W251 Homework 8

# Exercise 1 Classify each of the following NoSQL databases as either (a) key-value store, (b) column store/column family, (c) document store, (d) graph database, or (e) other.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Database** | **(a) key-value store** | **(b) column store/column family** | **(c) document store** | **(d) graph database** | **(e) other** |
| Accumulo | x | x |  |  |  |
| Aerospike | x |  |  |  |  |
| Amazon SimpleDB | x |  |  |  |  |
| Apache CouchDB | x |  | x |  |  |
| Azure Table Storage | x |  |  |  |  |
| BerkeleyDB | x |  |  |  |  |
| Cassandra | x |  | x |  |  |
| Cloudant | x |  | x |  |  |
| Couchbase | x |  | x |  |  |
| Datomic | x |  |  |  |  |
| DynamoDB | x |  | x |  |  |
| Elasticsearch |  |  |  |  | search server |
| FlockDB |  |  |  | x |  |
| FoundationDB | x |  |  |  |  |
| GenieDB | x |  |  |  |  |
| Graphbase |  |  |  | x |  |
| HBase | x | x |  |  |  |
| Hibari | x |  |  |  |  |
| Hypertable | x | x |  |  |  |
| Infinite Graph (by Objectivity) |  |  |  | x |  |
| LevelDB | x |  |  |  |  |
| MarkLogic Server |  |  | x |  |  |
| MemcacheDB | x |  |  |  |  |
| MongoDB | x |  | x |  |  |
| Neo4J |  |  |  | x |  |
| RavenDB | x |  | x |  |  |
| Redis |  |  |  |  | data structure server |
| RethinkDB | x |  | x |  |  |
| Riak | x |  |  |  |  |
| RocksDB | x |  |  |  |  |
| Scalaris | x |  |  |  |  |
| Titan |  |  |  | x |  |

# Exercise 2: Quorum and Dynamo Inspired Systems

1. Name at least three systems that implement quorum protocols.

S3, DynamoDB, Cassandra, Google Spanner, CouchDB

1. Define the following:
   * ‘W’ minimum number of nodes that must participate in a successful read operation
   * ‘R’ minimum number of nodes that must participate in a successful write operation
   * ‘N’ number of nodes/hosts that data is replicated on, only counting nodes that are healthy and can be accessed
   * ‘Q’ number of partitions that hashed space is partitioned equally into
2. Why is ‘N’ generally chosen to be an odd integer?

So that when a vote takes place, it is easy to decide R or W which wins, for example R=W=1 N =2, it may be a tie, but if N=3, then R or W has to be bigger than 2.

1. What condition relating ‘W’, ‘R’, and ‘N’ must be satisfied to "yield a quorum like system"?

R is the minimum number of nodes that must participate in a successful read operation. W is the minimum number of nodes that must participate in a successful write operation. Setting R and W such that R + W > N yields a quorum-like system.

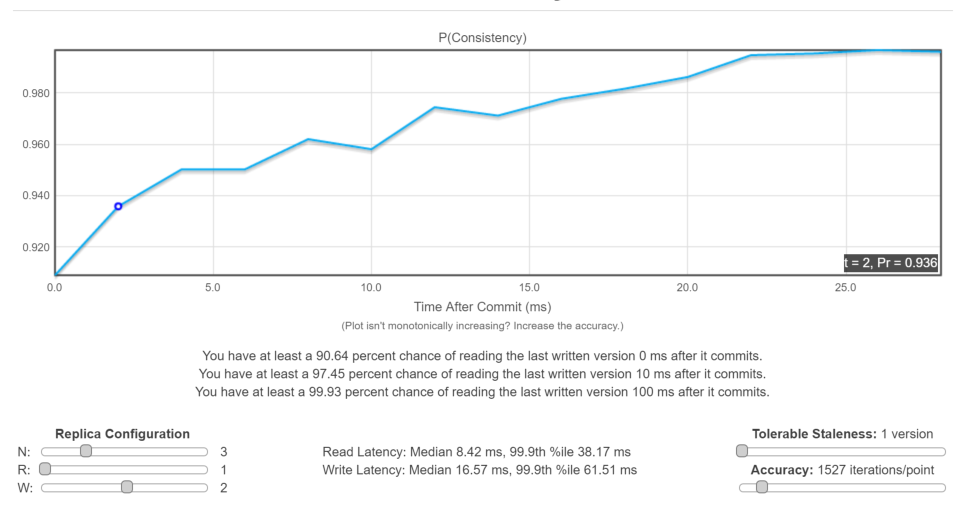
1. In the paper "Probabilistically Bounded Staleness," Berkeley researchers derive an analytic framework for the probability of reading a "stale" version of an object in a Dynamo-like system that implements quorum. Using [this tool](http://pbs.cs.berkeley.edu/#demo) (lambda=0.1 for all latencies, tolerable staleness=1 version, 15,000 iterations/point), answer the following questions:
   * With what probability are you reading "fresh" data for n=3, w=2, r=2?

