Bases de dades NOSQL.

Distributed architectures - consistent hashing

index: associa clau amb direcció física

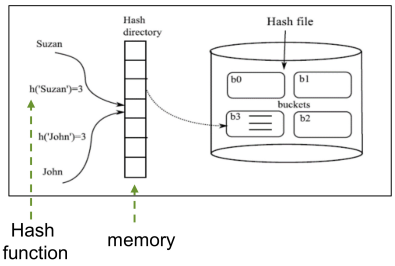
↳ . Tree ⇝ complexitat de busca logarítmica

↳ ' hash tables ⇝ complexitat de busca constant

✓ Consultes puntuals

✗ Queries de rang

✗ Dynamic connections (canvien ràpidament)



Què passa si les dades creixen molt?

→ Distribució de hash.

- Distributed hashing challenges
  - Dynamicity: grow and shrink rapidly
    - Distribution: Assign buckets to participating nodes

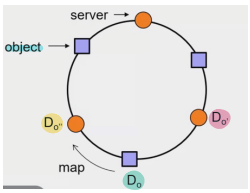
    *all the nodes should share the hash function for it to work

    E.g., h(x) = x % #servers

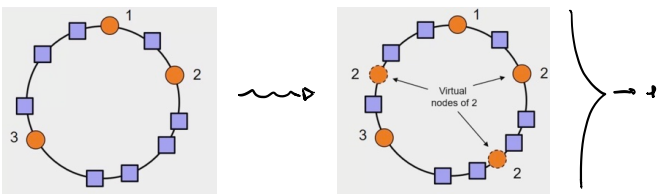
    Adding a new server implies modifying h...
    - Re-hashing all the objects
    - Communicating the new *h'* to all servers
  - Location of the hash directory: any access must go through the hash directory
    - Potential bottleneck

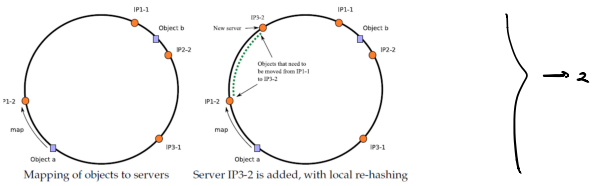
→ Consistent hashing

- Results of the most common queries are in *caches* (i.e., in-memory) of several servers
- A dedicated proxy machine records which server stores which query results
  - Queries are assigned to servers according to a hash function over the query
- Copying with dynamicity:
  - The hash function **never** changes
    - Choose a very large domain *D* (address space) and map server IP addresses and object *keys* to such domain
    - Organize *D* as a ring in clockwise order so each node has a successor
    - Objects are assigned as follows:
      - For an object O, f(O) = D_o
      - Let $D_{o'}$ and $D_{o''}$ be the two nodes in the ring such that
        - $D_{o'} < D_o <= D_{o''}$
      - O is assigned to $D_{o''}$
  - Adding a new server is straightforward
    - It is placed in the ring and part of its successors objects transferred



Exemples





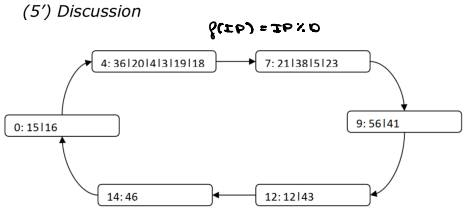
- Further refinements:
  1. Assign to the same server several hash values (virtual servers) to balance load
  2. Lazy update of the hash directory

Activity - consistent hashing

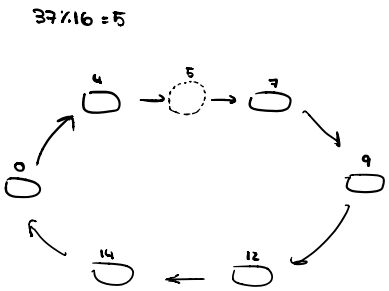
Objective: Understand how the consistent hash works

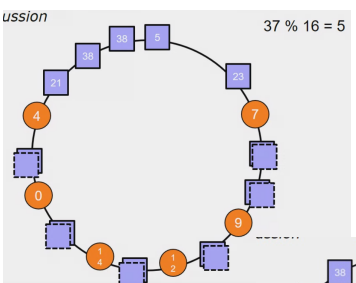
Tasks:

1. *(5') By pairs, solve the following exercise*
   - What happens in the structure when we register a new server with IP address "37"? Draw the result.
2. *(5') Discussion*

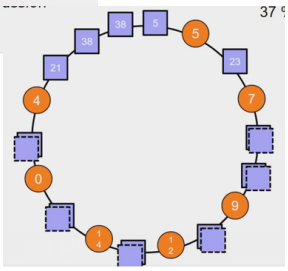


Current state of the consistent hash (D=16)





per mòduls

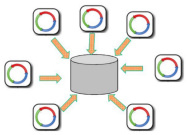


KEY-VALUE ENHANCEMENTS
DOCUMENT-ORIENTED DBS

Applications Databases.

