**Provide examples when aggregate-oriented databases are not an efficient way to store data**

1.Aggregate oriented databases help us tackle the need for a cluster based meta-model where much of the information processed stays in the same node.  However aggregate meta models have limitations in scenarios when we need to update multiple aggregates we can encounter failure partway through.

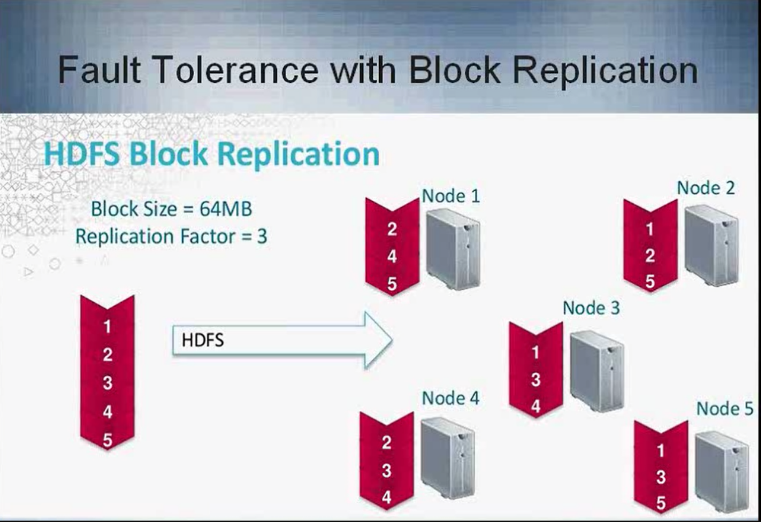
Unlike relational databases that have ACID transaction properties, aggregate databases have reduced means of ensuring consistency. If we have data based on lots of relationships, then a relational database may be more suited than a No-SQL option. In scenarios where we want to ensure consistency and enable concurrency with referential integrity between the meta-model entities and a strong set of relationships we need to follow a normalized data model of a relational database.

For example, a banking enterprise needs to ensure atomicity, consistency, isolation, durability of its client transactions so they would use a relational model instead of an aggregate model.

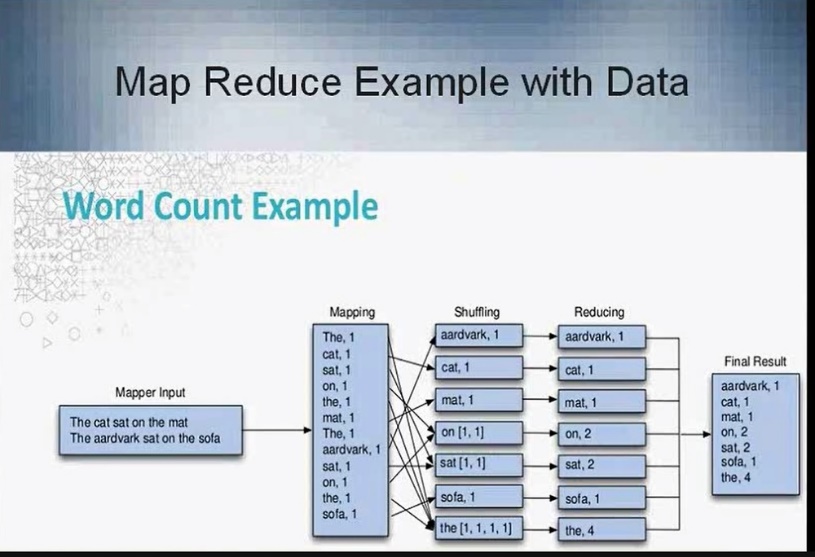
2. I think that we have potential bottlenecks at the physical level and the logical level for no sql databases. In the physical level the bottlenecks are avoided by clustering and making the data storage redundant and fault tolerant. At the logical level the bottlenecks are avoided by making use of precomputed views and massively parallel processing using Map and Reduce APIs. Once these two performance bottlenecks are tackled we can ensure that the no sql database and its business use case are being successfully met.

**Explain why aggregate-oriented databases make it easier for the database to manage data storage over clusters.**

1.One of the key features of NoSQL databases is greater fault tolerance. Each HDFS Block is replicated across multiple nodes allowing for data to be backed up across massively distributed processing jobs.



The simple MapReduce program below illustrates how a sentence can be parsed and processed across multiple nodes in a sparse data scenario. Semantic analytics can be then applied to the data set accordingly.



