



ОНЛАЙН-ОБРАЗОВАНИЕ

Онлайн-образование



Меня хорошо видно && слышно?

Ставьте , если все хорошо
Напишите в чат, если есть проблемы
заодно проверяем, включена ли запись занятия





Кластерные возможности MongoDB

Курочкин Константин
Ведущий администратор баз данных
«Medindex»
telegram https://t.me/konstantin_kurochkin

Не забыть включить запись!



Правила вебинара



Активно участвуем



Задаем вопрос в чат



Вопросы вижу в чате, могу ответить не сразу

Маршрут вебинара



Цели вебинара | После занятия вы сможете

1 Выбирать правильный вариант репликации

2 Понимать как работает концепция кворума

3 Настраивать шардирование

4 Эффективно выбирать правильный ключ шардирования

Варианты репликации





Варианты репликации

Зачем она нужна?

1. Высокая доступность. Бэкап это хорошо, но нужно время на его развертывание.

Зачем она нужна?

1. Высокая доступность. Бэкап это хорошо, но нужно время на его развертывание.
2. Что делать, когда закончились физические ядра и память у сервера? горизонтально масштабировать

Зачем она нужна?

1. Высокая доступность. Бэкап это хорошо, но нужно время на его развертывание.
2. Что делать, когда закончились физические ядра и память у сервера? горизонтально масштабировать
3. Бэкап лучше делать с реплики, а не мастера.

Зачем она нужна?

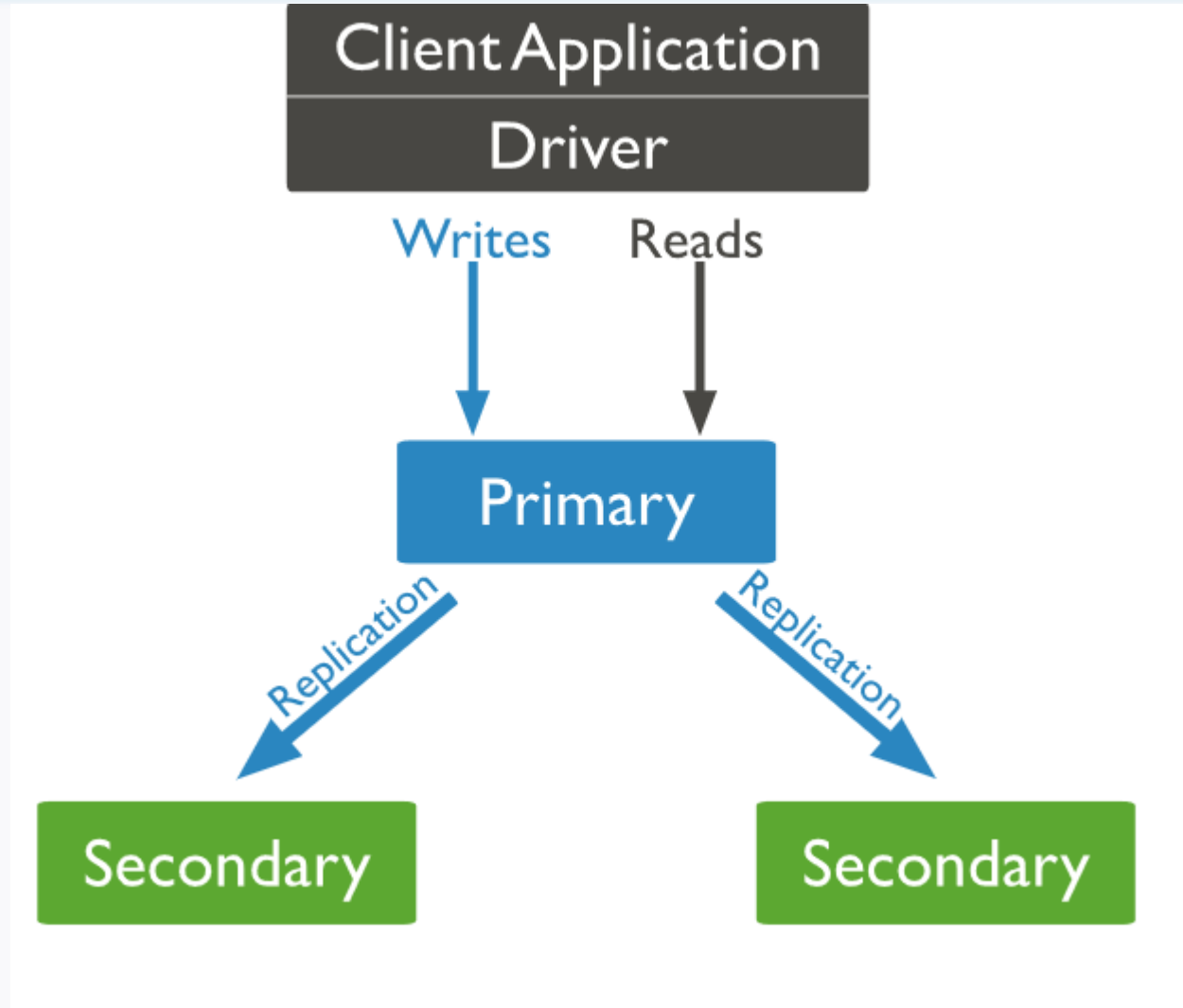
1. Высокая доступность. Бэкап это хорошо, но нужно время на его развертывание.
2. Что делать, когда закончились физические ядра и память у сервера? горизонтально масштабировать
3. Бэкап лучше делать с реплики, а не мастера.
4. Геораспределение нагрузки.

