Хранилища "ключ-значение"

Объектные базы данных. Лекция 2

Хранилища "ключ-значение"

- Хранилища "ключ-значение" являются простейшим хранилищем данных, использующим ключ для доступа к значению. Такие хранилища используются для хранения изображений, создания специализированных файловых систем, в качестве кэшей для объектов, а также в системах, спроектированных с прицелом на масштабируемость.
- Примеры СУБД: Redis, Memcached, Riak, Berkeley DB, Amazon DynamoDB, LevelDB

Riak

- ***Riak*** – распределенная NoSQL-система управления базами данных с открытым исходным кодом типа "ключ-значение", созданная Basho Technologies под влиянием идей DynamoDB компании Amazon.

- Riak написана преимущественно на Erlang и частично на C/C++ и JavaScript.
- Первый выпуск: 2009 г.
- Последняя версия: 2.1.0 (16 апреля 2015 года)
- Лицензия: Apache license 2.0
- Существует Riak Enterprise — это платная версия Riak с коммерческой поддержкой
- Riak используется в Google, GitHub, AOL, Dell, Wikia, BazQux и др.

Riak

- В Riak значением может быть что угодно – простой текст, документ в формате JSON или XML, изображение или видеоклип.
- Riak предоставляет REST-интерфейс поверх HTTP.
- В Riak нет хорошей поддержки произвольных запросов, а хранилища ключей и значений, по своей природе, плохо связываются друг с другом (иными словами, понятие внешнего ключа отсутствует).
- Riak – отказоустойчивая система. Любой сервер может быть остановлен или запущен в любой момент, точки общего отказа не существует. Кластер продолжает работать при удалении, добавлении или аварийном отказе серверов.
- Riak – выбор для центров обработки данных, которые должны обслуживать много запросов с низкой задержкой.

Представление данных в Riak

<ключ> : <значение>

- Примеры:
  - dog1 : "Тузик"
  - dog2 : "{"nickname" : "Sweet Polly Purebred", "breed" : "Purebred"}"
  - dog3 :

"‰PNG

IHDR Лm8 sBIT|d€ sRGB ®Ой gAMA ±Ц
ьа pHYs Д Д•+ tEXtSoftware www.inkscape.org›o<
™IDAT(S…QЫ+ŕq~sп“ЫЪж†
‡Ж=Й1"ўЬLYD№q!k9ьJ"Ьs±"¬№Eie8)cM¬ШВ6vИNYлзз·/µ+ПХysпУ{x^ђ°d4‘
л•eВюа жн]1Jŕ‡€X,&ILH$ьц,O'ЏE(
¦фоПБюшНьН$,·6Ыъъ УЏбэЭ‹Р(Њх<Ґ¶
›Ы1ЁЖкЪ:д2¤R)’Й$jZУ
ї5ШЫ1ГйtA©РЂJ№Т*…уГ,|>?Э2гSCrh rN№ЊZ,Ђр8mO?LЛ‹Р)•a-§g(S—@ЫХ
!КА“ЛЂгРРђя;‹rz/Апя@МuЛUV"iЃ<ПБлу1MIq«cxu5iиИЬњд©Тш¦'D#Q&€'МЉs
ЮпйG±d\Ґа~Ґбг©пq"ЄХ™ЋDSkУя}щДr*2ВЁgДjSfьO~ОиЊцц_’ЌН-
вv{Иллщшыэ"ЭШп‰Гб$?ЋГ4УLф‰ IEND®B`,"

REST-интерфейс

- ***REST (Representational State Transfer)*** - архитектурный стиль взаимодействия компонентов распределенного приложения в сети.
- Системы, поддерживающие REST, называются RESTful-системами.
- REST – это рекомендация по отображению ресурсов на URL-адреса и взаимодействию с ресурсами путем использования операций CRUD.

Операции CRUD

- ***CRUD*** — сокращенное именование 4-х базовых функций, используемых при работе с хранилищами данных:
  - создание (***Create***);
  - чтение (***Read***);
  - редактирование (***Update***);
  - удаление (***Delete***).

Операция	**Оператор в языке SQL**	**Операция в протоколе HTTP**
Создание (create)	INSERT	POST
Чтение (read)	SELECT	GET
Редактирование (update)	UPDATE	PUT или PATCH
Удаление (delete)	DELETE	DELETE

Пример REST-интерфейса

- Каждая единица информации однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных:
  - третья книга с книжной полки:
    /book/3
  - 35 страница в этой книге:
    /book/3/page/35
- Пример запросов:
  - GET /book/ — получить список всех книг
  - GET /book/3/ — получить книгу номер 3
  - PUT /book/ — добавить книгу (данные в теле запроса)
  - POST /book/3 – изменить книгу (данные в теле запроса)
  - DELETE /book/3 – удалить книгу

Простой запрос к Riak

- Добавление записи в хранилище Riak:

```
$curl -v -X PUT http://localhost:8091/riak/favs/db \
-H "Content-Type: text/html" \
-d "<html><body><h1>My new DB</h1></body></html>"
```

ключ

значение

- Параметр -X PUT говорит утилите cURL выполнить HTTP-действие PUT, чтобы сохранить явно заданное значение ключа.
- Флаг -H означает, что следующая за ним строка должна быть передана в виде HTTP-заголовка (MIME-тип содержимого HTML).
- Флаг -d (тело запроса) интерпретирует последующую строку, как новое значение.

Модельная база данных "Гостиница для собак"

Реляционная схема

Animals

KeyA*	Nickname	Breed	Photo
ace	The Wonder Dog	German Shepherd	ace_image.jpg
polly	Sweet Polly Purebred	Purebred	polly_image.jpg
...	...	...	...

Cages

KeyC*	Room	Contains	Next_to
1	101	polly	null
2	101	ace	1
...	...	...	...