2.Aggregate is a collection of related objects that we treat as a unit. Sadalage describes the aggregate as a unit of data manipulation and management of consistency. We update aggregates via atomic operations and communicate with our data storage in terms of aggregates. Dealing in aggregates makes it easier for databases to operate on a cluster, the aggregate makes a natural unit f or replication and sharding. (Sadalage, p.14).

3. In relational databases we have ACID that ensures transactional integrity.

ACID properties ensure that a transaction meets the standards established for maintaining data integrity in a database.

A transaction is a sequence of database operations that access the database. Transactions have four main properties these are atomicity, consistency, durability and isolation.  Two main statements of COMMIT and ROLLBACK define the implementation and undo of a transaction. The transaction log keeps a track of all transactions that occur in the database. Concurrency control defines the simultaneous nature of some transaction events.  Often time’s concurrent execution of transactions can lead to lost data and updates.

A scheduler determines the order in which concurrent transactions take place.  The scheduler uses time stamping, locking and optimistic methods to ensure the synchronized nature of individual threads and guarantees serial transactions.

There are locks that prevent simultaneous access to a data object when someone is using it.  Binary locks can have only two states, locked or unlocked. A shared lock is used when transaction wants to read data from a database.  An exclusive lock is issued once a transaction updates to the database and no other locks hold the data.  Two phase locking guarantees that a transaction has all the locks it needs in place during the growing phase and a release of all locks during the shrinking phase. There can also be a deadlock when two transactions are waiting on each other and these can be avoided via prevention, detection and avoidance.

Time stamping methods ensure that a particular transaction is rolled back or executes. These are the wait/die scheme and the wound/wait scheme. Having transaction control with optimistic methods ensures that transactions are executed using private, temporary copies of data. ANSI defines four levels of transaction isolation which are Read uncommitted, Read committed, Repeatable Read and Serializable.

 References:

Coronel, Carlos & Morris, Steven. (2015). Database Systems: Design, Implementation, and Management   11th Edition. Cengage Learning.

**Explain why we must chose two of the three components of CAP theorem which are Consistency, Availability and Partition Tolerance.  Which of the two components does HBase favor?  Why?**

The CAP theorem states that we have to assess our choices when it comes to supporting partitions, availability and consistency. Databases that are partitioned and available are different from databases that are partitioned and consistent.  HBase in this example is a database that is partitioned and consistent. HBase allows automatic partitioning and is a schemaless, column oriented database. HBase is built for low latency operations and supports random read and write operations.Today, HBase doesn’t address two important areas:

1. Write availability during failures.
2. Consistent reads and writes across datacenters.

References:

Retrieved from http://hortonworks.com/blog/apache-hbase-high-availability-next-level/

**When using a real cluster of many servers, explain how the region servers function.  How many servers are the minimum for HBase according to Chapter 4 in the text book?**

In HBase rows are sorted by row key. A region is a chunk of rows identified by starting key(inclusive) and ending key(exclusive). Regions never overlap and each is assigned to a specific region server in the cluster. In a simple configuration the stand alone server is the only region server. (Redmond, p.111)

In distributed clusters we will have many region servers. A HBase cluster is typically made up of at least five nodes.

References:

Redmond, E. & Wilson, J. R. (2012). Seven databases in seven weeks. Dallas, TX: Pragmatic Programmers, LLC. (The Pragmatic Bookshelf)

The text discussed peer to peer replication as a means to ensure that the master is not the single point of failure or bottleneck.  In peer to peer replication all replicas have equal weight, can accept all writes and the loss of any of them does not prevent access to the data store.  However, we can have issues like write-write conflict and lead to inconsistencies.

**Describe the circumstances when HBase is appropriate to use and when it is not appropriate to use.**

HBase is advantageous when we have a large scale project. There are a minimum of five nodes recommended in a cluster.  HBase is deployed in conjunction with other ecosystem technologies like Hadoop, HDFS, Zookeeper.

HBase has an issue where it does not scale down properly, Documentation for specific use cases is evolving but not complete. Administration becomes tedious with large clusters and complex ecosystems. Also sorting and indexing options in HBase are limited aside from row keys.

Datatypes in HBase are all casted to byte arrays for which we need to have application specific byte array translation methods.

The Gartner Hype Cycle attributes the BI& A 1.0 being dependent on column based DBMS, In-memory DBMS, involving real time decisions and data mining workbenches.  BI&A 2.0 is defined as web based operating on unstructured content and focused on information retrieval and extraction, operation mining, question answering, social media analytics, social network analysis, spatial temporal analysis. The Gartner Hype cycle attributes information semantic services, natural language question answering, content and text analytics to BI &A 2.0.