- SQL and relational databases played a key role as **integration mechanism between applications**
  - Multiple applications using a common integrated database

- More complex
- Changes by different apps need to be coordinated
- Different apps have different performance needs, thus call for different index structures
- Complex access policies



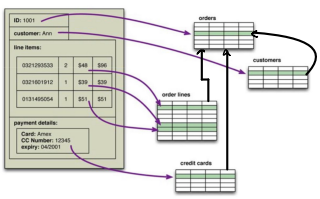
A different approach, treat your database as an **application database**

- An application database is only directly accessed by a single application, which makes it much **easier to maintain and evolve**
- Interoperability concerns can now shift to the interfaces of the application:
  - During the 2000s we saw a shift to web services, where applications would communicate over HTTP
- With a service you are able to use **richer data structures** (compared to SQL)
  - Usually represented as documents in XML or, more recently JSON

Aggregate data models

- The relational model divides the information that we want to store into **tuples** (rows): this is a very simple structure for data
- **Aggregate orientation** takes a different approach. It recognizes that often you want to operate on data in **units that have a more complex structure**
  - Think of it as a complex record that allows lists and other record structure to be nested inside
- Key-value, document and column-family DBs can all be seen as aggregate-oriented databases
  - They differ in how they structure the aggregate and consequently how they allow for it to be accessed

- What is good about these models?
  - Dealing with aggregates makes it much easier for the databases to handle **operating on a cluster**, since the aggregate makes a natural unit for replication and sharding.
    - Also a natural unit to use for distribution
  - Also, it may help **solving the impedance mismatch problem**, i.e., the difference between the relational model and the in-memory data structures
    - The impedance mismatch is naturally solved in document-stores



An order, which looks like a single aggregate

Structuring "value"

- Essentially, they are key-value stores
  - Same design and architectural features
- The value is a document
  - XML (e.g., eXist)
  - JSON (e.g., MongoDB and CouchDB)
- Tightly related to the Web
  - Normally, they provide RESTful HTTP APIs
- So… what is the benefit of having documents?
  - ✓ New data model (collections and documents)
    - **New atom: from rows to documents**
  - ✓ Indexing

Types of document-stores

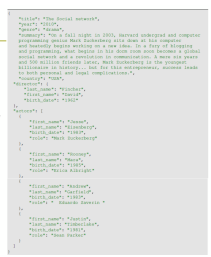
- JSON-like documents
  - MongoDB
  - CouchDB
- JSON is a lightweight data interchange format
  - Brackets ([]) represent ordered lists
  - Curly braces ({}) represent key-value dictionaries
    - Keys must be strings, delimited by quotes (")
    - Values can be strings, numbers, booleans, lists, or key-value dictionaries
- Natively compatible with JavaScript
  - Web browsers are natural clients for MongoDB / CouchDB

  http://www.json.org/index.html

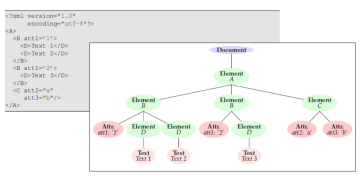
- XML-like documents
  - eXist, MarkLogic
    - Natively supported
  - Relational extensions for Oracle, PostgreSQL, etc.
    - Mapped to relational (impedance mismatch!)
- XML is a semistructured data model proposed as the standard for data exchange on the Web
  - Can be elegantly represented as a tree
    - Document: the root node of the XML document, denoted by "/"
    - Element: element nodes that correspond to the tagged nodes in the document
    - Attribute: attribute nodes attached to Element nodes
    - Text: text nodes, i.e., untagged leaves of the XML tree
- Support Xpath, Xquery and XSLT
  - Xpath is a language for addressing portions of an XML document
    - Subset of XQuery
  - XQuery is a query language for extracting information from collections of XML documents
  - XSLT is a language for specifying transformations (from XML to XML)

JSON Example

- Definition:
  A document is an object represented with an unbounded nesting of array and object constructs



XML Example



Labeled Unranked ordered tree.

MONGO DB