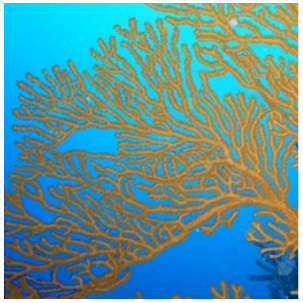
An Evaluation of Distributed Datastores Using the AppScale Cloud Platform



AppScale

Chris Bunch[1], ***Navraj Chohan***[1], Chandra Krintz[1], Jovan Chohan[1], Jonathan Kupferman[1], Puneet Lakhina[1], Yiming Li[1], Yoshihide Nomura[2]

UC Santa Barbara[1],
Fujitsu Labs[2]

UCSB

Overview

- Google App Engine (GAE)
- AppScale
- Datastore Interface to AppScale
- Distributed Datastores
- Evaluation

GAE in the Cloud Stack

Software-as-a-Service (SaaS)

Platform-as-a-Service (PaaS)

Azure

amazon web services™

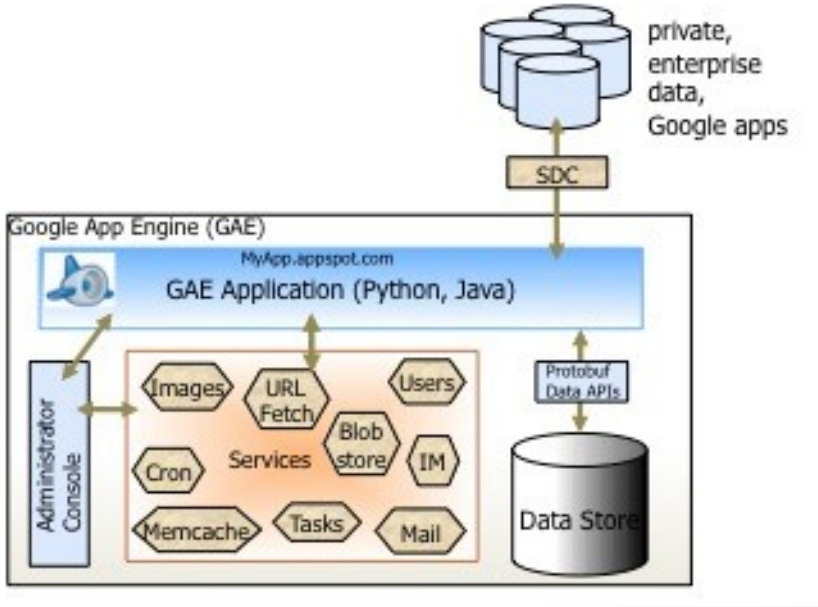
Infrastructure-as-a-Service (IaaS)

Eucalyptus

UCSB

GAE Overview

- Write your application in Python or Java
- Test locally
- Deploy on Google infrastructure
- Automatic Scaling
- Pay-as-you-go

