INSTITUTE OF TECHNOLOGY TRALEE

WINTER EXAMINATION AY 2016- 2017

Advanced Database Programming

Module Code DBMS 81001
CRN 48064

External Examiner: Mr Sean McHugh **Internal Examiner:** Mr. P Given **Duration:** 2 Hours

Instructions to Candidates:

i) Answer any **three** questions.

ii) All questions carry equal marks. Submit all your rough-work, marks may be lost otherwise.

Question 1:

i) Compare and contrast **CouchDB** with **PostgreSQL**. **(14 marks)**

(7)

- CouchDB is a NoSQL document Store
  PostgreSQL is primarily a relational database

- CouchDB documents stored in JSON within key-value pairs
  PostgreSQL are stored in Tables.

- CouchDB data types - JSON object types e.g. number, array, boolean, object
  PostgreSQL has predefined and typed data types.

- CouchDB uses RESTful API to connect to database.
  PostgreSQL uses custom interface over TCP protocol.

- PostgreSQL has libraries on multiple programming languages which help with connecting to the database e.g: PostgreSQL JDBC Driver.

- CouchDB uses Master-Master replication
  PostgreSQL uses Master-Slave replication
- PostgreSQL uses SQL to query database,

CouchDB uses views to query database

- CouchDB write operations don't block reads (No transactions or locking)
  Postgres uses locks to implement concurrent ACID transactions

- <Replication/Sharding>
- <Concurrency / Transactions>
- <Transactions>
- <Main Differentiator / Weakness>

//*****
E:\4th_Year_Mi\First Semester\Advanced Database Programming\NOSQL summary\comparing databases.docx
//*****

CouchDB - Genre: Document, Version: 1.1, Datatypes: Typed, Data Relations: None.
- Standard Object: JSON, Written in Language: Erlang, Interface Protocol: HTTP, HTTP/REST: Yes.
- Ad Hoc Query: Temporary views, Mapreduce: Javascript, Scalable: Datacenter (via BigCouch), Durability: Crash-only.
- Secondary Indexes: Yes, Versioning: Yes, Bulk Load: Bulk Doc API, Very Large Files: Attachments.

- Main Differentiator: Durable and embeddable clusters, Weaknesses: Query-ability.
//***

PostgreSQL - Genre: Relational, Version: 9.1, Datatypes: Predefined and typed, Data Relations: Predefined.
- Standard Object: Table, Written in Language: C, Interface Protocol: Custom over TCP, HTTP/REST: No.
- Ad Hoc Query: SQL, Mapreduce: No, Scalable: Cluster (via HA), Durability: ACID.
- Secondary Indexes: Yes, Versioning: No, Bulk Load: COPY command, Very Large Files: BLOBs.

- Main Differentiator: Best of OSS RDBMS model, Weaknesses: Distributed availability.
//**********

ii) Describe the features of CouchDB which make it a good fit in asynchronous environments. **(9 marks)**

- Master-Master Replication
  - All databases will have the same data.
  - _rev field
  - conflicts
  - old values

iii) Write a note on the use of JSON and _rev in CouchDB. **(10 marks)**

JSON - JavaScript Object Notation - used to store information in organized, easily accessible format. CouchDB is a native JSON document database -> built around the format.

CouchDB uses JSON as its storage and communication language

JSON objects consist of key-value pairs - values may be any of several types, including other objects nested to any depth.

Revision string format = integer dash unique pseudorandom string.

integer = numerical revision e.g: 1
e.g. rev=1-917fa2381192822767f010b95b45325b

CouchDB uses _rev value to implement **Multi-Version Concurrency Control (MVCC)**

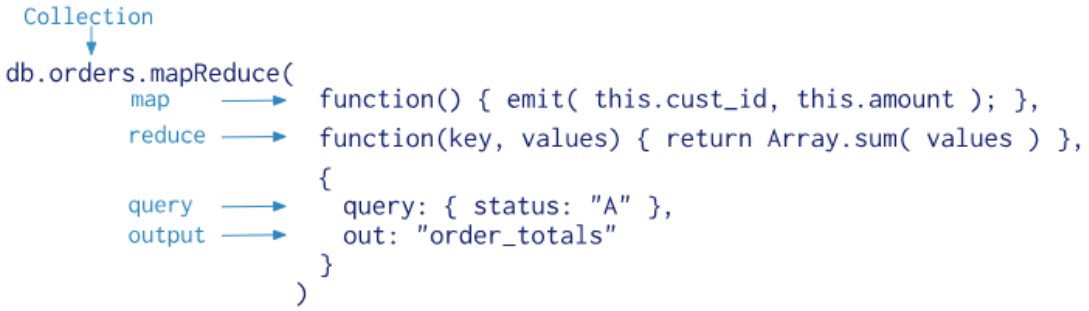
Requiring_rev during operations, CouchDB ensures that the document you think you're modifying hasn't been altered behind your back while you weren't looking.