Сегменты множества ключей

- Riak разбивает все множество ключей на ***сегменты (bucket)***, чтобы избежать коллизий.
- URL-адрес устроен следующим образом:

```
http://SERVER:PORT/riak/BUCKET/KEY
```

- Пример создания сегмента animals и добавление ключа ace:

```
$curl -v -X PUT http://localhost:8091/riak/animals/ace \
-H "Content-Type: application/json" \
-d ‘{"nickname" : "The Wonder Dog", "breed" : "German Shepherd"}’
```

- Список созданных сегментов:

```
$curl -X GET http://localhost

{"buckets":["favs","animals"]}
```

Операции с ключами

- Автоматическая генерация ключа:

```
$curl -i -X POST http://localhost:8091/riak/animals \
-H "Content-Type: application/json" \
-d ‘{"nickname" : "Sergeant Stubby", "breed" : "Terrier"}’
```

- Удаление ключа:

```
$curl -i -X DELETE
http://localhost:8091/riak/animals/6VZc2o7zKxq2B34kJrm1S0ma3PO
```

- Список ключей в сегменте:

```
$curl http://localhost:8091/riak/animals?keys=true
```

Ссылки

- ***Ссылки (link)*** – метаданные , ассоциирующие один ключ с другими.
- Ссылка, указывающая на ключ key в сегменте bucket:

```
Link: </riak/bucket/key>; riaktag=\"whatever\"
```

Ключ объекта, на который ссылаемся

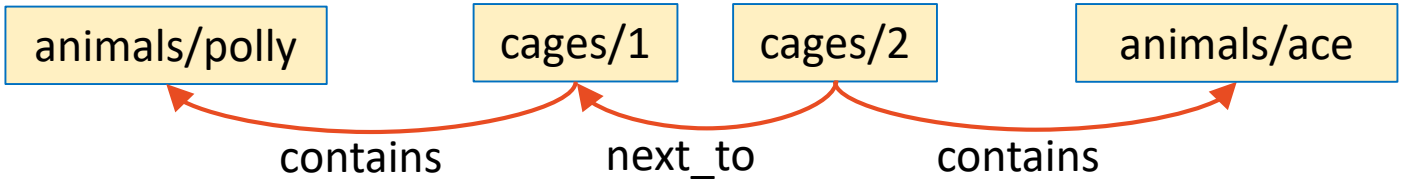
Тег - название связи

- Пример:

```
$curl -X PUT http://localhost:8091/riak/cages/1 \
-H "Content-Type: application/json" \
-H "Link: </riak/animals/polly>; riaktag=\"contains\"" \
-d ‘{"room" : 101}’
```

- Множество ссылок разделяется запятыми:

```
$curl -X PUT http://localhost:8091/riak/cages/2 \
-H "Content-Type: application/json" \
-H "Link:</riak/animals/ace>;riaktag=\"contains\",
</riak/cages/1>;riaktag=\"next_to\"" \
-d ‘{"room" : 101}’
```



Обход ссылок (link-walking)

- Для получения связанных данных мы добавляем в URL *спецификацию ссылки*:

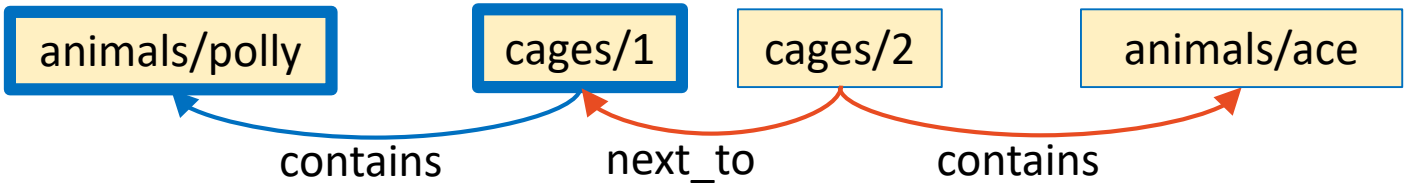
  `/<сегмент>,<тег>,<признак_keep>.`

- Пример: выберем все ссылки, ведущие из клетки 1

  `$curl http://localhost:8091/riak/cages/1/_,_,_`

- Результат:

```
…
Location: /riak/animals/polly
Content-Type: application/json
Link: </riak/animals>; rel="up"
…
{"nickname" : "Sweet Polly Purebred", "breed" : "Purebred"}
…
```



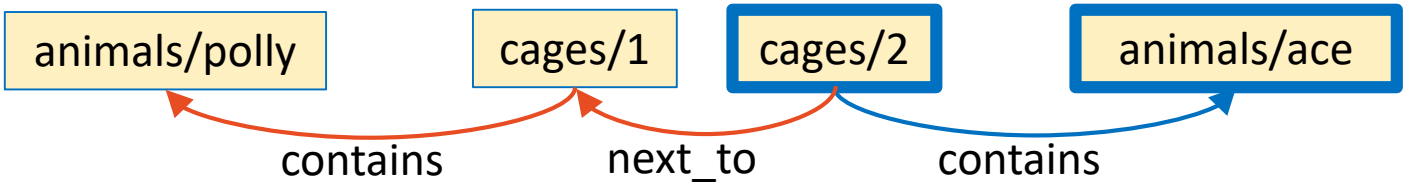
Обход ссылок (link-walking)

- Для получения связанных данных мы добавляем в URL *спецификацию ссылки*:

  `/<сегмент>,<тег>,<признак_keep>.`

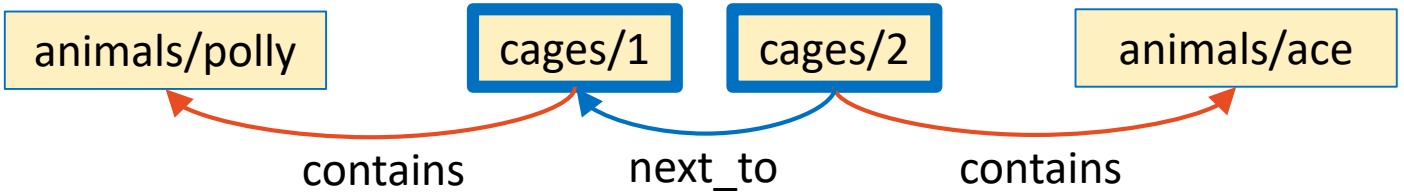
- Пример: выберем все ссылки, ведущие из клетки 2 (только в сегменте animals)