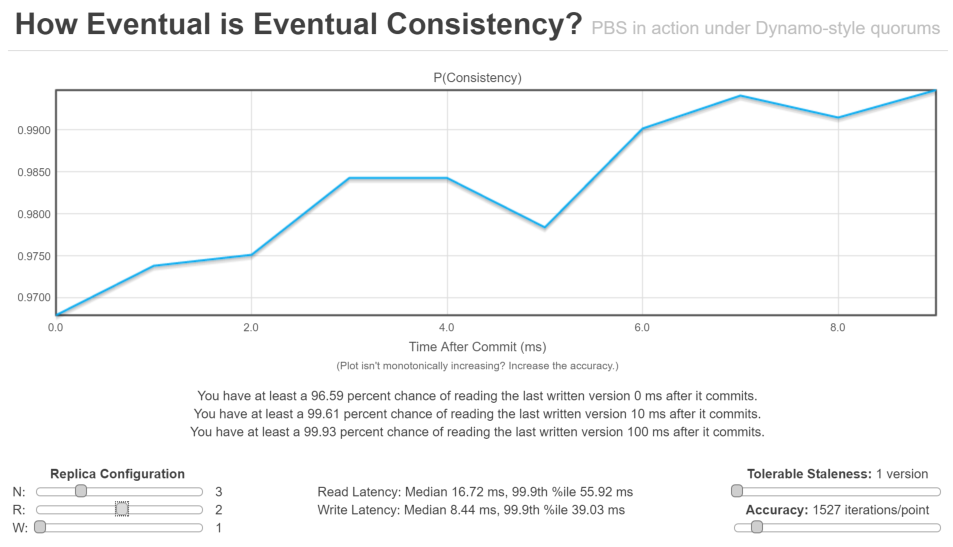
1

* + Does it depend on time? If so, why? If so, why not?

No. because when R+W > N you are guaranteed to read latest data right away.

* + Compare the scenarios for (w,r,n) = (2,1,3) and (1,2,3).





R is 1, write is slower and read is faster but read is more likely to get inconsistent result. R is 2, read is slower and read is less likely to get inconsistent result.

* + Write down and explain the differences (if any) for the time dependence of P(consistent).

As time goes by, it is more likely to get higher consistency. Because data will be eventually replicated to all nodes.

* + Is the (2,1,3) state symmetric with (1,2,3)?

No

* + Compare both P(consistent) and the median and 99.9% latencies.

When R is 1, read latency median is 8.42 ms, and 99.9th percentile is 38.17 ms. When R is 2, read latency increased to median 16.72 ms, 99.9th percentile is 55.92 ms. P(consistent) increase faster when R is 2.

* + Provide an intuitive explanation for your results.

When R is 1, read can take place on any one node, hence faster. But R is 2, it requires at least two node to match in order to return a value, so it wait longer, hence slower, but more likely to return consistent result.

* + Do either of these states favor consistency or availability? If so, why?

Both favor availability over consistency, one favor faster write and one favor faster read. Because the data returned could be inconsistent right after write in both cases.

* + Perform a similar comparison for the (3,1,3) and (1,3,3) states. Do either of these states favor consistency or availability? If so, why?

