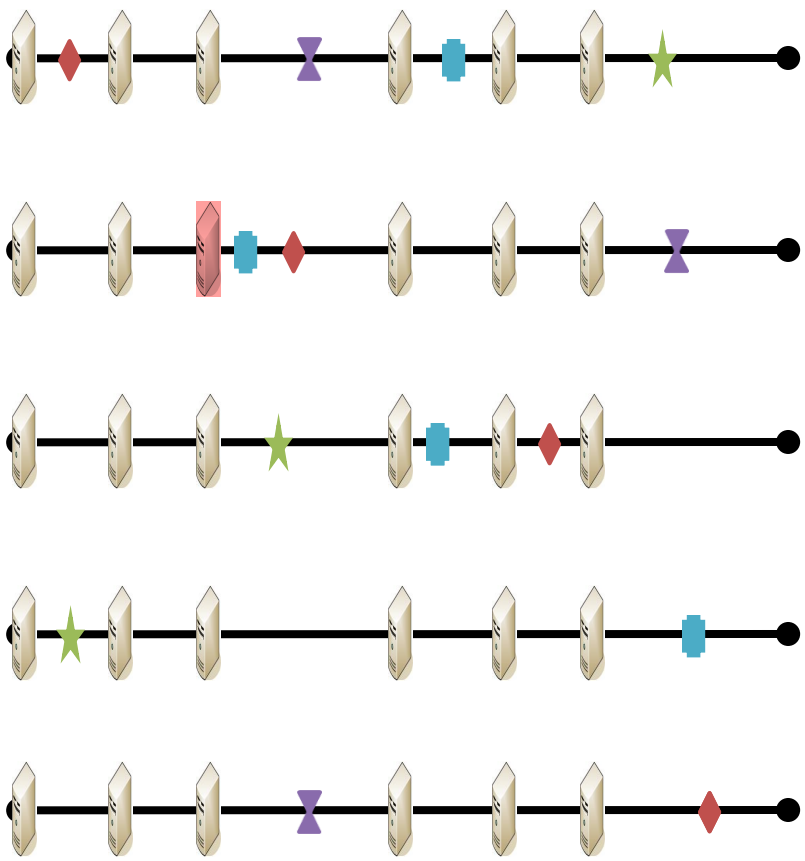
- **Segmentation:**
  - Divide storage servers into **k** segments
  - Each segment is an independent hash-based system
- **Failure Isolation:**
  - Servers with shared components in the same segment
  - **r** replicas per item, each on a different segment
  - Reduces impact of correlated failures

Kinesis Design => Freedom of Choice



- Balanced Resource Usage:
  - Multiple-choice paradigm
  - Write: r out of k choices
  - Read: 1 out of r choices

Kinesis Design => Scattered Data Placement



- Independent hash functions
  - Each server stores different set of items
- Parallel Recovery
  - Spread recovery load among multiple servers uniformly
  - Recover faster

Motivation =>
Data Placement Models

Hash-based

- Data is stored on a server determined by a hash function
- Server identified by a local computation during reads
- Low overhead
- Limited control in placing data items and replicas

Directory-based

- Any server can store any data
- A directory provides the server to fetch data form
- Expensive to maintain a globally-consistent directory in a large-scale system
- Can place data items carefully to maintain load balance, avoid correlated failure, etc.

Kinesis => Advantages

- Enables hash-based storage systems to have advantages of directory-based systems
  - Near-optimal load balance
  - Tolerance to shared-component failures
  - Freedom from problems induced by correlated replica placement
- No bottlenecks induced by directory maintenance
- Avoids slow data/replica placement algorithms

Evaluation

- Theory
- Simulations
- Experiments

Theory =>
The "Balls and Bins" Problem

- Load balancing is often modeled by the task of throwing balls (items) into bins (servers)
- Throw m balls into n bins:
  - Pick a bin uniformly at random
  - Insert the ball into the bin
- Single-Choice Paradigm:

Max Load: (with high prob.)	$\frac{m}{n} + \sqrt{\frac{m \log n}{n}}$	$m \geq n \log n$

Theory => Multiple-Choice Paradigm

- Throw m balls into n bins:
  - Pick d bins uniformly at random ($d \geq 2$)
  - Insert the ball into the **less-loaded** bin

- Excess load is independent of m, number of balls! [BCSV00]

Max Load: (with high prob.)	$\frac{m}{n} + \frac{\log \log n}{\log d}$	$m \geq n \log n$