  `$curl http://localhost:8091/riak/cages/2/`**`animals,_,_`**



- Пример: узнать о клетках рядом с клеткой 2

  `$curl http://localhost:8091/riak/cages/2/_,`**`next_to`**`,_`



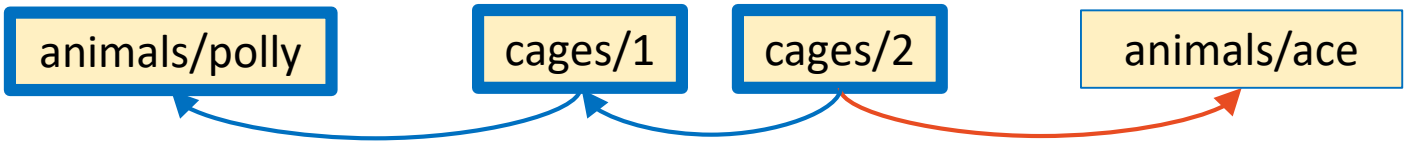
Обход ссылок (link-walking)

- `признак`_keep - принимает значения 1 или 0. При "1" происходит следование по ссылкам второго порядка, то есть ведущим из объекта, на который ведет первая ссылка.

- Пример:
  `$curl http://localhost:8091/riak/cages/2/_,next_to,1/_,_,_`

- Результат:

```
…
Location: /riak/cages/1
Content-Type: application/json
Link: </riak/animals/polly>; riaktag="contains", </riak/cages>;
rel="up"
…
{"room" : 101}
…
Location: /riak/animals/polly
Content-Type: application/json
Link: </riak/animals>; rel="up"
…
{"nickname" : "Sweet Polly Purebred",
```



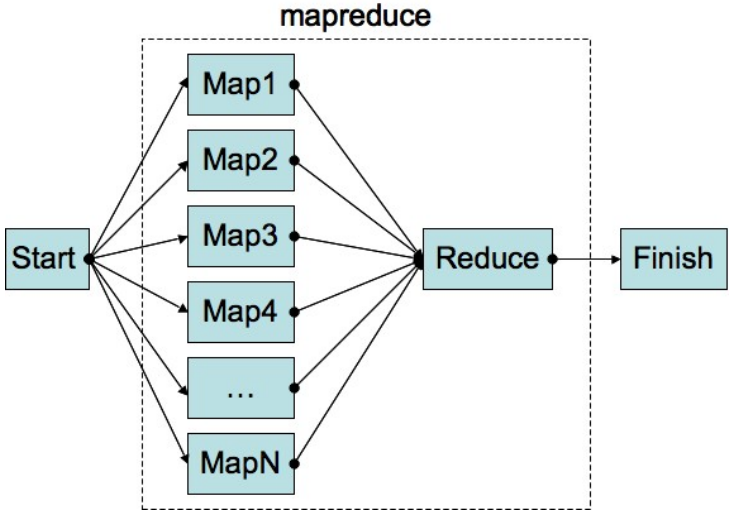
Типы MIME в Riak

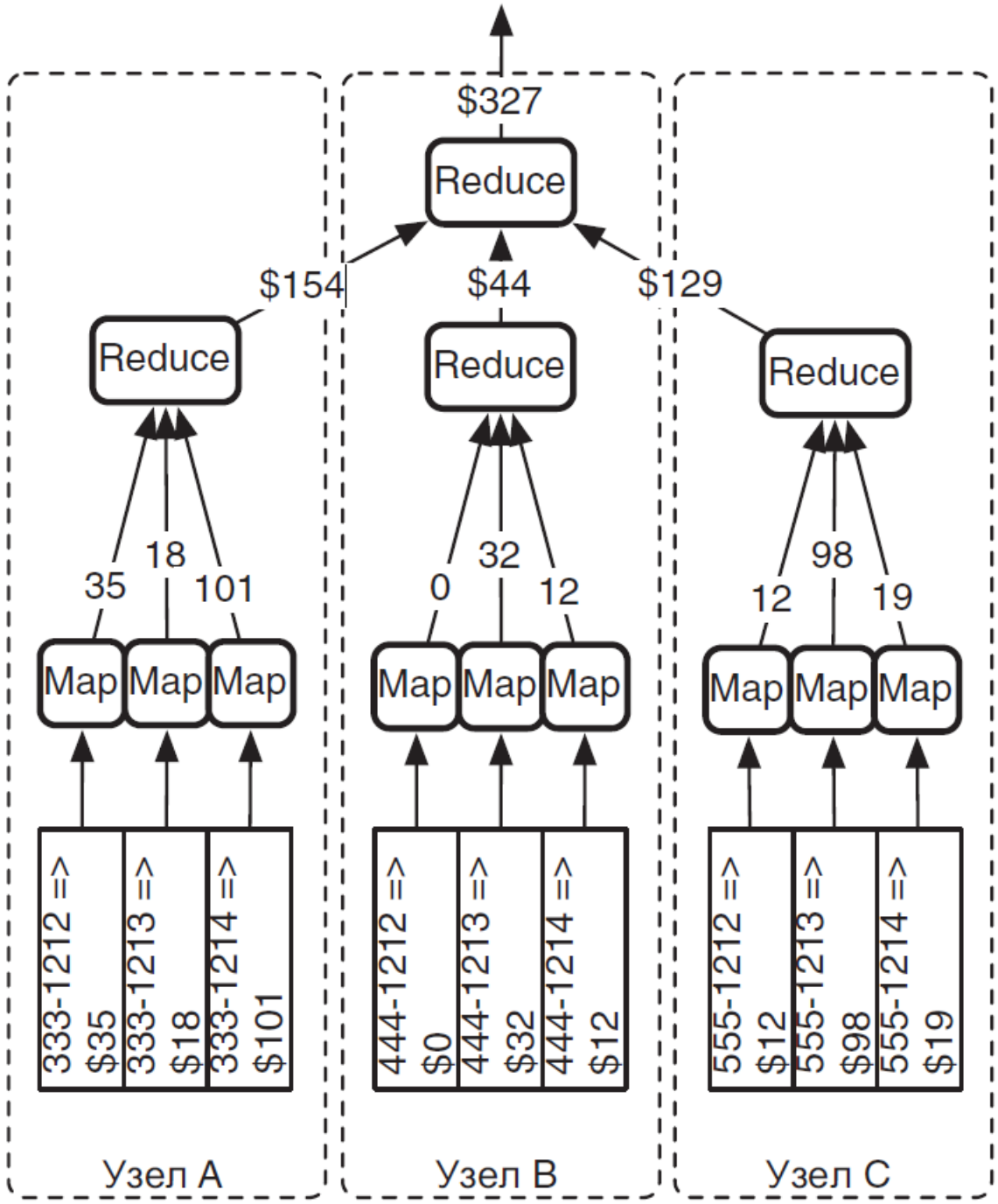
- Riak хранит все данные в двоичном виде (как принято в HTTP). Тип MIME сообщает, как интерпретировать двоичные данные.
- Пример: добавление фотографии Полли

```
$ curl -X PUT http://localhost:8091/riak/photos/polly.jpg \
-H "Content-type: image/jpeg" \
-H "Link: </riak/animals/polly>; riaktag=\"photo\"" \
--data-binary @polly_image.jpg
```