The _rev field gets new value every document change.

Question 2:

i) Explain how Map-Reduce works in MongoDB, giving examples to support your answer. **(13 marks)**

Diagram



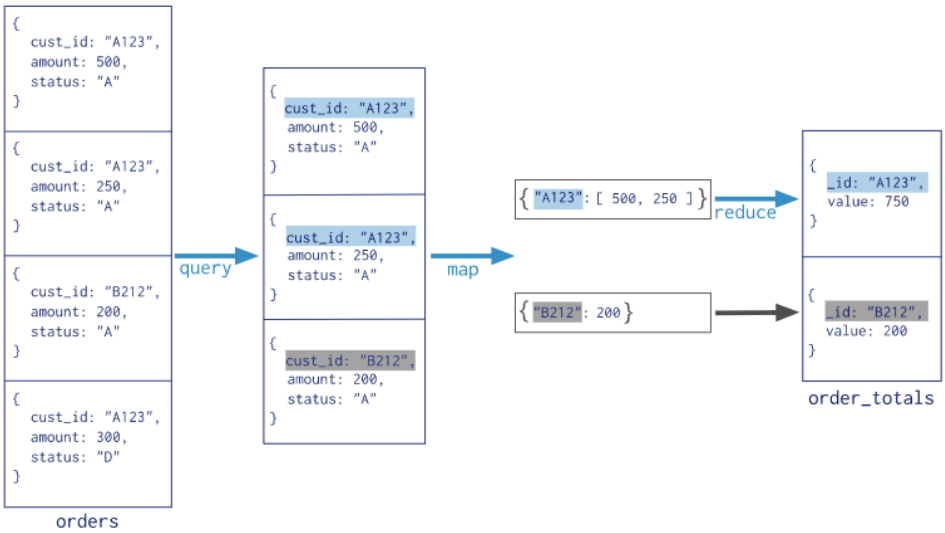
All map-reduce functions in MongoDB are JavaScript. It may use custom JavaScript functions.
Three functions: map(), reduce(key, values), finalize()
map(): emits key-value pairs. Gets applied to each input document (documents that match the query condition)

reduce: Collects data which has the same key and condenses aggregated data
finalize: Optional, further condense or process the results of the aggregation
Input: Documents of a single collection
Ouput: Write into a collection or return the results inline. If written to a collection, subsequent map-reduce functions can be performed, allows new results with previous results. Inline results must adhere to the BSON document size limit of currently 16 MB.

- In this map-reduce operation, MongoDB applies the *map* phase to each input document (i.e. the documents in the collection that match the query condition). The map function emits key-value pairs.
- For those keys that have multiple values, MongoDB applies the *reduce* phase, which collects and condenses the aggregated data. MongoDB then stores the results in a collection.
- Optionally, the output of the reduce function may pass through a *finalize* function to further condense or process the results of the aggregation.

Map-reduce is supported on sharded collection.

Process Of Map Reduce in action,



ii) Write a note on a) server side commands and b) gridFS in MongoDB, giving examples to support your answer. **(10 marks)**

a)

Server-Side commands is JavaScript that is run on the server itself instead of the client, this reduces chatter between the server and the client.

There are also prebuilt commands in Mongo which are executed on the server, some can only be executed under the admin database.

It is possible to leverage the concept of executing server-side code for our own gain to create something in MongoDB that's similar to the stored procedures like in PostgreSQL

The eval command passes the JavaScript code to the server, evaluates it and runs it and returns the results.

