Database - P82

CouchDB

filter function : Changes API provide way to get notified of event in single stream
take document (and HTTP request)
make decision about whether document permitted through the filter
return value : Boolean
ex curl -u admin:admin -X PUT \
"${COUCH_ROOT_URL}/<db>/_design/<design_document>" \
-H "Content-Type: application/json" \
-d '{"language": "javascript",
"filter": { "by_country":
"function(doc,req){return doc.country === req.query.country;}"
}}'
compare the document's country
to the parameter passed into request's query string named "country"
curl -u admin:admin "${COUCH_ROOT_URL}/<db>/_changes \
?filter=<design_document>/by_country&country=RUS"
can contain arbitrary JavaScript : put sophisticated logic
regular expression for testing property
filter by date range
user-based filter using user context property on request object (req.userCtx)

replicating : CouchDB all about asynchronous environment and data durability
support multi-master (or master-master) replication data
each CouchDB server equally able to receive update / respond to request / delete
regardless of ability to connect to any other server
change selectively replicated in one direction
all data subject to replication in the same way
Update Seq value in replica may not match that of original
only insertion replicated to bring replica up to speed
update / deletion happened in original database not be replicated
resolving conflict : CouchDB pick one of updates and call it winner
all CouchDB nodes use same deterministic algorithm
"losing" document stored, client could review later
- conflicts field when conflicts=true URL parameter

Database - P83

CouchDB

replicating : CouchDB not try to intelligently merge conflicting changes
how conflicts are resolved is highly application-specific function
generalized solution not make sense

strength : robust and stable NoSQL database
built on philosophy : network unreliable, hardware failure happen at any time
provide decentralized approach to data storage
small enough to be run on smartphone
big enough to support enterprise application
very flexible database afford variety of deployment situation

weakness : not well suited for everything
mapreduce based view, while innovative
can not provide all of flexibility expect from relational database
work best when have good sense of what going to need in advance
know all of queries in advance, so that can create view

great at putting up with harsh realities of Internet-based application
feel like natural choice whenever web technology present
by leveraging common web concept : HTTP, REST, JSON
unique feature : ease of backup
binary attachment to document through CouchDB
CouchApps : system for developing/deploying application directly

Neo4j

member of family of database referred to as graph database
whiteboard friendly : virtually any diagram can draw using box and line
can be stored in Neo4j
focus more on relationship between values
rather than on commonality among set of values
store highly variable data in natural and straightforward way
small enough to be embedded into any application
can also run in large cluster of servers using master-slave replication
can handle about any size problem

Database - P84

Neo4j

saying in relational database world :
on a long enough timeline, all fields become optional
Neo4j handle implicitly by providing value/structure only where necessary
no schema to adjust in graph database
relationships are all that matter
two version : community (free) edition / enterprise edition

great graph data browser : bundled web administration tool and data browser
:server connect : to connect the data base

Cypher : Neo4j's graph query language
working with graph all about understanding pattern of data
rich, Neo4j-specific graph traversal language
other way : client library in multiple programming languages
REST API
other querying language : Gremlin
graph data point referred to node
graph in Cypher consist of node rather than vertex
connection between nodes called relationship rather than edge
Cypher 中的i语句基本结构为
$MATCH <pattern> WHERE <condition> RETURN <expression>
MATCH <pattern> : graph pattern to match
most common way to get data from graph
WHERE <condition> : part of MATCH/OPTIONAL MATCH/WITH
add constraint to pattern
filter intermediate result passing through WITH
RETURN <expression> : expression to return
eg MATCH (n) RETURN n; 等价于 SELECT * FROM <table>
access all nodes (and relationships) in the graph
$ CREATE/DELETE : create/delete node/relationship
$ SET/REMOVE : set/remove value/label to/from property/node
$ MERGE : match existing / create new node and pattern
especially useful together with unique constraint

Database - P85

Neo4j

Cypher <pattern> : 用于 MATCH / CREATE / MEAGE / pattern expression

(<n>) 表示 node , [<r>] 表示 relationship

可以在其中定义 label , 如 (<n>:<label>)