Mapreduce

- MapReduce — модель распределенных вычислений, представленная компанией Google, используемая для параллельных вычислений над большими наборами данных в компьютерных кластерах.
- При использовании mapreduce задача разбивается на две части:
  1) ***распределение*** - список данных преобразуется в новый список посредством функции *map()*.
  2) ***редукция*** - полученный список преобразуется в одно или несколько скалярных значений с помощью функции *reduce().*





Пример

- Функция map поставляет исходные данные редукторам, а вычисленные теми данные передаются редукторам следующего уровня

Mapreduce в Riak

- Для настройки функций map и reduce необходимо задать язык программирования и исходный код.

```
$ curl -X POST -H "content-type:application/json" \
http://localhost:8091/mapred --data @-
{  "inputs":"<Входные данные>",
   "query":[
     { "map":{
                 "language":"javascript",
                 /* Функция */ }},
     { "reduce":{
                 "language":"javascript",
                 /* Функция */ }}
   ]
}
```

- Варианты реализации функции:
  - передача исходного кода в REST-запросе
  - создание хранимой функции
  - использование встроенных функций Riak

Передача исходного кода в REST-запросе

```
$ curl -X POST -H "content-type:application/json" \
http://localhost:8091/mapred --data @-
{ "inputs":[
      ["rooms","101"],["rooms","102"],["rooms","103"]    ],
  "query":[
      {"map":{
          "language":"javascript",
          "source":
             "function(v) {
             /* Достать данные из Riak и разобрать их как JSON */
             var parsed_data = JSON.parse(v.values[0].data);
             var data = {};
             /* Словарь вместимостей с ключом “тип комнаты” */
             data[parsed_data.style] = parsed_data.capacity;
             return [data]; }"
}} ] }
```

Результат: `[{"suite":6},{"single":1},{"double":1}]`

Хранимые функции

- Riak предоставляет возможность хранить функцию map в качестве значения сегмента.

```
$ curl -X PUT -H "content-type:application/json" \
http://localhost:8091/riak/my_functions/map_capacity --data @-
function(v) {
var parsed_data = JSON.parse(v.values[0].data);
var data = {};
data[parsed_data.style] = parsed_data.capacity;
return [data];
}
```

- Выполнение хранимой функции:

```
…
"query":[
{ "map":{
    "language":"javascript",
    "bucket":"my_functions",
    "key":"map_capacity"
}}   ]
…
```

Встроенные функции Riak

- Все встроенные функции Riak находятся в файле mapred_builtins.js.
- mapValuesJson - извлечение из объектов значения и возврат их в формате JSON.

- Пример:

```
curl -X POST http://localhost:8091/mapred \
-H "content-type:application/json" --data @-
{
"inputs":[
  ["rooms","101"],["rooms","102"],["rooms","103"]  ],
"query":[
  {"map":{
     "language":"javascript",
     "name":"Riak.mapValuesJson"
  }  }
```

Функция reduce

```
"reduce":{
   "language":"javascript",
   "source":
      "function(v) {
         var totals = {};
         for (var i in v) {
         for(var style in v[i]) {
         if( totals[style] )
            totals[style] += [i][style];
         else
            totals[style] = v[i][style];  }  }
         return [totals];  }”
}
```

Фильтры ключей

- ***Фильтры ключей*** - это набор команд, которые применяются для обработки каждого ключа перед подачей его на вход mapreduce.
- Фильтры позволяют сэкономить на загрузке ненужных данных.

```
"inputs":{
  "bucket":"rooms",
  "key_filters":[["string_to_int"], ["less_than", 1000]]
},
```

Обход ссылок в mapreduce

- В секции *query* наряду с параметрами map и reduce появляется еще один – link.

- Если объединить функции map и reduce, следование по ссылкам и фильтры ключей, то можно будет исполнять произвольные запросы к широкому спектру ключей Riak. Это гораздо эффективнее, чем просматривать все данные на стороне клиента. Поскольку такие запросы обычно исполняются одновременно на нескольких серверах, то долго ждать не придется.

Согласованность и долговечность

- Невозможно создать распределенную базу данных, которая была бы полностью согласованной, доступной и устойчивой к потере связности (когда некоторые сообщения пропадают). В *любой момент времени в*озможно выполнение только двух условий. Это утверждение известно под названием ***теорема CAP***.
- Riak позволяет выбирать доступность или согласованность на уровне отдельных запросов.

