the globus alliance
www.globus.org

Design, Performance and Scalability of a Replica Location Service

Ann L. Chervenak

Robert Schuler, Shishir Bharathi

USC Information Sciences Institute

Replica Management in Grids

Data intensive applications produce terabytes or petabytes of data

- Hundreds of millions of data objects

Replicate data at multiple locations for reasons of:

- Fault tolerance
  - Avoid single points of failure
- Performance
  - Avoid wide area data transfer latencies
  - Achieve load balancing

A Replica Location Service

- **A Replica Location Service (RLS)** is a distributed registry that records the locations of data copies and allows replica discovery
  - Must perform and scale well: support hundreds of millions of objects, hundreds of clients
- E.g., LIGO (Laser Interferometer Gravitational Wave Observatory) Project
  - RLS servers at 8 sites
  - Maintain associations between 3 million logical file names & 30 million physical file locations
- RLS is one component of a Replica Management system
  - Other components include consistency services, replica selection services, reliable data transfer, etc.

Talk Outline

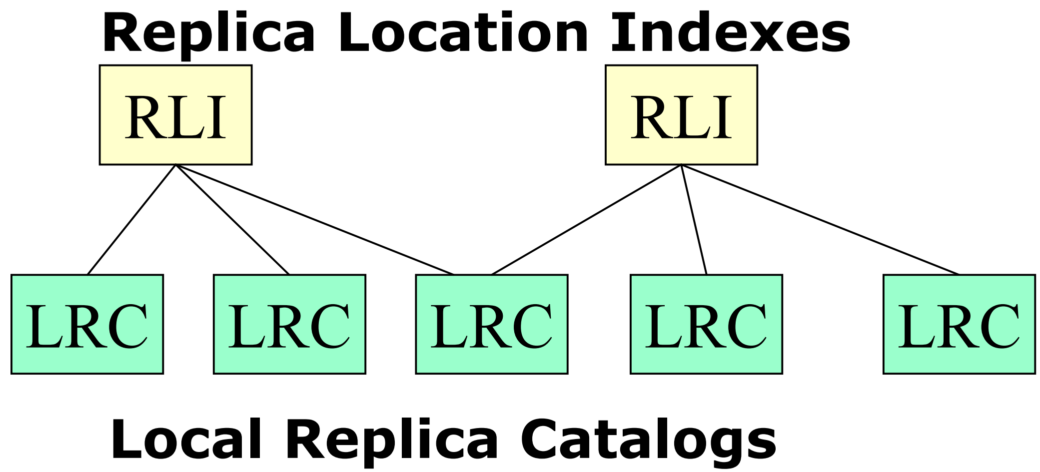
- Replica Location Service Overview
- An RLS Implementation
- Performance and Scalability Study
  - Individual server performance
  - Updates among distributed servers
- In Development: A Data Publication Service
- Research: A Peer-to-Peer RLS Implementation
- Summary and ongoing/future work

A Replica Location Service

- **A Replica Location Service (RLS)** is a distributed registry that records the locations of data copies and allows discovery of replicas
- RLS maintains mappings between *logical* identifiers and *target names*
- An RLS framework was designed in a collaboration between the Globus project and the DataGrid project (SC2002 paper)

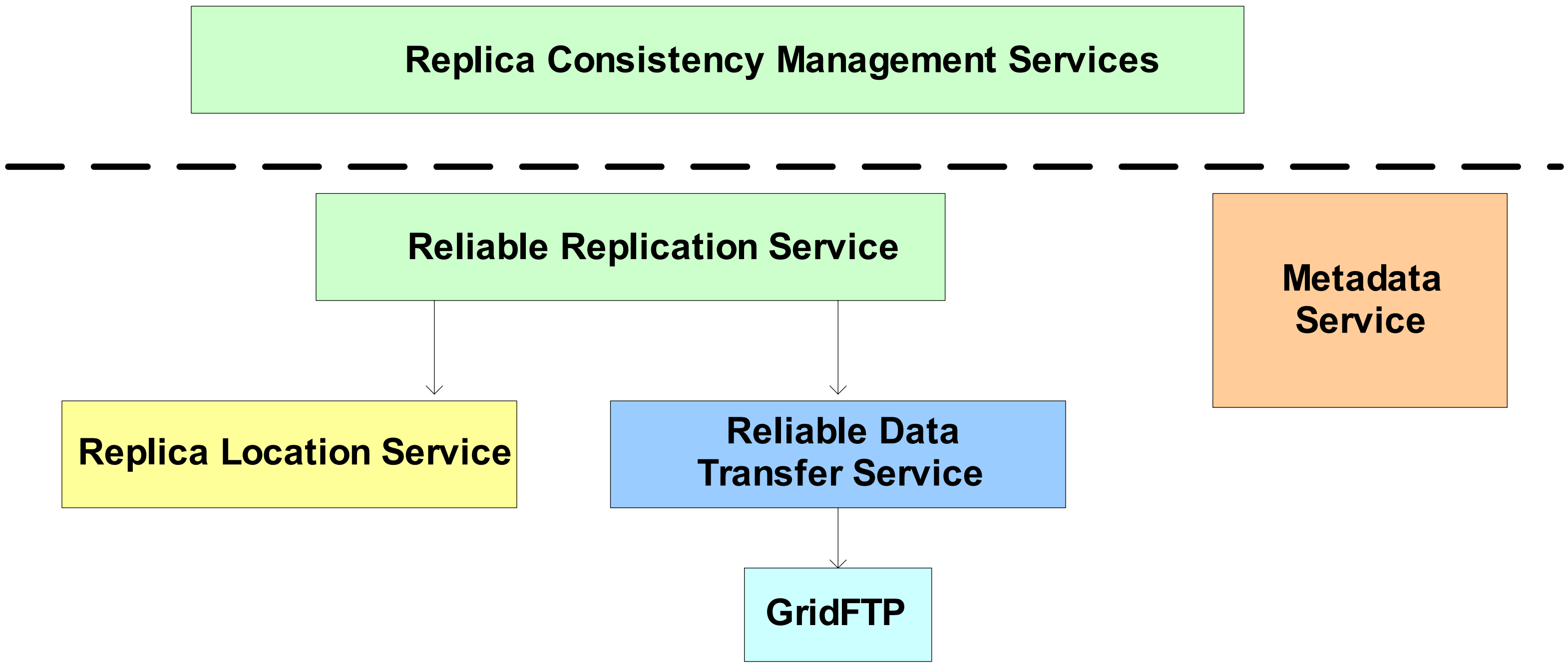
RLS Framework

- Local Replica Catalogs (LRCs) contain consistent information about logical-to-target mappings



- Replica Location Index (RLI) nodes aggregate information about one or more LRCs
- LRCs use soft state update mechanisms to inform RLIs about their state: relaxed consistency of index
- Optional compression of state updates reduces communication, CPU and storage overheads
- Membership service registers participating LRCs and RLIs and deals with changes in membership

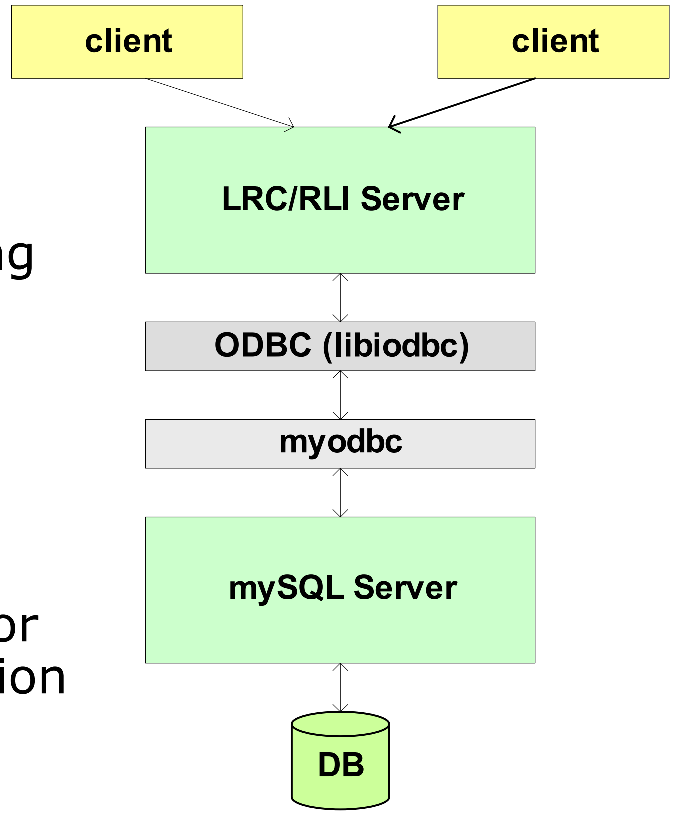
Replica Location Service In Context



- The Replica Location Service is one component in a layered data management architecture
- Provides a simple, distributed registry of mappings
- Consistency management provided by higher-level services