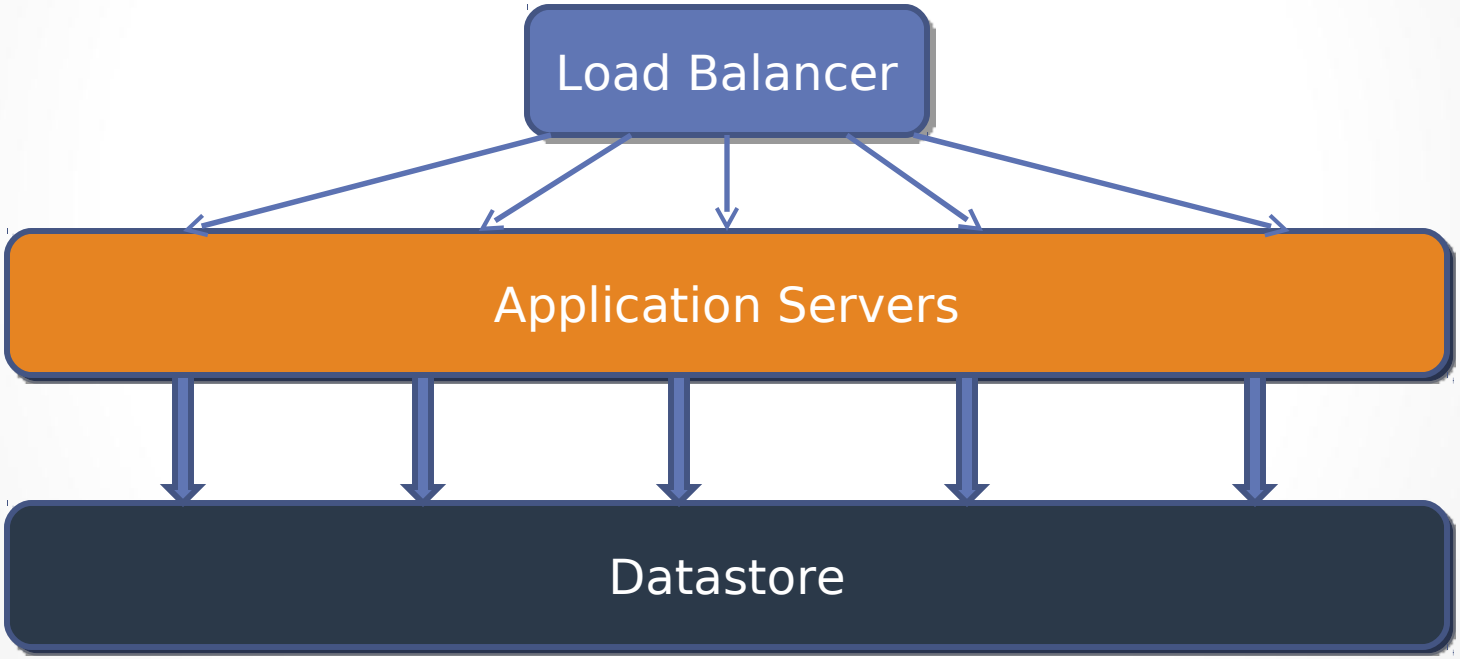


AppScale

- Open source GAE Implementation
- Distributed and scalable API implementations
  - Login
  - Cron
  - Tasks
  - Memcache
  - Scalable Datastores
  - MapReduce
- You choose the infrastructure
  - KVM
  - Xen
  - Eucalyptus
  - Amazon EC2
- An open option to avoid lock-in
- Keep your data/code

UCSB

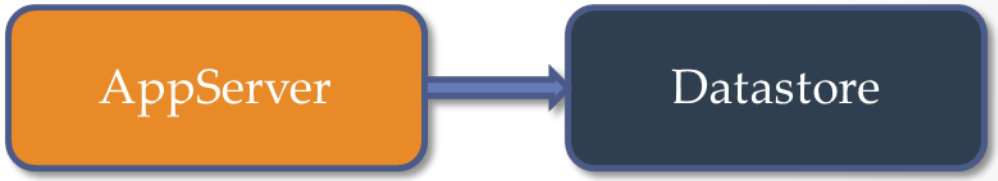
AppScale Design



UCSB

AppServer to Datastore Interface

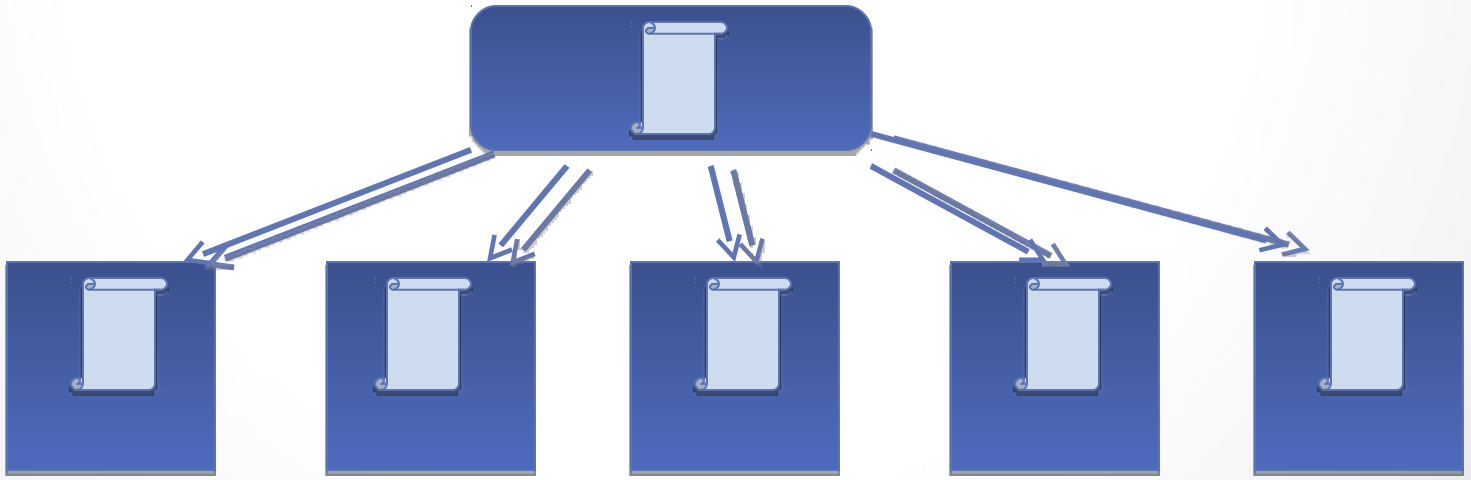
- Key/Value storage
- AppScale DB API:
  - GET
  - PUT
  - GET_TABLE
  - DELETE