Репликация

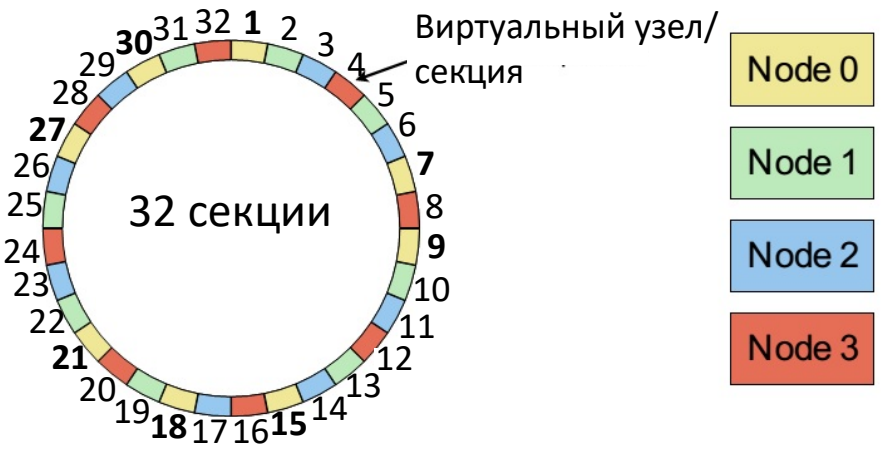
- **Репликация** — одна из техник масштабирования баз данных, которая состоит в том, что данные с одного сервера базы данных постоянно копируются (**реплицируются**) на один или несколько других (называемые **репликами**).
- Для приложения появляется возможность использовать не один сервер для обработки всех запросов, а несколько. Таким образом появляется возможность распределить нагрузку с одного сервера на несколько.

Репликация в Riak

- При записи объекта в базу данных Riak мы можем затребовать репликацию на несколько узлов. Если один сервер выйдет из строя, то останется копия данных на другом сервере.
- Данные ***автоматически распределяются*** между узлами и ***перераспределяются*** при добавлении нового узла.
- В Riak нет единой точки отказа. Все узлы в кластере являются равноправными (нет master’ов и slave’ов).

Кольцо Riak

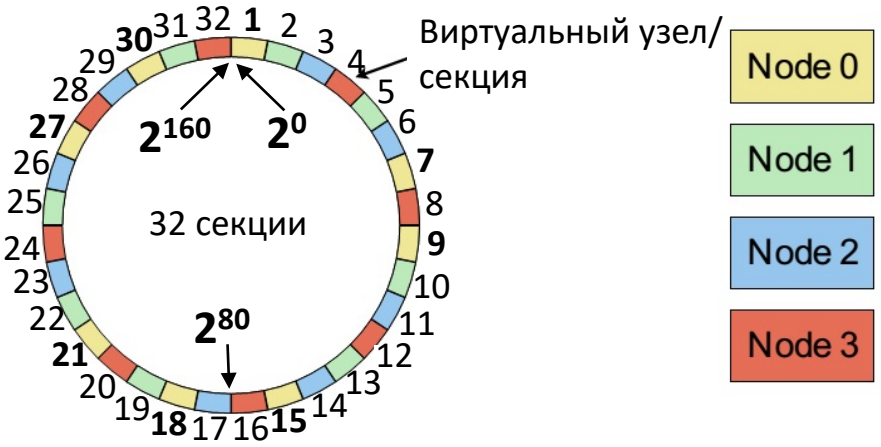
- Riak разбивает множество серверов на ***секции***. Секции представляются в виде кольца.
- Количество секций указывается при настройке Riak. Это число должно быть степенью двойки.
- Каждая секция хранится на ***виртуальном узле***, или ***v-узле (vnode).***
- Каждый виртуальный узел соответствует некоторому физическому серверу/узлу кластера.



Пример кольца Riak для 32 секций

Кольцо Riak

- На кольцо отображаются последовательно числа от 2^0 до 2^{160}, которые представляют собой ***хэш-значения для ключей объектов «ключ-значения»***.
  - Т.о. каждый виртуальный узел представляет диапазон хэш-значения для ключей.
- Вычисляется хеш ключа, Riak отображает его на секцию, кольцо определяет, на каком сервере будет храниться значение.



Пример кольца Riak для 32 секций

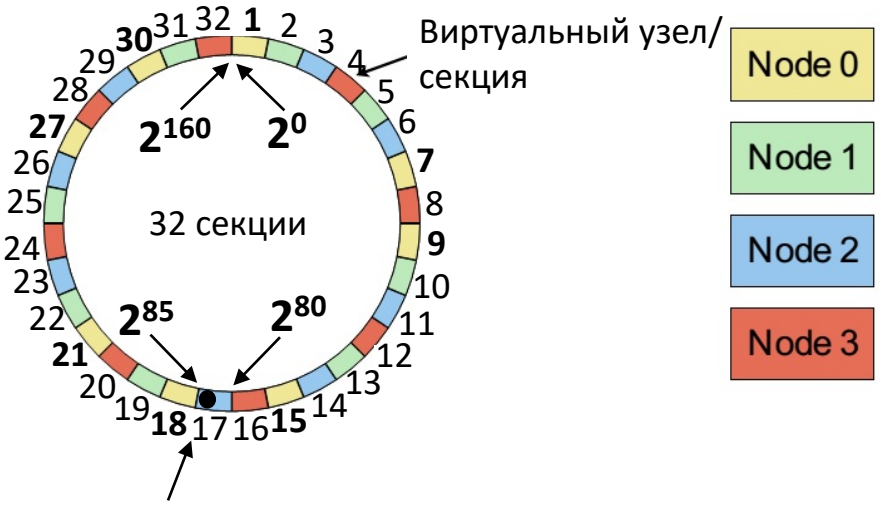
Кольцо Riak: пример

- При вставке данных

  `"101" : "suite"`

  хеш-значение ключа 101 находится, например, в диапазоне от 2^{80} до 2^{85}.