Components of RLS Implementation

- **Common server implementation for LRC and RLI**
- **Front-End Server**
  - Multi-threaded
  - Written in C
  - Supports GSI Authentication using X.509 certificates
- **Back-end Server**
  - MySQL or PostgreSQL Relational Database (later versions support Oracle)
  - No database back end required for RLIs using Bloom filter compression
- **Client APIs: C and Java**
- **Client Command line tool**



RLS Implementation Features

- Two types of soft state updates from LRCs to RLIs
  - Complete list of logical names registered in LRC
  - Compressed updates: Bloom filter summaries of LRC
- Immediate mode
  - Incremental updates
- User-defined attributes
  - May be associated with logical or target names
- Partitioning (without bloom filters)
  - Divide LRC soft state updates among RLI index nodes using pattern matching of logical names
- Currently, static membership configuration only
  - No membership service

Alternatives for Soft State Update Configuration

- LFN List
  - Send list of Logical Names stored on LRC
  - Can do exact and wildcard searches on RLI
  - Soft state updates get increasingly expensive as number of LRC entries increases
    - space, network transfer time, CPU time on RLI
  - E.g., with 1 million entries, takes 20 minutes to update mySQL on dual-processor 2 GHz machine (CPU-limited)
- Bloom filters
  - Construct a summary of LRC state by hashing logical names, creating a bitmap
  - Compression
  - Updates much smaller, faster
  - Supports higher query rate
  - Small probability of false positives (lossy compression)
  - Lose ability to do wildcard queries

Immediate Mode for Soft State Updates

- Immediate Mode
  - Send updates after 30 seconds (configurable) or after fixed number (100 default) of updates
  - Full updates are sent at a reduced rate
  - Tradeoff depends on volatility of data/frequency of updates
  - Immediate mode updates RLI quickly, reduces period of inconsistency between LRC and RLI content
- Immediate mode usually sends less data
  - Because of less frequent full updates
- Usually advantageous
  - An exception would be initially loading of large database

globus-rls-admin: Command Line Administration Tool

globus-rls-admin option **[** ***rli*** **] [** ***server*** **]**

-p: verifies that server is responding

-A: add RLI to list of servers to which LRC sends updates

-s: shows list of servers to which updates are sent

-c all: retrieves all configuration options

-S: show statistics for RLS server

-e: clear LRC database

Examples of globus-rls-admin commands

```
% globus-rls-admin -p rls://smarty
ping rls://smarty: 0 seconds

% globus-rls-admin -s rls://smarty
rls://smarty.isi.edu:39281      all LFNs
```

% globus-rls-admin -S rls://smarty

```
Version:    2.0.9
Uptime:     383:27:39
LRC stats
  update method: lfnlist
  update method: bloomfilter
  updates lfnlist:      rls://smarty.isi.edu:39281 last 01/21/04
    11:09:35
  lfnlist update interval: 3600
  bloomfilter update interval: 900
  numlfn: 10719
  numpfn: 33560
  nummap: 33560
RLI stats
  updated by: rls://smarty.isi.edu:39281 last 01/21/04 11:35:45
  updated by: rls://sukhna.isi.edu:39281 last 01/20/04 17:33:17
  updated via lfnlists
  numlfn: 11384
  numlrc: 2
  nummap: 15363
```

globus-rls-cli: Client Command Line Tool

globus-rls-cli [-c] [-h] [-l reslimit] [-s] [-t timeout] [-u] [command] rls-server

- If command is not specified, enters interactive mode

- Create an initial mapping from a logical name to a target name:

globus-rls-cli create logicalName targetName1 rls://myrls.isi.edu

- Add a mapping from same logical name to a second replica/target name:

globus-rls-cli add logicalName targetName2 rls://myrls.isi.edu

Examples of simple create, add and query operations

```
% globus-rls-cli create ln1 pn1 rls://smarty

% globus-rls-cli query lrc lfn ln1 rls://smarty
  ln1: pn1

% globus-rls-cli add ln1 pn2 rls://smarty

% globus-rls-cli query lrc lfn ln1 rls://smarty
  ln1: pn1
  ln1: pn2
```

globus-rls-cli Attribute Functions

Attribute Functions

- **globus-rls-cli attribute add** <object> <attr> <obj-type> <attr-type>
  - Add an attribute to an object
  - *object* should be the lfn or pfn name
  - *obj-type* should be one of lfn or pfn
  - *attr-type* should be one of date, float int, or string
- **attribute modify <object> <attr> <obj-type> <attr-type>**
- **attribute query <object> <attr> <obj-type>**

globus-rli-client Bulk Operations

- **bulk add <lfn> <pfn> [<lfn> <pfn>**
  - Bulk add lfn, pfn mappings
- **bulk delete <lfn> <pfn> [<lfn> <pfn>**
  - Bulk delete lfn, pfn mappings
- **bulk query lrc lfn [<lfn> ...]**
  - Bulk query lrc for lfns
- **bulk query lrc pfn [<pfn> ...]**
  - Bulk query lrc for pfns
- **bulk query rli lfn [<lfn> ...]**
  - Bulk query rli for lfns
- Others: bulk attribute adds, deletes, queries, etc.

Examples of Bulk Operations

% globus-rls-cli bulk create ln1 pn1 ln2 pn2 ln3 pn3 rls://smarty

% globus-rls-cli bulk query lrc lfn ln1 ln2 ln3 rls://smarty

ln3: pn3

ln2: pn2

ln1: pn1

Registering a mapping using C API

globus_module_activate(GLOBUS_RLS_CLIENT_MODULE)

globus_rls_client_connect (serverURL, serverHandle)

globus_rls_client_lrc_create (serverHandle, logicalName1, targetName1)

globus_rls_client_lrc_add (serverHandle, logicalName1, targetName2)

globus_rls_client_close (serverHandle)

Registering a mapping using Java API

```
RLSClient rls = new RLSClient(URLofServer);

RLSClient.LRC lrc = rls.getLRC();

lrc.create(logicalName1, targetName1);

lrc.add(logicalName1, targetName2);

rls.Close();
```

Talk Outline

- Replica Location Service Overview
- An RLS Implementation
- Performance and Scalability Study
  - Individual server performance
  - Updates among distributed servers
- In Development: A Data Publication Service
- Research: A Peer-to-Peer RLS Implementation
- Summary and ongoing/future work

Performance Testing

- Extensive performance testing reported in HPDC 2004 paper
- Performance of individual LRC (catalog) or RLI (index) servers
  - Client program submits operation requests to server
- Performance of soft state updates
  - Client LRC catalogs sends updates to index servers

