Apache Cassandra:

Distributed DB for Massive Scale

Austin JUG

Stu Hood (@stuhood) – Technical Lead, Rackspace

January 26th 2010

rackspace HOSTING

My, what a large/volatile dataset you have...

- Large
  - Larger than 1 node can handle
- Volatile
  - More than 25% (ish) writes
- Expensive
  - More than you can afford with a commercial solution

My, what a large/volatile dataset you have...

- For example
  - Event/log data
  - Output of batch processing or log analytics jobs
    - In Hadoop, particularly
  - Social network relationships/updates

What about sharding?

- shard n. - *A horizontal partition in a database*
  - Example: Sharding by userid
- Limitations (for the partitioned dimension)
  - No more joins
  - No more indexes

What about sharding?

- Options
  - Provided by ORM?
    - Fixed partitions: manual rebalancing
  - Developing from scratch?
    - Adding/removing nodes
    - Handling failover
    - As a library? As a middle tier?

What about sharding?

Pain in the ass.

Case Study: Digg

1. Vertical partitioning and master/slave trees
2. Developed sharding solution
   - IDDB
   - Awkward replication, fragile scaling
3. Began populating Cassandra in parallel
   - Initial dataset for 'green badges'
     - 3 TB
     - 76 billion kv pairs
   - Most applications being ported to Cassandra

Relationship trouble: The NoSQL Movement

Lessons from the NoSQL Movement

- Not Only SQL: Specialization!
  - Relational is not always the answer
  - Need highly specialized systems to distribute:
    - Transactions
    - Joins
  - Normalization strives to remove duplication
    - But duplication is an interesting alternative to joins

Lessons from the NoSQL Movement

- Understanding your data, finding the most efficient model
  - Key-value / Document
    - Hint: already denormalized? fits in memcache?
    - Cassandra, Riak, Voldemort, CouchDB, MongoDB
  - Graph/RDF
    - Hint: frequent schema changes or deeply nested joins?
    - Neo4j, Sesame
  - Column store
    - Hint: all writes go to the end of a few tables?
    - Cassandra, SDB, Hbase, Hypertable

Lessons from the NoSQL Movement

- Replication is not one-size-fits-all
  - Facebook multi datacenter caching
    - Writes occurring in one DC need to expire caches elsewhere
    - Label events as requiring cache expiration
  - Google semi-synchronous replication patches
    - Selective synchronous replication
  - Fragile write master
    - DRBD for high availability

Lessons from the NoSQL Movement

- Partition tolerance vs. consistency
  - Brewer's **CAP** Theorem
    - **C**onsistency, **A**vailability, **P**artition tolerance
    - Choose 2:
      - CA – Corruption possible if live nodes can't communicate (network partition)
      - CP – Completely inaccessible if any nodes are dead
      - AP – Always available, but may not always read most recent
  - Tunables!
    - Cassandra chooses AP, but makes consistency configurable

Cassandra's Elders

Standing on the shoulders of: Amazon Dynamo

- No node in the cluster is special
  - No special roles
  - No scaling bottlenecks
  - No single point of failure
- Techniques
  - Gossip
  - Eventual consistency

Standing on the shoulders of: Google Bigtable

- Column family data model
- Range queries for rows:
  - Scan rows in order
- Memtable/SSTable structure
  - Always writes sequentially to disk
  - Bloom filters to minimize random reads
  - Trounces B-Trees for big data
    - Linear insert performance
    - Log growth for reads

Enter Cassandra

- Hybrid of ancestors
  - Adopts listed features
- And adds:
  - A sweet logo!
  - Pluggable partitioning
  - Multi datacenter support
    - Pluggable locality awareness
  - Datamodel improvements