  Данный диапазон принадлежит виртуальному узлу 17, и тогда объект "101" : "suite" будет храниться в узле 2.



Node 2: `"101" : "suite"`

Операции чтения-записи

- Riak позволяет управлять операциями чтения-записи в кластере с помощью трех параметров: ***N***, ***W*** и ***R***.

Параметр N

- N – это количество узлов, которые будут содержать правильное значение ***в конечном счете***. Это не означает, что мы должны дождаться, пока значение будет реплицировано на все узлы, и только потом вернуть управление. Иногда требуется вернуть результат клиенту немедленно и разрешить Riak продолжить репликацию в фоновом режиме. А иногда, наоборот, желательно дождаться завершения репликации на все N узлов (безопасности ради).
- По умолчанию в Riak происходит репликация на три узла (N = 3).

Параметр W

- Величина *W* определяет, на скольких узлах запись должна завершиться успешно, прежде чем можно будет считать успешной операцию в целом. В конечном счете значение будет записано на все три узла, но если сделать W равным 1, то операция записи вернет управление после успешной записи всего одной копии.
- Репликация на оставшиеся два узла произойдет в фоновом режиме.

Параметр R

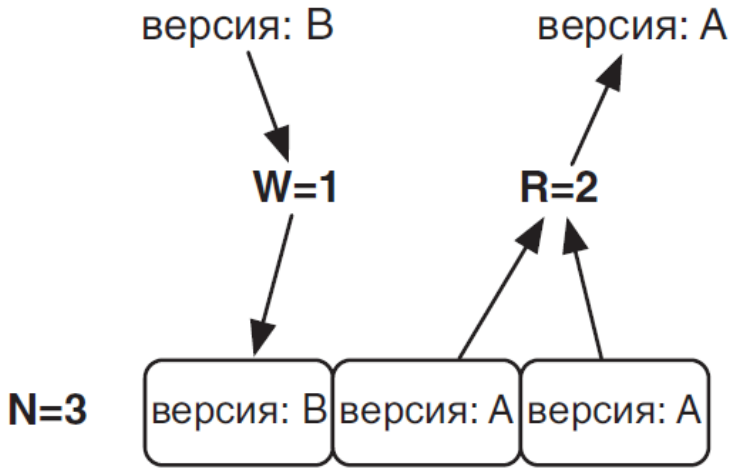
- *R* – это количество узлов, откуда должно быть прочитано значение, чтобы операция чтения считалась успешной.
- Можно задать величину *R* по умолчанию. Но Riak предлагает и более гибкое решение. Мы можем указывать, со скольких узлов следует прочитать значение, задавая параметр r *в конкретном запросе*:

```
curl http://localhost:8091/riak/animals/ace?r=3
```

Зачем считывать данные с нескольких узлов?

Согласованность в конечном счете

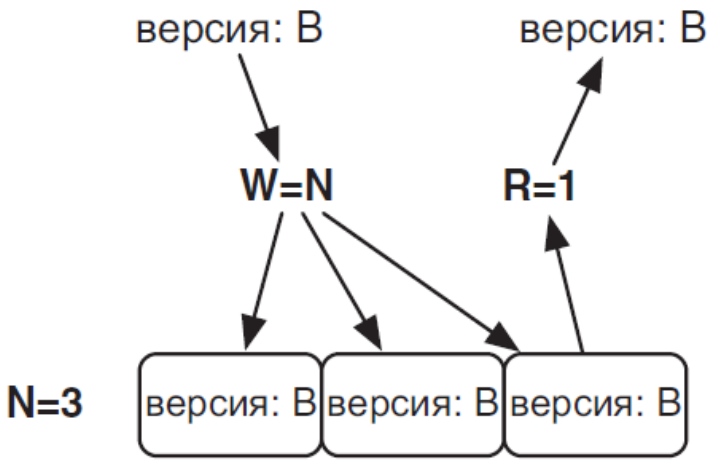
- Предположим, что тройка NRW задана следующим образом {“n_val”:3, “r”:2, “w”:1}
- В этом случае система будет **быстрее отзываться на операции записи**, поскольку перед возвратом управления необходимо успешно завершить операцию только на одном узле.
- НО не исключено, что какой-то другой процесс начнет операцию чтения до того, как все узлы синхронизируются. **Даже при чтении с двух узлов есть шанс получить старое значение.**



Согласованность в конечном счете: W+R <= N

Согласованность за счет записи

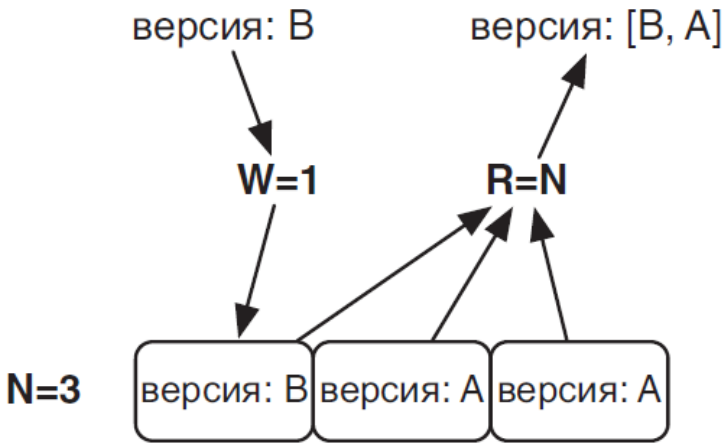
- Один из способов гарантировать получение актуального значения – установить W=N и R=1: {“n_val”:3, “r”:1, “w”:3}. То есть поступить точно так же, как делают реляционные СУБД, которые обеспечивают непротиворечивость, **не возвращая управление, пока запись не будет завершена**.
- Это **ускорит чтение**, так как нам нужно обратиться только к одному узлу. Зато **запись может заметно замедлиться**.



