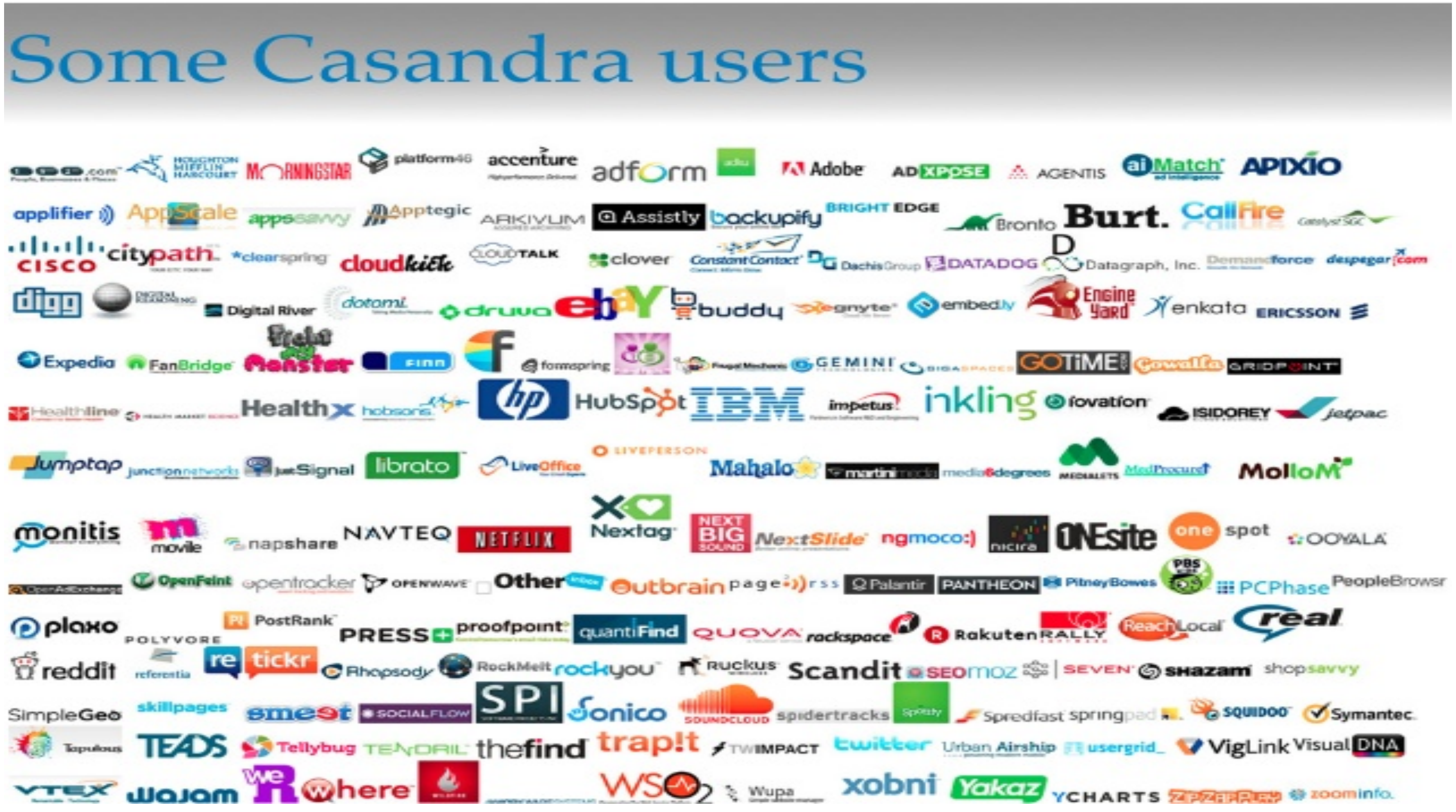
# Cassandra Overview

Nick Carey, Vaibhav Mohan, Shaojie Chen, Ethan Holly

# Some Cassandra Users



source: Cassandra at NoSql Matters 2012 from jbellis

## Industries & Use Cases

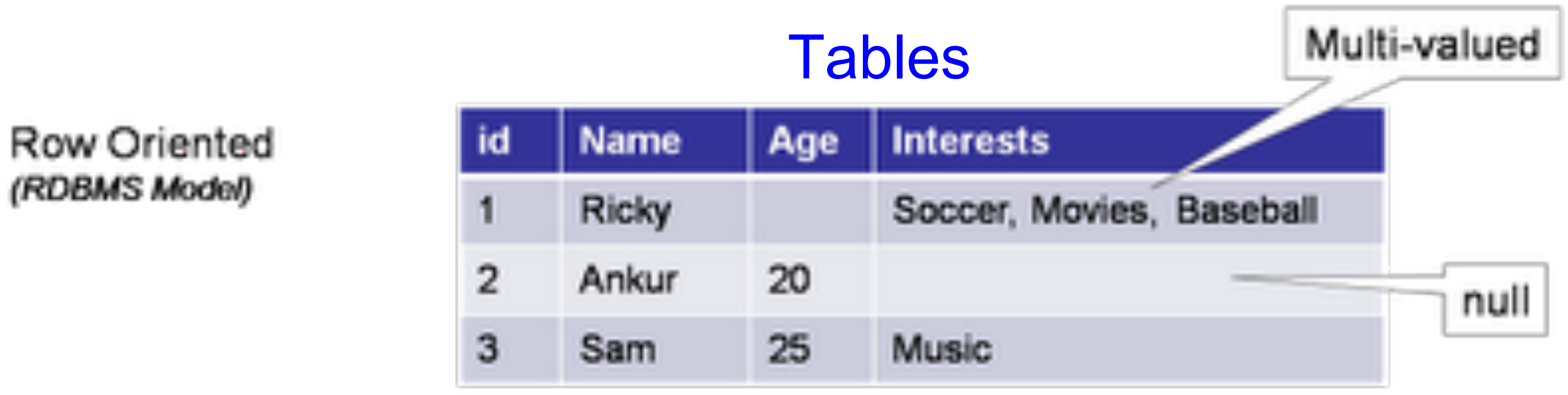
- Financial
- Social Media
- Advertising
- Entertainment
- Energy
- E-tail
- Health care
- Government
- Ad tracking

- Time series data
- Messaging
- Data mining