//*****************************

b)

GridFS is a distributed filesystem in Mongo, which allows the storage of files. GridFS runs out of the box with mongo through a command-line tool. In the mongoshell it is possible to call the file collections to see the files. Since those files are stored within a collection, they can be replicated or queried.

iii) Explain the CAP theorem and describe how MongoDB keeps its CAP theorem guarantees. **(10 marks)**

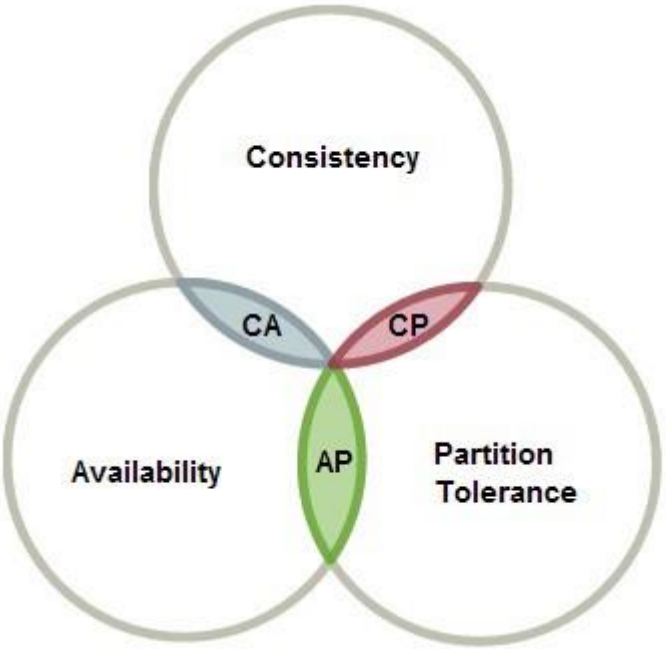
CAP Theorem is a concept that a distributed database system can only have 2 of the 3: Consistency, Availability and Partition Tolerance.

Consistency: writes are atomic and all subsequent requests retrieve the new value

Availablility: The database will always return a value as long as a single server is running

Partition Tolerant: The system will still function even if server communication is temporarily lost that is , a network partition

- Consistency means that when two users access the system at the same time they should see the same data.
- Availability means up 24/7 and responds in a reasonable time.
- Partition Tolerance means if part of the system fails, it is possible for the system as a whole to continue functioning.



MongoDB is generally consistent and partition tolerant (CP).

MongoDB is strongly consistent by default - if you do a write and then do a read, assuming the write was successful you will always be able to read the result of the write you just read. This is because MongoDB is a single-master system and all reads go to the primary by default.

- Automatic failover in case of partitioning
- If a partition occurs the system will stop accepting writes until it believes that it can safely complete them