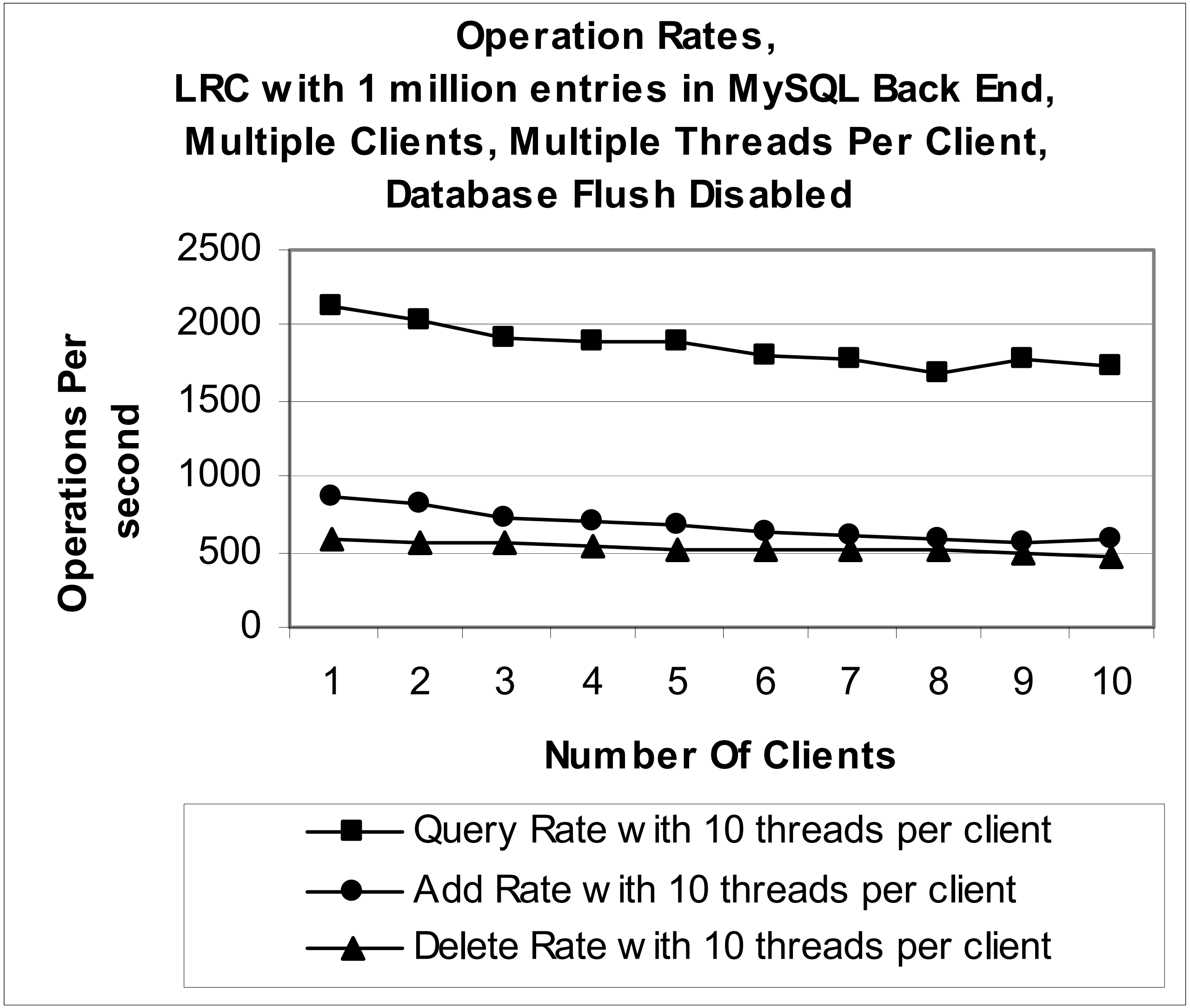
Software Versions:

- Replica Location Service Version 2.0.9
- Globus Packaging Toolkit Version 2.2.5
- libiODBC library Version 3.0.5
- MySQL database Version 4.0.14
- MyODBC library (with MySQL) Version 3.51.06

Testing Environment

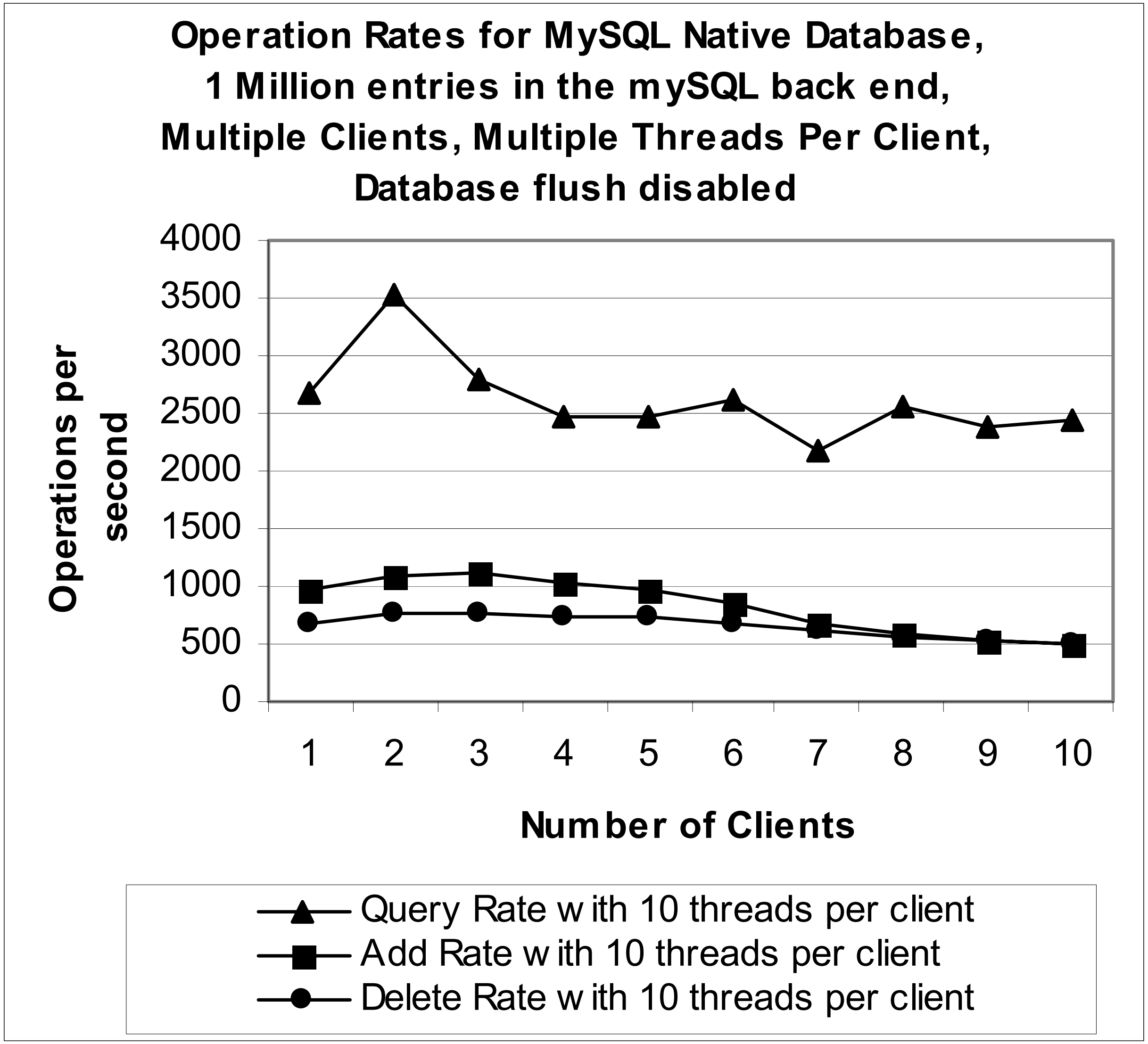
- Local Area Network Tests
  - 100 Megabit Ethernet
  - Clients (either client program or LRCs) on cluster: dual Pentium-III 547 MHz workstations with 1.5 Gigabytes of memory running Red Hat Linux 9
  - Server: dual Intel Xeon 2.2 GHz processor with 1 Gigabyte of memory running Red Hat Linux 7.3
- Wide Area Network Tests (Soft state updates)
  - LRC clients (Los Angeles): cluster nodes
  - RLI server (Chicago): dual Intel Xeon 2.2 GHz machine with 2 gigabytes of memory running Red Hat Linux 7.3

LRC Operation Rates (MySQL Backend)



- Up to 100 total requesting threads
- Clients and server on LAN
- Query: request the target of a logical name
- Add: register a new <logical name, target> mapping
- Delete a mapping