Finally, BI &A 3.0 characteristics and capabilities as highlighted as focused on mobile and sensor based content which has location aware analysis, person centered analysis, context relevant analysis and mobile visualization and HCI in general expressed over Mobile BI.

Its interesting to see how databases are evolving and now there is a push to enable real time analytics within the database that enables us to create something like an enterprise data hub.

I found the discussion in the Seven weeks’ textbook of the use case of maintaining versioned articles from Wikipedia interesting. Since hbase allows us to keep versioning history by use of timestamps I could see its application in maintaining a article revision history or an editors change log relevant.

I also think that data structures like bloom filters when used with discretion are helpful in determining if a data has entered the database previously or if a row key exists.

**Describe the circumstances when Cassandra is appropriate to use and when it not appropriate to use.**

I found the following article that shows ebay's use of Cassandra. The tech writers mention that when data modeling with Cassandra you should.

* Not think of a relational table, but think of a nested sorted map data structure while designing Cassandra column families.
* Model column families around query patterns. But start your design with entities and relationships, if you can.
* De-normalize and duplicate for read performance. But don’t de-normalize if you don’t need to.
* Remember that there are many ways to model. The best way depends on your use case and query patterns.

References:

retrieved from <http://www.ebaytechblog.com/2012/07/16/cassandra-data-modeling-best-practices-part->

Sadalage and Fowler mention the following use cases appropriate for Cassandra.

1. Event logging- column family databases are great when it comes to store event information such as application state or errors.  The timestamp feature of Cassandra works great to track historical event data.

2. Content Management Systems and Blogging Platforms-  We can store blog entries with tags, categories, links and trackbacks in different columns.

3. Counters- You can use count and categorize visitors of a page to help in traffic analytics.

4. Expiring Usage- You can configure expiring columns defined with TTL (time to live) which helps with timing certain promotions or features.

According to Sadalage and Fowler Cassandra is not great for early prototypes or initial tech spikes. Since during early stages we are not sure how the query patterns might change and as query patterns change the column family design has to adapt accordingly. In Cassandra the cost is higher for query change as compared to schema change. (Sadalage, p.109)

Cassandra is also not suited in scenarios where we need to meet transaction integrity that follow ACID guidelines.

Spotify uses Cassandra to manage embedded version control for a billion playlist objects. Each row primary key is the playlist id and each subsequent column stores a mutated version of the playlist.  It becomes easy to track back the changes to a playlist over time due to the change history maintained in each column.

Here is the Cassandra use case for Spotify for more details.

<http://www.slideshare.net/JimmyMrdell/playlists-at-spotify-cassandra-summit-london-2013>

**Discuss why a ring of peer-to-peer servers offers high availability.**

I found the section by Sadalage on write and read operation consistency insightful.

Write operations: Sadalage mentions that as a write is received by Cassandra it is first logged in commit log and then moved to an in memory structure known as mem table. Writes are batched in memory and periodically written to structures known as SSTable. SSTables are not written to after they are flushed. If there are changes to data new SSTables are written.  Unused SSTables are reclaimed by compaction.

Cassandra writes to one nodes commit log and returns a response to the client. Some writes can be lost if the ndoe goes down before the write is replicated to other nodes.

Read operations: When a read request is made Cassandra returns data from first replica even if data is stale. In a process called read repair subsequent nodes are queried for new data  in order to get latest data.  There is low consistency.

We can use Quorum consistency setting to ensure that newest timestamp is returned back to client and outdated replicas are fixed by read repair operations. We can specify alternately a ALL consistency setting which requires that all nodes respond to reads or writes and does not tolerate node failures or outages.

Sadalage points in his text that peer to peer master less cluster configuration allows for high availability. Furthermore consistency constraints can be defined on nodes by the quorum consistency setting where the newest timestamp is returned for reads and writes.  And alternately the all consistency option where the cluster is made intolerant to faults.

Sadalage also provides the following equation to balance availability and consistency.

R+ W > N. (Read and write consistency depends on number of nodes carrying out replication

W is minimum number of nodes where write must be successfully written.

R is minimum number of nodes that must respond successfully to read.

N is number of nodes participating in the replication of data.

Sadalage and Fowler discuss that Cassandra follows a peer to peer node architecture. The cluster does not have a master node so any read and write can be handled by any node on the cluster. (Sadalage, p.100) Scaling an existing Cassandra cluster means adding more nodes and horizontally increasing maximum uptime and supporting more writes and reads.

**How is the Cassandra Counter Column Type like a sequence in Oracle and how is it different.**

In oracle the sequence function operates on a numerical value by incrementing the current value to next value.  This increment is used to handle the need for non-intelligent surrogate keys where primary keys might be cumbersome.