Advanced Database Programming\NOSQL summary\The Cap Theorm .docx

//*********

week 2 and 3

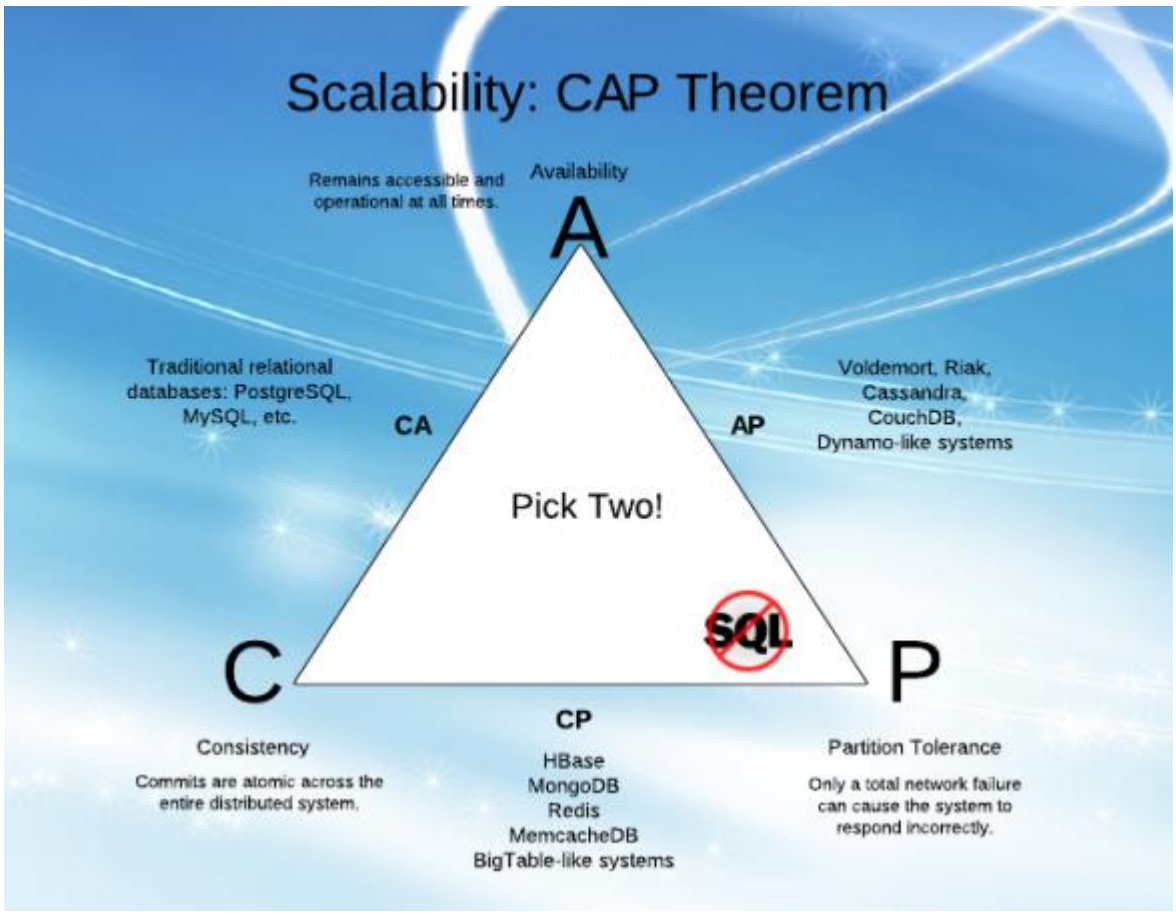
CAP Theorem

It's very common to invoke the 'CAP theorem' when designing, or talking about designing, distributed data storage systems. The theorem, as commonly stated, gives system designers a choice between three competing guarantees:

http://www.youtube.com/watch?v=Jw1iFr4v58M

- Consistency: every read would get you the most recent write
- Availability: every node (if not failed) always executes queries

- Partition-tolerance: even if the connections between nodes are down, the other two (A & C) promises, are kept.



http://architects.dzone.com/articles/better-explaining-cap-theorem

//**********

Question 3:

i) Discuss the strengths and weaknesses of Redis. **(10 marks)**

- Redis resides within memory, which makes it a very fast database but also creates a durability problem, since if you shutdown the database, the data is gone from memory and if not snapshotted beforehand it will be lost.
- Since Redis is stored in memory, it limits the size of the database based on the physical RAM. This means that data sets aren't allowed to be bigger than the available RAM.
- Redis is able to store lists, hashes, sets and retrieve them based through operations specific to those datatypes.
- Redis has a few persistence options to choose from: none, snapshotting, append-only file. Which option to choose from depends on the usage intent. If the database should be used purely as a basic caching server it is most probably best to have no persistence since it can increase latency. Snapshotting saves data to a file in certain intervals and based on how many entries have changed.Append-Only File persistence option saves all write commands within a file, which allows the database

to recover from a crash by reexecuting the commands stored within the file. These options allow redis to trade speed for data safety.

- Redis clustering is immature and has an unclear future????

week 4

Redis's Strengths

The obvious strength of Redis is speed, like so many key-value stores of its ilk.

But more than most key-value stores, Redis provides the ability to store complex values like lists, hashes, and sets, and retrieve them based through operations specific to those datatypes.

Beyond even a data structure store, however, Redis's durability options allow you to trade speed for data safety up to a fairly fine point.

Built-in master-slave replication is another nice way of ensuring better durability without requiring the slowness of syncing an append-only file to disk on every operation.

Additionally, replication is great for very high-read systems.

Redis's Weaknesses

Redis is fast largely because it resides in memory.

Some may consider this cheating, since of course a database that never hits the disk will be fast.

A main memory database has an inherent durability problem; namely, if you shut down the database before a snapshot occurs, you can lose data.

Even if you set the append-only file to disk sync on every operation, you run a risk with playing back expiry values, since time-based events can never be counted on to replay in exactly the same manner—though in fairness this case is more hypothetical than practical.

Redis also does not support datasets larger than your available RAM (Redis is removing virtual memory support), so its size has a practical limitation.

Although there is a Redis Cluster currently in development to grow beyond a single-machine's RAM requirements, anyone wanting to cluster Redis must currently roll their own with a client that supports it.

//**********

week 4

ii) Write a note on a) the durability model in Redis and b) publish subscribe in Redis.

(10 marks)

a)

Redis has a few persistence options to choose from: none, snapshotting, append-only file. Which option to choose from depends on the usage intent. If the database should be used purely as a basic caching server it is most probably best to have no persistence since it can increase latency. Snapshotting saves data to a file in certain intervals and based on how many entries have changed.Append-Only File persistence option saves all write commands within a file, which allows the database to recover from a crash by re executing the commands stored within the file.

Durability

Redis has a few persistence options.

