A Global In-memory Data System for MySQL

Daniel Austin, PayPal Technical Staff

Percona Live! MySQL Conference
Santa Clara, April 12th, 2012 v1.3

Intro: Globalizing NDB

Proposed Architecture

What We Learned

Q&A

Mission YesSQL

"Develop a globally distributed DB For user-related data."

- Must Not Fail (99.999%)
- Must Not Lose Data. Period.
- Must Support Transactions
- Must Support (some) SQL
- Must WriteRead 32-bit integer globally in 1000ms
- MinMax Data Volume: 10-100 TB (working)
- Must Scale Linearly with Costs

THE FUNDAMENTAL PROBLEM IN DISTRIBUTED DATA SYSTEMS

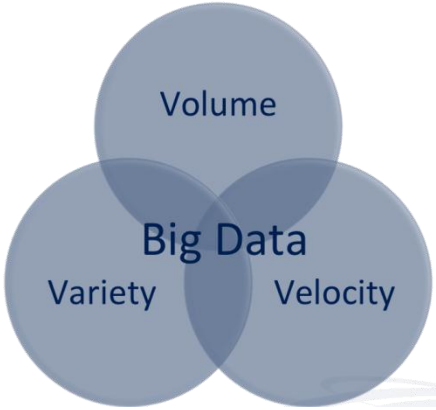
"How Do We Manage Reliable Distribution of Data Across Geographical Distances?"

One Approach: The NoSQL Solution

- NoSQL Systems provide a solution that relaxes many of the common constraints of typical RDBMS systems
  - Slow - RDBMS has not scaled with CPUs
  - Often require complex data management (SOX, SOR)
  - Costly to build and maintain, slow to change and adapt
  - Intolerant of CAP models (more on this later)
- Non-relational models, usually key-value
- May be batched or streaming
- Not necessarily distributed geographically

Big Data Myth #1: Big Data = NoSQL

- 'Big Data' Refers to a Common Set of Problems
  - Large Volumes
  - High Rates of Change
    - Of Data
    - Of Data Models
    - Of Data Presentation and Output
  - Often Require 'Fast Data' as well as 'Big'
    - Near-real Time Analytics
    - Mapping Complex Structures



Takeaway: Big Data is the problem, NoSQL is one (proposed) solution

NoSQL Hype Cycle: Where Are We Now?

There are currently more than **120+** NoSQL databases listed at nosql-database.org!

As the pace of new technology solutions has slowed, some clear winners have emerged.

3 Essential Kinds of NoSQL Systems

1. Columnar K-V Systems
   – Hadoop, Hbase, Cassandra, PNUTs
2. Document-Based
   – MongoDB, TerraCotta
3. Graph-Based
   – FlockDB, Voldemort

Takeaway: These were originally designed as solutions to specific problems because no commercial solution would work.

Intro: Globalizing Big Data

Proposed Architecture

What We Learned

Q&A

OUR APPROACH: THE YESSQL SOLUTION

Use MySQL/NDB

Use AWS

Key Tradeoffs:

- Cost vs. Performance
- HA vs. CAP
- Complexity vs. RDBMS features

WHY NDB?