Theory vs. Practice

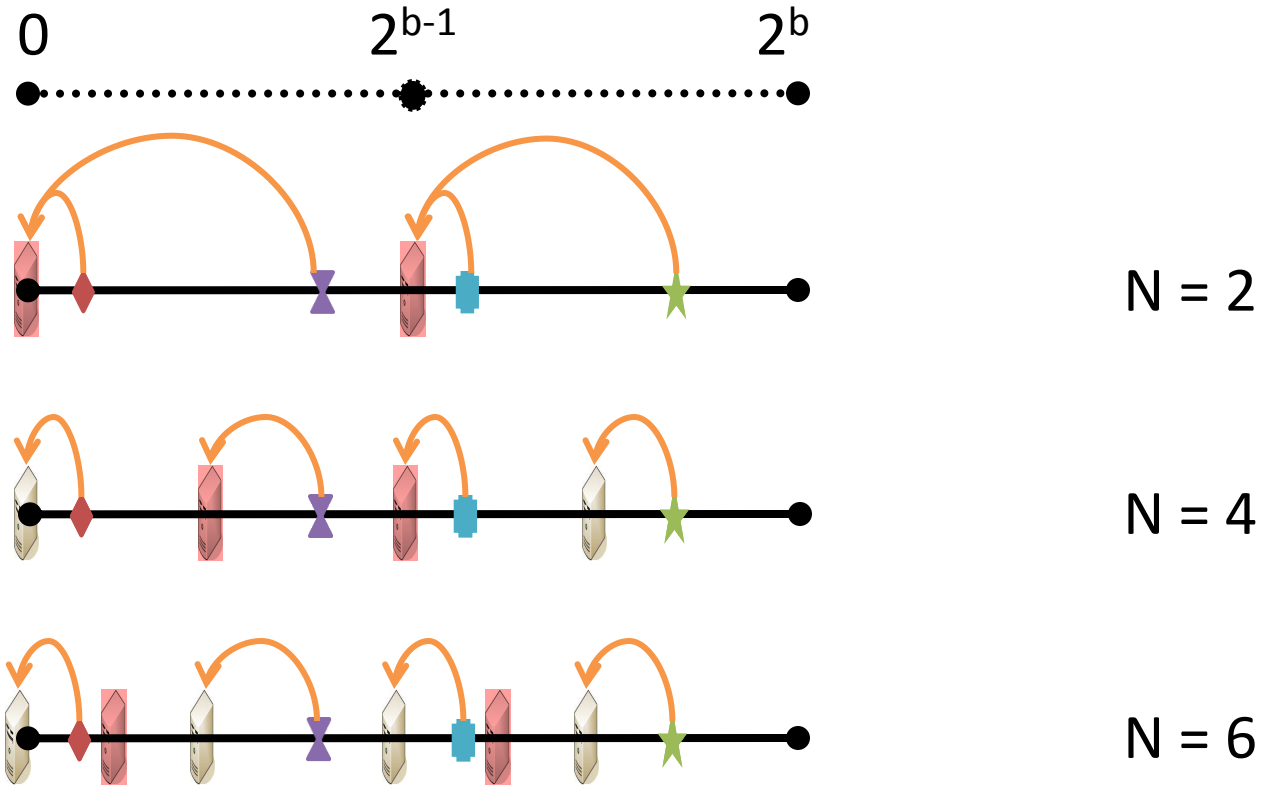
- ✓ Storage load vs. network load: [M91]
  - Network load is not persistent unlike storage load
- ✓ Non-uniform sampling: [V99],[W07]
  - Servers chosen based on consistent (or linear) hashing
- ✓ Replication: [MMRWYZ08]
  - Choose r out of k servers instead of 1 out of d bins
- ? Heterogeneity:
  - ✓ Variable-sized servers [W07]
  - ? Variable-sized items [TW07]

Theory => Planned Expansions

- Adding new, empty servers into the system
- Creates sudden, large imbalances in load
  - Empty servers vs. filled servers
- ? Eventual storage balance
  - Do subsequent inserts/writes fill up new servers?
- ? Eventual network load balance
  - Are new items distributed between old and new servers?
  - New items are often more popular than old items

Theory =>
Planned Expansions

- Expanding a linear-hashing system:



Theory => Planned Expansions

- Before Expansion:
  - Assume there are $k\ n$ servers on k segments
  - Storage is evenly balanced
- Expansion:
  - Add $\alpha\ n$ new servers to each segment
- After expansion:
  - $R = (1 - \alpha)\ n$ servers with twice the load as
  - $L = 2\ \alpha\ n$ servers (new servers + split servers)

Theory => Planned Expansions

- **Relative Load Distribution: R_L**
  - Ratio of expected number of replicas inserted in L servers
  - to expected number of replicas inserted on all servers

- $R_L > 1$ => eventual storage balance!
  - $R_L < 1$ => no eventual storage balance

✓ **Theorem: $R_L > 1$ if $k \geq 2r$** [MMRWYZ08]

Evaluation

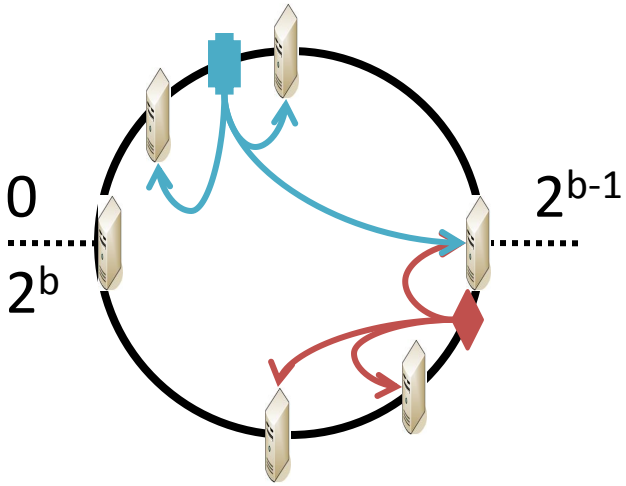
- ✓ Theory
- Simulations
- Experiments

Simulations => Overview

- Real-World Traces:

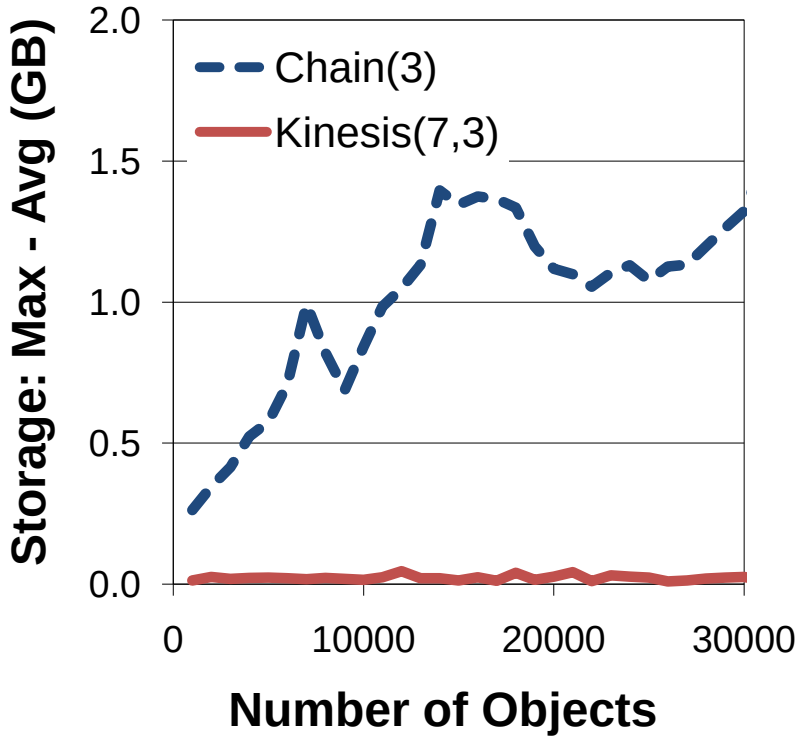
Trace	Num Files	Total Size	Type
MSNBC	30,656	2.33 TB	Read only
Plan-9	1.9 Million	173 GB	Read/write

- Compare with Chain:
  - one segment
  - single-choice paradigm
  - chained replica placement
  - E.g. PAST, CFS, Boxwood, Petal