- User activity streams
- User sessions
- Anything requiring **scalable performant + highly available**

source: **Cassandra at NoSql Matters 2012** from **jbellis**

# Architecture - Implements BigTable Data Model

Tables

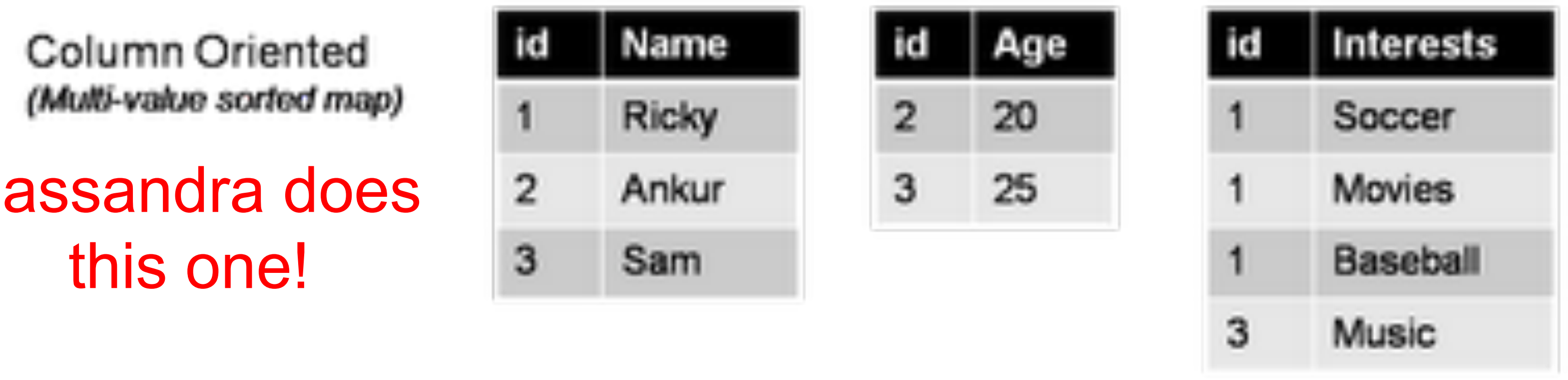


Row Oriented
*(RDBMS Model)*

| id | Name | Age | Interests |
|---|---|---|---|
| 1 | Ricky | | Soccer, Movies, Baseball |
| 2 | Ankur | 20 | |
| 3 | Sam | 25 | Music |