Pro

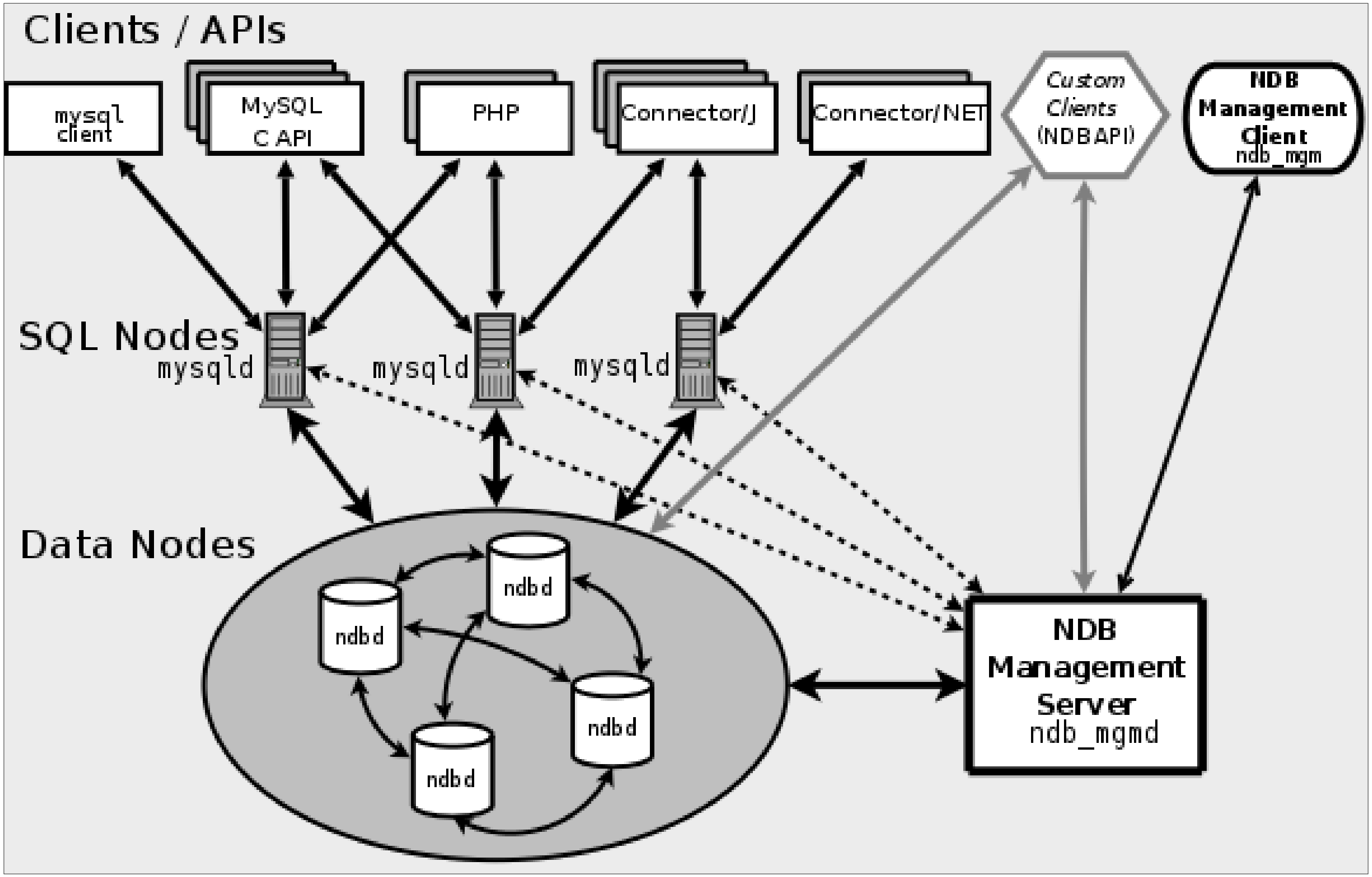
- True HA by design
  - Fast recovery
- Supports (some) X-actions
- Relational Model
- In-memory architecture = high performance
- Disk storage for non-indexed data