rackspace HOSTING

Enter Cassandra

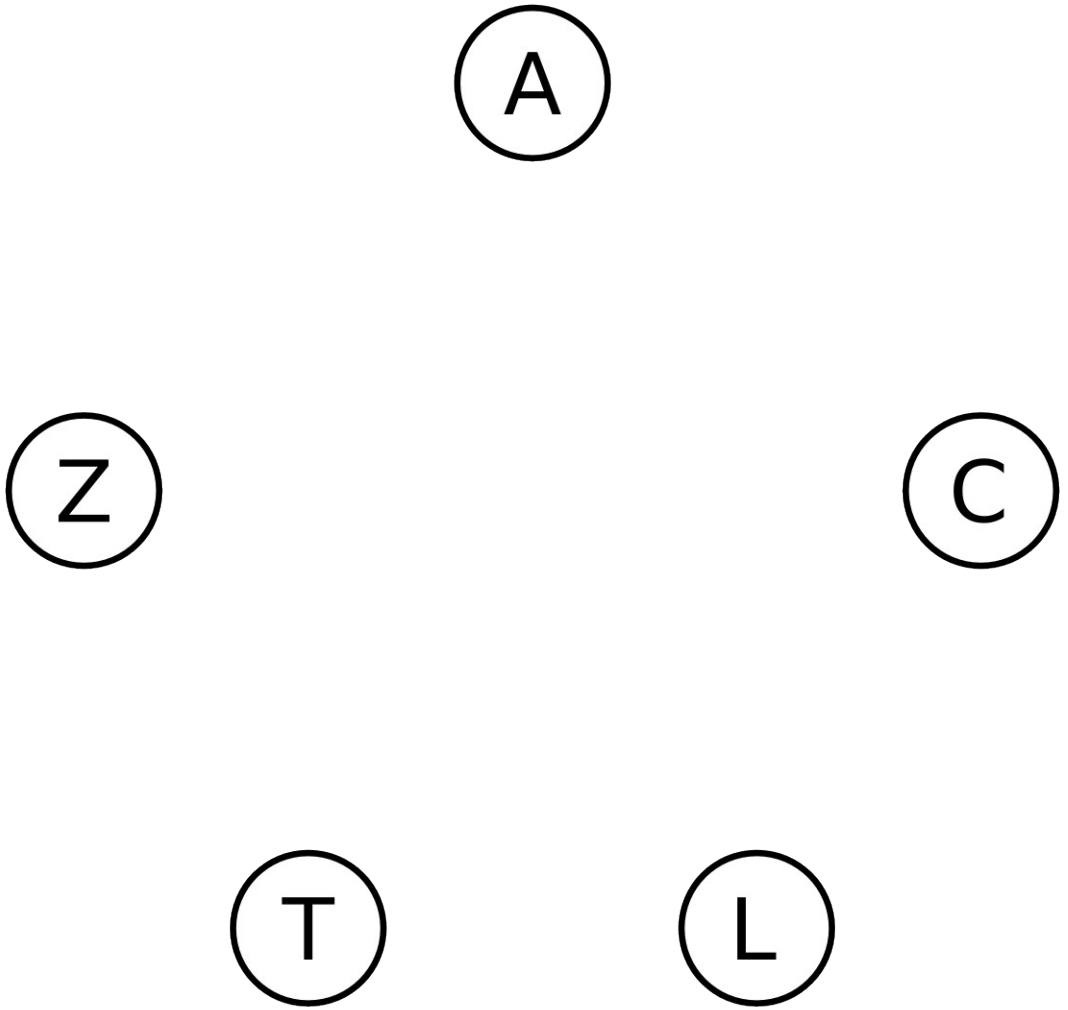
- Project status
  - Open sourced by Facebook (no longer active)
  - Apache License
  - Incubating with Apache: graduation in progress
  - Major releases: 0.3, 0.4, 0.5 (this past weekend!)

Enter Cassandra

- The code base
  - Java, Apache Ant, Git/SVN
  - 5+ committers from 3+ companies
- Known deployments at:
  - Rackspace, Digg, Twitter, Mahalo, SimpleGeo, Cloudkick