First is no persistence at all, which will simply keep all values in main memory.

If you're running a basic caching server, this is a reasonable choice since durability always increases latency.

One of the things that sets Redis apart from other fast-access caches like memcached is its built-in support for storing values to disk.

By default, key- value pairs are only occasionally saved.

You can run the LASTSAVE command to get a Unix timestamp of the last time a Redis disk write succeeded, or you can read the last_save_time field from the server INFO output.

You can force durability by executing the SAVE command (or BGSAVE, to asynchronously save in the background).

SAVE

// Image here

If you read the redis-server log, you will see lines similar to this:

[46421] 10 Oct 19:11:50 * Background saving started by pid 52123
[52123] 10 Oct 19:11:50 * DB saved on disk
[46421] 10 Oct 19:11:50 * Background saving terminated with success

// Image here

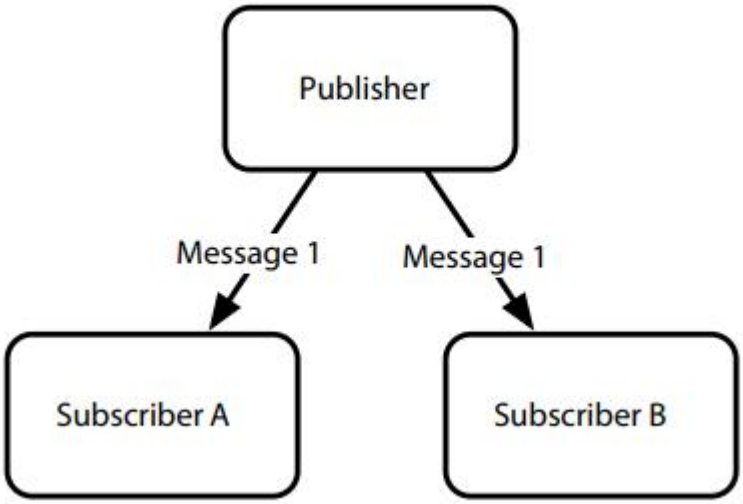
Another durability method is to alter the snapshotting settings in the configuration file.

//**********

b)

Publish-Subscibe allows a user to publish messages to multiple subscribers.Redis offers the following commands for this functionality: subscribe <key> and publish <key> <message>. Subscribers connect to a publishers with a key e.g: comments. To publish a message through the publisher the command publish gets used e.g: publish comments "Check this out.". The subscribers will then receive a multibulk reply of three items

showing the type, key of publisher and the content in this case a message. To unsubscribe from a publisher, the user can use the command unsubscribe <key> or unsubscribe to disconnect from all publishers.



Publish-subscribe

Earlier we were able to implement a rudimentary blocking queue using the list datatype. We queued data that could be read by a blocking pop command.

Using that queue, we made a very basic publish-subscribe model.

Any number of messages could be pushed to this queue, and a single queue reader would pop messages as they were available.

This is powerful but limited.

Under many circumstances we want a slightly inverted behavior, where several subscribers want to read the announcements of a single publisher, as shown on page 278.

Redis provides some specialized publish-subscribe (or pub-sub) commands.

Let's improve on the commenting mechanism we made yesterday using blocking lists, by allowing a user to post a comment to multiple subscribers (as opposed to just one).

We start with some subscribers that connect to a key, known as a *channel* in pub-sub nomenclature.

Let's start two more clients and subscribe to the comments channel.

Subscribing will cause the CLI to block.

SUBSCRIBE comments
Reading messages... (press Ctrl-C to quit)
1) "subscribe"

2) "comments"
3) (integer) 1

// image here

// image here

With two subscribers, we can publish any string we want as a message to the comments channel.

The PUBLISH command will return the integer 2, meaning two subscribers received it.

PUBLISH comments "Check out this shortcoded site! 7wks"
(integer) 2

Both of the subscribers will receive a *multibulk reply* (a list) of three items: the string "message," the channel name, and the published message value.

1) "message"
2) "comments"
3) "Check out this shortcoded site! 7wks"

When your clients want to no longer receive correspondence, they can execute the UNSUBSCRIBE comments command to disconnect from the comments channel or simply UNSUBSCRIBE alone to disconnect from all channels.

However, note in redis-cli that you will have to press CTRL+C to break the connection.