In Cassandra we have an option to create a column counter and increment it whenever data in the associated columns is persisted in the database. (Sadalage, p.108) The column counter can help in determining analytics type metrics for the number of times a particular attribute was accessed by users. We can increment a counter in CQL using the following query by specifying the factor of the increment and the key whose counter gets incremented.

UPDATE visit\_counter SET home = home + 1 WHERE KEY = 'mfowler'

References:

Sadalage, P. & Fowler, M. (2013). *NoSQL distilled*. Upper Saddle River, NJ: Addison-Wesley

**What types of documents can be stored in MongoDB?**

Mongo DB stores documents in BSON format, which is binary format of JSON documents and contains more fields than JSON. The maximum BSON document size is 16 MB.  If you want to store larger documents, you can use Grid FS API.

We can have the following document structure for example which has in itself nested documents, and different data types and their values:

var mydoc = {

\_id: ObjectId("5099803df3f4948bd2f98391"),

name: { first: "Alan", last: "Turing" },

birth: new Date('Jun 23, 1912'),

death: new Date('Jun 07, 1954'),

contribs: [ "Turing machine", "Turing test", "Turingery" ],

views : NumberLong(1250000)

}

The above fields have the following data types:

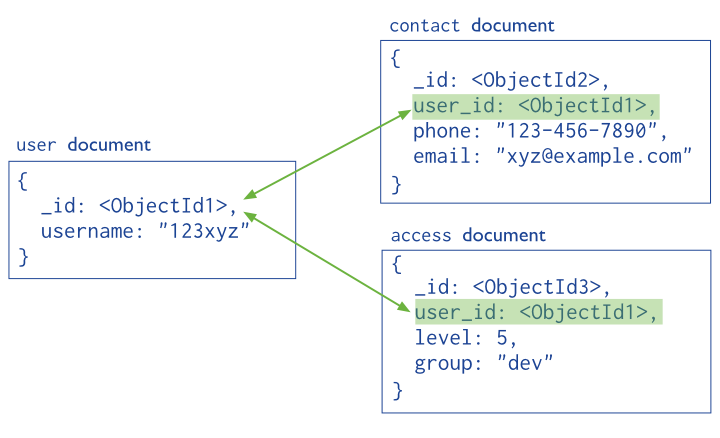
* \_id holds an [*ObjectId*](https://docs.mongodb.com/manual/reference/bson-types/#objectid).
* name holds an *embedded document* that contains the fields first and last.
* birth and death hold values of the *Date* type.
* contribs holds an *array of strings*.
* views holds a value of the *NumberLong* type.

References:

retrieved from <https://docs.mongodb.com/manual/core/document/>

**Explain the difference between a document and relational database. Provide a concrete example to support your statements.**

Mongo DB maintains relationships between documents or rows by associating certain reference fields within related documents. This is analogous to foreign keys in a relational table.



It seems like Mongo DB is not schema less but has dynamic schema which allows us to define a collection of documents each with customizable set of nested attributes derived for business level use cases. A traditional table or collection of documents can be organized dynamically keeping the available attributes in mind.  We can use GridFS API to break down large documents and their metadata into chunks and handle those with MongoDB.

As one white paper outlines,

"Schema design determines the way an application handles its data. With traditional relational databases, you must define your schema before you can add any data. This inflexibility means you can’t change your schema as your data, application requirements or business evolves. In today’s world hyper-competitive, global business environment, this can hamper your efforts to innovate and stay on top of the competition.

NoSQL databases arose to address this limitation by allowing you to insert data without a predefined schema. Because of this, you can easily make changes to an application without interruption. The result is more reliable code integration, faster development, and database administration time.

MongoDB provides a document data model that lets you store and combine any type of data while also offering sophisticated data access and rich indexing features. With MongoDB, you don’t have to reinvent the wheel every time your business requirements change or evolve over time."

References:

retrieved from https://www.mongodb.com/scale/dynamic-schema-design

A relational database has a fixed schema and integrity constraints. Each row in a relational database table is identified by a unique primary key.

A document database is schema less and allows for nested values in its key value ordering of data. The values themselves can be of various types such as other documents, arrays, multiple data types.  Each document id identified by a mandatory id field which acts like a primary key. Attributes can be inconsistent across documents and dont need to be consistent as they would be in a relational schema.  In documents there are no empty attributes.