[<r>:<label>:<multi-label>]

可以在其中定义 property : 如 (<n>:{<field1>:<value1>, ⋯})

[<r>:{<field1>:<value1>, ⋯}]

通过 (a)-->(b) 表示 (a,b)∈E

(a)--(b) 表示 (a,b)∈E ∨ (b,a)∈E

即 点a与点b之间存在边，但无所谓方向

注意，等价于 (a)<-->(b)

(a)-->(b)-->(a) 表示 (a,b)∈E ∧ (b,a)∈E

通过 (a)-[r]->(b) 表示从a到b的边r

(a)-[*i..j]->(b) 表示长度从i到j的路径(path)

如 [*2] 表示长度为2的 path

[*2..5] 表示长度为 [2,5] 的 path

[*2..] 表示长度至少为2的 path

[*..5] 表示长度至多为5的 path

可以连续地使用 如 (a)-[r]->(b)-[s]->(c)

name can be omitted when unnecessary in Cypher query

特别注意 only directed relationship supported in CREATE

即不可使用 如 (a)--(b) 和 (a)<-->(b)

WHERE : 逻辑运算 : AND(与), OR(或), NOT(非), XOR(异或)

比较运算 : < (lt), <= (le), = (eq), <> (ne), >= (ge), > (gt)

筛选属性 : 如 n.age > 18 , r.length > 11

dynamically-computed node property

如 WITH 'AGE' as propname

MATCH (n) WHERE n[toLower(propname)] > 18

RETURN n

判断属性是否存在 : 如 exists(n.age)

字符串比较 : 提供 STARTS WITH (开始于), ENDS WITH (结束于)

CONTAINS (包含) 指定的字符串

使用 Java regular expression , 如 n.name =~ 'regex'

Database - P86

Neo4j

WHERE：Neo4j支持直接在WHERE中使用<pattern>作为筛选条件
pattern is expression in Cypher, return list of path
empty list represent false, non-empty represent true
但是注意 pattern 在MATCH和WHERE中的行为不同
对于 MATCH (a)-[*]->(b)
produce subgraph for every path find between (a) and (b)
而对于 WHERE (a)-[*]->(b)
eliminate subgraph without directed relationship chain between (a) and (b)
IN 运算符：用于判断给定元素是否存在于列表中
如 MATCH (a) WHERE a.name IN ['abc', 'XYZ']
RETURN a
对于 missing property，Neo4j 默认为 false
即 missing property evaluate to null
comparison evaluate to false for node without specific property
IS NULL / IS NOT NULL 运算符：与SQL类似
用于判断一个值或一个变量是否为 null

SET：在Neo4j中用于UPDATE已经存在的 node 和 relationship
基本形式为 MATCH <pattern>，如 (a)-[r]->(b)
SET a.name = NULL
RETURN a
也可以将一个 node/relationship 的 property 复制并覆盖另一个 node/relationship
如 SET a = b
也可以向已经存在的结点添加新属性
如 SET a += {<field1>: <value1>, ...}
也可以向已经存在的结点添加新的 label
如 SET a[:<label>]+
结点的 label 可以通过 labels(n) 函数查询
另外设置 label 是幂等的(idempotent)，即添加已存在的标签不产生效果

REMOVE：在Neo4j中用于UPDATE已经存在的 node / relationship
基本形式为 MATCH (a)-[r]->(b)
REMOVE a.name，移除属性 a.name
(idempotent) REMOVE a[:<label>]*，移除 a 的 label