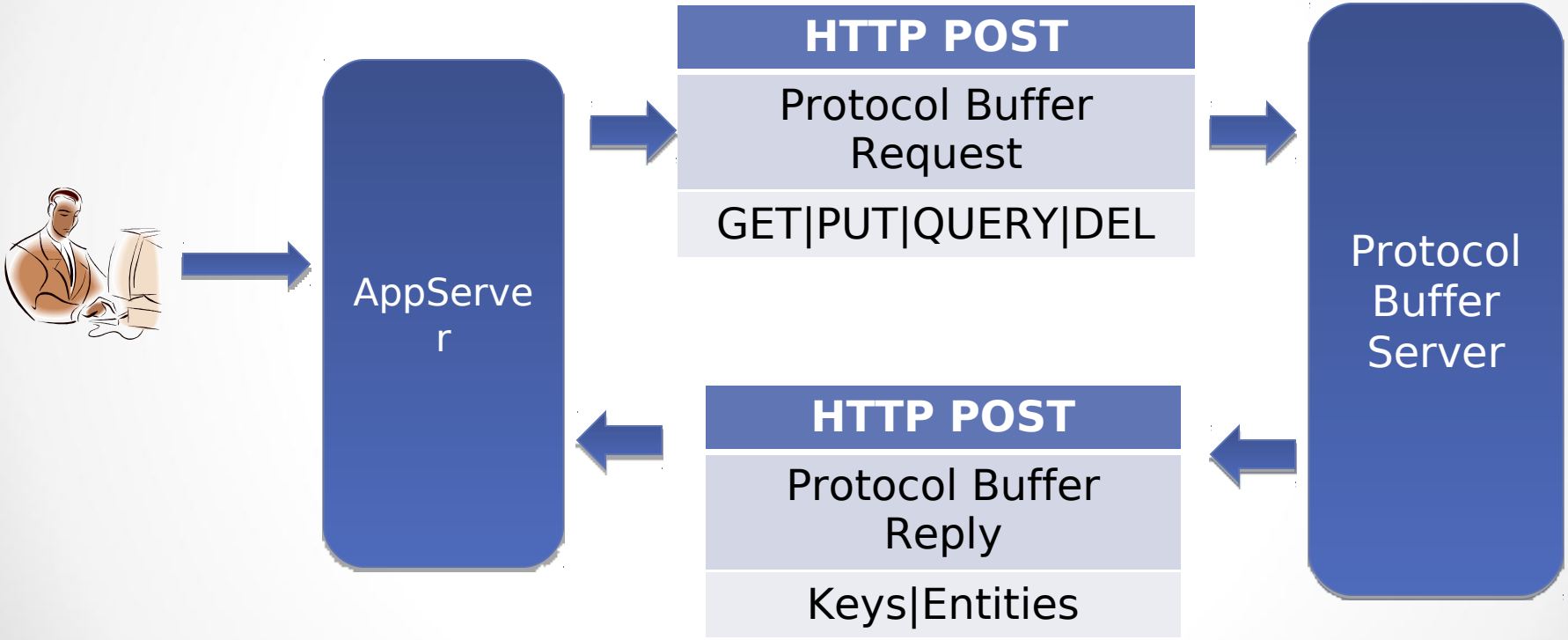
- All AppScale datastores implement this interface
- Automatic Deployment
- RPC calls
  - HTTP Post request
  - Protocol Buffers format
- Store serialized entities