  (since 5.1)
- APIs, APIs, APIs

Con

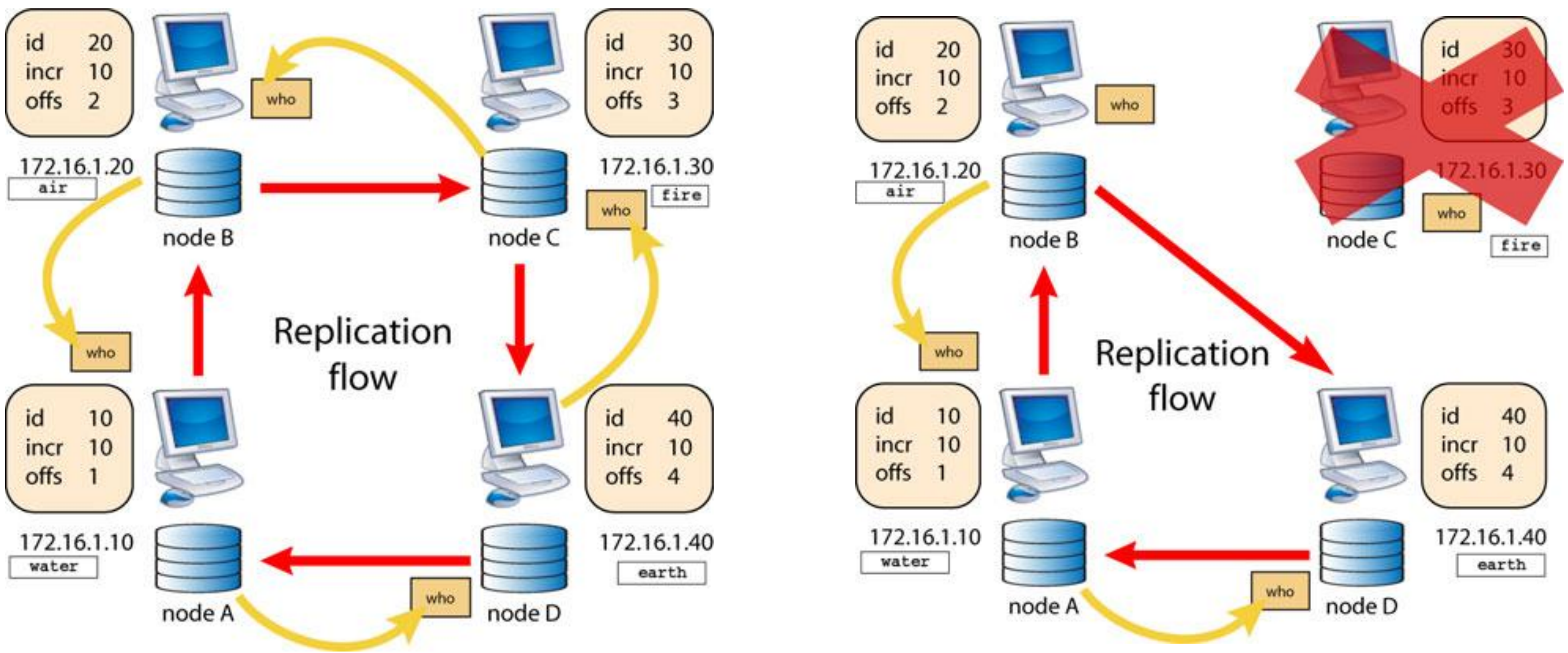
- Some semantic limitations on fields
- Size constraints (2 TB?)
  - Hardware limits also
- Higher cost/byte
- Requires reasonable data partitioning
- Higher complexity

