IF4031 Pengembangan Aplikasi Terdistribusi

EKSPLORASI PEMROGRAMAN RIAK

oleh

Erick Wijaya / 13515057 / K1

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2018

Daftar Isi

A. Output *member status* cluster Riak

Perintah: `sudo riak-admin member-status`

1. Output *member status* dari node `167.205.35.19`, node `167.205.35.20`, node `167.205.35.21`, dan node `167.205.35.22`

```
================================= Membership ==================================
Status     Ring    Pending    Node
-------------------------------------------------------------------------------
valid      25.0%      --      'riak@167.205.35.19'
valid      25.0%      --      'riak@167.205.35.20'
valid      25.0%      --      'riak@167.205.35.21'
valid      25.0%      --      'riak@167.205.35.22'
-------------------------------------------------------------------------------
Valid:4 / Leaving:0 / Exiting:0 / Joining:0 / Down:0
```

B. Output status setiap node (*ring members* dan *connected nodes*)

URL: `http://<ip_address>:8098/stats`

1. Output dari `http://167.205.35.19:8098/stats`

```
"connected_nodes":["riak@167.205.35.21","riak@167.205.35.20","riak@167.2
05.35.22"]
"ring_members":["riak@167.205.35.19","riak@167.205.35.20","riak@167.205.
35.21","riak@167.205.35.22"]
```

2. Output dari `http://167.205.35.20:8098/stats`

```
"connected_nodes":["riak@167.205.35.22","riak@167.205.35.21","riak@167.2
05.35.19"]
"ring_members":["riak@167.205.35.19","riak@167.205.35.20","riak@167.205.
35.21","riak@167.205.35.22"]
```

3. Output dari `http://167.205.35.21:8098/stats`

```
"connected_nodes":["riak@167.205.35.22","riak@167.205.35.20","riak@167.2
05.35.19"]
"ring_members":["riak@167.205.35.19","riak@167.205.35.20","riak@167.205.
35.21","riak@167.205.35.22"]
```

4. Output dari `http://167.205.35.22:8098/stats`

```
"connected_nodes":["riak@167.205.35.21","riak@167.205.35.20","riak@167.2
05.35.19"]
"ring_members":["riak@167.205.35.19","riak@167.205.35.20","riak@167.205.
35.21","riak@167.205.35.22"]
```

C. Output status setelah perintah riak stop

Perintah: `sudo riak stop` (dilakukan pada node `167.205.35.19`)

URL: `http://<ip_address>:8098/stats`

1. Output dari `http://167.205.35.20:8098/stats`
`"connected_nodes":["riak@167.205.35.22","riak@167.205.35.21"]`
`"ring_members":["riak@167.205.35.19","riak@167.205.35.20","riak@167.205.35.21","riak@167.205.35.22"]`

2. Output dari `http://167.205.35.21:8098/stats`
`"connected_nodes":["riak@167.205.35.22","riak@167.205.35.20"]`
`"ring_members":["riak@167.205.35.19","riak@167.205.35.20","riak@167.205.35.21","riak@167.205.35.22"]`

3. Output dari `http://167.205.35.22:8098/stats`
`"connected_nodes":["riak@167.205.35.21","riak@167.205.35.20"]`
`"ring_members":["riak@167.205.35.19","riak@167.205.35.20","riak@167.205.35.21","riak@167.205.35.22"]`

D. Output percobaan dengan perintah curl

No	1
Deskripsi	Cek koneksi ke node `167.205.35.19`, node `167.205.35.20`, node `167.205.35.21`, dan node `167.205.35.22`
Perintah	`curl http://<node_ip_address>:8098/ping`
Output	OK

No	2
Deskripsi	Penyimpanan data key-value pada Riak

Perintah	`curl -v -X PUT http://167.205.35.19:8098/riak/wijayaerick/mykey \`
	`    -H "Content-type: text/html" \`
	`    -d "<html><body><h1>My Data for key: mykey</h1></body></html>"`