Automatic Deployment

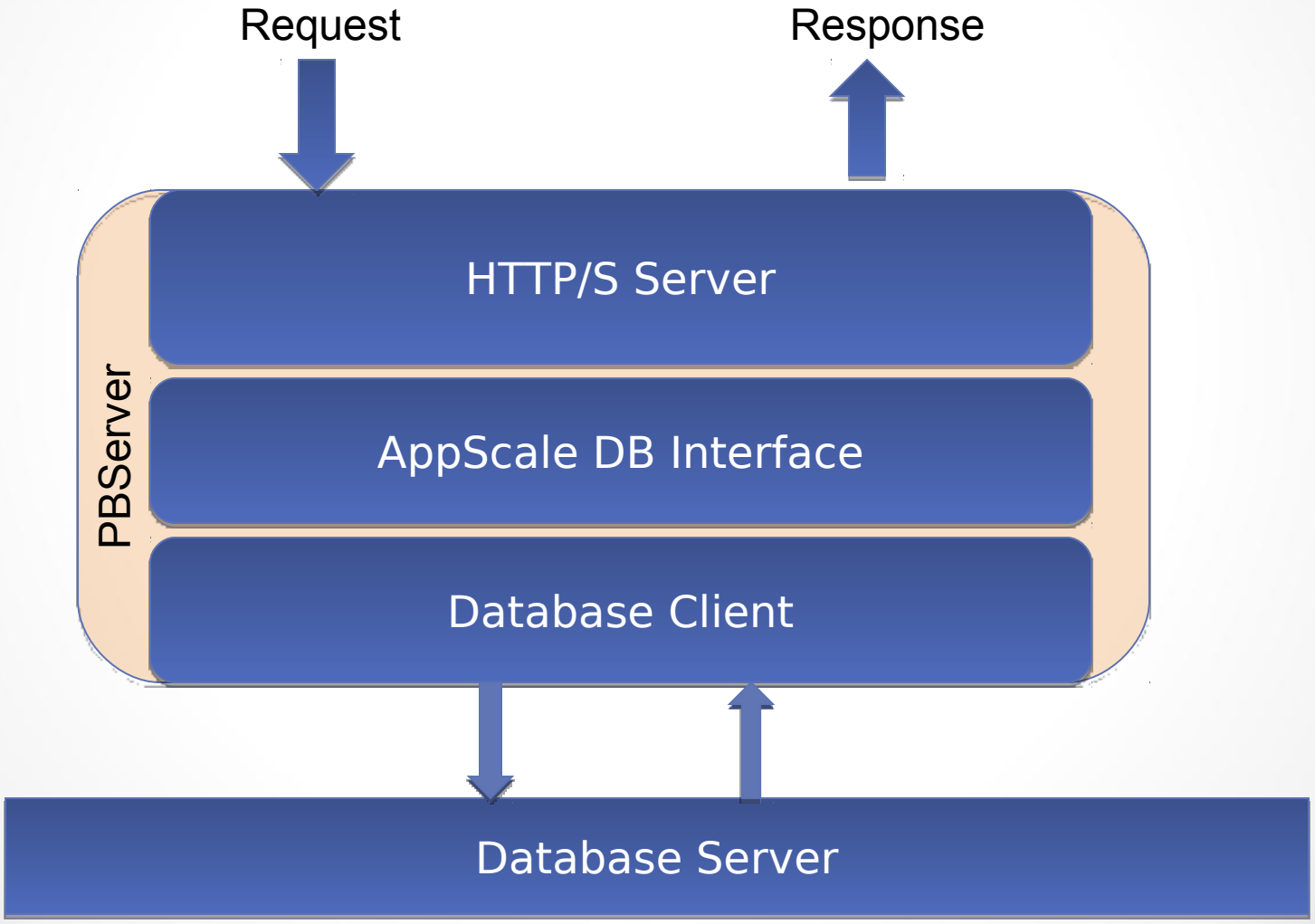
- Setup configuration files
- Spawn processes with correct command line args
- Prime databases for Users and Applications table
- Automatic termination



Datastore Usage



Protocol Buffer Server



Datastores

- Peer to Peer
  - Cassandra
  - Voldemort
- Master/Slave
  - HBase
  - Hypertable
  - MongoDB
  - MemcacheDB
- Relational
  - MySQL Cluster

Datastores

	Replication	Consistency	CAP	Data Model	Range Queries
Cassandra	Yes	Eventual	AP	Column oriented	Yes
Hbase	Yes	Strong	CP	Column oriented	Yes
Hypertable	Yes	Strong	CP	Column oriented	Yes
MemcacheDB	Yes	Strong	CP	Key/Value	Yes
MongoDB	Yes	Strong	CP	Document	Yes
MySQL	Yes	Strong	CP	Relational	Yes
Voldemort	Yes	Ev...	AP	Key/Valu	No

