Cassandra

A few very practical examples

Create and verify key spaces

AT NODE #0:

```
CREATE KEYSPACE one_replicated
WITH REPLICATION = {'class' : 'SimpleStrategy',
                    'replication_factor' : 1};
CREATE KEYSPACE two_replicated
WITH REPLICATION = {'class' : 'SimpleStrategy',
                    'replication_factor' : 2};


SELECT * FROM system.schema_keyspaces;
```

Prepare first table

AT NODE #0:

```
CREATE TABLE one_replicated.users (
    id int,
    name text,
    surname text,
    PRIMARY KEY(id, surname))
WITH CLUSTERING ORDER BY (surname ASC)
AND read_repair_chance = 0.0;

INSERT INTO one_replicated.users(id, name, surname)
VALUES (0, 'Jan', 'Kowalski');
```

Where does Kowalski live?

AT NODE #0 WITH #1 AND THEN #2 DISABLED; THEN AT #1 WITH @0 DISABLED:

```
CONSISTENCY ONE;

SELECT * FROM users;

SELECT * FROM users WHERE id = 0;
```

Where does Kowalski live?

AT NODE #0 WITH #1 AND THEN #2 DISABLED; THEN AT #1 WITH @0 DISABLED:

CONSISTENCY ONE;

SELECT * FROM users; #will fail

SELECT * FROM users WHERE id = 0; #may fail
(too strong CL or RPC timeout – FD not fast enough)

Kowalski has a neighbour!

AT NODE #0

```
INSERT INTO one_replicated.users(id, name, surname)
VALUES (1, 'Tomasz', 'Iksiński');


SELECT * FROM users WHERE id = 0
ORDER BY surname DESC;
```

Kowalski has a neighbour!

AT NODE #0

```
INSERT INTO one_replicated.users(id, name, surname)
VALUES (1, 'Tomasz', 'Iksiński');

SELECT * FROM users ORDER BY surname DESC;
#why can’t this just work?!
```

Kowalski has a neighbour!

AT NODE #0

```
INSERT INTO one_replicated.users(id, name, surname)
VALUES (1, 'Tomasz', 'Iksiński');


SELECT * FROM users WHERE id = 0 ORDER BY surname
DESC; #it’s OK now
```

Replication

For now Replication Factor (RF) = 2

Let’s have some strings

AT NODE #0

```
USE two_replicated;


CREATE TABLE strings (
    string text PRIMARY KEY)
WITH read_repair_chance = 0.1;


CONSISTENCY one;
```

Let’s have some strings

AT NODE #0

```
INSERT INTO strings(string) VALUES ('test1');
INSERT INTO strings(string) VALUES ('test2');
INSERT INTO strings(string) VALUES ('test3');
INSERT INTO strings(string) VALUES ('test4');
INSERT INTO strings(string) VALUES ('test5');
```

And where does ‘test5’ live?

AT NODE #0

SELECT token(string) FROM strings

WHERE string = 'test5';

$ nodetool getendopints two_replicated strings ‘test5’

Consistency

A bit more complicated stuff

A map now

AT NODE #0

```
USE two_replicated;

CREATE TABLE map (
    key int PRIMARY KEY,
    value text)
WITH read_repair_chance = 0.0; #sorry

CONSISTENCY ONE;

INSERT INTO map (key, value) values (0, 'version1'); #and
    where exactly this entry lives?
```

We’re updating our map

AT NODE #2 AFTER IT ACKNOWLEDGED FAILURE OF NODE #0

```
USE two_replicated;

CONISITENCY ONE;

UPDATE map SET value = 'version2' WHERE key = 0;

SELECT value FROM map WHERE key = 0;
```

Failure scenarios

AT NODE #2 AFTER IT ACKNOWLEDGED RETURN OF NODE #0 AND FAILURE OF #1

SELECT value FROM map WHERE key = 0; #now only NODE0 should be queried

#is the result correct?!

How about disabling hinted-handoff?

…by editing config file and restarting the service

Soft reset

AT NODE #2 WHEN ALL THREE NODES ARE UP & RUNNING

```
CONSISTENCY ALL;

UPDATE map SET value = 'version1' WHERE key = 0;

SELECT value FROM map WHERE key = 0;

CONSISTENCY ONE;
```

Soft reset

AT #0, #1 AND #2

```
SELECT value FROM map WHERE key = 0;# should
    produce the same result everywhere
```

We’re updating our map

AT NODE #2 AFTER IT ACKNOWLEDGED FAILURE OF NODE #0 (#1 IS UP)

```
UPDATE map SET value = 'version2' WHERE key = 0;

SELECT value FROM map WHERE key = 0;# result should
    be up to date
```

Failure scenarios

AT NODE #2 AFTER IT ACKNOWLEDGED RETURN OF NODE #0 AND FAILURE OF #1

SELECT value FROM map WHERE key = 0; #now only NODE0 should be queried

#the result is obsolete – that's our goal!

Now, how to repair this?!

AT NODE #2 AFTER IT ACKNOWLEDGED RETURN OF NODE #1 (#0 IS STILL UP)

```
CONSISTENCY QUORUM;

SELECT value FROM map WHERE key = 0; #produces an
    error as long as #1 is not UP

ALTER TABLE map WITH read_repair_chance = 1.0;

SELECT value FROM map WHERE key = 0; #a few times;
    #0 should be finally updated
```

Let’s check if NODE0 has been repaired

AT NODE #2 AFTER IT ACKNOWLEDGED FAILURE OF NODE #1

```
CONSISTENCY ONE;

SELECT value FROM map WHERE key = 0; #finally the
    correct result!
```

Thank you

http://sirius.cs.put.poznan.pl/~inf89719/cassandra/