Comparison of LRC to Native MySQL Performance



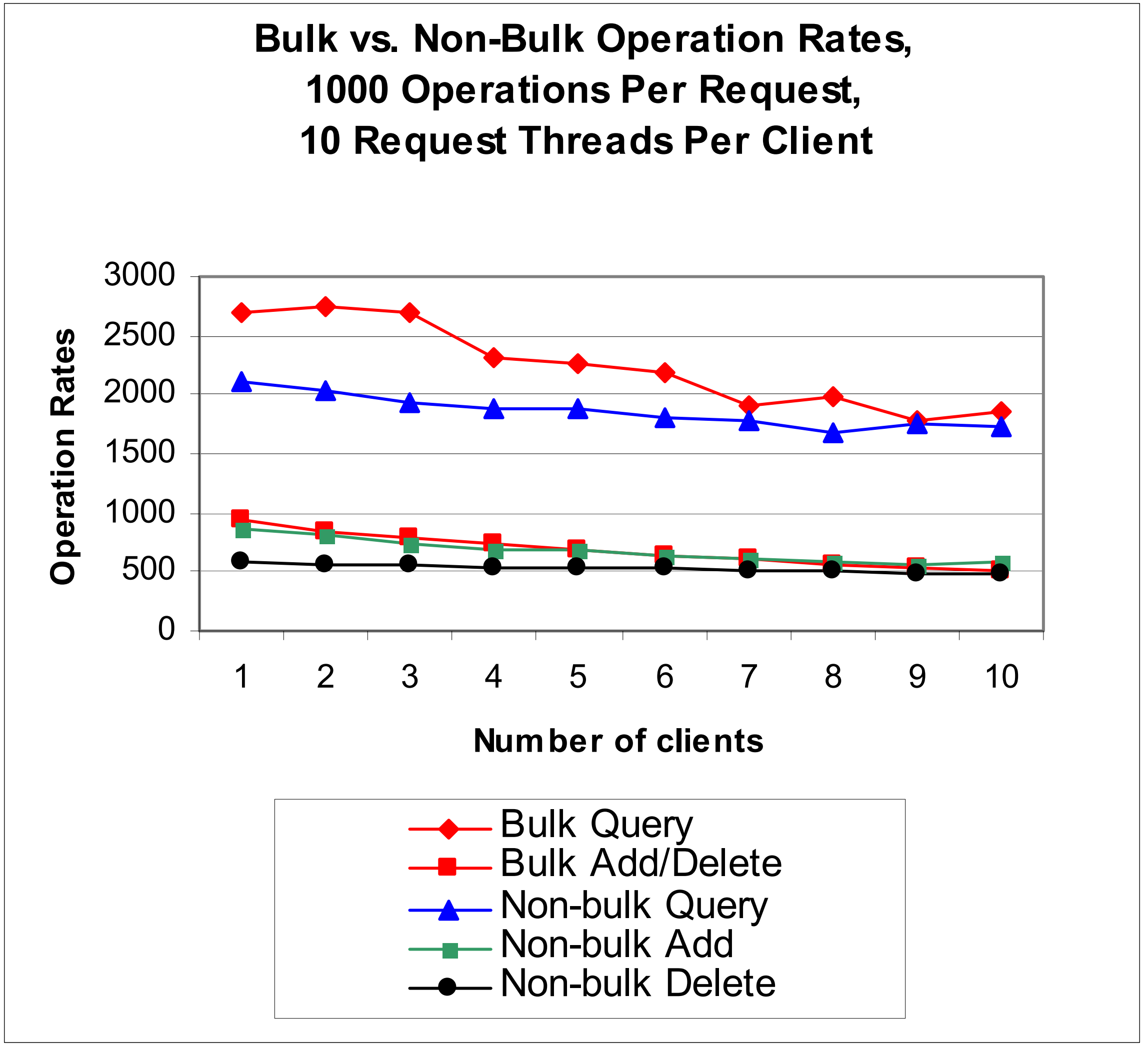
LRC Overheads

Highest for queries: LRC achieve 70-80% of native rates

Adds and deletes: ~90% of native performance for 1 client (10 threads)

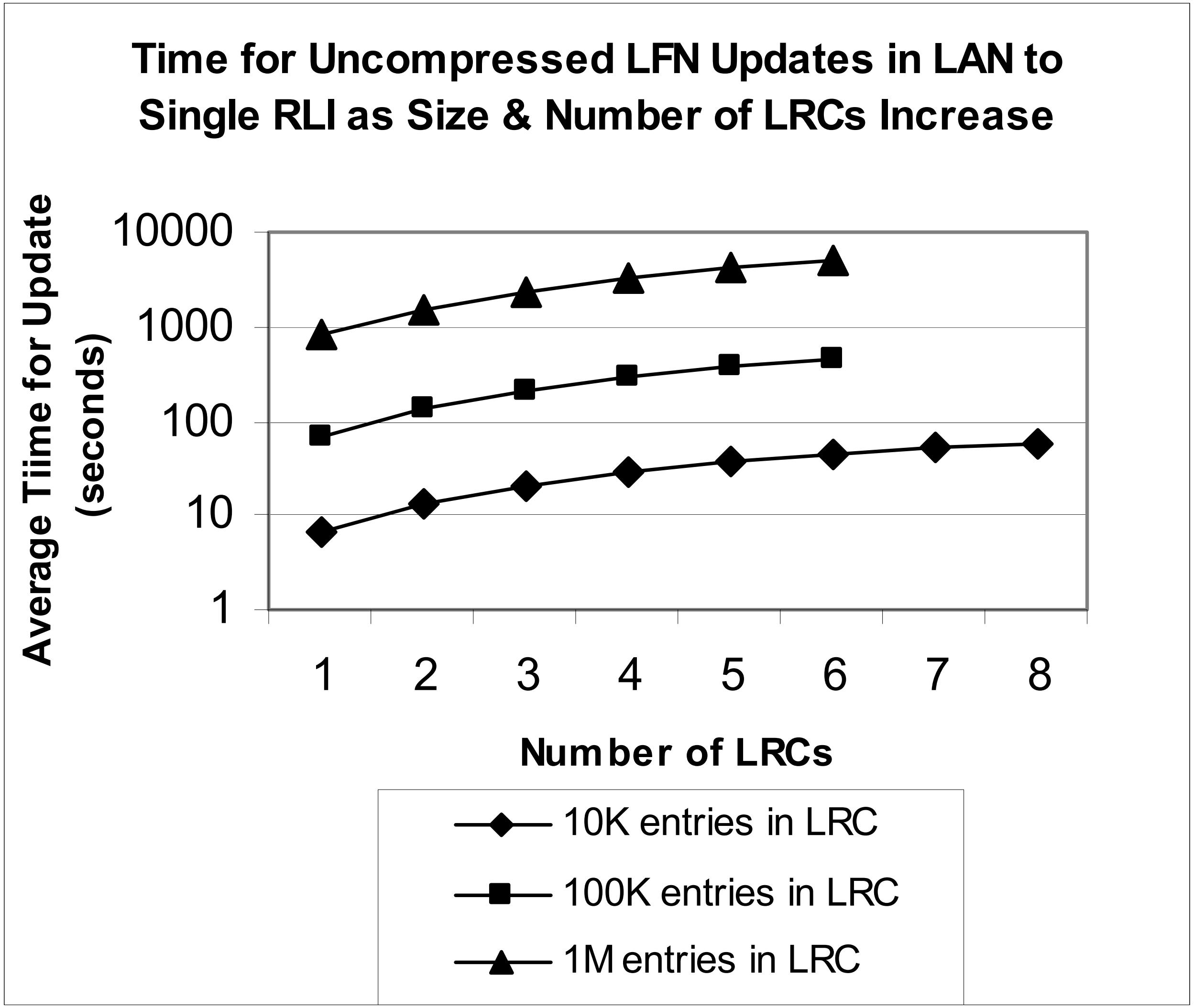
Similar or better add and delete performance with 10 clients (100 threads)

Bulk Operation Performance



- For user convenience, server supports bulk operations
- E.g., 1000 operations per request
- Combine adds/deletes to maintain approx. constant DB size
- For small number of clients, bulk operations increase rates
- E.g., 1 client (10 threads) performs 27% more queries, 7% more adds/deletes

Uncompressed Soft State Updates



- Perform poorly when multiple LRCs update RLI
- E.g., 6 LRCs with 1 million entries updating RLI, average update ~5102 seconds in Local Area
- Limiting factor: rate of updates to an RLI database
- Advisable to use incremental updates