Multi-valued
null

Column-Families



Column Oriented
*(Multi-value sorted map)*

| id | Name |
|---|---|
| 1 | Ricky |
| 2 | Ankur |
| 3 | Sam |

| id | Age |
|---|---|
| 2 | 20 |
| 3 | 25 |

| id | Interests |
|---|---|
| 1 | Soccer |
| 1 | Movies |
| 1 | Baseball |
| 3 | Music |

Cassandra does this one!

Source: http://1.bp.blogspot.com/_j6mB7TMmJJY/TK1npAatLqI/AAAAAAAAAd4/TscPInSeUoo/s400/p1.png

## Architecture - P2P, DHT

- Uses Chord-like distributed hash table for distributing keys among nodes.

- Data is stored redundantly across multiple nodes.

# Architecture - CAP Tradeoffs

- Values AP; Consistency/latency tradeoff tunable
- What does Cassandra do with queries in the case of a network partition/node failure?
  - Behavior can be specified per operation with keywords in CQL (Cassandra Query Language)
  - ONE keyword used to request some instance of the datum, regardless of whether it is consistent with other versions
  - QUORUM keyword used to request value that *most* nodes agree on
  - ALL keyword used to require consistency

Source:http://ria101.wordpress.com/2010/02/24/hbase-vs-cassandra-why-we-moved/

## Architecture - CAP Tradeoffs

- It is possible to ensure consistency without requiring complete ACKs on all operations.
  - Writes with ALL, Reads with ONE