Database - P87

Neo4j

相比于关系型数据库将 entity 和 relation 都实现为 table

Neo4j represent entity as node in the graph

create relationship between nodes

	Cypher clause	description
reading	MATCH	specify pattern to search for in the database
	OPTIONAL MATCH	specify pattern to search for in the database while using null for missing part of pattern
projecting	RETURN .. [AS]	define what to include in the query result set
	WITH .. [AS]	allow query part to be chained together in the next piping result from one to be used as starting point/criteria
	UNWIND .. [AS]	expand list into sequence of rows
reading sub-clause	WHERE	add constraint to pattern in MATCH / OPTIONAL MATCH clause or filter result of WITH clause
	ORDER BY	[ASC[ENDING] | DESC[ENDING]] sub-clause following RETURN / WITH specify output should be sorted is ascending (default) / descending order
	SKIP	define from which row to start including rows in the output
	LIMIT	constraint the number of rows in the output
reading hint	USING INDEX	index hint used to specify which index, if any, the planner should use as starting point
	USING INDEX SEEK	index seek hint instruct planner to use index seek for this clause
	USING SCAN	scan hint used to force planner to do label scan followed by filtering operation instead of using index
	USING JOIN	join hint used to enforce join operation at specific point
writing	CREATE	create node and relationship
	DELETE	delete graph element : node / relationship / path node deleted must have all associated relationship explicitly deleted
	DETACH DELETE	delete node, all associated relationship automatically deleted
	SET	update label / property on node / relationship
	REMOVE	remove label / property from node / relationship
	FOREACH	updata data within list, whether component of path or result of aggregation

Database - P88

Neo4j	Cypher clause	description
reading/writing clause	MERGE	ensure pattern exist in graph, either already exist, or need to be created
	└ ON CREATE	write sub-clause used in conjunction with MERGE specify action if pattern need to be created
	└ ON MATCH	write sub-clause used in conjunction with MERGE specify action if pattern already exist
	CALL [.. YIELD]	invoke procedure deployed in database and return any result
set operation	UNION	combine result of multiple queries, duplicate removed
	UNION ALL	combine result of multiple queries, duplicate retained
importing data	LOAD CSV	import data from CSV file
	└ USING PERIODIC COMMIT	query hint used to prevent out-of-memory error when importing large amount of data using LOAD CSV
schema	CREATE | DROP	CONSTRAINT create/drop index on all nodes with particular label and property
	CREATE | DROP	INDEX create/drop constraint pertaining to node label / relationship type / property

constraint cannot be applied to label already has index
index created automatically if create constraint on specific label/property
: schema 命令用于返回当前 database 中所有 INDEX / CONSTRAINT
Neo4j not fundamentally schema-driven as relational database

RESTful interface : like HBase, MongoDB, CouchDB
allow language-agnostic interaction via standard interface
response from request : node path in header
body of metadata about the node
all data retrievable via GET
on given header Location value
or self property in metadata
GET appending /properties : fetch node properties
further appending property name :
to fetch individual property

Database - P89

Neo4j

RESTful access: POST request data to starting node [redacted]/paths URL
find path between two nodes
request must be JSON string denoting node path to
type of relationship to follow
path-finding algorithm to use
shortestPath / allPaths / allSimplePaths / dijkstra

Indexing: different path when accessing data via REST
indexing mechanism actually separate service
[redacted] simplest kind of index: key-value / hash style
full-text: full-text search inverted index
building require entire dataset
can also be built on relationship

Cypher plugin: send Cypher console command via REST

```
$o $ curl -X POST \
    http://localhost:7474/db/data/cypher \
    -H "Content-Type: application/json" \
    -d '{"query": "MATCH ()-[r]-() RETURN r;"}'
```

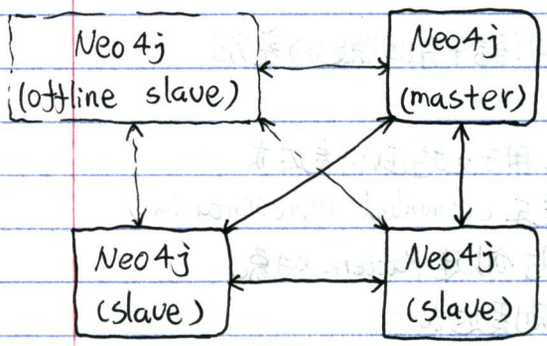
shortestPath: Cypher function to easily calculate distance between nodes

distributed high availability: while Neo4j can scale
write to one slave not immediately synchronized with all other slaves
danger of losing consistency for period of time while change replicating
lose pure ACID-compliant tranction
Neo4j HA recommanded mostly as method
for increasing read capacity of cluster
servers in cluster will elect master like MongoDB
hold primary responsibility for managing data distribution in cluster
unlike MongoDB, slave node in Neo4j accept write
slave writes will synchronize to master node, which propagate them out