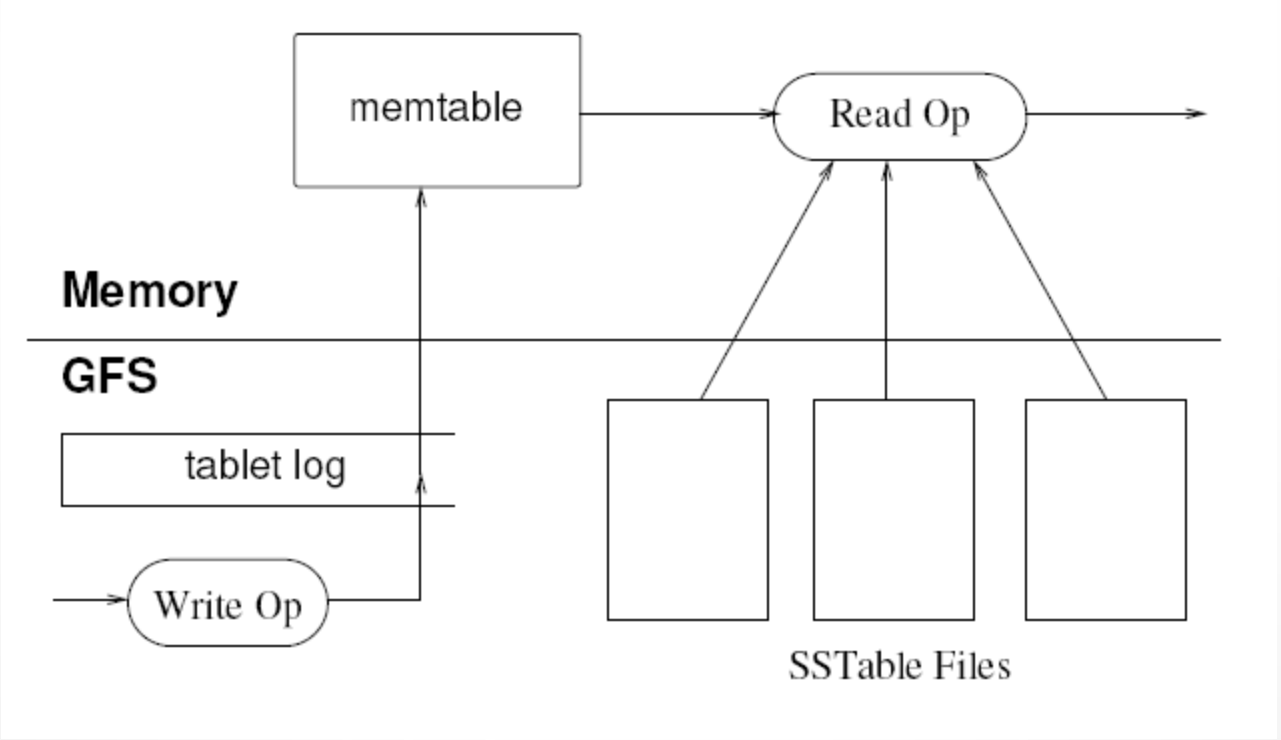
UCSB

Big Table

- Introduced by Google at OSDI in 2006 (Chang et al.)
- Column oriented storage
- Row key, column key, and timestamp -> string
- Master/Slave architecture
- Range queries
- On top of a Distributed File System

Big Table

- Minor Compactions
- Major Compactions



Big Table Clones

- Developed by Zevents
- Written in C++
- Current version: 0.9.3.3 alpha
- Evaluated version: 0.9.2.5 alpha

- Open sourced by PowerSet
  - Now an Apache Project under Hadoop
- Written in Java
- Current version: 0.20.5
- Evaluated version: 0.20.3

HBase and Hypertable

- Hadoop Distributed File System (HDFS)
  - Evaluated with version 0.20.0
  - Replication
  - Fault tolerance
- Thrift interface
  - Communicate with PBServer

Voldemort

- Dynamo clone
- Eventually consistent
- Multiple versions may be returned on a Get
- Uses Berkeley DB for persistence
- Thrift interface
- Written in java
- Current version: 0.81
- Version evaluated: 0.51

Big Table/Dynamo Hybrid

- Originally from Facebook
  - Now an Apache project
- Eventually consistent
- Big Table data model
- Highly available
- Partition tolerant
- Writes directly to the FS
- Written in Java
- Thrift interface
- Most recent version: 0.6.3
- Version evaluated: 0.5.0

MemcacheDB

- Memcached with persistence
- Uses Memcache API
- Uses Berkelely DB
- Master/Slave
- Read from any slave
- Write only to the master
- Version evaluated: 1.2.1 Beta

MySQL Cluster

- Relational DB
- We use it as a column oriented datastore
- Master is only needed for initialization and monitoring
- Node types:
  - API node
  - Data node
  - Management node
- Native MySQL python binding
- Supports transactions
- Version evaluated: 6.3.20