Bloom Filter Compression

- Construct a summary of each LRC's state by hashing logical names, creating a bitmap
- RLI stores in memory one bitmap per LRC

Advantages:

- Updates much smaller, faster
- Supports higher query rate
  - Satisfied from memory rather than database

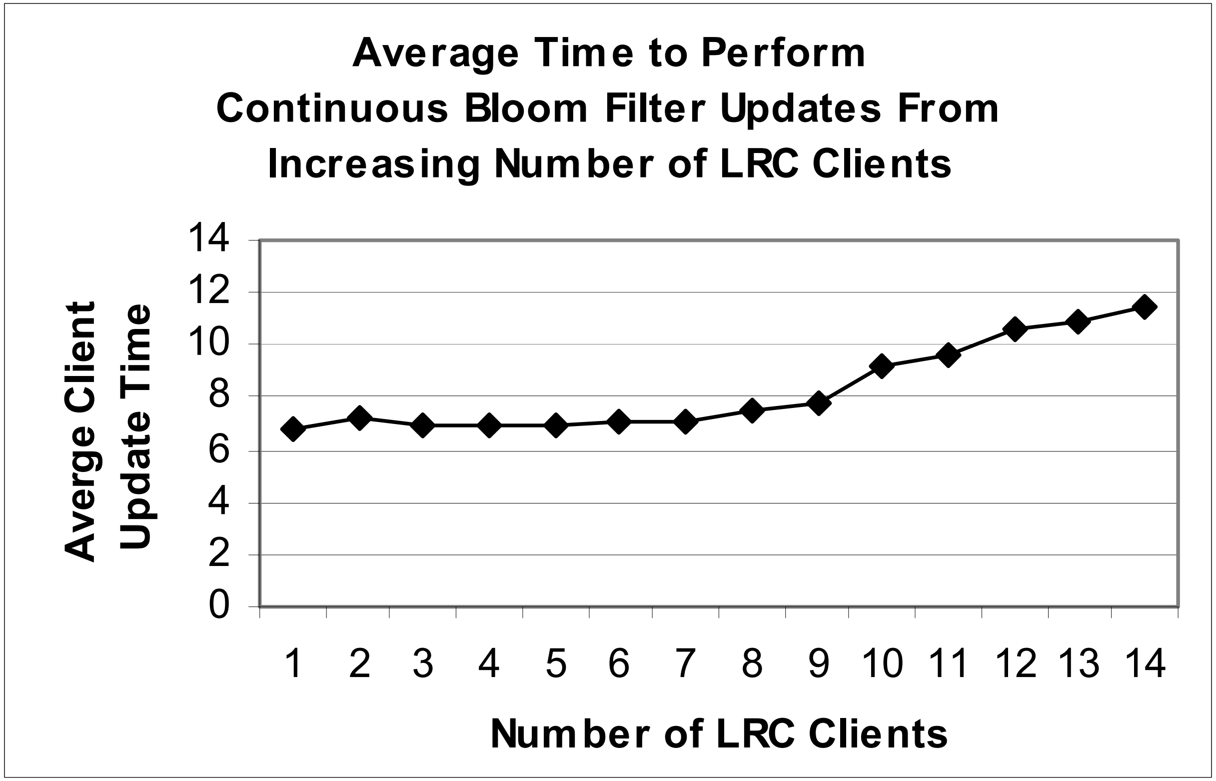
Disadvantages:

- Lose ability to do wildcard queries, since not sending logical names to RLI
- Small probability of false positives (configurable)
  - Relaxed consistency model

Bloom Filter Performance: Single Wide Area Soft State Update (Los Angeles to Chicago)

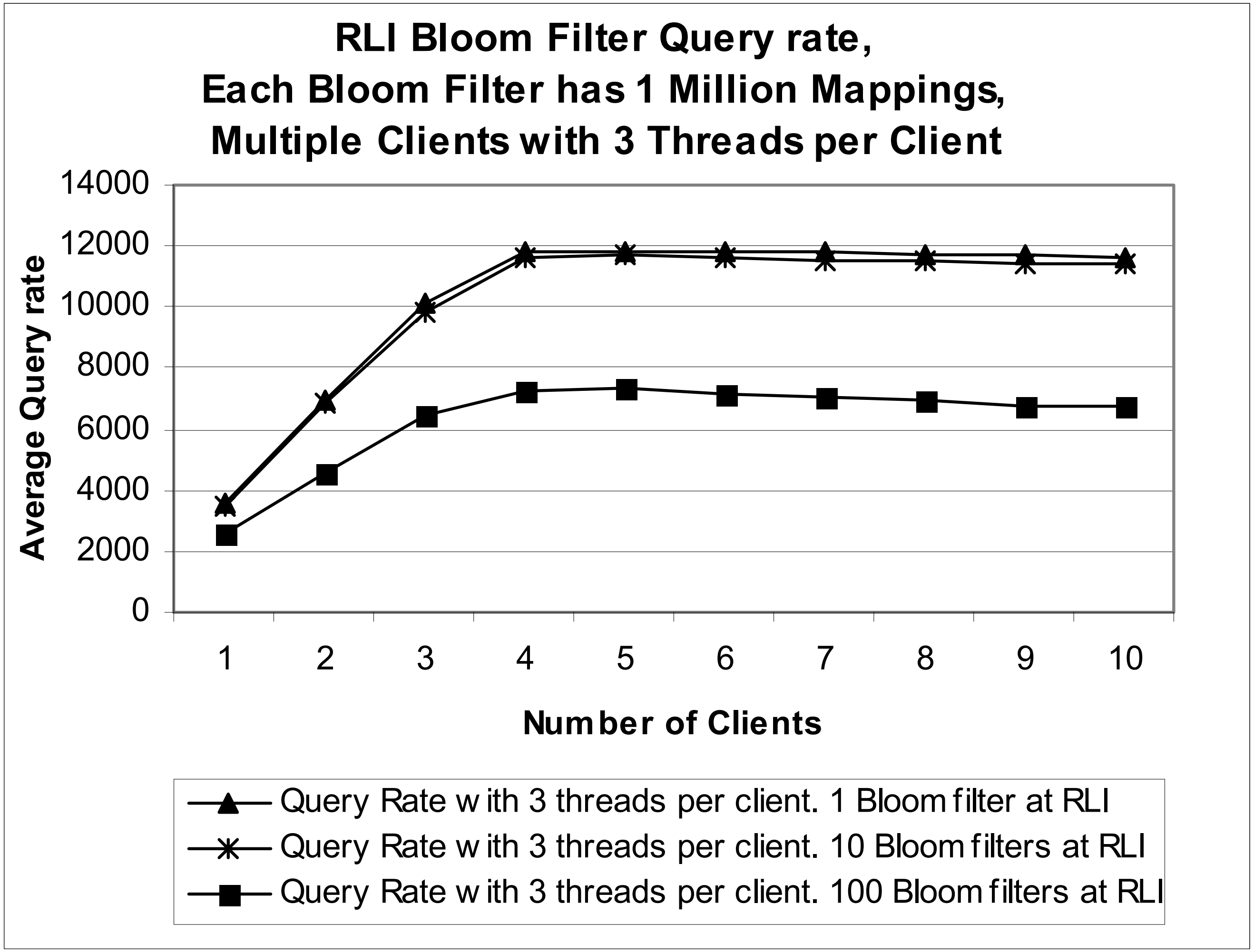
LRC Database Size	Avg. time to send soft state update (seconds)	Avg. time for initial bloom filter computation (seconds)	Size of bloom filter (bits)
100,000 entries	Less than 1	2	1 million
1 million entries	1.67	18.4	10 million
5 million entries	6.8	91.6	50 million

Scalability of Bloom Filter Updates



- 14 LRCs with 5 million mappings send Bloom filter updates continuously in Wide Area (unlikely, represents worst case)
- Update times increase when 8 or more clients send updates
- 2 to 3 orders of magnitude better performance than uncompressed (e.g., 5102 seconds with 6 LRCs)

Bloom Filter Compression Supports Higher RLI Query Rates



- Uncompressed updates: about 3000 queries per second
- Higher rates with Bloom filter compression
- Scalability limit: significant overhead to check 100 bit maps
- Practical deployments: <10 LRCs updating an RLI

RLS Performance Summary

Individual RLS servers perform well and scale up to

- Millions of entries
- One hundred requesting threads

- Soft state updates of the distributed index scale well when using Bloom filter compression

- Uncompressed updates slow as size of catalog grows
  - Immediate mode is advisable

Talk Outline

- Replica Location Service Overview
- An RLS Implementation
- Performance and Scalability Study
  - Individual server performance
  - Updates among distributed servers
- In Development: A Data Publication Service
- Research: A Peer-to-Peer RLS Implementation
- Summary and ongoing/future work

WS-RF Data Publishing and Replication Service

- Being developed for the Tech Preview of GT4.0 release
- Based in part on Lightweight Data Replicator system (LDR) developed by Scott Koranda from U. Wisconsin at Milwaukee

- Ensures that a specified set of files exist on a storage site
  - Compares contents of a local file catalog with a list of desired files
  - Transfers copies of missing files other locations
  - Registers them in the local file catalog

- Uses a pull-based model
  - Localizes decision making
  - Minimizes dependency on outside services

the globus alliance
www.globus.org

Publishing and Replication Service (Cont.)