Согласованность за счет записи : W=N, R=1

Согласованность за счет чтения

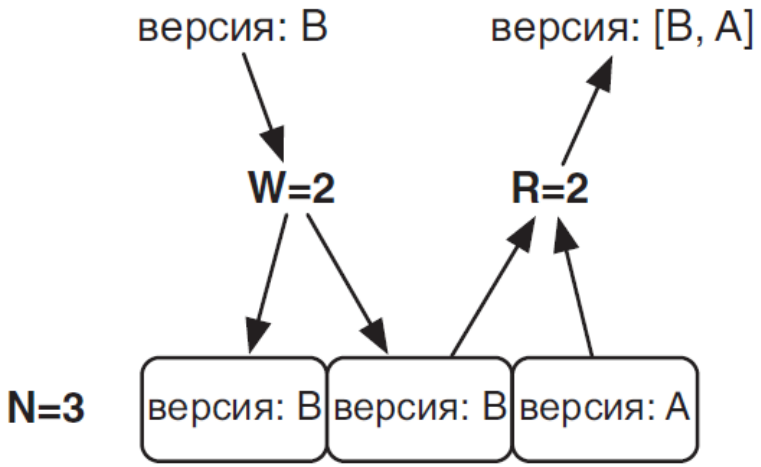
- Можно **писать на один узел, а читать со всех**, т. е. задать W=1, R=N: {“n_val”:3, “r”:3, “w”:1}.
- Не исключено, что среди них окажется несколько устаревших значений, но **гарантируется, что последнее записанное значение будет прочитано**. Останется только выяснить, что это за значение (применяется алгоритм векторных часов).
- Здесь возникает проблема – **медленное чтение**.



Согласованность за счет чтения: W=1, R=N

Согласованность за счет кворума

- Пусть W=2 и R=2: {“n_val”:3, “r”:2, “w”:2}. Здесь **требуется завершить запись более чем на половине узлов и читать также более чем с половины узлов**.
- Обеспечивается согласованность, распределяя временные задержки поровну между операциями чтения и записи. Это называется ***кворумом*** и дает минимальное число операций, необходимое для поддержания согласованности данных.



Согласованность за счет кворума : W+R>N

Предустановленные параметры для N, R, W

Значение	Определение
one	Это просто 1. Присваивание такого значения параметру W или R означает, что операция будет считаться успешной после завершения хотя бы на одном узле.
all	Значение, совпадающее с N. Присваивание такого значения параметру W или R означает, что операция будет считаться успешной после того, как завершится на всех реплицированных узлах.
quorum	Равно N/2+1. Означает, что операция должна успешно завершиться на большинстве узлов.
по умолчанию	Значение W или R, заданное для сегмента. По умолчанию равно 3.

Перечисленные значения можно задавать в качестве свойств сегмента:

```
curl http://localhost:8091/riak/animals/ace?r=all
```

Запись и долговечная запись

- Для операций записи в Riak ***не гарантируется долговечность***, то есть данные не пишутся на диск немедленно. Даже если *запись на узел* сочтена успешной, все равно не исключено, что в результате сбоя данные в этом узле будут потеряны – даже в случае, когда W=N. Перед записью на диск данные некоторое время хранятся в буфере в памяти, и вот эта-то доля миллисекунды и есть опасная зона.
- В Riak имеется специальный параметр ***DW (durable write, долговечная запись)***. В этом режиме операция производится медленнее, но риск снижается, так как Riak не возвращает управление, пока объект не будет физически записан на диск на заданном числе узлов.