Output	`*   Trying 167.205.35.19...`
	`* TCP_NODELAY set`
	`* Connected to 167.205.35.19 (167.205.35.19) port 8098 (#0)`
	`> POST /riak/wijayaerick/mykey HTTP/1.1`
	`> Host: 167.205.35.19:8098`
	`> User-Agent: curl/7.58.0`
	`> Accept: */*`
	`> Content-type: text/html`
	`> Content-Length: 57`
	`>`
	`* upload completely sent off: 57 out of 57 bytes`
	`< HTTP/1.1 204 No Content`
	`< Vary: Accept-Encoding`
	`< Server: MochiWeb/1.1 WebMachine/1.10.9 (cafe not found)`

```
< Date: Wed, 10 Oct 2018 06:44:34 GMT
< Content-Type: text/html
< Content-Length: 0
<
* Connection #0 to host 167.205.35.19 left intact
```

No	3
Deskripsi	Akses data (dari browser)
URL 1	`http://167.205.35.19:8098/riak/wijayaerick/mykey`
Output 1	`<html><body><h1>My Data for key: mykey</h1></body></html>`
URL 2	`http://167.205.35.19:8098/riak/wijayaerick/key2`
Output 2	`<html><body><h1>My Data for key: key2</h1></body></html>`

No	4
Deskripsi	PUT data baru

Perintah

```
curl -v -X PUT http://167.205.35.19:8098/riak/animals/ace \
    -H "Content-Type: application/json" \
    -d '{"nickname" : "The Wonder Dog", "breed" : "German Shepherd"}'
```

Output

```
*   Trying 167.205.35.19...
* TCP_NODELAY set
* Connected to 167.205.35.19 (167.205.35.19) port 8098 (#0)
> PUT /riak/animals/ace HTTP/1.1
> Host: 167.205.35.19:8098
> User-Agent: curl/7.58.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 60
>
* upload completely sent off: 60 out of 60 bytes
< HTTP/1.1 204 No Content
< Vary: Accept-Encoding
< Server: MochiWeb/1.1 WebMachine/1.10.9 (cafe not found)
< Date: Wed, 10 Oct 2018 06:48:07 GMT
< Content-Type: application/json
< Content-Length: 0
<
* Connection #0 to host 167.205.35.19 left intact
```

No	5
Deskripsi	Melihat hasil (GET seluruh bucket)
Perintah	`curl -X GET http://<node_ip_address>:8098/riak?buckets=true`

Output

```
{"buckets":["girvandi","bucketA","wijayaerick","test","testleo","Orders",
"majors","students","OrderSummaries","kristiantokarim","gua","animals",
"sashi","malik","test2","adonankue","jonathanchristopher","Customers",
"kharis","w_equals_0","michael","ade_surya","asdf","tifani","azzahid",
"pendisaurus","books"]}
```

No	6
Deskripsi	Menghapus data

Perintah

```
curl -v -X DELETE \
    http://167.205.35.19:8098/riak/wijayaerick/mykey
```

Output

```
*   Trying 167.205.35.19...
* TCP_NODELAY set
* Connected to 167.205.35.19 (167.205.35.19) port 8098 (#0)
> DELETE /riak/wijayaerick/mykey HTTP/1.1
> Host: 167.205.35.19:8098
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 204 No Content
< Vary: Accept-Encoding
< Server: MochiWeb/1.1 WebMachine/1.10.9 (cafe not found)
< Date: Wed, 10 Oct 2018 08:18:39 GMT
< Content-Type: text/html
< Content-Length: 0
<
* Connection #0 to host 167.205.35.19 left intact
```

No	7
Deskripsi	Menampilkan daftar key (sebelum dan sesudah DELETE)
Perintah	`curl http://167.205.35.19:8098/riak/wijayaerick?keys=true`
Output 1 (sebelum delete)	`"keys":["mykey","key2"]`
Output 2 (setelah delete)	`"keys":["key2"]`