The Datamodel

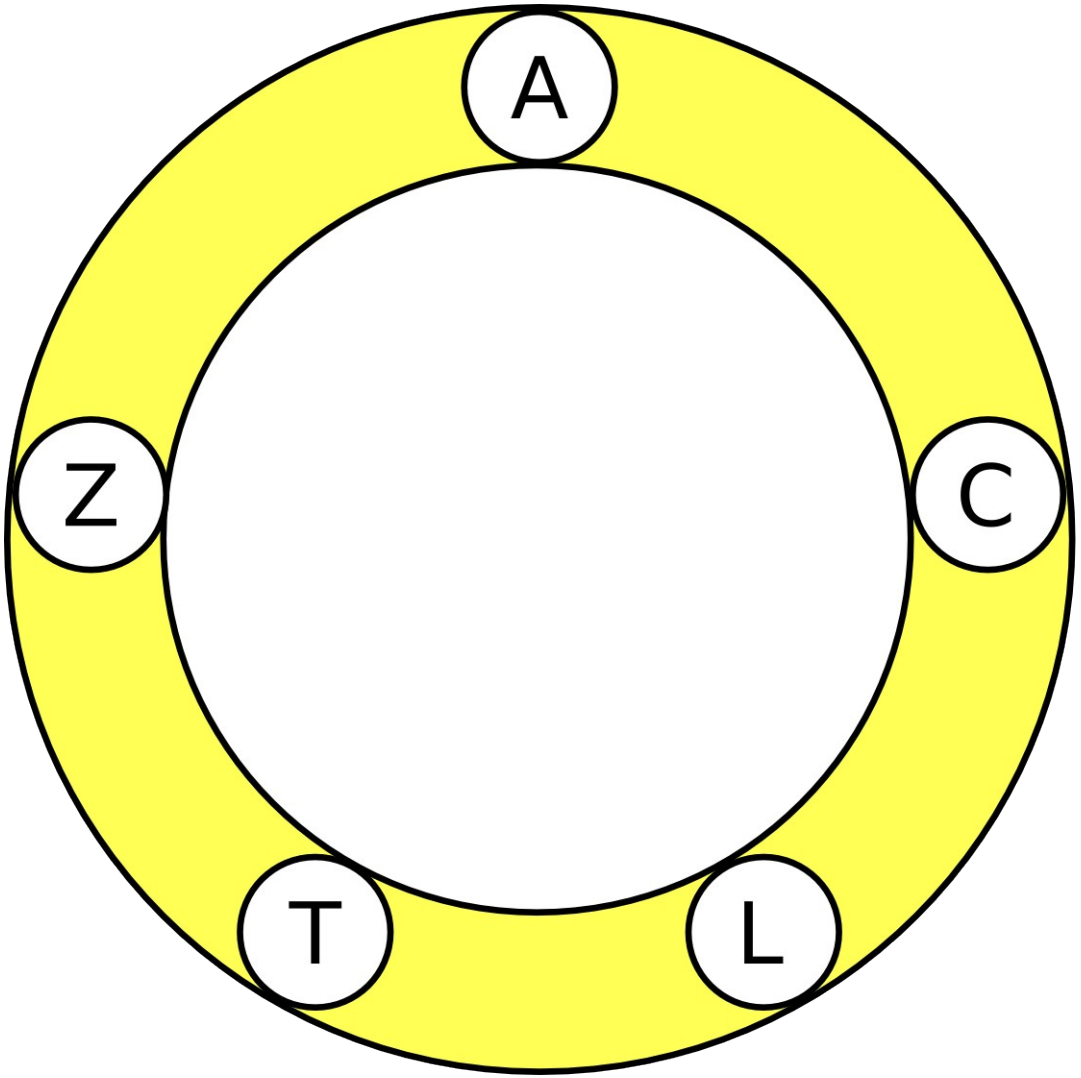
Cluster



rackspace HOSTING

The Datamodel

Cluster > **Keyspace**



Partitioners:

OrderPreservingPartitioner

RandomPartitioner

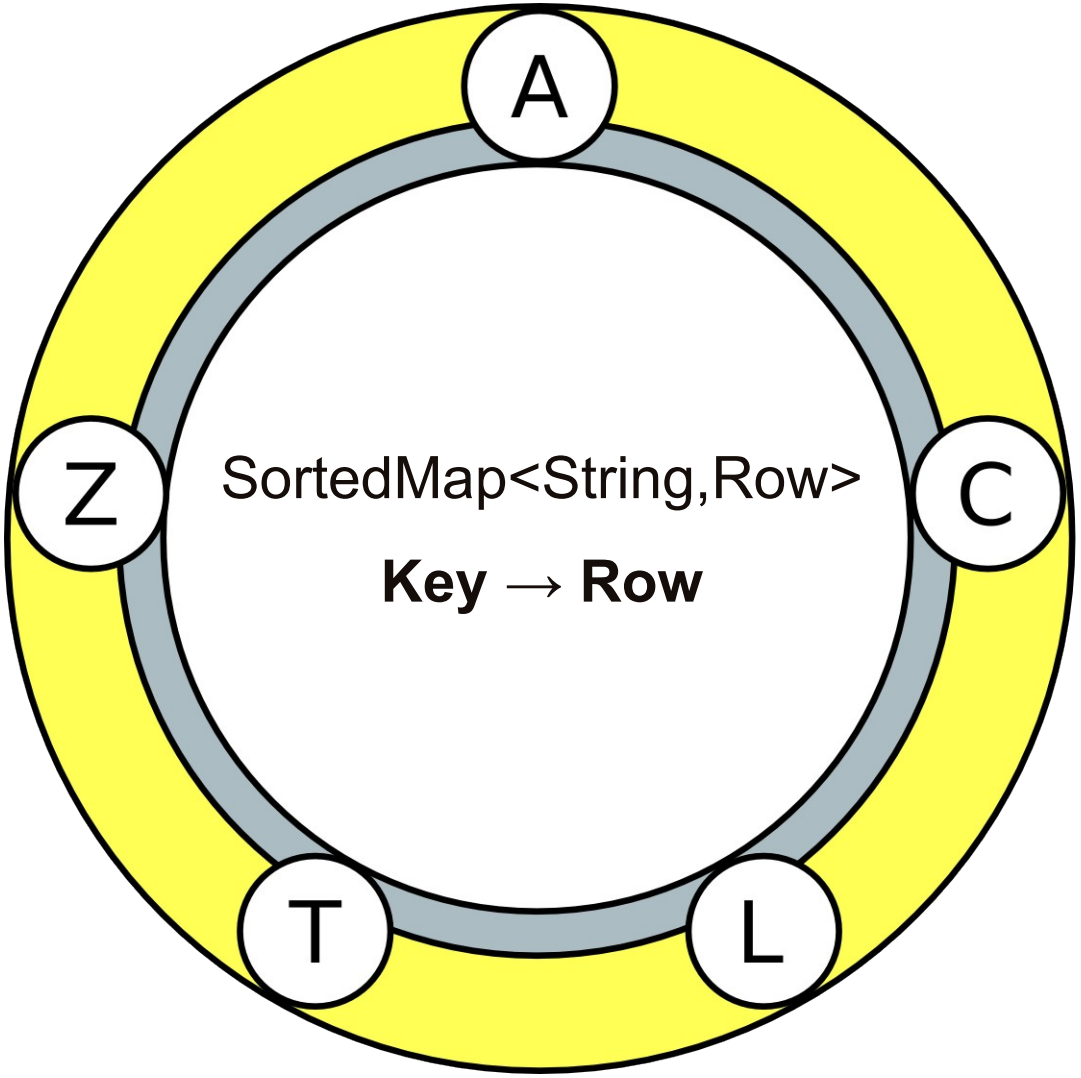
Like an RDBMS schema:

Keyspace per application

rackspace HOSTING

The Datamodel

Cluster > Keyspace > **Column Family**



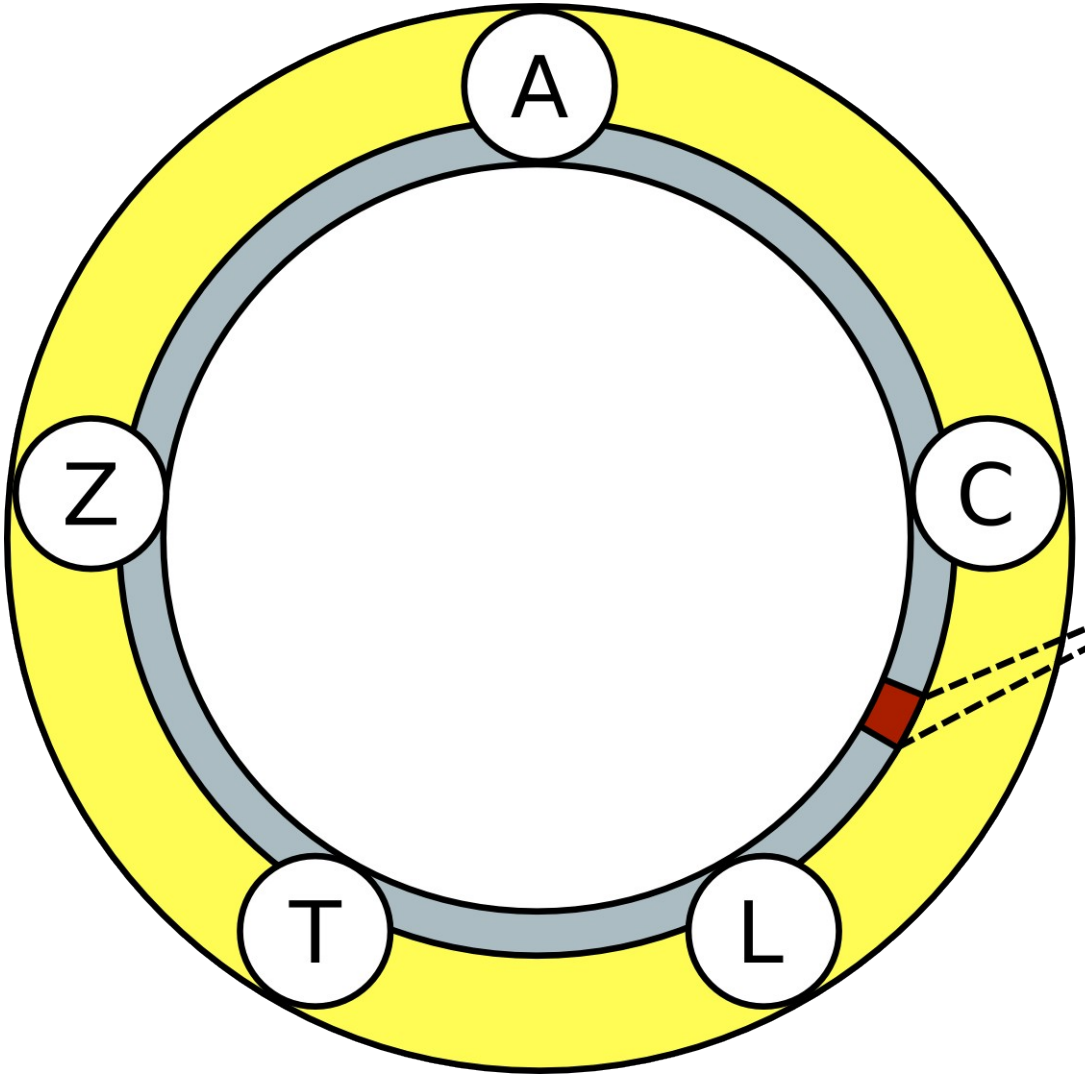
Like an RDBMS table:

Separates types in an app

rackspace HOSTING

The Datamodel

Cluster > Keyspace > Column Family > **Row**



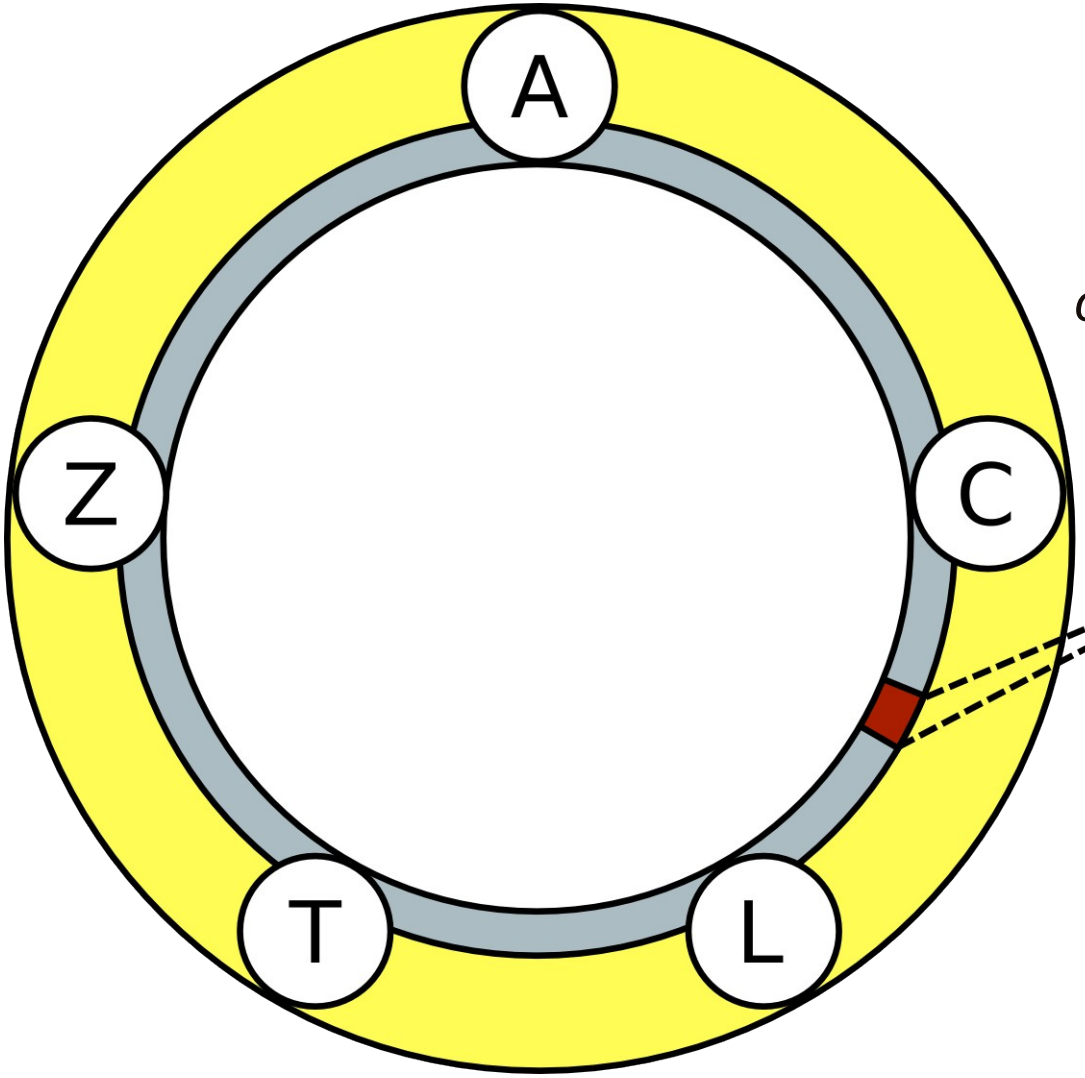
SortedMap<Name,Value>

...

rackspace HOSTING

The Datamodel

Cluster > Keyspace > Column Family > Row > **“Column”**



<u>Not</u> like an RDBMS column:

Attribute of the row: each row can contain millions of different columns

…

Name → Value

byte[] → byte[]

+version timestamp

rackspace HOSTING

StatusApp: another Twitter clone.

StatusApp Example

<ColumnFamily Name=”Users”>

- Unique id as key: name->value pairs contain user attributes.

{key: “strangeloop_stl”, row: {“fullname”: “Dr. Strange”, “joindate”: “20091023” … }}

StatusApp Example

<ColumnFamily Name=”Statuses”>

- Unique id as key: name->value pairs contain status attributes

{key: “status19”, row: {“userid”: “user19”, “content”: “Excited to be at #strangeloop!” … }}

StatusApp Example

<ColumnFamily Name=”Timelines” ColumnType=”Super”>

- User id as key: row contains lists of timelines that the user cares about. Each list contains name:value pairs representing ordered statuses

{key: “user19”, row: {“personal”: [<timeuuid>: “status19”, …], “following”: [<timeuuid>: “status21”, …] }}

Client API

- Thrift generates client bindings for (almost) any language
  - RPC framework
- Most popular:
  - Python
  - Ruby
  - Java
  - PHP

Client API

1. Get the full name for a user:

- get(keyspace, key, [column_family, column_name], consistency_level)
- get(“statusapp”, “userid19”, [“Users”, “fullname”], ONE)

> “Dr. Strange”

Client API

2. Get the 50 most recent statuses in a user's personal timeline:

- get_slice(keyspace, key, [column_family, column_name], predicate, consistency_level)
- get_slice(“statusapp”, “userid19”, [“Timelines”, “personal”], {start: ””, count: 50}, QUORUM)

> [<timeuuid>: “status19”, <timeuuid>: “status24” …]

Consistency Levels?

- Eventual consistency
  - Synch to Washington, asynch to Hong Kong
- Client API Tunables
  - Synchronously write to **W** replicas
  - Confirm **R** replicas match at read time
  - *of* **N** *total replicas*
- Allows for almost-strong consistency
  - When **W** + **R** > **N**

Caveat consumptor

- No secondary indexes:
  - Querying the content of rows always involves iterating over them
  - Example: Finding users with a certain last name
- Solution: Manually maintain indexes
  - Implications:
    - Indexes must by synced with the indexed data manually
    - New indexes mean new writes, reads

Caveat consumptor

- No joins:
  - No server side method to re-query based on a first query
- Solution: Denormalize some queries, client join the rest

The bright side: Ops

- Need N new nodes?
  - Start more nodes with the same config file
  - *New nodes request load information from the cluster and join with a token that balances the cluster*