- WS-RF interface allows a client to explicitly specify the list of files that should exist at the local site
  - associates priorities with files should they need to be replicated from another site
  - allows clients to remove files from this list
- Each storage site uses the Replica Location Service (RLS) to determine
  - what files from the desired set are missing from the local storage system
  - where missing files exist elsewhere in the Grid
- Missing files are replicated locally
  - Issue requests to pull data to the local site from remote copies using the Reliable File Transfer Service (RFT)
- After files are transferred, they are registered in the Local Replica Catalog

Talk Outline

- Replica Location Service Overview
- An RLS Implementation
- Performance and Scalability Study
  - Individual server performance
  - Updates among distributed servers
- In Development: A Data Publication Service
- Research: A Peer-to-Peer RLS Implementation
- Summary and ongoing/future work

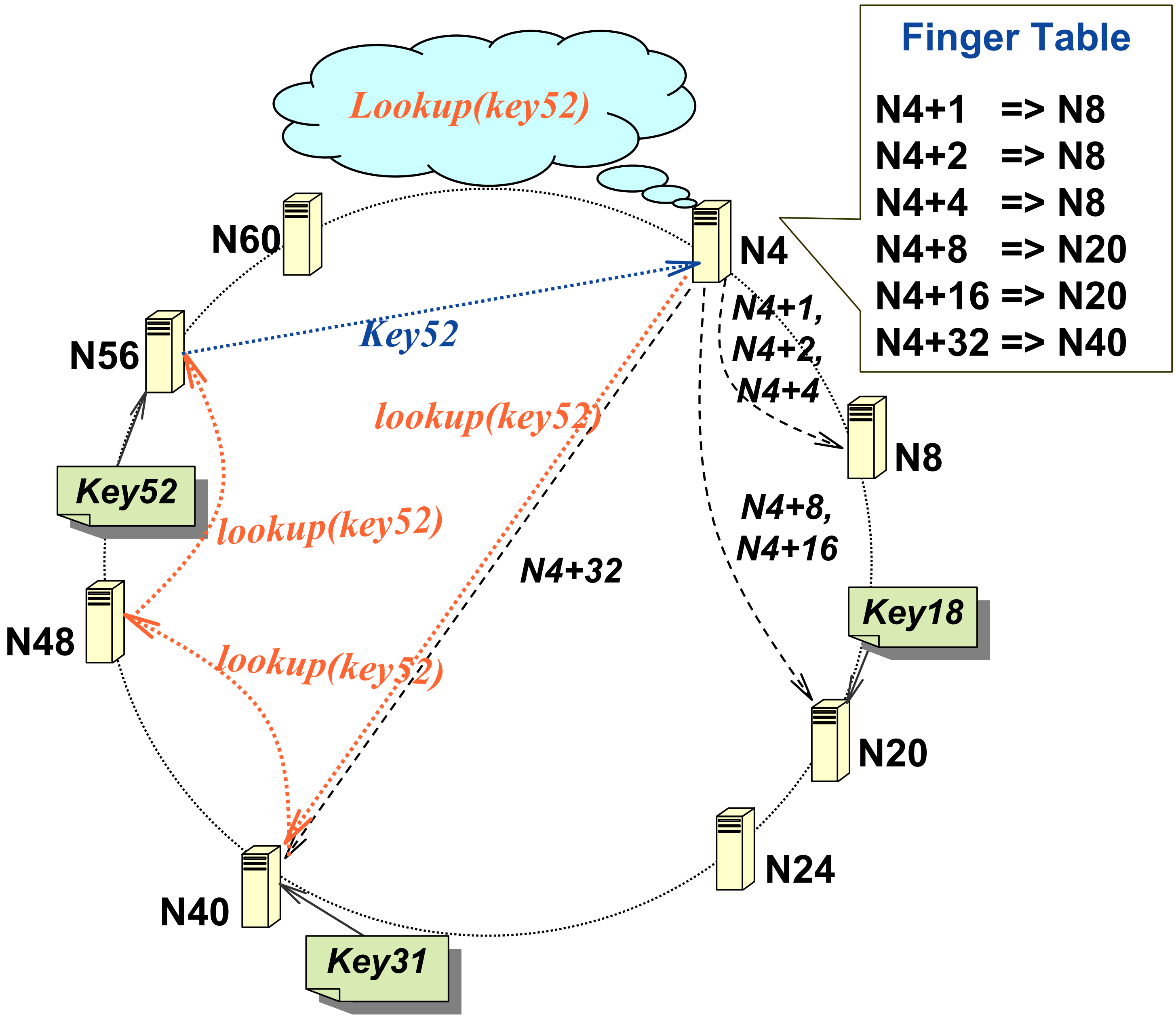
Motivation for a Peer-to-Peer RLS

- Each RLS deployment is statically configured
  - If upper level RLI fails, the lower level LRCs need to be manually redirected
- More automated and flexible membership management is desirable for:
  - larger deployments
  - dynamic environments where servers frequently join and leave
- We use a peer-to-peer approach to provide a distributed RLI index for {*logical-name, LRC*} mappings with properties of:
  - self-organization
  - greater fault-tolerance and availability
  - improved scalability for large number of RLS nodes

Peer-to-Peer Replica Location Service (P-RLS) Design

- Work of Min Cai, Ph.D. student
- A P-RLS server consists of:
  - An unchanged Local Replica Catalog (LRC) to maintain consistent {*logical-name, target-name*} mappings
  - A Peer-to-Peer Replica Location Index node (P-RLI)
- The P-RLS design uses a Chord overlay network to self-organize P-RLI servers
  - Chord is a distributed hash table that supports scalable key insertion and lookup
  - Each node has log (N) neighbors in a network of N nodes
  - A key is stored on its *successor node* (first node with ID equal to or greater than key)
  - Key insertion and lookup in log (N) hops
  - Stabilization algorithm for overlay construction and topology repair