E. Akses data pada Riak dengan Java Library

Kode program:

```java
/*
    Source: https://raw.githubusercontent.com/basho/basho_docs/master/extras/code-examples/TasteOfRiak.java
 */

import com.basho.riak.client.api.RiakClient;
import com.basho.riak.client.api.commands.kv.DeleteValue;
import com.basho.riak.client.api.commands.kv.FetchValue;
import com.basho.riak.client.api.commands.kv.StoreValue;
import com.basho.riak.client.api.commands.kv.UpdateValue;
import com.basho.riak.client.core.RiakCluster;
import com.basho.riak.client.core.RiakNode;
import com.basho.riak.client.core.query.Location;
import com.basho.riak.client.core.query.Namespace;
import com.basho.riak.client.core.query.RiakObject;
import com.basho.riak.client.core.util.BinaryValue;

import java.net.UnknownHostException;

public class TasteOfRiak {
    // A basic POJO class to demonstrate typed exchanges with Riak
    public static class Book {
        public String title;
        public String author;
        public String body;
        public String isbn;
        public Integer copiesOwned;
    }

    // This will allow us to update the book object handling the
    // entire fetch/modify/update cycle.
    public static class BookUpdate extends UpdateValue.Update<Book> {
        private final Book update;
    public BookUpdate(Book update){
        this.update = update;
    }

    @Override
    public Book apply(Book t) {
        if (t == null) {
            t = new Book();
        }

        t.author = update.author;
        t.body = update.body;
        t.copiesOwned = update.copiesOwned;
        t.isbn = update.isbn;
        t.title = update.title;

        return t;
    }
}

// This will create a client object that we can use to interact with Riak
private static RiakCluster setUpCluster() throws UnknownHostException {
    RiakNode node = new RiakNode.Builder()
            .withRemoteAddress("167.205.35.19")
            .withRemotePort(8087)
            .build();

    // This cluster object takes our one node as an argument
    RiakCluster cluster = new RiakCluster.Builder(node)
            .build();

    // The cluster must be started to work, otherwise you will see errors
    cluster.start();

    return cluster;
}

public static void main( String[] args ) {
    try {
        // First, we'll create a basic object storing a movie quote
        RiakObject quoteObject = new RiakObject()
                // We tell Riak that we're storing plaintext, not JSON, HTML, etc.
                .setContentType("text/plain")
                // Objects are ultimately stored as binaries
                .setValue(BinaryValue.create("You're dangerous, Maverick"));
        System.out.println("Basic object created");

        // In the new Java client, instead of buckets you interact with Namespace
        // objects, which consist of a bucket AND a bucket type; if you don't
        // supply a bucket type, "default" is used; the Namespace below will set
        // only a bucket, without supplying a bucket type
        Namespace quotesBucket = new Namespace("quotes");

        // With our Namespace object in hand, we can create a Location object,
        // which allows us to pass in a key as well
        Location quoteObjectLocation = new Location(quotesBucket, "Iceman");
        System.out.println("Location object created for quote object");

        // With our RiakObject in hand, we can create a StoreValue operation
        StoreValue storeOp = new StoreValue.Builder(quoteObject)
                .withLocation(quoteObjectLocation)
                .build();
        System.out.println("StoreValue operation created");

        // And now we can use our setUpCluster() function to create a cluster
        // object which we can then use to create a client object and then
        // execute our storage operation
        RiakCluster cluster = setUpCluster();
        RiakClient client = new RiakClient(cluster);
        System.out.println("Client object successfully created");

        StoreValue.Response storeOpResp = client.execute(storeOp);
        System.out.println("Object storage operation successfully completed");

        // Now we can verify that the object has been stored properly by
```

```java
            // creating and executing a FetchValue operation
            FetchValue fetchOp = new FetchValue.Builder(quoteObjectLocation)
                    .build();
            RiakObject fetchedObject = client.execute(fetchOp).getValue(RiakObject.class);
            assert(fetchedObject.getValue().equals(quoteObject.getValue()));
            System.out.println("Success! The object we created and the object we fetched have the same 
value");

            // Now update the fetched object
            fetchedObject.setValue(BinaryValue.create("You can be my wingman any time."));
            StoreValue updateOp = new StoreValue.Builder(fetchedObject)
                    .withLocation(quoteObjectLocation)
                    .build();
            StoreValue.Response updateOpResp = client.execute(updateOp);
            updateOpResp = client.execute(updateOp);

            // And we'll delete the object
            DeleteValue deleteOp = new DeleteValue.Builder(quoteObjectLocation)
                    .build();
            client.execute(deleteOp);
            System.out.println("Quote object successfully deleted");

            Book mobyDick = new Book();
            mobyDick.title = "Moby Dick";
            mobyDick.author = "Herman Melville";
            mobyDick.body = "Call me Ishmael. Some years ago...";
            mobyDick.isbn = "1111979723";
            mobyDick.copiesOwned = 3;
            System.out.println("Book object created");

            // Now we'll assign a Location for the book, create a StoreValue
            // operation, and store the book
            Namespace booksBucket = new Namespace("books");
            Location mobyDickLocation = new Location(booksBucket, "moby_dick");
            StoreValue storeBookOp = new StoreValue.Builder(mobyDick)
                    .withLocation(mobyDickLocation)
                    .build();
            client.execute(storeBookOp);
            System.out.println("Moby Dick information now stored in Riak");

            // And we'll verify that we can fetch the info about Moby Dick and
            // that that info will match the object we created initially
            FetchValue fetchMobyDickOp = new FetchValue.Builder(mobyDickLocation)
                    .build();
            Book fetchedBook = client.execute(fetchMobyDickOp).getValue(Book.class);
            System.out.println("Book object successfully fetched");

            assert(mobyDick.getClass() == fetchedBook.getClass());
            assert(mobyDick.title.equals(fetchedBook.title));
            assert(mobyDick.author.equals(fetchedBook.author));
            // And so on...

            // Now to update the book with additional copies
            mobyDick.copiesOwned = 5;
            BookUpdate updatedBook = new BookUpdate(mobyDick);
            UpdateValue updateValue = new UpdateValue.Builder(mobyDickLocation)
                    .withUpdate(updatedBook).build();
            UpdateValue.Response response = client.execute(updateValue);

            System.out.println("Success! All of our tests check out");

            // Now that we're all finished, we should shut our cluster object down
            cluster.shutdown();

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