//**********

iii) Discuss, using an example, the architecture of HBase tables and discuss the advantage of using column families in HBase. **(13 marks)**

HBase stores data in buckets it calls *tables*, which contain *cells* that appear at the intersection of *rows* and *columns*.
In HBase, tables don't behave like relations, rows don't act like records, and columns are completely variable (not enforced by a schema description).
HBase has some built-in features that other databases lack, such as versioning, compression, garbage collection (for expired data), and in-memory tables.

Schema design is still important, since it informs the performance characteristics of the system.
A map is a key-value pair, like a hash in Ruby or a hashmap in Java.

A table in HBase is basically a big map. Well, more accurately, it's a map of maps

A row is itself a map, in which keys are called *columns* and values are uninterpreted arrays of bytes.
Columns are grouped into *column families*, so a column's full name consists of two parts: the column family name and the *column qualifier*. Column Family: 'family:qualifier'

Each column family's performance options are configured independently. These settings affect things such as read and write speed and disk space consumption.

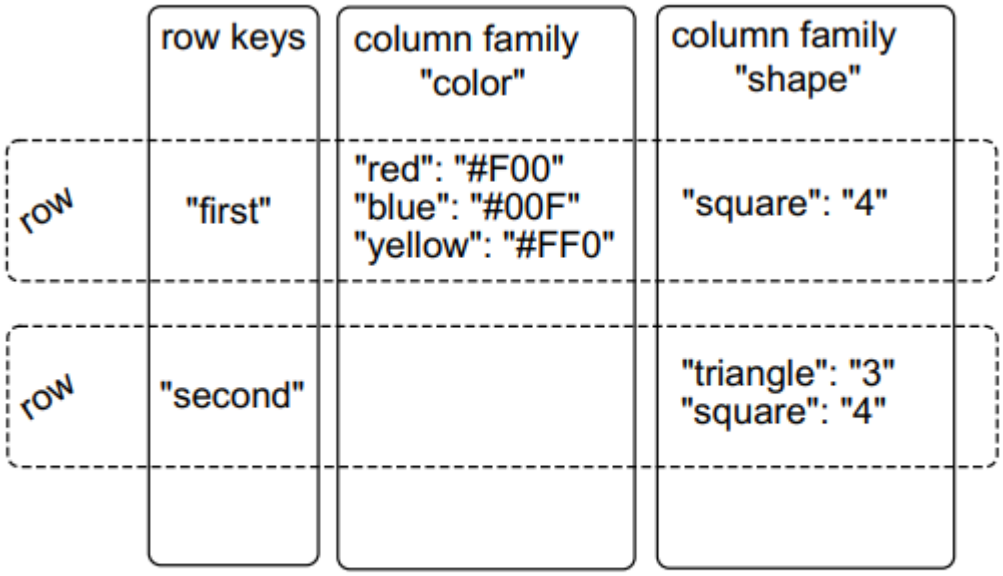


Figure 13—HBase tables consist of rows, keys, column families, columns, and values.

The combination of row key and column name (including both family and qualifier) creates an address for locating data.

//**********

Question 4:

i) Discuss the strengths and weaknesses of RIAK. **(12 marks)**

Week 7

Riak's Strengths

If you want to design a large-scale ordering system a la Amazon, or in any situation where high availability is your paramount concern, you should consider Riak.

Hands down, one of Riak's strengths lies in its focus on removing single points of failure in an attempt to support maximum uptime and grow (or shrink) to meet changing demands.

If you do not have complex data, Riak keeps things simple but still allows for some pretty sophisticated data diving should you need it.

There is currently support for about a dozen languages (which you can find on the Riak website) but is extendable to its core if you like to write in Erlang.

And if you require more speed than HTTP can handle, you can also try your hand at communicating via Protobuf, which is a more efficient binary encoding and transport protocol.

Riak's Weaknesses

If you require simple queryability, complex data structures, or a rigid schema or if you have no need to scale horizontally with your servers, Riak is probably not your best choice.

One of our major gripes about Riak is it still lags in terms of an easy and robust ad hoc querying framework, although it is certainly on the right track.

Mapreduce provides fantastic and powerful functionality, but we'd like to see more built-in URL-based or other PUT query actions.

The addition of indexing was a major step in the right direction and a concept we'd love to see expanded upon.

Finally, if you don't want to write Erlang, you may see a few limitations using JavaScript, such as slower mapreduce execution.

//**********

ii) Describe how Riak controls reads and writes to the cluster and compare consistency by read and consistency by write in Riak. **(12 marks)**

//**********

iii) Using and example, discuss how and why RIAK uses vector clocks. **(9 marks)**

Advanced Database Programming\week 7\new Riak.docx

//**********

//Riak is not coming up in the exam.