UCSB

MongoDB

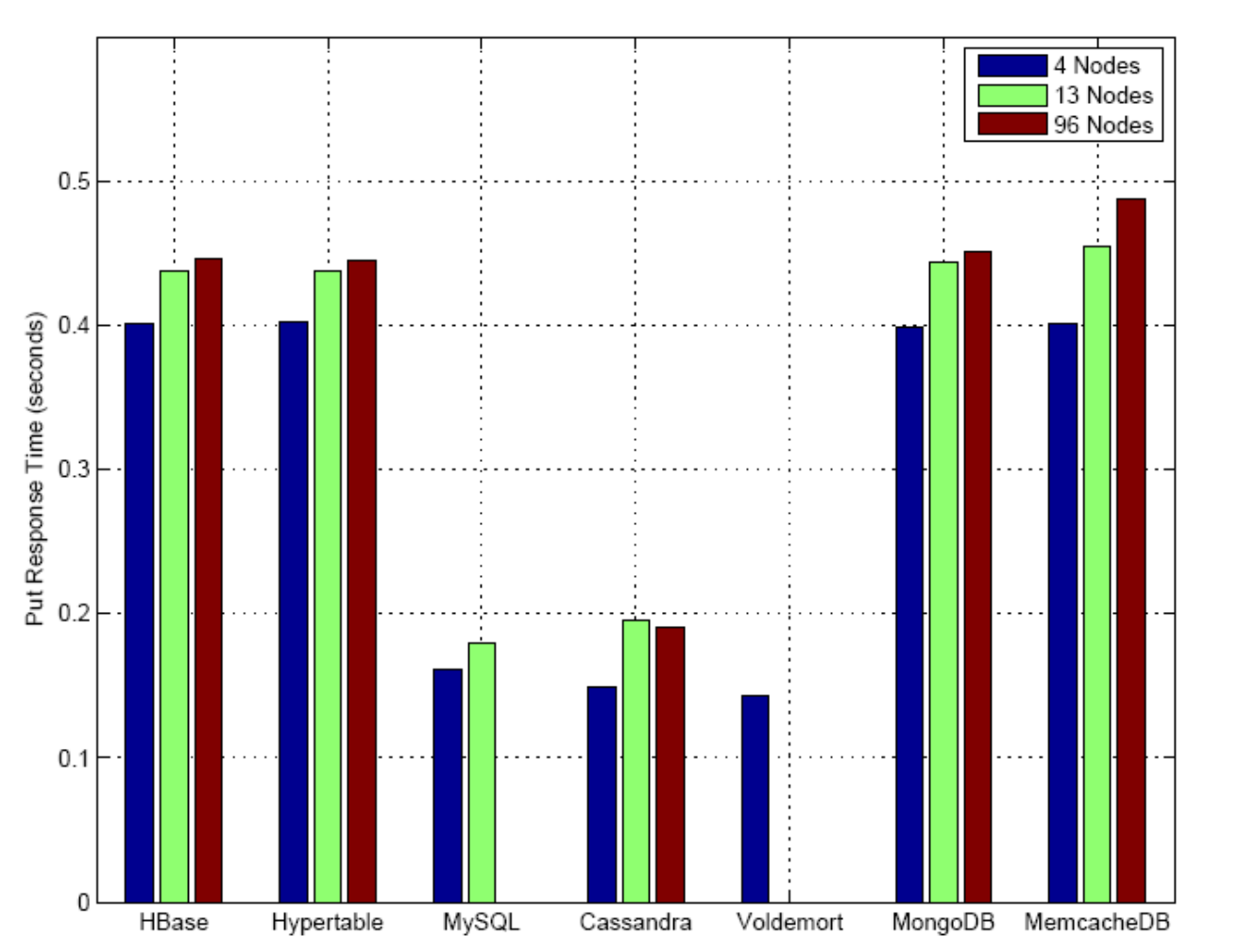
- Released by 10gen as open source
- Document–oriented
  - Documents stored as JSON objects
- All writes and reads are through the master
- Written in C++
- Native Python bindings
- Simple configuration
- Version evaluated: 1.0.0
- Current version: 1.4.4