Database - P90

Neo4j

distributed high availability : cluster can choose own master / slave setup
re-coordinate when server go offline



master election happen automatically
when master server go offline
other will elect leader amongst themselves
if old master start again and rejoin cluster
as slave instead of master
allow very read-heavy system
to deal with replicating graph across multiple servers
thereby sharing the load

eventually consistent : cluster as a whole
configuration setting to reduce chance of stale read
pegging session to one server

able to support graph with billions of nodes / edges
and nearly any number of request

backups : necessary aspect of any professional database
although effectively built-in with highly available cluster recovery
periodic backups stored off-side always prudent choice for disaster
neo4j-admin : tool of Neo4j Enterprise
perform wide variety of action, including backups
most powerful backup method when running HA server
to craft full backup command to copy database file
from cluster to date-stamped file on mounted drive
backup directory created by full backup
standalone usable copy of database
replace the installation data directory with backup directory
all have to do when need to recover database
incremental backup : "delta backup" choosen once full backup created
only back up change occured since last full backup

Database - P91

Neo4j

Strength : one of best examples of open source graph database
graph database perfect for unstructured data
may even more than document database
typeless / schemaless / no constraint on how data related
support fast lookup using Lucene
robust query language in Cypher
provide REST interface
can support maximum of 34.4 billion nodes and 34.4 billion relationships
graph traversal is constant time
moving from node A to node B always one step operation
if they share one relationship

weakness: nomenclature add complexity to communicating about Neo4j
node instead of vertex / relationship instead of edge
HA great at replicating full graph to other server
cannot currently shard subgraph, limit size of graph
feature needed in production deployment
not free and require using Enterprise tool

CAP : strategy can be summed up by term "high availability cluster"
available and partition-tolerant (AP) : Neo4j HA
each slave return only what currently has
may be out of sync with master node temporarily
although can reduce update latency by increasing slave pull interval
still eventually consistent system
Neo4j HA recommended for "read-mostly" requirement

Redis

(REmote DIctionary Service), simple-to-use key-value store
contain sophisticated set of command
hard to beat in term of speed
can handle upwards of 100,000 SET operations per second
refer to Redis as data structure server
due to nuanced handling of complex datatype, and other feature

Database - P92

Redis

basic level is key-value store, difficult to classify Redis
support advanced data structure, though not as sophisticated as document database
support set-based query operation
    but not with granularity or type support as relational database
trade durability for raw speed
in addition to being advanced data structure server
    also used as blocking queue (stack)
        as publish-subscribe system
configurable expiration policy / durability level / replication option
expansive list of client library, make Redis drop-in option for programming languages

command-line interface (CLI):
    important to user of Redis and to Redis development team

SET <key> <value> [expiration EX <seconds> | PX <milliseconds>] [NX | XX]
    set string <value> of a <key>
        EX <seconds> : set specified expire time in seconds
        PX <milliseconds> : set specified expire time in milliseconds
        NX : only set the key if not already exist
        XX : only set the key if already exist
    等价于 SETEX <key> <seconds> <value>
        PSETEX <key> <milliseconds> <value>
        SETNX <key> <value>

GET <key> : get <value> of <key>, only handle string
    return special value nil if key not exist
    return error if value stored at key not string

MSET [<key> <value>]+ : set multiple keys to multiple values
MSETNX [<key> <value>]+ : set multiple values only if none of key exist
    atomic command, all given keys set at once

MGET [<key>]+ : get values of all given key, never fail
    return nil for every key not hold string value / not exist