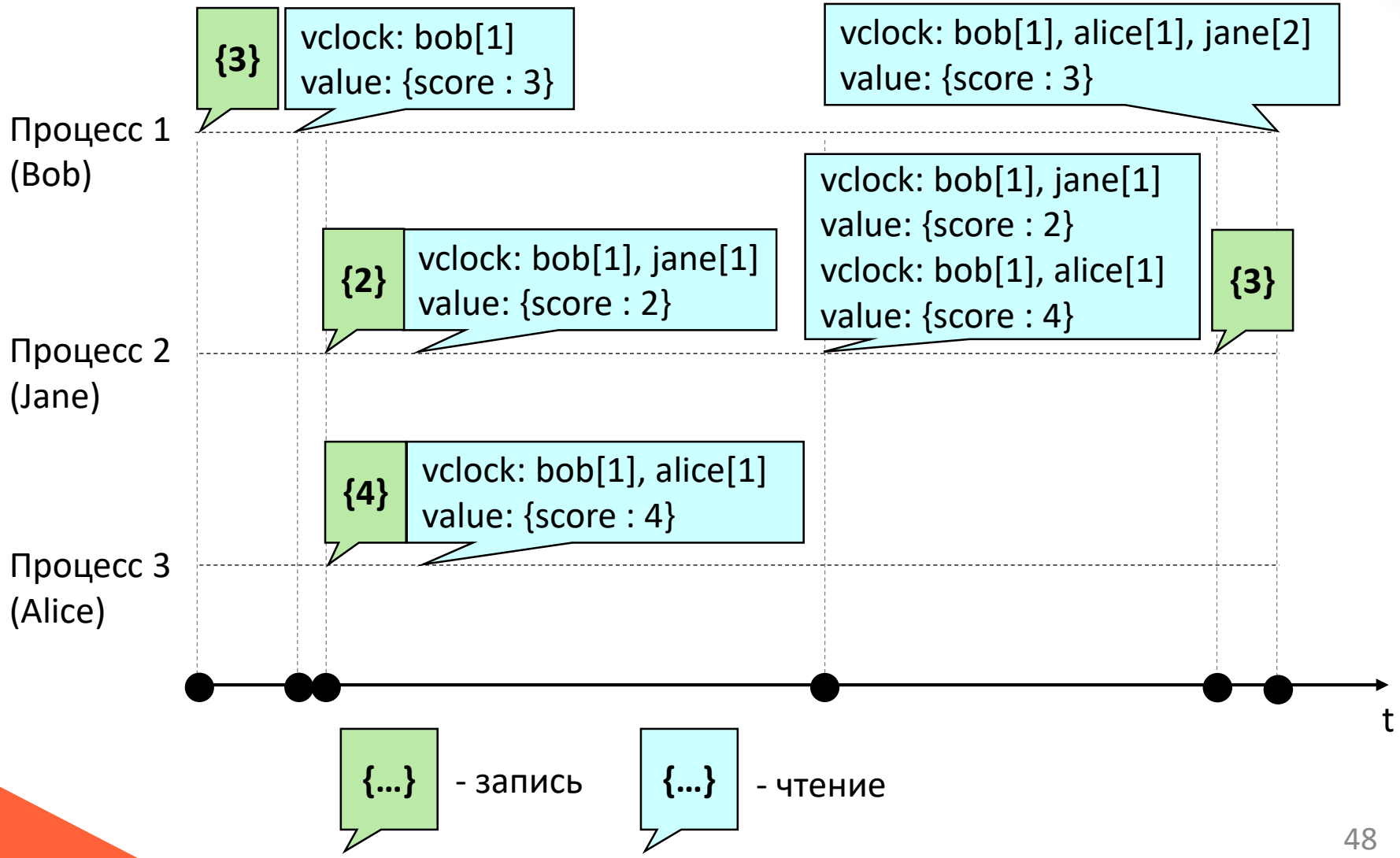
О временной передаче ответственности

- В случае отказа одного из узлов, Riak записывает данные на расположенный рядом узел, где они хранятся до тех пор, пока не появится возможность вернуть их на ранее недоступный узел.
- Все запросы, адресованные серверу A, передаются серверу B, нагрузка на сервер B возрастает вдвое. Если в результате откажет и сервер B, тогда нагрузка ляжет на сервер C. Такое явление называется ***каскадным отказом***.

Алгоритм «векторные часы»

- ***Векторные часы (vector clock)*** – это алгоритм, который в распределенных системах используется для сохранения порядка конфликтующих обновлений пар ключ-значение.
- Векторные часы решают проблему, снабжая каждое событие (создание, обновление или удаление пары ключ-значение) меткой , содержащей информацию о том, какой клиент произвел изменение и в каком порядке. В результате сами клиенты или разработчик приложения могут решать, кто побеждает в случае конфликта.

Векторные часы в Riak



Riak — редкая база данных

- Riak ***избыточен для большинства проектов***: избыточность заключается в том, что для небольших проектов MySQL или Postgresql покрывает все требования к хранению данных, а грамотная репликация дает хорошую отказоустойчивость.
- Вторая проблема заключается в том, что для хорошей производительности Riak ***в кластере должно быть пара десятков машин***. Если вы рассчитали, что вам хватит пяти-десяти машин, то скорее всего вам достаточно трех-четырех машин с MySQL. Достаточно большой кластер Riak сложен в обслуживании, несмотря на все его достоинства и организацию.
- ***Riak нужен, если у вас действительно много данных*** и вы точно знаете, что делаете.

Рейтинг СУБД 2015 издания DB-Engines

Rank Jan 2016	Rank Dec 2015	Rank Jan 2015	DBMS	Database Model	Score Jan 2016	Score Dec 2015	Score Jan 2015
1.	1.	1.	Oracle	Relational DBMS	1496.08	-1.47	+56.92
2.	2.	2.	MySQL	Relational DBMS	1299.26	+0.72	+21.75
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1144.06	+20.90	-54.55
4.	4.	↑ 5.	MongoDB +	Document store	306.03	+4.64	+55.13
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	282.40	+2.31	+27.91
6.	6.	6.	DB2	Relational DBMS	196.37	+0.24	-3.76
7.	7.	7.	Microsoft Access	Relational DBMS	134.04	-6.17	-5.10
8.	8.	8.	Cassandra +	Wide column store	130.95	+0.11	+32.20
9.	9.	9.	SQLite	Relational DBMS	103.74	+2.89	+7.54
10.	10.	10.	Redis +	Key-value store	101.16	+0.62	+6.92
11.	11.	11.	SAP Adaptive Server	Relational DBMS	83.18	+1.71	-0.60
12.	↑ 13.	↑ 16.	Elasticsearch	Search engine	77.21	+0.65	+28.17
13.	↓ 12.	↓ 12.	Solr	Search engine	75.39	-3.75	-1.35
14.	14.	↓ 13.	Teradata	Relational DBMS	74.95	-0.77	+7.90
15.	15.	↑ 17.	Hive	Relational DBMS	53.58	-1.69	+18.19
16.	16.	↓ 14.	HBase	Wide column store	53.37	-0.88	-0.22
17.	17.	↓ 15.	FileMaker	Relational DBMS	48.83	-1.29	-2.86
18.	18.	↑ 20.	Splunk	Search engine	43.12	-0.74	+10.05
19.	19.	↑ 21.	SAP HANA	Relational DBMS	38.61	-0.24	+8.71
20.	20.	↓ 18.	Informix	Relational DBMS	34.88	-1.52	+0.06
21.	21.	↑ 23.	Neo4j +	Graph DBMS	33.01	-0.18	+8.58
22.	22.	↓ 19.	Memcached	Key-value store	29.97	-0.96	-4.40
23.	23.	↑ 27.	MariaDB +	Relational DBMS	27.76	+0.02	+10.34
24.	24.	24.	Couchbase +	Document store	26.09	-0.17	+3.50
25.	25.	↓ 22.	CouchDB	Document store	24.12	-0.92	-2.20

Redis

- Точно отнести Redis к какой-нибудь категории довольно трудно. На самом базовом уровне это хранилище ключей и значений. Но кросе этого Redis поддерживает сложные структуры данных, хотя и не до такой степени, как документо-ориентированные базы данных.
- Она поддерживает запросы, возвращающие множества, но не таком уровне детальности, как реляционные СУБД, и без поддержки типов.
- Redis является еще и блокирующей очередью (или стеком), а также системой публикации-подписки.