UCSB

Evaluation Methodology

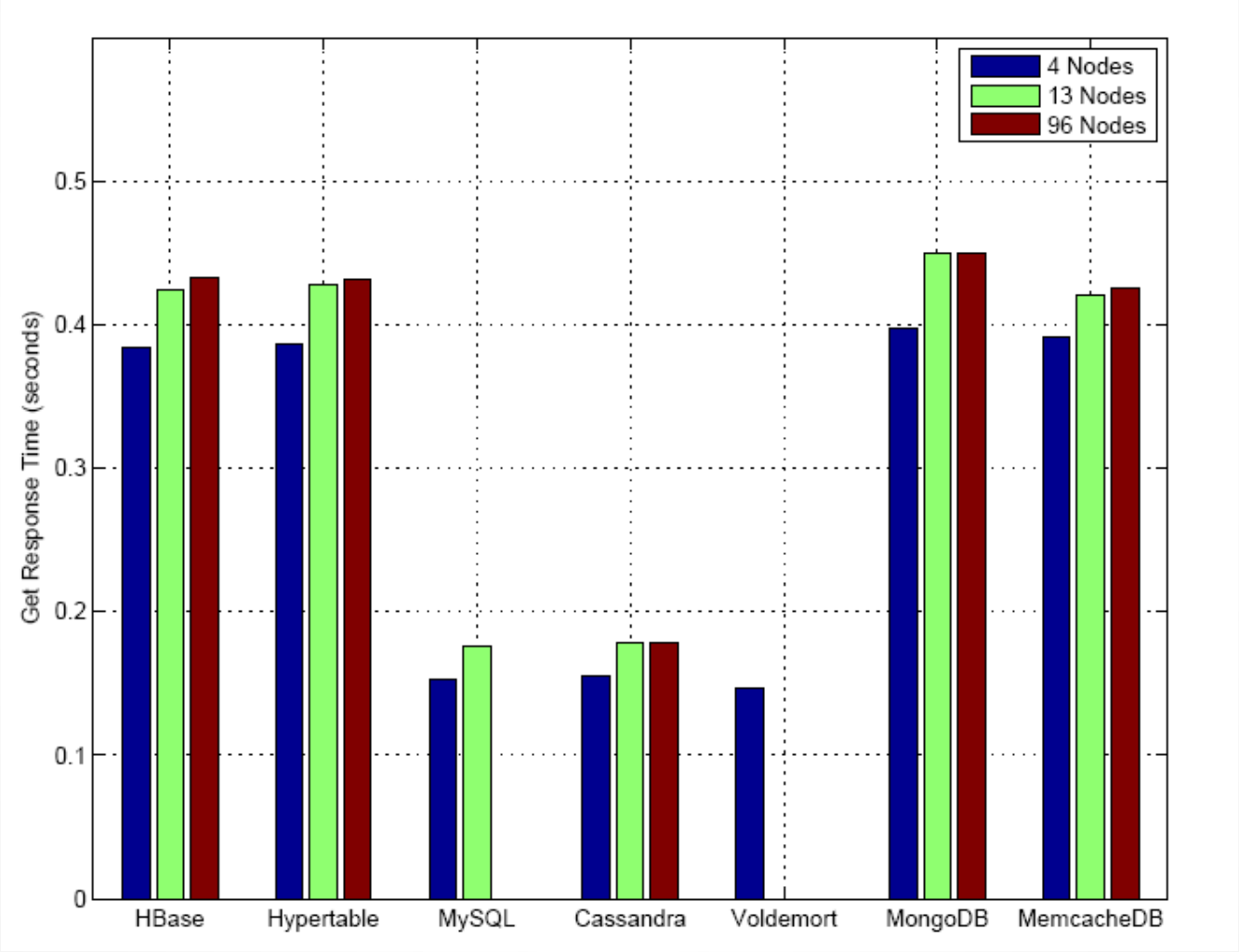
- Run an App with a DB REST interface
  - Exposes GAE API to clients
  - 1000 puts, 1000 gets, 1000 deletes, 100 queries
- Different Scales
  - 1, 2, 4, 13, 96 nodes
  - Xen guestVM
  - 1, 2, 4 node configuration: 4 GB Ram, 2 Virtual Cores
  - 4, 13, 96 node configuration: 1 GB Ram, 1 Virtual Core
- Three loads
  - Low: 1 thread
  - Medium: 3 threads
  - High: 9 threads
- Measuring end-to-end round trip time

Puts


4 Nodes
13 Nodes
96 Nodes
Put Response Time (seconds)
0
0.1
0.2
0.3
0.4
0.5
HBase
Hypertable
MySQL
Cassandra
Voldemort
MongoDB
MemcacheDB