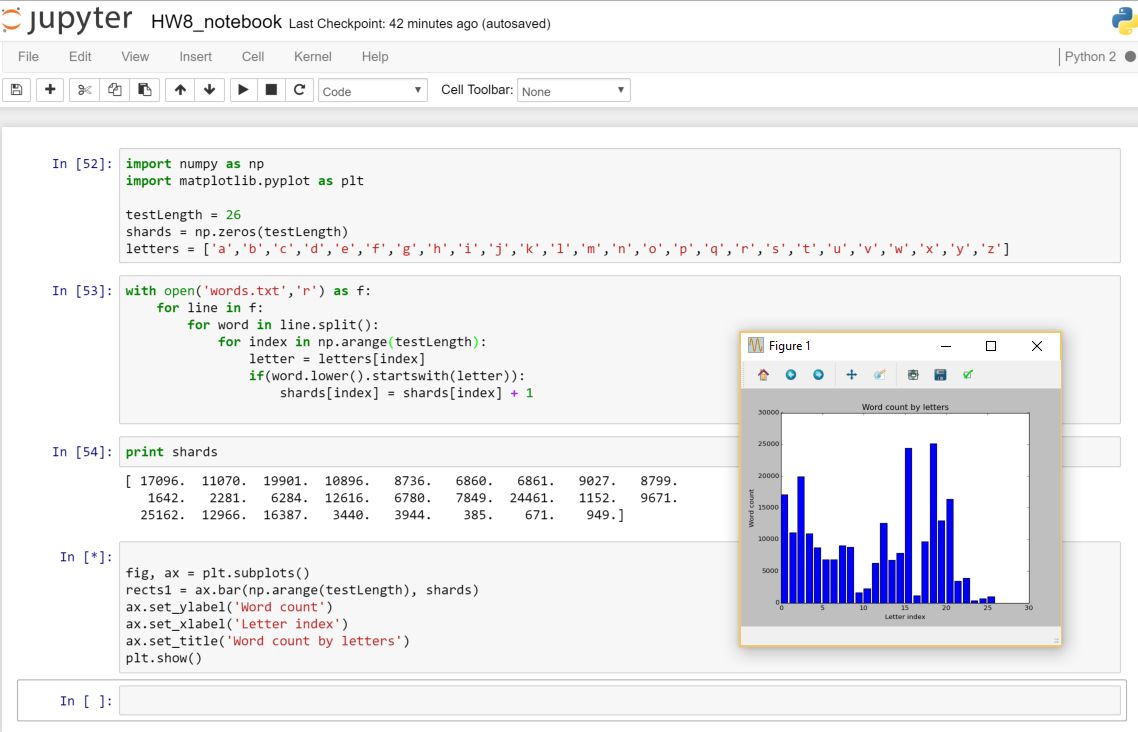
Both favor consistency, because both will always return consistent result.

* + In your opinion, assuming an n=3 system, what do you think is a reasonable choice for write heavy, read heavy, and read~=write workloads?

N= 3, R=W=2.

**Range Partitioning**

1. Build a simple Python class or structure that implements a range-partition map. Your map should consist of 26 "shards," where shard\_0 contains words starting with ‘a’, shard\_1 contains words starting with ‘b’, etc.



1. Generate a plot or table listing the number of words stored in each shard. Is the mapping of objects to shards uniform?