Each mongo db instance has multiple databases and each database can have multiple collections.  An RDBMS instance is the same as a Mongo DB instance, schemas in RDBMS are similar to Mongo DB Databases and RDBMS Tables are collections in MongoDB. (Sadalage, p.94)

References:

<https://docs.mongodb.com/manual/core/document/>

Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled. Upper Saddle River, NJ: Addison-Wesley

In Mongo DB, each table is a collection and each collection has a documents or rows.  Each document is identified by a unique id field which acts as a primary key and each document can hold a variable list of attributes of different data types.

The link below shows common SQL relational command comparison to MongoDB commands.

<https://docs.mongodb.com/manual/reference/sql-comparison/>

You can create a collection(table) in Mongo DB as follows and add data to it as follows:

| **SQL Schema Statements** | **MongoDB Schema Statements** |
| --- | --- |
| CREATE TABLE users (  id MEDIUMINT NOT NULL  AUTO\_INCREMENT,  user\_id Varchar(30),  age Number,  status char(1),  PRIMARY KEY (id)  ) | Implicitly created on first [insert()](https://docs.mongodb.com/manual/reference/method/db.collection.insert/#db.collection.insert) operation. The primary key \_id is automatically added if \_id field is not specified.  db.users.insert( {  user\_id: "abc123",  age: 55,  status: "A"  } )  However, you can also explicitly create a collection:  db.createCollection("users")   | **SQL INSERT Statements** | **MongoDB insert() Statements** | | --- | --- | | INSERT INTO users(user\_id,  age,  status)  VALUES ("bcd001",  45,  "A") | db.users.insert(  { user\_id: "bcd001", age: 45, status: "A" }  ) | |

**Describe two different use cases for document databases**

A document database can be used for the following use cases:

Event Logging: Document databases can store different types of events and act as a central data store for event storage. Events can be sharded by the name of the application where the event originated or by the type of event.

Content Management Systems, Blogging Platforms: Document databases come without pre-defined schemas and work well in content management systems or applications for publishing websites, managing user comments, user registrations, profiles or web facing documents. (Sadalage, p.98)

Real time Analytics: Document databases can store data for real time analytics. Parts of the document can be updated capturing unique visitors or page views.

References:

Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled. Upper Saddle River, NJ: Addison-Wesley.

So since a document database model makes attributes diverse it might not be ideal for obtaining historical data in a data warehousing/business intelligence scenario. In data warehouses before a dimensional model is constructed we need to have conformed or consistent dimensions that are linked to measurable facts. Since a document database can have a variety of forms in which a name can be expressed it won’t fit into a conformed dimensional model intuitively.

**How does MongoDB support high availability and performance throughput?**

**Describe Map Reduce functions**

Map Reduce API is an open source API that provides fasts data analytics services.  MapReduce distributes the processing of data among thousands of nodes in parallel. MapReduce works with structured and unstructured data. The MapReduce framework provides two main functions, Map and Reduce.  In general terms, the Map function takes a job and divides it into smaller units of work; the Reduce function collects all the output results generated from the nodes and integrates them into a single result set.

An example of a map and reduce function would be as follows(attached).

References:

retrieved from <https://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/>

retrieved from <http://www.slideshare.net/andreaiacono/mapreduce-34478449>

**Explain the difference between a key/value and relational database. Provide a concrete example to support your statements**

I found the data model of Riak similar to XML document.

For example, in each XML document we have a namespace and then associated complexTypes or business objects which are a collection of name value pairs. Each complexType with its associated namespace can help identify a interface and its schema objects for any application/webservice.

Similarly in Riak we have Buckets with an associated namespace and then key and values. The nested structure seems very intuitive for me and differs greatly from a normalized relational entity model.

It seems to me that the most useful use cases in relational databases are in seeing the intersections of sorted data attributes, in the form of various joins.

For NoSQL databases the most useful functions are in feeding a set of logically sorted data to parallel computing algorithms. For example a bucket and its key value(blob) pairs for Riak are fed into a map reduce algorithm which allows us to see the common attributes within the bucket value objects(blobs).

The terminology analogy between Oracle and Riak is as follows.

A database instance in Oracle is analogous to to a Riak cluster.  A relational database table maps to a bucket which is a set of keys that are collectively assigned a common namespace . Each row in a relational table maps to a key-value pair. Each row id maps to an individual key. ( Sadalage , p.81)

References:

Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled. Upper Saddle River, NJ: Addison-Wesley

**Describe three different built-in MapReduce functions in Riak and how to add custom functions.**

1. In Riak, we can use the Map reduce function for example to count the number of times a particular word ("demo") appears in a set of documents.

Demo Data
        map_demo/key1.txt
          Random boring demo data for map demo

        map_demo/key2.txt
          Mo...

Results




basho
 

2. We can alternately apply a sort reduce function and sort documents based on the number of times a word("demo") appears within them.