How NDB Works in One Slide



Graphics courtesy dev.mysql.com

CIRCULAR REPLICATION/FAILOVER



Graphics courtesy O'Reilly OnLamp.com

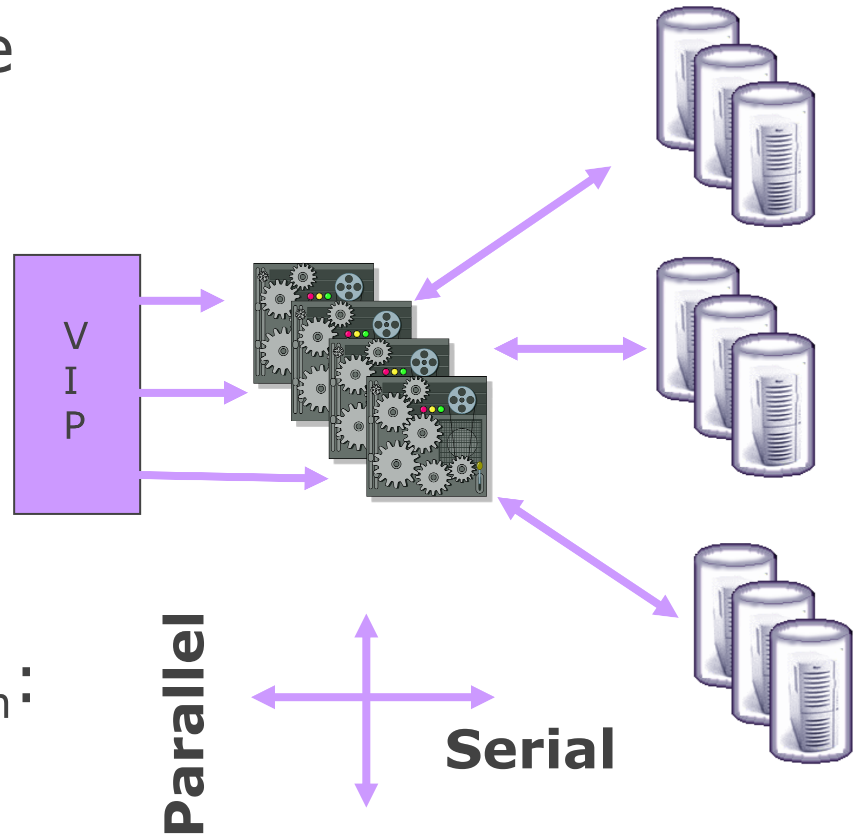
SYSTEM AVAILABILITY DEFINED

- Availability of the entire system:

$$A_{sys} = 1 - \prod_{i=1}^{n}\left(1 - \prod_{j=1}^{m} r_i\right)_j$$

- Number of Parallel Components Needed to Achieve Availability A_{min}:

$$N_{min} = [\ln(1-A_{min})/\ln(1-r)]$$



Big Data Myth #2: The CAP Theorem Doesn't Say What You Think It Does

- Consistency, Availability, (Network) Partition
  - Pick any two? Not really.
- The Real Story: These are not Independent Variables
- AP =CP (Um, what? But…A != C)
- Variations:
  - PACELC (adds latency tolerance)
  - VPAC (adds variability)

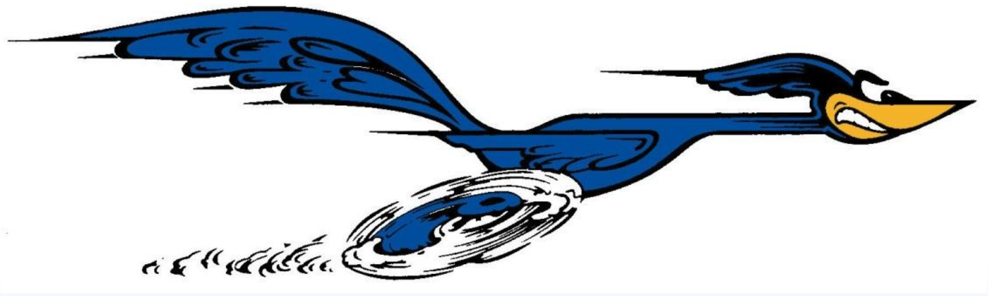
Takeaway: the real story here is about the tradeoffs made by designers of different systems, and the main tradeoff is between consistency and availability, usually in favor of the latter.

What about "High Performance"?

- Maximum lightspeed distance on Earth's Surface: ~**67** ms
- Target: data available worldwide in < 1000 ms

Sound Easy?

Think Again!



AWS Meets NDB

- Why AWS?
  - Cheap and easy infrastructure-in-a-box

    (Or so we thought! Ha!)
- Services Used:
  - EC2 (Centos 5.3, small instances for mgm & query nodes, XL for data
  - Elastic IPs/ELB
  - EBS Volumes
  - S3
  - Cloudwatch