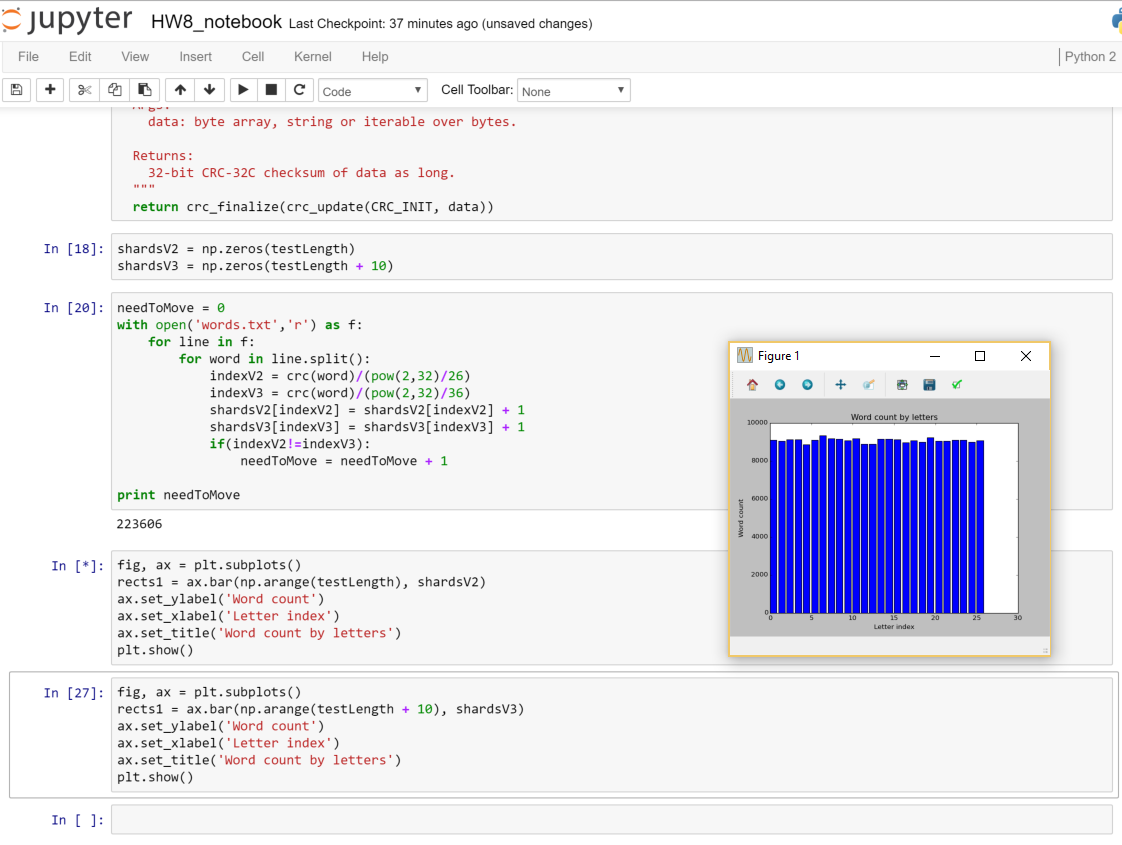
No.

1. Suppose that each shard now lives on a separate database server. Under what workloads (e.g., specific queries) would range partitioning be a good/bad choice?

If index in each range will be queried for similar amount of frequency, range partitioning makes sense, otherwise, this partitioning is not efficient.

**Consistent Hashing**

1. Building on your previous example, implement a hashing approach. Use [this Python file](http://googleappengine.googlecode.com/svn/trunk/python/google/appengine/api/files/crc32c.py) as your hash function.



1. Again, create 26 "shards" but with equidistant boundaries linear on the crc32 interval [0, pow(2,32)], and assign objects to shards based on ‘crc32(word)’.
2. Generate a plot or table listing the number of words stored in each shard.
3. Is the mapping of objects to shards uniform?

Almost uniform, much better than the counting by first letter method.

1. Suppose that each shard now lives on a separate database server. Under what workloads (e.g., specific queries) would range partitioning be a good/bad choice?

When queries mostly interact with one partition at a time, for example mostly select string equals a value. Instead of select strings across different partitions.

1. Suppose that you need to grow your "cluster" by adding 10 additional nodes to the distributed hashing "ring" you built in Exercise 2. By any means you choose, count the total number of objects that would migrate from one shard to a new shard if you were to divide the interval [0,pow(2,32)] into 36 equidistant shards instead of the preexisting 26 shards.

There are 223606 (roughly 95% of all data)

1. Augment the algorithm to be consistent hashing now. Instead of equidistant shards, name each node, hash that name and use the number as the shard boundary. For each node, define a certain number of virtual nodes that represent it on the ring.
2. For a consistent-hashing ring, what is your expectation for the average number of keys that need to be remapped under a table resize if you have ‘K’ keys and ‘n’ shards? How is that value different for standard hash tables?

When the table is resized, expect to have only k/n keys need to be remapped on average. When it is standard hashing table, resize table often result in remap of almost all keys.

1. Name at least one popular NoSQL project that uses range partitioning.

HBase, MongoDB

1. Name at least three popular NoSQL projects that use consistent hashing.

Openstack, Dynamo, Apache Cassandra