The bright side: Ops

- Dead drive?
  - Swap the drive, restart, run 'repair'
  - *Streams missing data from other replicas*

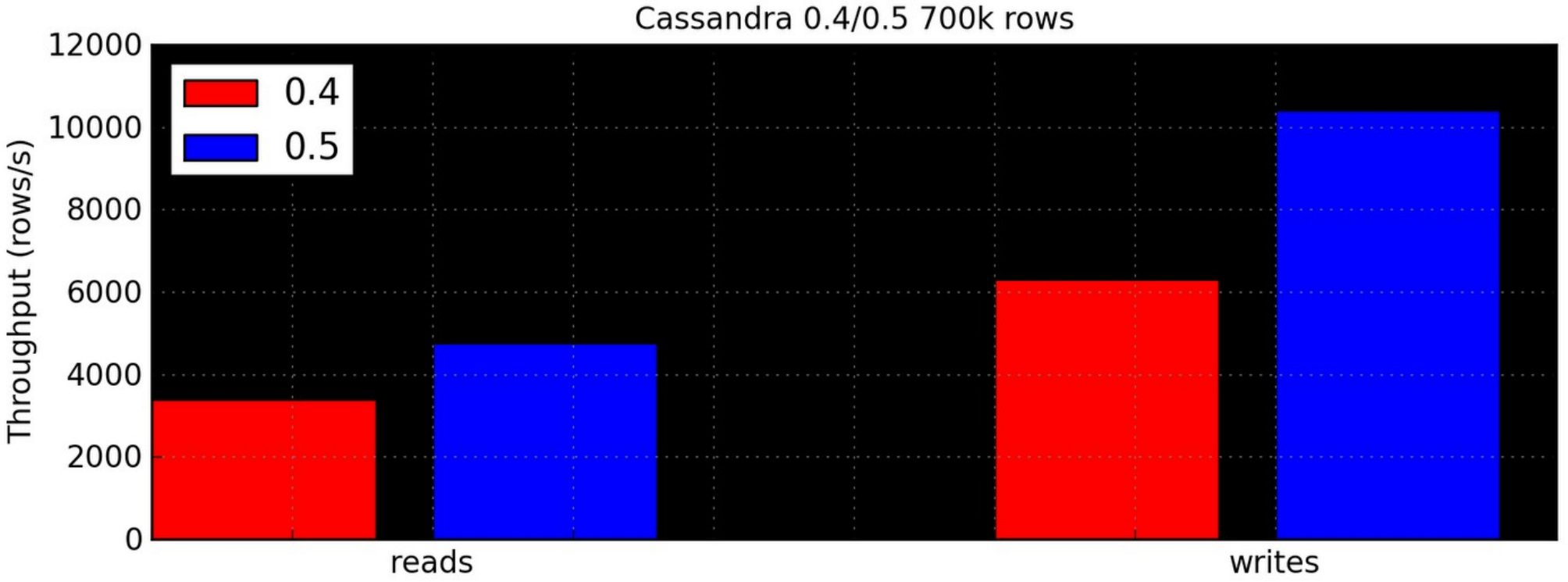
The bright side: Ops

- Dead node?
  - Start a new node with the same IP and token, run 'repair'

The bright side: Ops

- Adding a datacenter?
  - Extend the 'EndpointSnitch' class to describe the location of your nodes
  - Add new nodes as before

The bright side: Performance



rackspace HOSTING

Getting started

- http://incubator.apache.org/cassandra/
- Read "Getting Started"... Roughly:
  - Start one node
  - Test/develop app, editing node config as necessary
    - Don't skip: exposes partitioner, data model strengths/weaknesses
  - Launch cluster by starting more nodes with chosen config

Getting started

- Resources
  - http://incubator.apache.org/cassandra/
  - Wiki
  - Mailing List
  - #cassandra on freenode.net

Thanks!

Austin JUG

UT

Rackspace

rackspace HOSTING

Questions?

References

- Digg Technology Blog
  - http://about.digg.com/blog/looking-future-cassandra
  - http://about.digg.com/blog/introducing-digg's-iddb-infrastructure
- Cassandra Wiki
  - http://wiki.apache.org/cassandra/
- Werner Vogels – Eventual Consistency
  - http://www.allthingsdistributed.com/2008/12/eventually_consistent.html
- Jonathan Ellis's Blog
  - http://spyced.blogspot.com/2010/01/cassandra-05.html