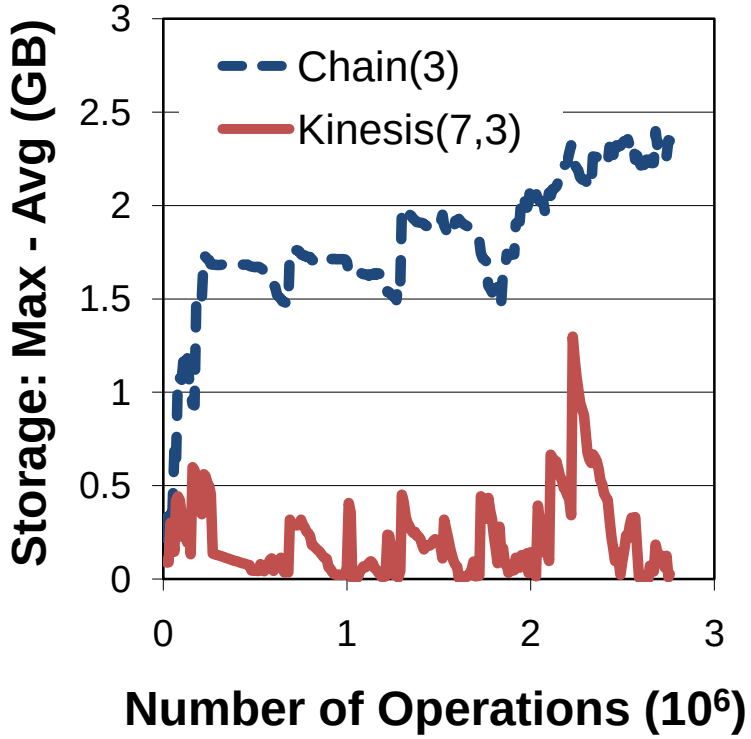


Simulations => Load Balance

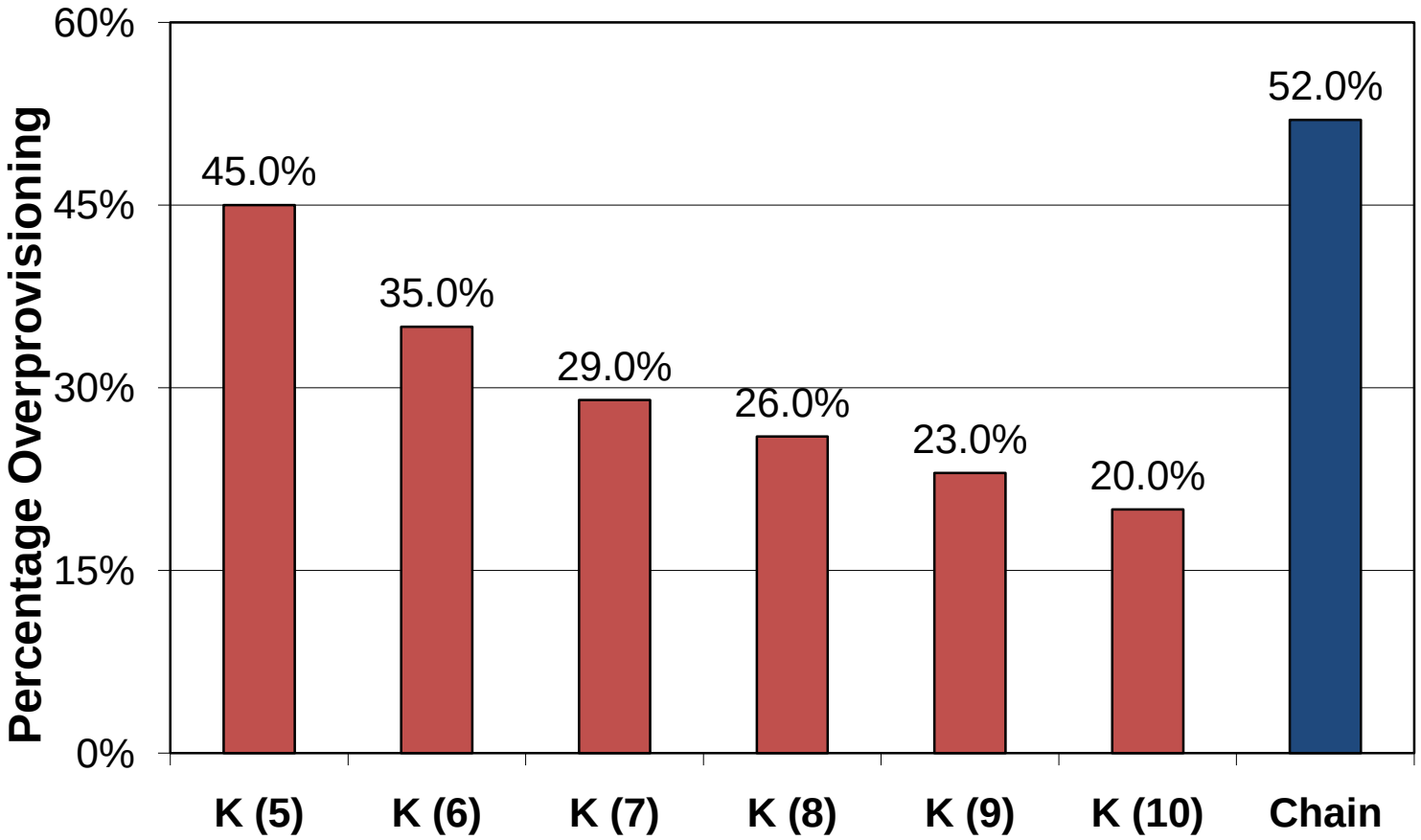
MSNBC Trace



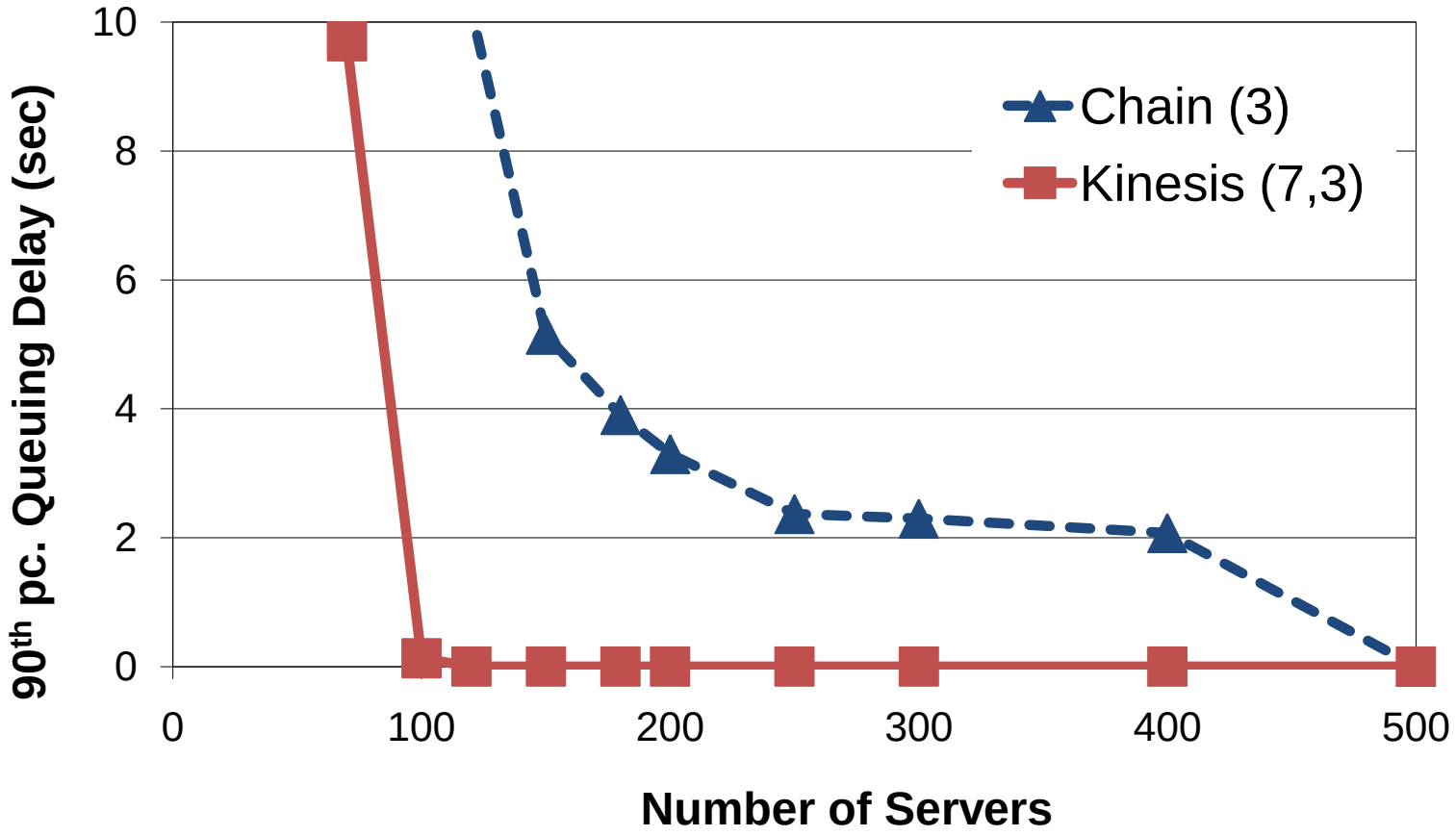
Plan-9 Trace



Simulations => System Provisioning



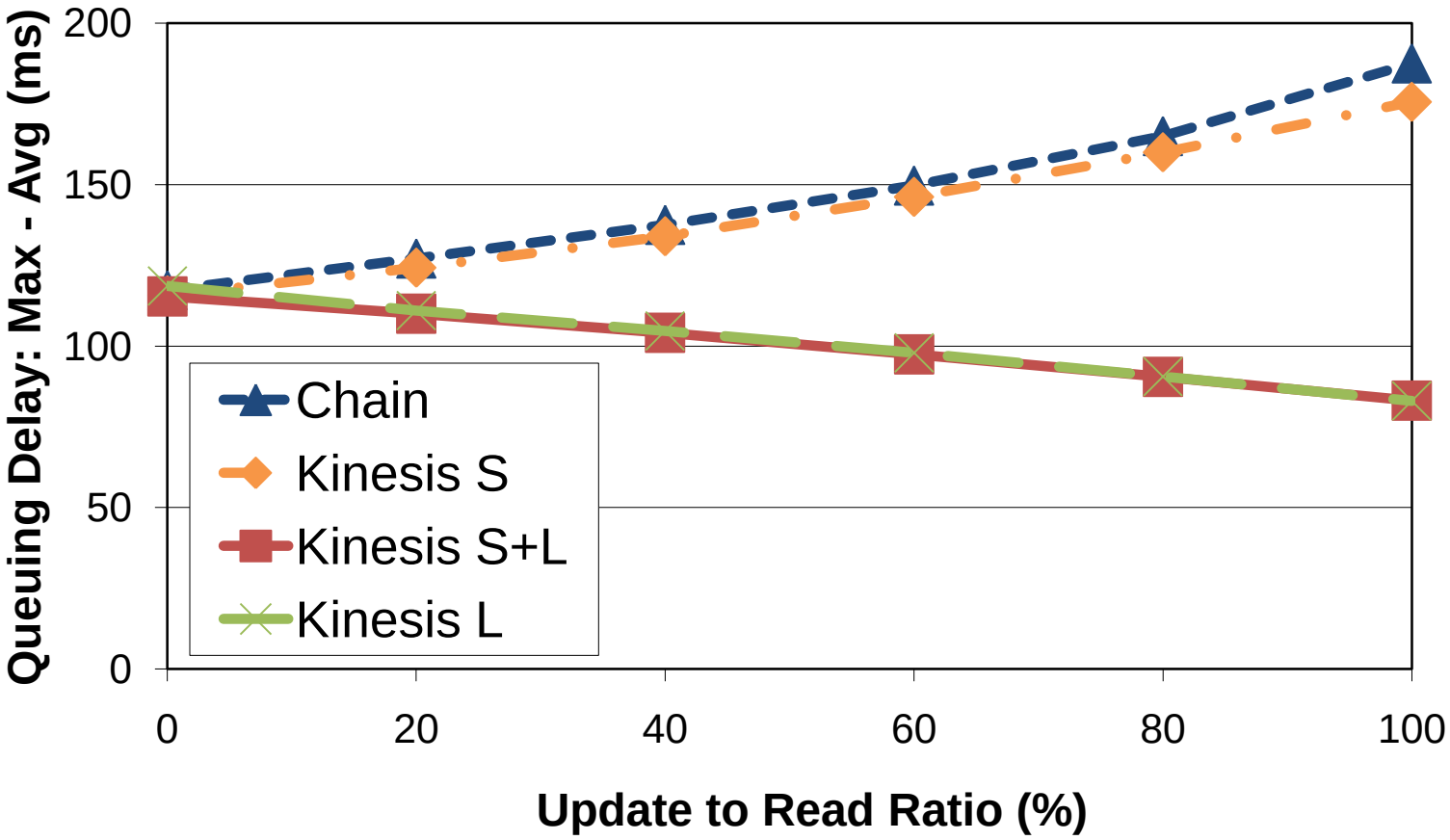
Simulations => User Experienced Delays



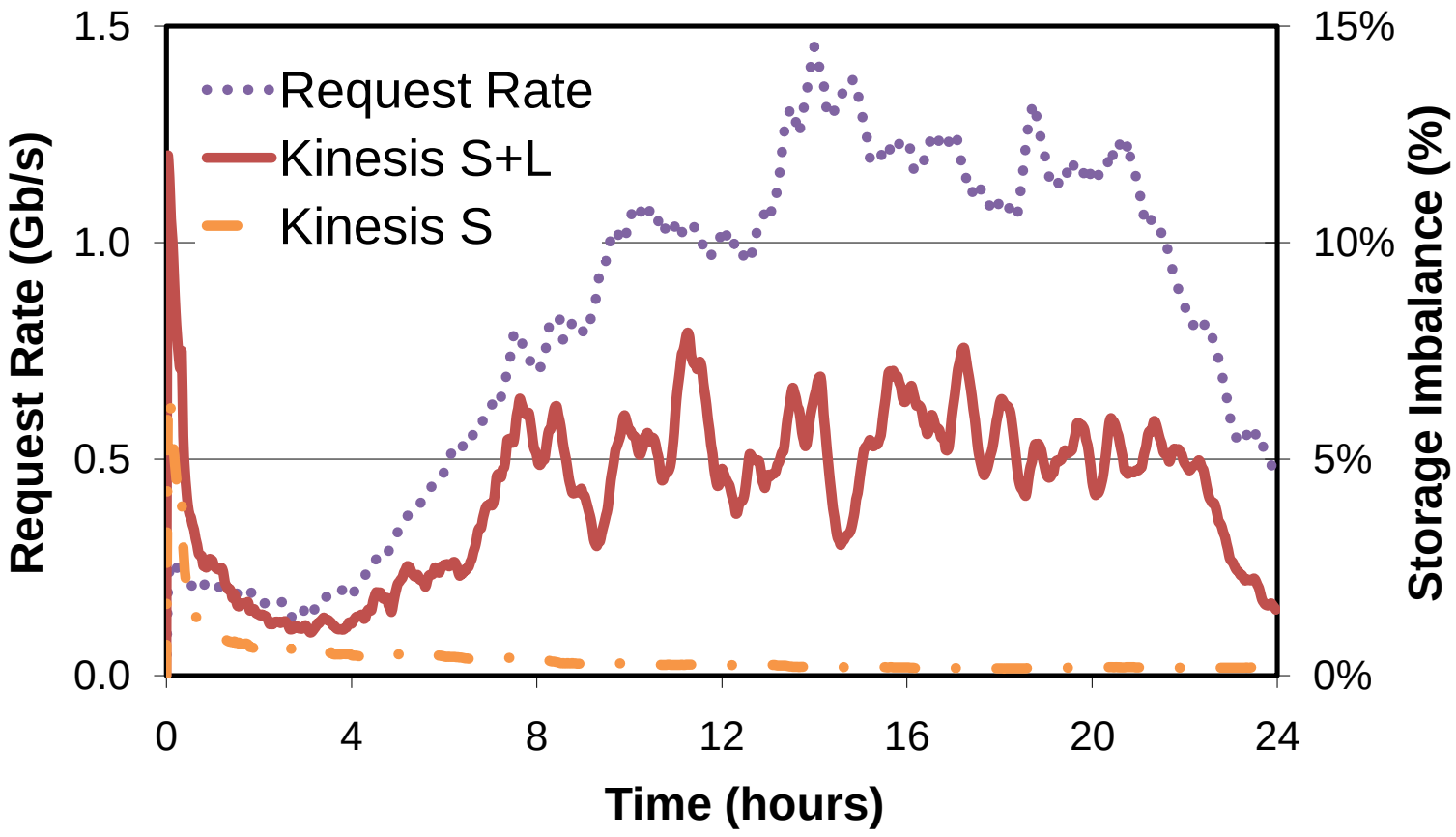
Simulations => Read Load vs. Write Load

- Read load: short term resource consumption
  - Network bandwidth, computation, disk bandwidth
  - Directly impacts user experience
- Write load: short and long term resource consumption
  - Storage space
- Solution (Kinesis S+L): Choose short term over long term
  - Storage balance is restored when transient resources are not bottlenecked

Simulations =>
Read Load vs. Write Load



Simulations =>
Read Load vs. Write Load



Evaluation

- ✓ Theory
- ✓ Simulations
- ➢ Experiments

Kinesis =>
Prototype Implementation

- Storage system for variable-sized objects
  - Read/Write interface
- Storage servers
  - Linear hashing system
  - Read, Append, Exists, Size (storage), Load (queued requests)
  - Failure recovery for primary replicas
    - Primary for an item is the highest numbered server with a replica