```
Basic object created
Location object created for quote object
StoreValue operation created
Client object successfully created
Object storage operation successfully completed
Success! The object we created and the object we fetched have the same
value
Quote object successfully deleted
Book object created
Moby Dick information now stored in Riak
Book object successfully fetched
Success! All of our tests check out
```

F. Tutorial query Riak

Kode program:

```java
package SipOfRiak;

import com.basho.riak.client.IRiakClient;
import com.basho.riak.client.IRiakObject;
import com.basho.riak.client.RiakException;
import com.basho.riak.client.RiakFactory;
import com.basho.riak.client.bucket.Bucket;
import com.basho.riak.client.query.indexes.BinIndex;
import com.basho.riak.client.query.indexes.IntIndex;

import java.util.ArrayList;
import java.util.List;

public class SipOfRiak {

    public static void main(String[] args) throws RiakException {

        System.out.println("Creating Data");
        Customer customer = createCustomer();
        ArrayList<Order> orders = createOrders();
        OrderSummary orderSummary = createOrderSummary(orders);


        System.out.println("Starting Client");
        IRiakClient client = RiakFactory.pbcClient("167.205.35.19", 8087);


        System.out.println("Creating Buckets");
        Bucket customersBucket = client.fetchBucket("Customers").lazyLoadBucketProperties().execute();
        Bucket ordersBucket = client.fetchBucket("Orders").lazyLoadBucketProperties().execute();
        Bucket orderSummariesBucket =
client.fetchBucket("OrderSummaries").lazyLoadBucketProperties().execute();

        System.out.println("Storing Data");
        customersBucket.store(String.valueOf(customer.CustomerId), customer).execute();
        for (Order order : orders) {
            ordersBucket.store(String.valueOf(order.OrderId), order).execute();
        }
        orderSummariesBucket.store(String.valueOf(orderSummary.CustomerId), orderSummary).execute();


        System.out.println("Fetching related data by shared key");
        String key = "1";
        String fetchedCust = customersBucket.fetch(key).execute().getValueAsString();
        String fetchedOrdSum = orderSummariesBucket.fetch(key).execute().getValueAsString();
        System.out.format("Customer     1: %s\n", fetchedCust);
        System.out.format("OrderSummary 1: %s\n", fetchedOrdSum);


        System.out.println("Adding Index Data");
        IRiakObject riakObj = ordersBucket.fetch("1").execute();
        riakObj.addIndex("SalespersonId", 9000);
        riakObj.addIndex("OrderDate", "2013-10-01");
        ordersBucket.store(riakObj).execute();

        IRiakObject riakObj2 = ordersBucket.fetch("2").execute();
        riakObj2.addIndex("SalespersonId", 9001);
        riakObj2.addIndex("OrderDate", "2013-10-15");
        ordersBucket.store(riakObj2).execute();

        IRiakObject riakObj3 = ordersBucket.fetch("3").execute();
        riakObj3.addIndex("SalespersonId", 9000);
        riakObj3.addIndex("OrderDate", "2013-11-03");
        ordersBucket.store(riakObj3).execute();

        System.out.println("Index Queries");
        // Query for orders where the SalespersonId index is set to 9000
        List<String> janesOrders = ordersBucket.fetchIndex(IntIndex.named("SalespersonId"))
                                       .withValue(9000).execute();