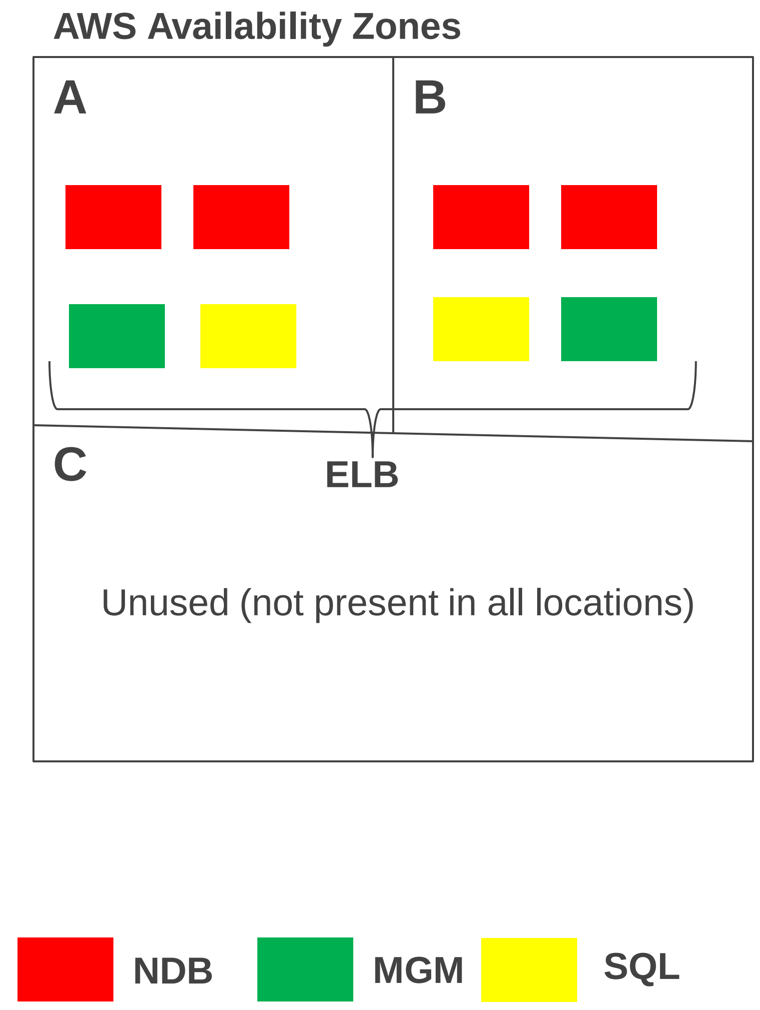
ARCHITECTURAL TILES

Tiling Rules

- Never separate NDB & SQL
- Ndb:2-SQL:1-MGM:1
- Scale by adding more tiles
- Failover 1st to nearest AZ
- Then to nearest DC
- At least 1 replica/AZ
- Don't share nodes
- Mgmt nodes are redundant