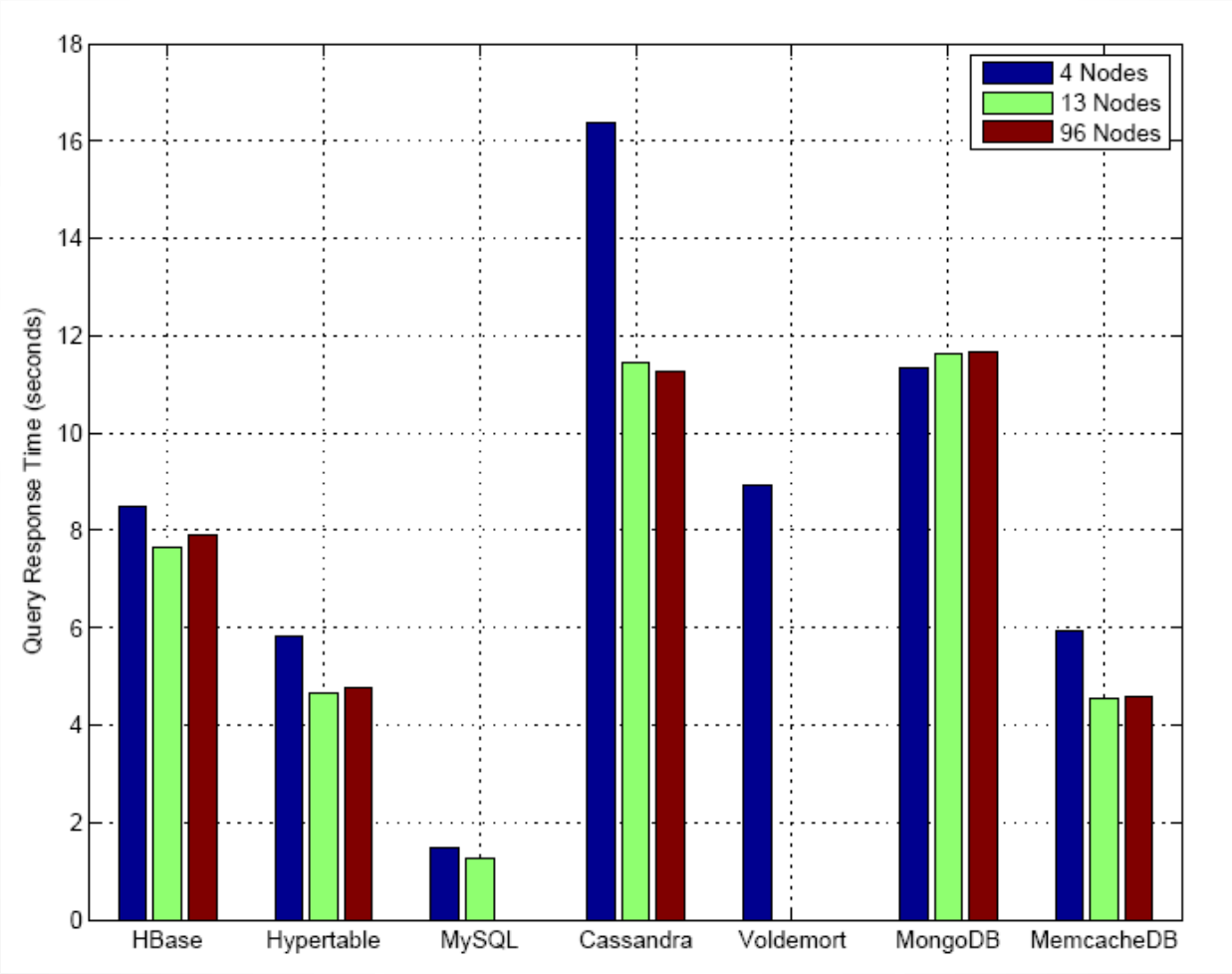
UCSB

Gets


4 Nodes
13 Nodes
96 Nodes
Get Response Time (seconds)
0
0.1
0.2
0.3
0.4
0.5
HBase
Hypertable
MySQL
Cassandra
Voldemort
MongoDB
MemcacheDB

UCSB

Queries



UCSB

Related Work

- YCSB from Yahoo!
- KVZone and the Search for a Write-Optimized Key-Value Store by Gokhale et al.

Current and Future Work

- Database-agnostic transactions
- Database migration
- Cost in the cloud
- Optimizing database interfaces
- AppScale version 1.4 coming soon!

http://appscale.cs.ucsb.edu

- Thanks
  - Google, NSF and IBM for funding this project
  - Chandra Krintz (AppScale Director)
  - Chris Bunch (AppScale co-lead)
  - The AppScale Contributors
    - Jovan Chohan
    - Nupur Gara
    - Matt Hubert
    - Jonathan Kupferman
    - Puneet Lakhina
    - Yiming Li
    - Gaurav Mehta
    - Nagy Mostafa
    - Yoshihide Nomura (Fujitsu)
    - Soo Hwan Park

UCSB