        System.out.format("Jane's Orders: %s\n", StringUtil.Join(", ", janesOrders));


        // Query for orders where the OrderDate index is between 2013-10-01 and 2013-10-31
        List<String> octoberOrders = ordersBucket.fetchIndex(BinIndex.named("OrderDate"))
                                       .from("2013-10-01").to("2013-10-31").execute();

        System.out.format("October's Orders: %s\n", StringUtil.Join(", ", octoberOrders));

        // clean up test data
        customersBucket.delete("1").execute();
        ordersBucket.delete("1").execute();
        ordersBucket.delete("2").execute();
        ordersBucket.delete("3").execute();
        orderSummariesBucket.delete("1").execute();

        client.shutdown();
    }

    private static Customer createCustomer() {
        Customer customer = new Customer();
        customer.CustomerId = 1;
        customer.Name = "John Smith";
        customer.Address = "123 Main Street";
        customer.City = "Columbus";
        customer.State = "Ohio";
        customer.Zip = "43210";
        customer.Phone = "+1-614-555-5555";
        customer.CreatedDate = "2013-10-01 14:30:26";
        return customer;
    }

    private static ArrayList<Order> createOrders() {
        ArrayList<Order> orders = new ArrayList<Order>();

        Order order1 = new Order();
        order1.OrderId = 1;
        order1.CustomerId = 1;
        order1.SalespersonId = 9000;
        order1.Items.add(
                new Item("TCV37GIT4NJ",
                        "USB 3.0 Coffee Warmer",
                        15.99));
        order1.Items.add(
                new Item("PEG10BBF2PP",
                        "eTablet Pro; 24GB; Grey",
                        399.99));
        order1.Total = 415.98;
        order1.OrderDate = "2013-10-01 14:42:26";
        orders.add(order1);

        Order order2 = new Order();
        order2.OrderId = 2;
        order2.CustomerId = 1;
        order2.SalespersonId = 9001;
        order2.Items.add(
                new Item("OAX19XWN0QP",
                        "GoSlo Digital Camera",
                        359.99));
        order2.Total = 359.99;
        order2.OrderDate = "2013-10-15 16:43:16";
        orders.add(order2);

        Order order3 = new Order();
        order3.OrderId = 3;
        order3.CustomerId = 1;
        order3.SalespersonId = 9000;
        order3.Items.add(
                new Item("WYK12EPU5EZ",
                        "Call of Battle = Goats - Gamesphere 4",
                        69.99));
        order3.Items.add(
                new Item("TJB84HAA8OA",
                        "Bricko Building Blocks",
                        4.99));
        order3.Total = 74.98;
        order3.OrderDate = "2013-11-03 17:45:28";
        orders.add(order3);
        return orders;
    }

    private static OrderSummary createOrderSummary(ArrayList<Order> orders) {
        OrderSummary orderSummary = new OrderSummary();
        orderSummary.CustomerId = 1;
        for(Order order: orders)
        {
            orderSummary.Summaries.add(new OrderSummaryItem(order));
        }
        return orderSummary;
    }

}
```

Output:

```
Creating Data
Starting Client
Creating Buckets
Storing Data
Fetching related data by shared key
Customer     1: {"CustomerId":1,"Name":"John Smith","Address":"123
Main
Street","City":"Columbus","State":"Ohio","Zip":"43210","Phone":"+1-
614-555-5555","CreatedDate":"2013-10-01 14:30:26"}
OrderSummary 1:
{"CustomerId":1,"Summaries":[{"OrderId":1,"Total":415.98,"OrderDate":"
2013-10-01 14:42:26"},{"OrderId":2,"Total":359.99,"OrderDate":"2013-
10-15 16:43:16"},{"OrderId":3,"Total":74.98,"OrderDate":"2013-11-03
17:45:28"}]}
Adding Index Data
Index Queries
Jane's Orders: 3, 1
October's Orders: 1, 2
```

G. Mekanisme replikasi pada Riak

a. **Bagaimanakah sebuah data disimpan pada node? Jelaskan pada kasus mesin di atas, dimana terdapat sebuah cluster yang terdiri dari 4 node.**

   Pada contoh kasus, cluster memiliki n = 3, sehingga sebuah objek direplikasi 3 kali. Sebuah data akan disimpan pada beberapa node, sesuai dengan konfigurasi yang dibuat user.

b. **Apakah yang dimaksud dengan *ring size*?**

   Ring size adalah jumlah partisi data yang menyusun sebuah cluster. Semakin besar *ring size*, semakin tinggi proses I/O pada disk dikarenakan bertambahnya jumlah database yang berjalan secara konkuren. Sedangkan semakin kecil *ring size*, semakin rendah penggunaan juga resources seperti CPU dan RAM. *Ring size* merupakan nilai perpangkatan dari 2. Umumnya jumlah partisi yang baik berkisar 10-50 per node.

c. **Apakah yang dimaksud dengan *vnode*?**

   Vnode atau *virtual nodes* merupakan istilah bagi proses-proses yang mengatur partisi pada *Riak ring*. Setiap partisi pada cluster memiliki *vnode*-nya masing-masing. *Vnode* dapat dianggap sebagai manager yang masing-masing bertanggung jawab untuk mengatur *incoming request* dari *node/vnode* lain, menyimpan objek pada storage backend yang sesuai, mengambil objek dari *backend*, mengintepretasi *causal context metadata* dari objek, bersifat sebagai *strong consistency ensembles*, dsb. *Vnodes* dapat dilihat sebagai sebuah *finite state machine* yang berusaha menjaga sistem Riak untuk *high availability* dan *fault tolerance*.

d. **Bagaimana algoritma quorum diterapkan pada Riak? Jelaskan menggunakan parameter N, R dan W.**

   N adalah jumlah replika data yang ingin disimpan dalam *cluster*.

   R adalah jumlah *server* yang harus merespon untuk dapat membaca *request*.

   W adalah jumlah *server* yang harus merespon untuk dapat menulis *request*.

   Quorum berarti mayoritas dari replika-replika harus merespon, yaitu bernilai $(N/2) + 1$. Artinya R dan W harus lebih besar dari $N/2$.

e. **Buatlah sebuah program pada Java yang menyimpan data pada Riak dengan menggunakan parameter W bernilai 0. Apakah data yang ditulis pada program ini akan tersimpan dengan baik pada database?**

Kode program:

```
IRiakClient client = RiakFactory.pbcClient("167.205.35.19", 8087);
Location locationKey = new Location(new Namespace("w_equals_0", "x"), "key");
RiakObject obj = new
RiakObject().setContentType("text/plain").setValue(BinaryValue.create("My Riak
Object"));
StoreValue store = new StoreValue.Builder(obj).withLocation("key").build();
client.execute(store);
```

Kesimpulan: Tidak dapat dipastikan bahwa data yang ditulis tersimpan dengan baik karena Riak tidak menunggu node untuk memberi respon selesai menulis untuk menganggap data sudah ditulis dengan baik

f. **Berapakah setting N, R dan W yang baik untuk program yang banyak melakukan: i. pembacaan, ii. penulisan**

i. Pembacaan

Untuk dapat melakukan pembacaan cepat, diperlukan nilai R yang kecil (misal $R = 1$) dan untuk menjaga data yang baik, diperlukan W yang memenuhi kuorum ($W = N/2 + 1$).

ii. Penulisan

Untuk dapat melakukan penulisan cepat, diperlukan nilai W yang kecil (misal $W = 1$) dan untuk menjaga data yang baik, diperlukan R yang memenuhi kuorum ($R = N/2 + 1$).

g. **Mungkinkan jika kita melakukan setting N = W, dan jika terjadi kegagalan node, data dapat hilang? Mengapa?**

Hal tersebut dimungkinkan jika jika $N = 1$, karena berarti data hanya dituliskan pada 1 replika, sehingga jika node tersebut gagal, maka data akan hilang. Jika $N > 1$ dan terjadi kegagalan node, maka tidak mungkin data hilang (kecuali semua node gagal). Namun meski demikian, data baru tidak dapat ditulis atau diupdate karena tidak mungkin memenuhi kriteria W sejumlah N dengan node yang tersisa.

Referensi

http://docs.basho.com/riak/kv/2.2.3/

https://docs.basho.com/riak/kv/2.2.3/developing/app-guide/replication-properties/

https://raw.githubusercontent.com/basho/basho_docs/master/extras/code-examples/TasteOfRiak.java

https://github.com/basho/taste-of-riak/tree/master/java/Ch02-Schemas-and-Indexes/src/SipOfRiak

https://github.com/tugas-itb-erick/riak-explore