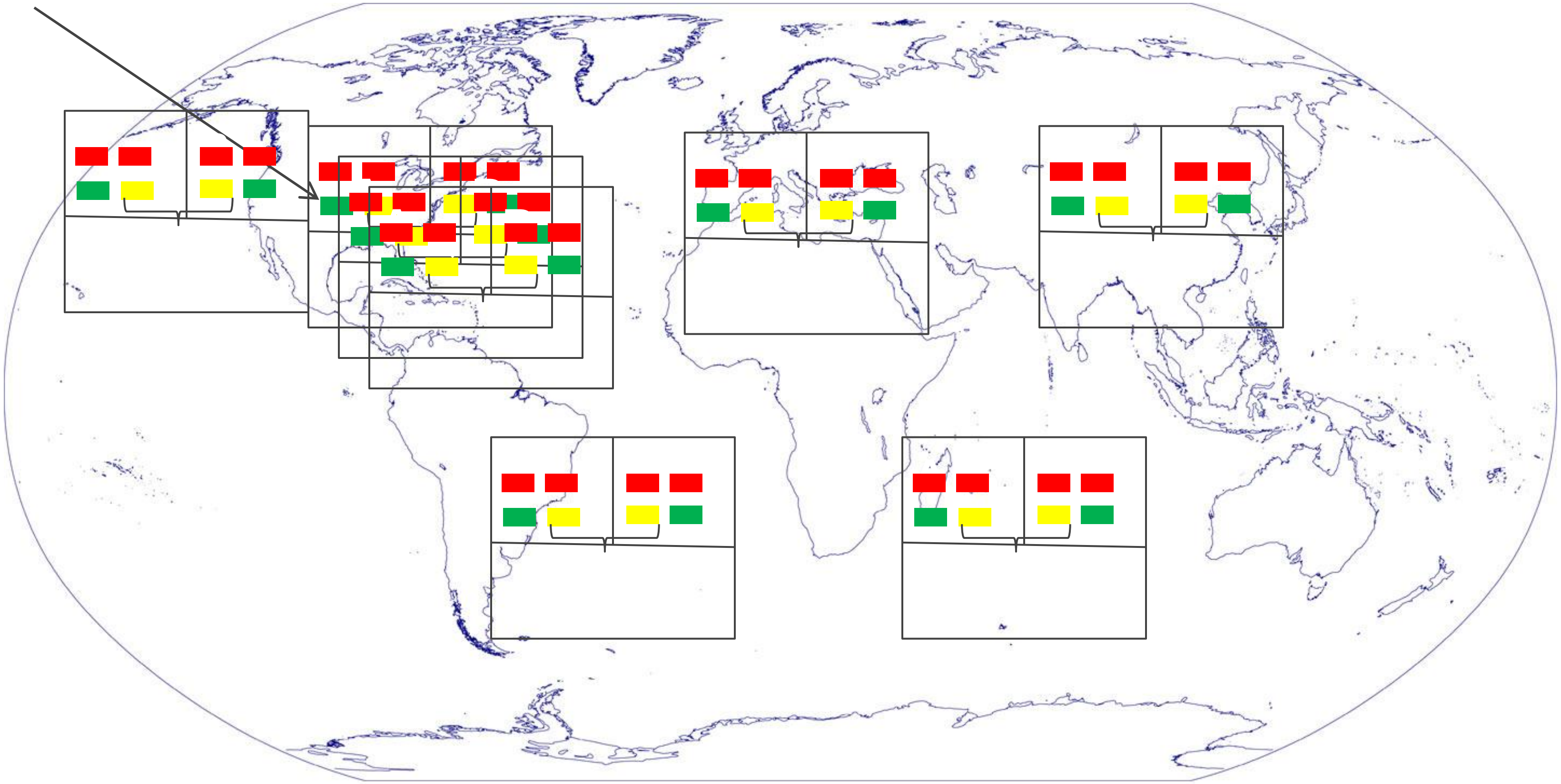
Limitations

- AWS is network-bound @ 250 MBPS – ouch!
- Need specific ACL across AZ boundaries
- AZs not uniform!
- No GSLB
- Dynamic IPs
- ELB sticky sessions !reliable



Architecture Stack

Scale by Tiling



7 AWS Data Centers:
US-E, US-W, US-W2, TK, EU, AS, SA-B

Other Technologies Considered

- Paxos
  - Elegant-but-complex consensus-based messaging protocol
  - Used in Google Megastore, Bing metadata
- Java Query Caching
  - Queries as serialized objects
  - Not yet working
- Multiple Ring Architectures
  - Even more complicated = no way

Intro: Globalizing Big Data

Proposed Architecture

What We Learned

Q&A

SYSTEM READ/WRITE PERFORMANCE (!)