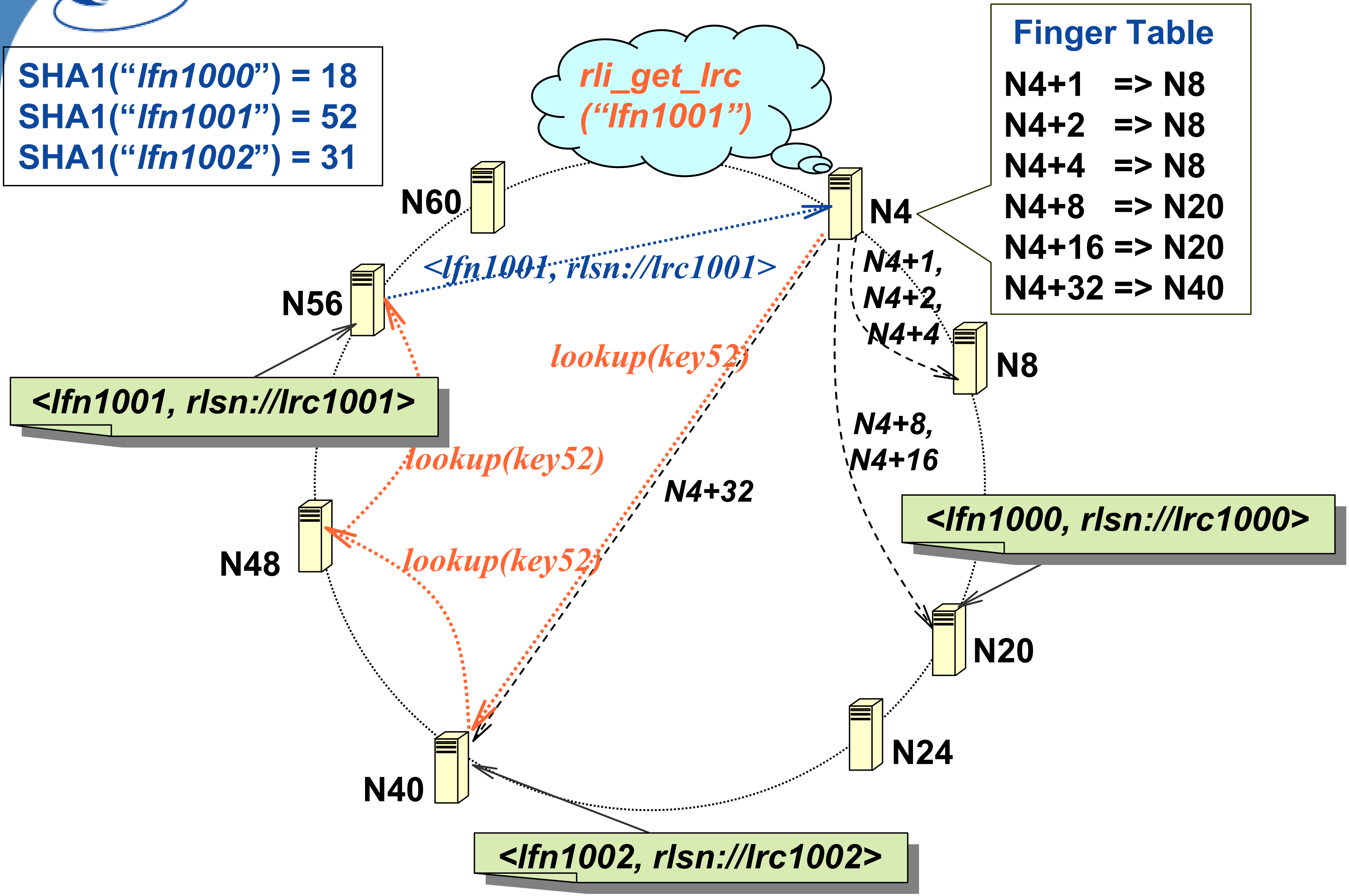
An Example of Chord Network



P-RLS Design (Cont.)

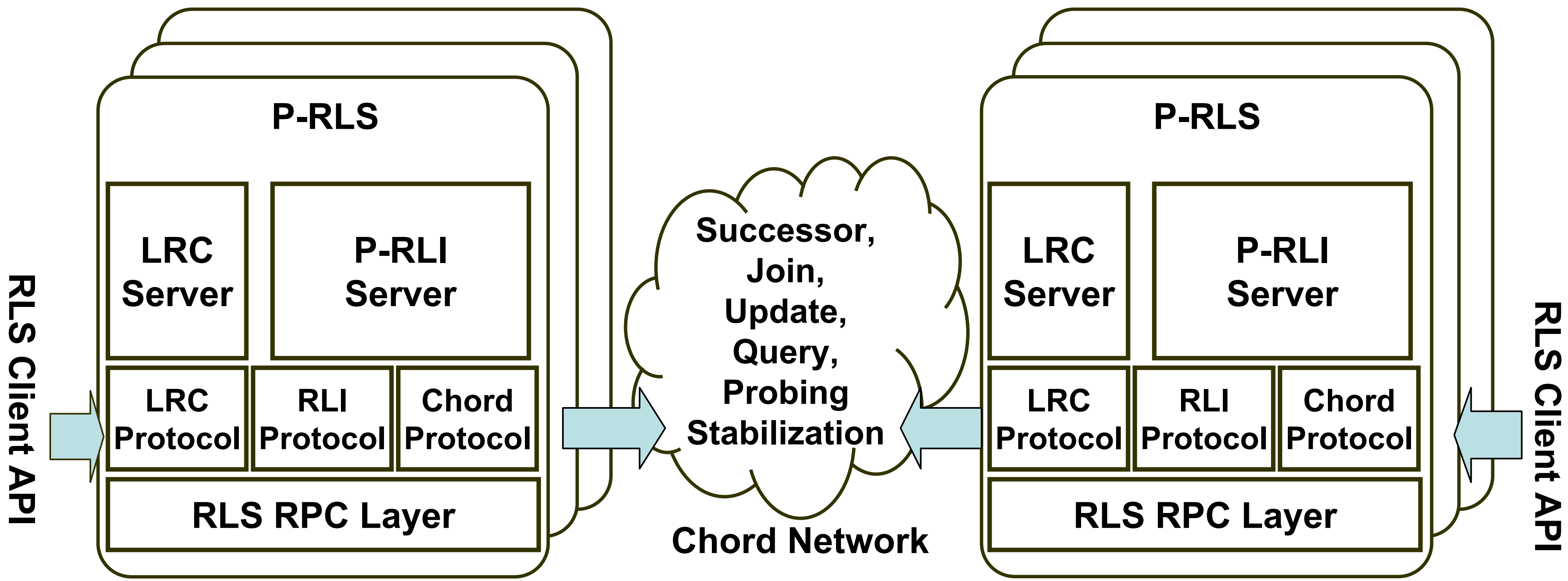
- Uses Chord algorithm to store mappings of logical names to LRC sites
  - Generates Chord key for a logical name by applying SHA1 hash function
  - Stores {*logical-name, LRC*} mappings on the P-RLI successor node, called the *root node* of the mapping
- When P-RLI node receives a query for LRC(s) that store mappings for a logical name:
  - Answers the query if it contains the logical-to-LRC mapping(s)
  - If not, routes query to the root node that contains the mappings
- Then query LRCs directly for mappings from logical names to replica locations