Request




basho
 

Results




basho
 

References:

retrieved from <http://www.slideshare.net/dreverri/riak-mapreduce>

The approach that Riak takes for map reduce is to bring the algorithm to the data instead of moving the data to the algorithm. (Redmond, p.65) This makes sure that the pattern for reducing is passed to database nodes which then reduce the result instead of collecting the data set from the nodes and then passing it to an API.

**How do the N, W, R values in the Riak Database support transactions in the database?**

In Riak data stores allow us to balance the CAP Theorem, we generally have a masterless architecture where neighboring nodes perform hinted handoff to update nodes that rejoin the cluster.

* N= number of nodes that store key value replicas
* R=  minimum number of nodes where data has to be fetched from to consitute a successful read operation.
* W= minimum number of nodes where data has to be written to constitute a successful write operation.

For example we can have a 5 node Riak cluster, where replication factor N is 5. Using the formula N-W = R we can compute the minimum threshold for node failure.

If R = 2, atleast two nodes have to reply to a GET request.

If W=3, atleast three nodes have to have confirmation of a write to  a PUT request.

Reference

Sadalage, P. J., & Fowler, M. (2012). *NoSQL Distilled.* Upper Saddle River, NJ: Addison-Wesley.

retrieved from http://basho.com/wp-content/uploads/2015/05/RelationaltoRiak.pdf

It appears that horizontal scaling in Riak makes the system highly available but unqueryable and inconsistent. Riak is more suited for write and store operations and when it comes to query and retrieve operations we get stuck with the impossible requirement of scaling consistency across a very spread out system.

**Describe three different use cases for KV databases**

I think gaming companies can use Riak effectively as it allows you to create buckets of session and profile related groupings which most games nowadays require. We can also link keys to determine in game behavior statistics between objects and players.

I would agree that Riak would be suited to store large amounts of Internet of Things(Sensor) data. Because the topography is very spread out and available we can write to the nodes in the cluster without interruption. But if we have a business intelligence dashboard where we need to read from the topology then we would not have consistent reads. So it might only work halfway

Riak is useful in the following scenarios:

1. Storing session information - we can aggregate session data and store  it via a single PUT request to a single object.

2. User Profiles, Preferences - we can create a preferences object with nested name value pairs of attributes. A single GET request can retrive the entire nested object for preferences/profiles.

3. Shopping Cart Data- we can manage multi tenancy via storing sessions for users in the key value store.

References:

Sadalage, P. J., & Fowler, M. (2012). NoSQL Distilled. Upper Saddle River, NJ: Addison-Wesley.

-

**Describe three different use cases for Graph database.**

Location based recommendation services like Yelp might be good candidates for graph database uses.

Yelp sponsored a dataset analysis challenge which asked users to use graph databases to find out cuisine related cultural trends, seasonal trends in restaurant sales, change points after they exported their data in json files.

References:

retreived from <http://datablend.be/?p=308>

retreived from <https://www.yelp.com/dataset_challenge>

I found the bioinformatics use case interesting. A protein interaction database would be a good candidate for a graph database use case. Graph databases have so far seen only limited use within bioinformatics ([Schriml](http://bioinformatics.oxfordjournals.org/content/early/2013/10/21/bioinformatics.btt549.full" \l "ref-3) *[et al.](http://bioinformatics.oxfordjournals.org/content/early/2013/10/21/bioinformatics.btt549.full" \l "ref-3)*[, 2012](http://bioinformatics.oxfordjournals.org/content/early/2013/10/21/bioinformatics.btt549.full" \l "ref-3)). However the graph database should be used only if the query requirements are appropriately satisfied. Graph databases are not good candidates when set operations are required on the data.

Visualization is a big part of a graph database and a graph makes it easy to visualize interactions when compared to a table. (see attached slide).

An example case study investigating the feasibility of graph database use cases in bioinformatics imported the human interaction network from STRING v9.05 ([Franceschini](http://bioinformatics.oxfordjournals.org/content/early/2013/10/21/bioinformatics.btt549.full" \l "ref-1) *[et al.](http://bioinformatics.oxfordjournals.org/content/early/2013/10/21/bioinformatics.btt549.full" \l "ref-1)*[, 2013](http://bioinformatics.oxfordjournals.org/content/early/2013/10/21/bioinformatics.btt549.full" \l "ref-1)), which is an approximately scale-free network with 20140 proteins and 2.2 million interactions.

A relational PostGRE database was used for comparison to a Neo4j NoSQL graph database model. The Neo4j graph data model used the property graph model, nodes and edges which had properties associated with them; this was used for storing the protein names and the confidence scores associated with the interactions. In PostgreSQL, we stored the graph as an indexed table of node pairs, which can be traversed with either logarithmic or constant look up complexity depending on the type of index used.