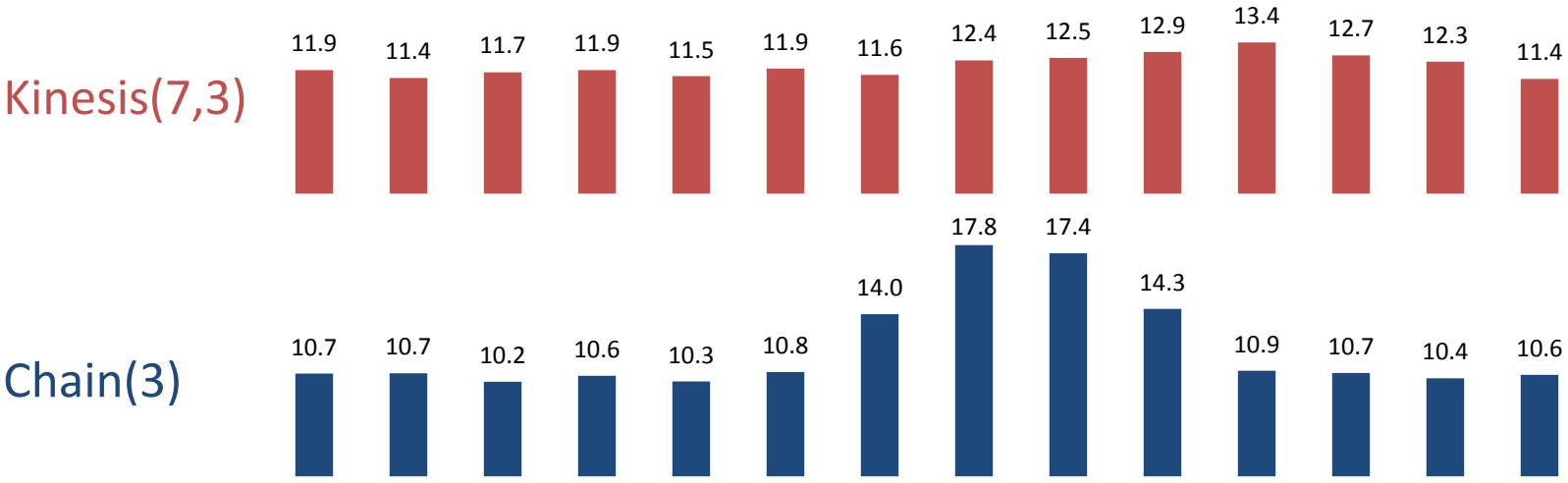
Kinesis => Prototype Implementation

- Front end
  - Two-step read protocol
    - 1) Query k candidate servers
    - 2) Fetch from least loaded server with a replica
  - Versioned updates with copy-on-write semantics
  - RPC-based communication with servers
- Non implemented!
  - Failure detector
  - Failure-consistent updates

Kinesis => Experiments

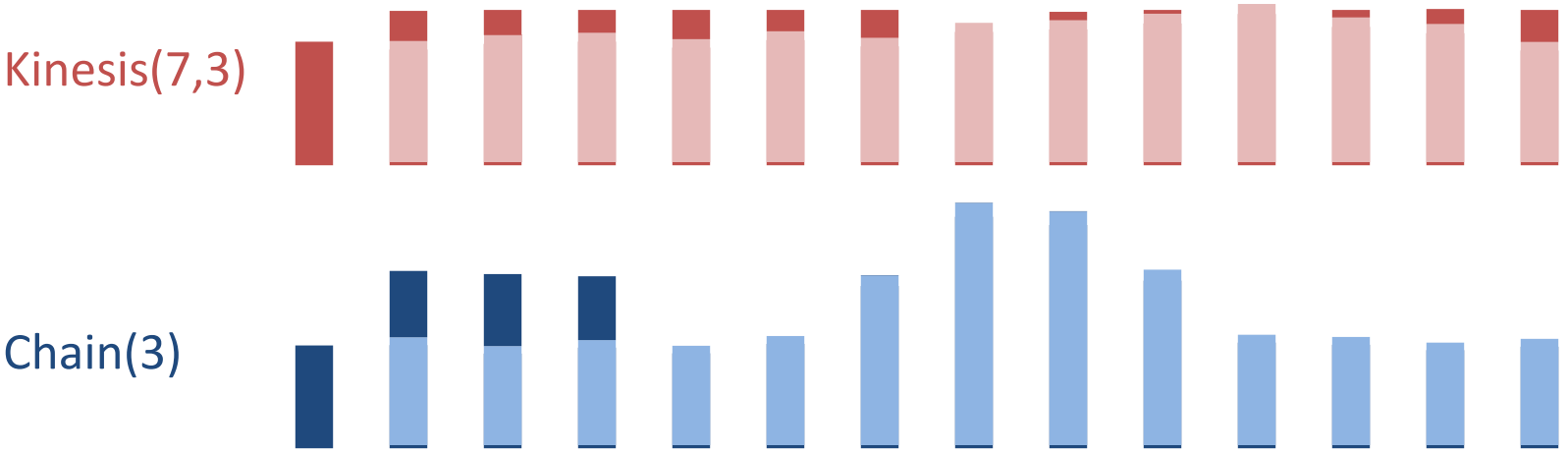
- 15 node LAN test-bed
- 14 storage servers and 1 front end
- MSNBC trace of 6 hours duration
  - 5000 files, 170 GB total size, and 200,000 reads
- Failure induced at 3 hours