    OR

  - Writes with ONE, Reads with ALL

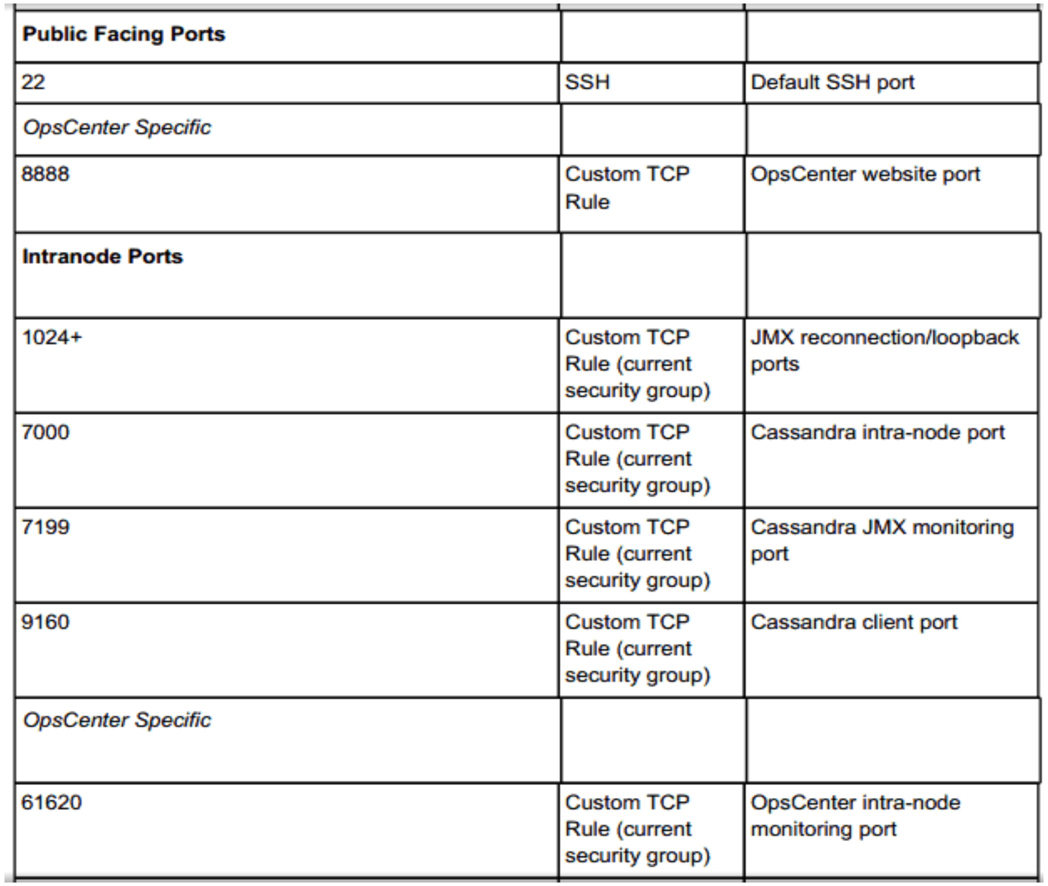
    OR

  - Writes with QUORUM, Reads with QUORUM
- Why is this useful?

Source: http://1.bp.blogspot.com/_j6mB7TMmJJY/TK1npAatLqI/AAAAAAAAAd4/TscPInSeUoo/s400/p1.png

## Setting up Cassandra

- We first created a special security group for Cassandra



| | | |
|---|---|---|
| **Public Facing Ports** | | |
| 22 | SSH | Default SSH port |
| *OpsCenter Specific* | | |
| 8888 | Custom TCP Rule | OpsCenter website port |
| **Intranode Ports** | | |
| 1024+ | Custom TCP Rule (current security group) | JMX reconnection/loopback ports |
| 7000 | Custom TCP Rule (current security group) | Cassandra intra-node port |
| 7199 | Custom TCP Rule (current security group) | Cassandra JMX monitoring port |
| 9160 | Custom TCP Rule (current security group) | Cassandra client port |
| *OpsCenter Specific* | | |
| 61620 | Custom TCP Rule (current security group) | OpsCenter intra-node monitoring port |

## Setting up Cassandra

- We then spun up 1,2,4,8 instances of M1 Large instance
- We used the AMI located on the following page:https://aws.amazon.com/amis/datastax-auto-clustering-ami-2-2
- parameters while launching AMI : --clustername cassandra --totalnodes #Num_nodes --version community
- We used cassandra cli for creating keyspace and column family.
- We used cql and python to interact with cassandra.

## Working With Cassandra

- Adding New Nodes On-the-Fly
    - It is possible to add nodes to Cassandra on-the-fly, but the server must be configured for this on initialization. Additionally, each node must have it’s own pre-calculated ‘initial_token’ parameter.
    - We decided just to start our clusters of various nodes separately.
- Downsides We Encountered:
    - No auto-incrementing keys
    - Must choose either load-balancing or sorted key indexing. Can’t have both.

## Performance Benchmarks - Experiment Set Up

- Dependent variables: time it took cassandra to write/read 100000 key-value pairs
- Independent variables:

  a. read/write to database

  b. number of client threads issuing queries simultaneously. Note that client threads are evenly spaced out over available machines in cassandra cluster

  c. number of machines in cassandra cluster

  d. length/size of value being inserted, length is in bytes

# Performance Benchmarks - Experiment Set Up

- Control variables:
  a. Each benchmark read/wrote 100000 key-value pairs
  b. At the start of write benchmark, there were 0 existing rows in the Cassandra database
  c. At the start of a read benchmark, there were always 100000 existing rows in the Cassandra database
  d. key accesses or key reads is always done in random order. While Cassandra does support ordered indices on keys, they heavily recommended against using them and by default Cassandra uses an unordered key index
  e. 0 Net partitions
  f. row access

## Performance Benchmarks

- See at our graphs!

## Performance Benchmarks - Sequential vs. Random Reads

- We did not expect to see a performance difference here. While Cassandra does support sequential indexing of keys, it would not balance the data across multiple nodes. Thus, we used the default setting of random key indexing.
- Because keys are indexed randomly, even if we queried them in order, we expect the actual database lookups to be random.