On these databases we set three benchmarking tasks for cypher query vs sql query.

These three bioinformatics graph processing problems were:

* finding immediate neighbors and their interactions,
* finding the best scoring path between two proteins
* finding the shortest path between them.

The researchers found that the fact that a certain dataset is a graph, however, does not necessarily imply that a graph database is the best choice; it depends on the exact types of queries that need to be performed. Graph queries formulated in terms of paths can be concise and intuitive compared with equivalent SQL queries complicated by joins. Nevertheless, declarative graph query languages leave much to be desired, both feature-wise and performance-wise. Relational databases are a better choice when set operations are needed.

References

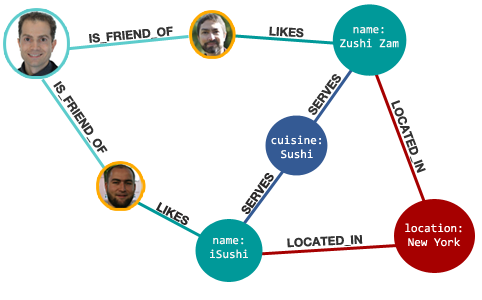
retrieved from <http://bioinformatics.oxfordjournals.org/content/early/2013/10/21/bioinformatics.btt549/F1.expansion.html>

retrieved from http://string905.embl.de/

Suitable use cases for the Neo4j graph database are as follows:

1. Connected Data- we can find graph databases useful in social network analysis where we can analyze nodes related by friend relationships and also nodes related by employee, expertise and work project relationships. Any link rich domain is well suited for graph databases. We can also take advantage of the ability to traverse across domains if we have domain entities and relationships between social, spatial and commerce domains.

2. Routing, Dispatch and Location Based services- we can find graph databases useful in delivery scenarios. Each node can be modeled as a address for delivery and the relationship between nodes can have properties like distance, roadblocks, rush hour times. We can alternately also have location based services like a restaurant rating website which can model nodes as clients and model relationships between customers as common cuisines they like.



**Discuss why relationships are so important to Neo4j.**

Relationships define the associations between nodes that are connected by them. We can have relationships that are unidirectional indicating that only one participant fosters the relationship to another or we can have bidirectional relationships to indicate that it is mutual. Having relationships between nodes gives the entities a way of achieving commonality, establishing order and expressing inheritance.

Relationships can have a type, a start node, an end node and properties of their own. Using properties in relationships we can add intelligence to the graph.

References:

Sadalage, P. J., & Fowler, M. (2013). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River, NJ: Pearson Education

**What are Nodes, Properties, Relationships, and Labels used for in Neo4J?**

Nodes are entities that have properties. Each node can have an edge which can be connected to another node' and form a relationship between the two nodes. Relationships can be unidirectional or bi directional. We can for example create the Martin node below and assign it properties such as name, age. We can then create relationships between the martin node and its friend nodes.

Node martin = graphDb.createNode();

martin.setProperty("name", "Martin");

Node pramod = graphDB.createNode();

pramod.setProperty("name", "Pramod");

martin.createRelationshipTo(pramod, FRIEND);

pramod.createRelationshiptTo(martin, FRIEND);

References

Sadalage, P. J., & Fowler, M. (2013). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River, NJ: Pearson Education

I only took upto Linear Algebra in undergrad so Graph theory is interesting area to explore. I found that there is a connection between the two in a concept known as adjacency matrixes.

 In graph theory, an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

I found a project that allows us to see the query that is developed for looking at the adjacency matrix for meetup groups. The detailed explanation for each query element use case is given on the referred source. Basically the person computed an array with the overlapping numbers given for each group as successive array elements. So for Group 1, it would be [0 x1, x2, x3... xn] where 0 is the Group #.

The use case is to create a social recommendation for a list of peer members that are in same groups.

|  |  |
| --- | --- |
| MATCH (g:Group)  WITH g  ORDER BY g.name  LIMIT 5    WITH COLLECT(g) AS groups  UNWIND groups AS g1  UNWIND groups AS g2  OPTIONAL MATCH path = (g1)<-[:MEMBER\_OF]-()-[:MEMBER\_OF]->(g2)    WITH g1, g2, CASE WHEN path is null THEN 0 ELSE COUNT(path) END AS overlap  ORDER BY g1.name, g2.name  RETURN g1.name, COLLECT(overlap)  ORDER BY g1.name | |
| +----------------------------------------------------------------------+  **|** g1.name **|** COLLECT**(**overlap**)** **|**  +----------------------------------------------------------------------+  **|** "Big Data / Data Science / Data Analytics Jobs" **|** **[**0,17,20,37,16**]** **|**  **|** "Big Data Developers in London" **|** **[**17,0,48,231,67**]** **|**  **|** "Big Data Jobs in London" **|** **[**20,48,0,189,70**]** **|**  **|** "Big Data London" **|** **[**37,231,189,0,417**]** **|**  **|** "Cassandra London" **|** **[**16,67,70,417,0**]** **|** |