Experiments => Kinesis vs. Chain



Average Latency	
Kinesis: 1.73 sec	Chain: 3 sec

Experiments => Failure Recovery



Total Recovery Time	
Kinesis: 17 min (12 servers)	Chain: 44 min (5 servers)

Related Work

- Prior work:
  - Hashing: Archipelago, OceanStore, PAST, CFS…
  - Two-choice paradigm: common load balancing technique
  - Parallel recovery: Chain Replication [RS04]
  - Random distribution: Ceph [WBML06]

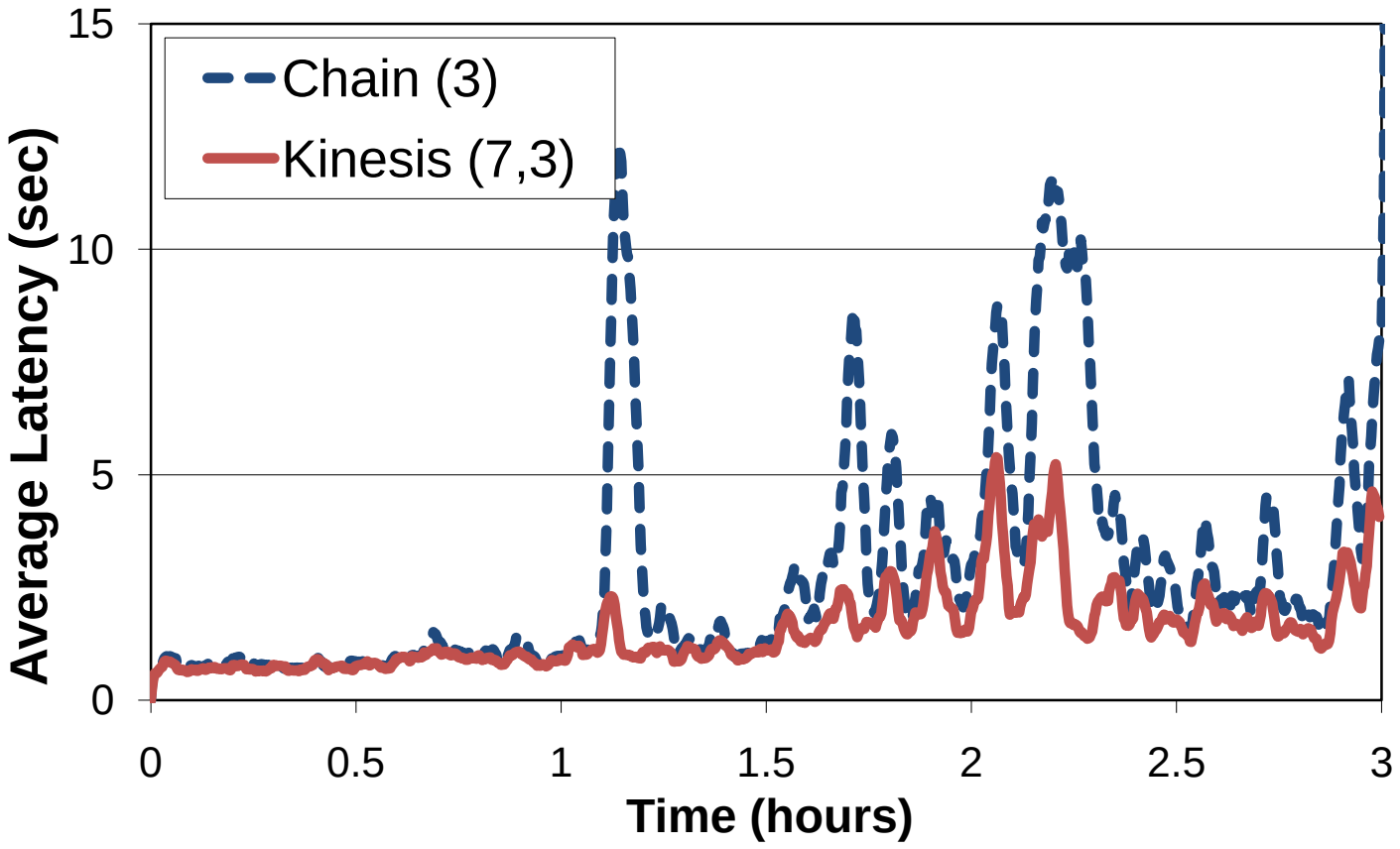
- Our contributions:
  - Putting them together in the context of storage systems
  - Extending theoretical results to the new design
  - Demonstrating the power of these simple design principles

Summary and Conclusions

- Kinesis: A data/replica placement framework for LAN storage systems
  - Structured organization, freedom-of-choice, and scattered distribution
- Simple and easy to implement, yet quite powerful!
  - Reduces infrastructure cost, tolerates correlated failures, and quickly restores lost replicas

Soon to appear in ACM Transactions on Storage, 2008
Download: http://research.microsoft.com/users/rama

Experiments => Kinesis vs. Chain



Experiments => Kinesis vs. Chain