An Example of P-RLS Network



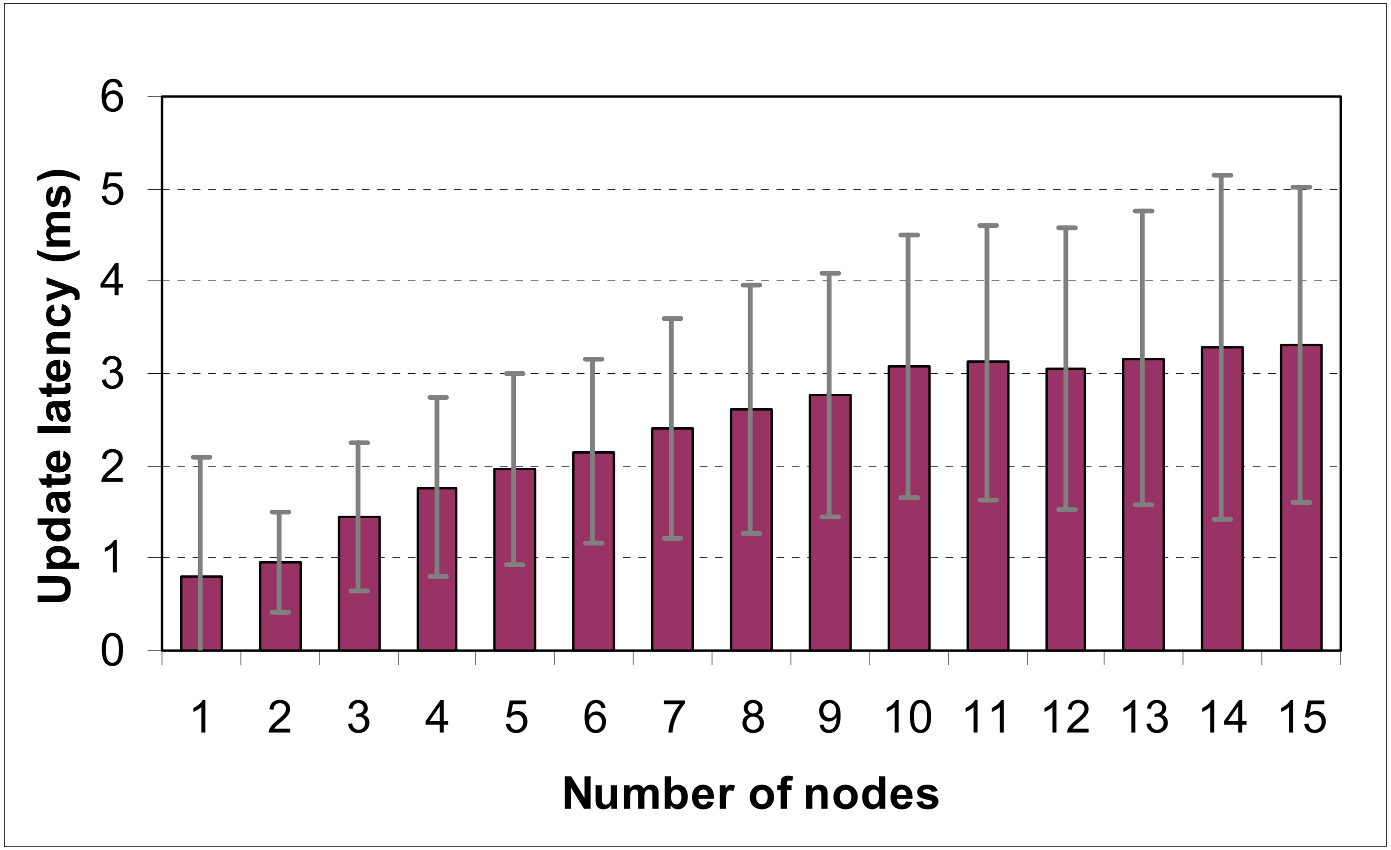
P-RLS Implementation

- Implemented a prototype of P-RLS
- Extends RLS implementation in Globus Toolkit 3.0
- Each P-RLS node consists of an unchanged LRC server and a peer-to-peer P-RLI server
- The P-RLI server implements the Chord protocol operations, including join, update, query, successor, probing & stabilization
- LRC, RLI & Chord protocols implemented on top of RLS RPC layer



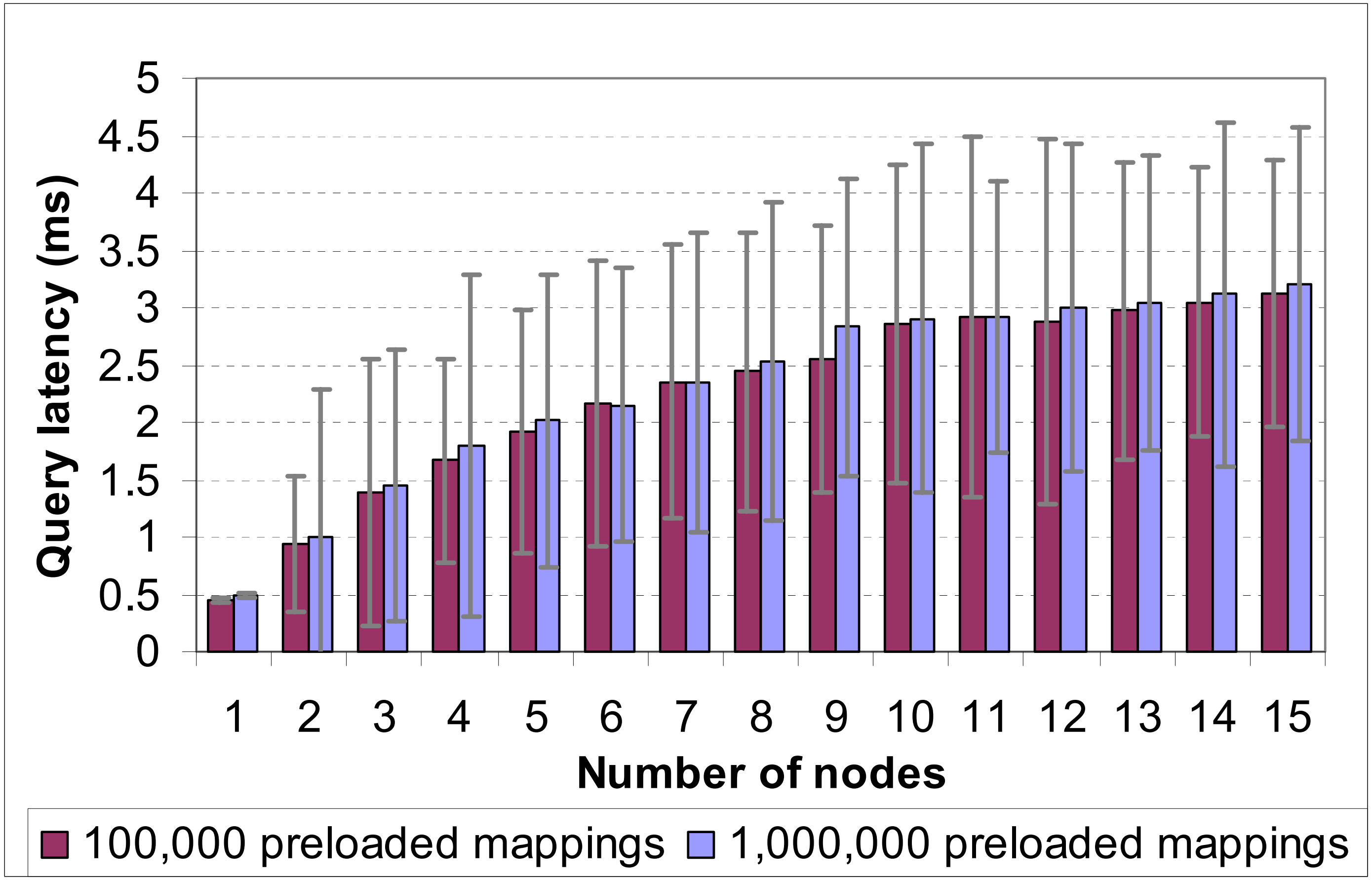
P-RLS Performance Measurements

- P-RLS network runs on a 16-node cluster
- 1000 updates (*add operations*) on each node, updates overwrite existing mappings, and maximum 1000 mappings in the network
- Update latencies increase on log scale with number of nodes



P-RLS Measurements (Cont.)

- Query latencies with 100,000 and 1 million mappings
- Total number of mappings has little effect on query times
  - Uses hash table to index mappings on each P-RLI node
- Query times increase on log scale with number of nodes



Additional P-RLS Topics

Replication schemes

- Replicate mappings in P-RLS network for better reliability
- Successor scheme: distributes mappings more evenly
- Predecessor scheme: reduces hotspots for popular mappings

Demonstrated:

- RPC calls performed for fixed number of updates
- Number of pointers to neighbors maintained by P-RLI node
    - Both increase on a log scale with size of P-RLS network
- Stabilization message traffic

Currently, P-RLS is a research project

- We will be investigating the possibility of incorporating peer-to-peer techniques into production RLS

Ongoing and Future Work

- Ongoing RLS scalability testing
- Incorporating RLS into production tools, such as POOL from the physics community
- Working on a publishing tool that uses RLS that is loosely based on the LDR system from the LIGO project
  - Each site compiles a list of published files to be copied locally
  - Invokes transfers using RFT, registers new files in RLS
  - Will be included in GT4.0 release as a technical preview
- Investigating peer-to-peer techniques
- OREP Working Group of the Global Grid Forum working to standardize a web services (WS-RF) interface for replica location services

Acknowledgements

- Thanks to these RLS users:
  - Scott Koranda and the LIGO Collaboration
  - Luca Cinquini and the Earth System Grid Project
  - Gaurang Mehta, Ewa Deelman and the Pegasus Project
  - Yujun Wu and the CMS Project
  - Rob Gardner and the Atlas Project
- Research supported in part by the DOE SciDAC Program
  - DE-FC02-01ER25449 (SciDAC-DATA)
  - DE-FC02-01ER25453 (SciDAC-ESG)

- Code and documentation available: www.globus.org/rls