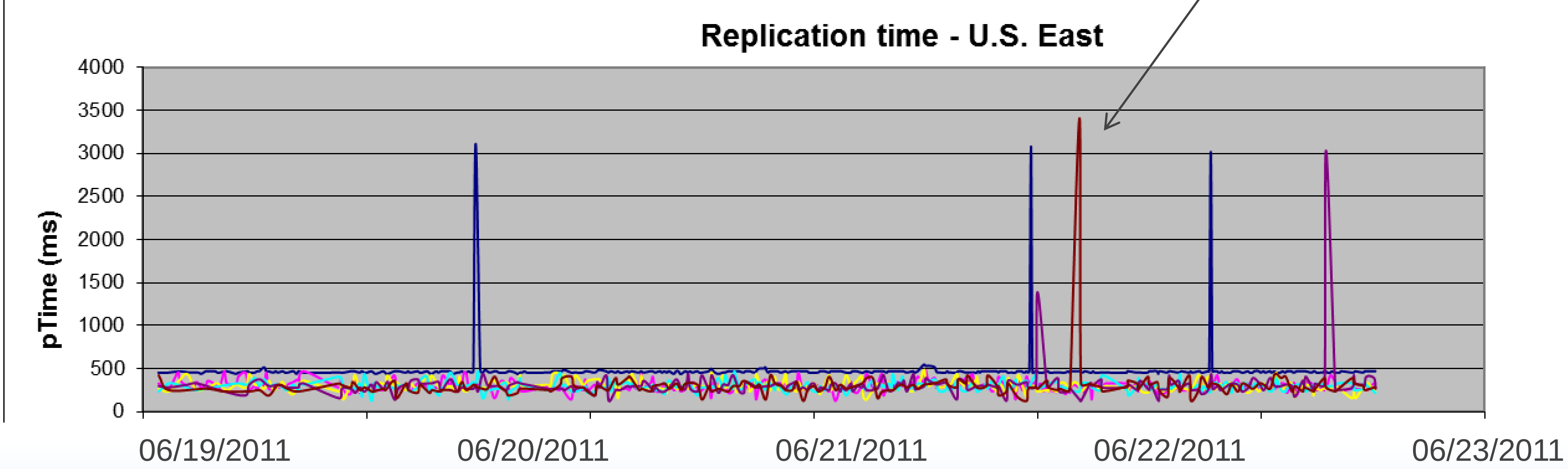
What we tested:

- 32 & 256 byte char fields
- Reads, writes, query speed vs. volume
- Data replication speeds

Results:

- Global replication < 350 ms
- 256 byte writeread < 1000ms worldwide

In-region replication tests



The Commit Ordering Problem

- Why does commit ordering matter?
- Write operators are non-commutative

  [W(d,t1),W(d,t2)] != 0 unless t1=t2

  - Can lead to inconsistency
  - Can lead to timestamp corruption
  - Forcing sequential writes defeats Amdahl's rule
- Can show up in GSLB scenarios

Dark Side of AWS

- Deploying NDB at scale on AWS is <u>hard</u>
  - Dynamic IPs (use hostfile)
  - DNS issues
  - Security groups (ec2-authorize)
  - Inconsistent EC2 deployments
    - Are availability zones independent network segments?
  - No GSLB (!) (rent or buy)

<u>Be prepared to struggle a bit!</u>

Future Directions for YesSQL

- Alternate solution using Pacemaker, Heartbeat
  - From Yves Trudeau @ Percona
  - Uses InnoDB, not NDB
- Implement Memcached plugin
  - To test NoSQL functionality, APIs
- Add simple connection-based persistence to preserve connections during failover
- Can we increase the number of data nodes?
- As the number of datacenters goes up, sync times go down. What’s the relation?
- So far only limited testing of hard configurations
  - Production systems will likely need more RAM for MGMs

Hard Lessons, Shared

- Be Careful…
  - With "Eventual Consistency"-related concepts
  - ACID, CAP are not really as well-defined as we'd like considering how often we invoke them
- NDB is a good solution
  - Real HA, real performance, real SQL
  - Notable limitations around fields, datatypes
  - Successfully competes with NoSQL systems for most use cases – better in many cases
- AWS makes the hard things possible
  - But not so much the easy things easier

Takeaway: You can achieve high performance and availability without giving up relational models and read consistency!

“In the long run, we are all ~~dead~~ eventually consistent.”

Maynard Keynes on NoSQL Databases

Twitter: @daniel_b_austin
Email: daaustin@paypal.com

With apologies and thanks to the real DB experts, Andrew Goodman, Yves Trudeau, Clement Frazer, Daniel Abadi, and everyone else!