References:

retrieved from http://www.markhneedham.com/blog/2014/05/20/neo4j-2-0-creating-adjacency-matrices/

**What are the differences between Pig and Python?  Which language do you prefer?**

Apache Pig provides a compiler that produces sequences of map-reduce programs. Pigs language layer consists of Pig Latin which is extensible, optimizable and easy to use. Pig operations are designed to transform data at one shot.  Pig Latin allows users to load data in the form of HDFS files. Once data is loaded it can be transformed using set transformations.  The transformed data can be dumped to a file or to the screen.  Pig Latin can be extended by UDF which can be written in Java, Python or JavaScript.

For example, using Apache Pig program we can stream in a text file from a Amazon s3 bucket. We can define a tokenizer for each line and group our tokenized words by similarity. Once the words are grouped we can count the member words in each group to conduct textual analytics.

**RAWINPUT = load '$INPUT' USING TextLoader as (RAWLINE:chararray); // loads an input text file from s3**

**RAWWORDS = foreach RAWINPUT generate flatten(TOKENIZE(RAWLINE)) as RAWWORD//tokenizes line words**

**grpd = group RAWWORDS by RAWWORD;//groups tokenized words.**

**cntd = foreach** **grpd generate group, COUNT(RAWWORDS);//counts group members**

**STORE cntd into '$OUTPUT';//stores count results in output file.**

Python is similarly an easy to use programming language that allows us to define logical operators and invoke custom methods. Python allows us to define programs that can also compute map reduce jobs. We can define a mapper.py and a reducer.py program similarly and generate textual analytics.

mapper.py{  
for line in sys.stdin:

# remove leading and trailing whitespace

line = line.strip()

# split the line into words

words = line.split()

# increase counters

for word in words:

# write the results to STDOUT (standard output);

# what we output here will be the input for the

# Reduce step, i.e. the input for reducer.py

#

# tab-delimited; the trivial word count is 1

print '%s\t%s' % (word, 1)

}  
  
reducer.py{  
from operator import itemgetter

import sys

current\_word = None

current\_count = 0

word = None

# input comes from STDIN

for line in sys.stdin:

# remove leading and trailing whitespace

line = line.strip()

# parse the input we got from mapper.py

word, count = line.split('\t', 1)

# convert count (currently a string) to int

try:

count = int(count)

except ValueError:

# count was not a number, so silently

# ignore/discard this line

continue

# this IF-switch only works because Hadoop sorts map output

# by key (here: word) before it is passed to the reducer

if current\_word == word:

current\_count += count

else:

if current\_word:

# write result to STDOUT

print '%s\t%s' % (current\_word, current\_count)

current\_count = count

current\_word = word

# do not forget to output the last word if needed!

if current\_word == word:

print '%s\t%s' % (current\_word, current\_count)  
}

References:

retrieved from <http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/>

retrieved from <http://hortonworks.com/hadoop-tutorial/how-to-use-basic-pig-commands/#what-is-pig>

retrieved from <https://pig.apache.org/>

**What is the value proposition of Cloud computing?  What are your opinions about the AWS Cloud computing environment?**

Amazon Web Services takes advantage of elastic compute resources that are available on demand to organizations throughout the world.  For AWS implementers investing in cloud computing makes it possible to efficiently allocate and save money that would be traditionally put in infrastructure maintenance tasks such as procuring new hardware, implementing software patches and investing in labor to upkeep servers.

AWS takes advantage of secure and atomic cloud computing offered via Identity and Access Management Services which allow you to provision unique keys and users for your compute resources. By enabling the delivery of compute(EC2), store(S3/Glacier) and database (RDS) services over a strong network connection AWS caters to application, database and storage requirements.

By offering Infrastructure as a service AWS makes it possible for entrepreneurs to allocate finances efficiently towards elastic computing requirements.  On demand computing tends to the flow and ebb of compute requirements accordingly as AWS allows you to auto scale and load balance your compute resources. AWS billing follows the utility model where you pay only for services that you use.

References:

retrieved from AWS Essentials Student Guide 2.5