Виды репликации

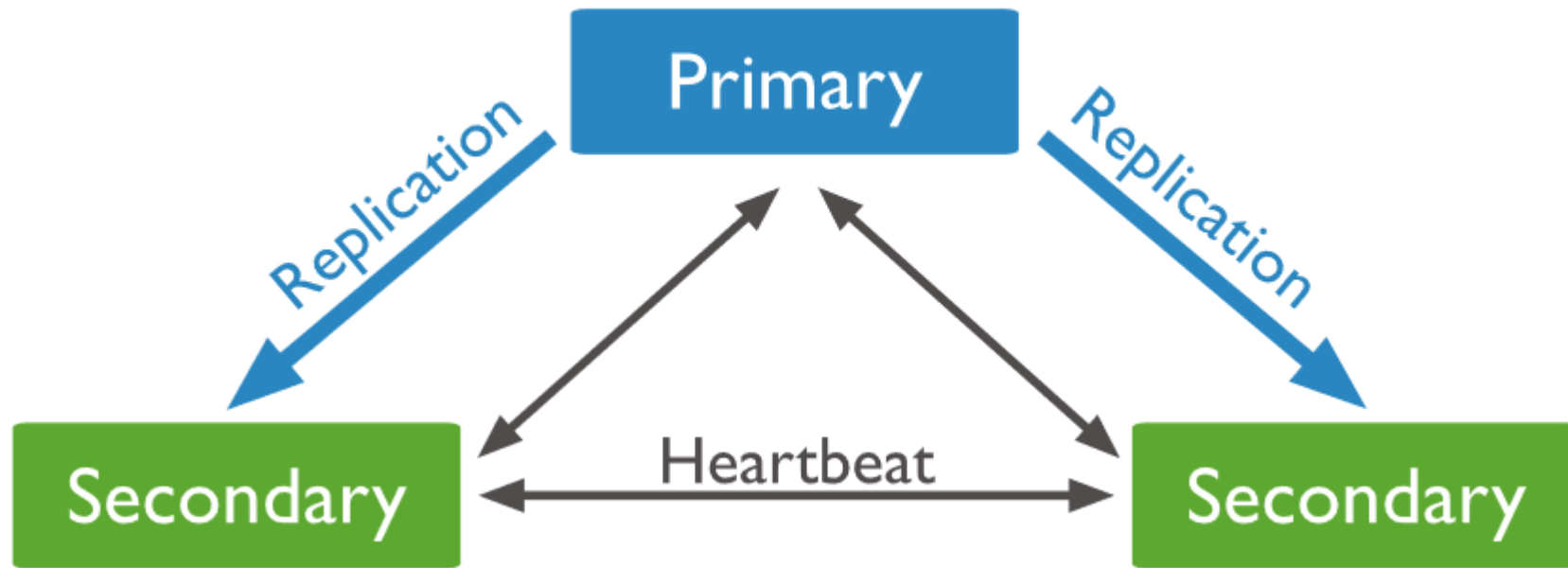
1. Replicaset

2. Master-slave 3.6

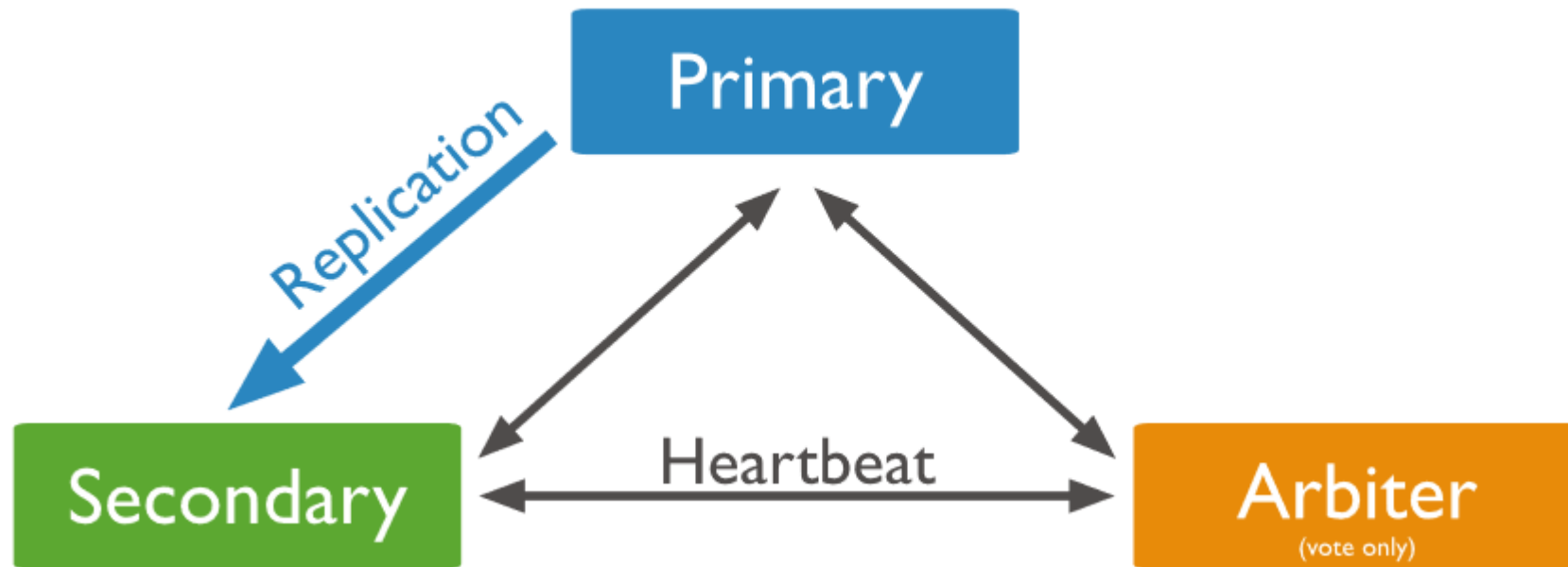
Виды репликации. Replicaset



Виды репликации. Replicaset



Виды репликации. Replicaset



Виды репликации. Основные особенности

- репликация асинхронная
- арбитр не содержит данных и поэтому очень легковесный
- primary может стать secondary и наоборот, а арбитр не может стать ни primary, ни secondary
- максимальное количество реплик 50, из них голосовать разрешено только 7
- запуск арбитра на той же машине, что и реплика не рекомендован (почему?)

Виды репликации. Практика. ReplicaSet

Можно настроить данный РепликаСет несколькими способами:

- Использовать 1 сервер и запустить 3 экземпляра самой `mongoDB`.
- Использовать 3 сервера с `mongoDB`.

Итак, для правильно работы ReplicaSet необходимо 3 запущенных экземпляра или сервера с монгой:

- Одна нода будет выступать как арбитр и не будет принимать на себя никакие данные. Его работа, заключается в том, чтобы выбрать того, кто будет сервером PRIMARY.
- Один сервер выступает в качестве PRIMARY сервера.
- Один сервер выступает в качестве SECONDARY сервера.
- *для доступа к экземпляру `mongo` из сети (не `localhost`) необходимо настроить встроенный файрвол и указать, например, `--bind_ip_all`*
- <https://docs.mongodb.com/manual/core/security-mongodb-configuration/>

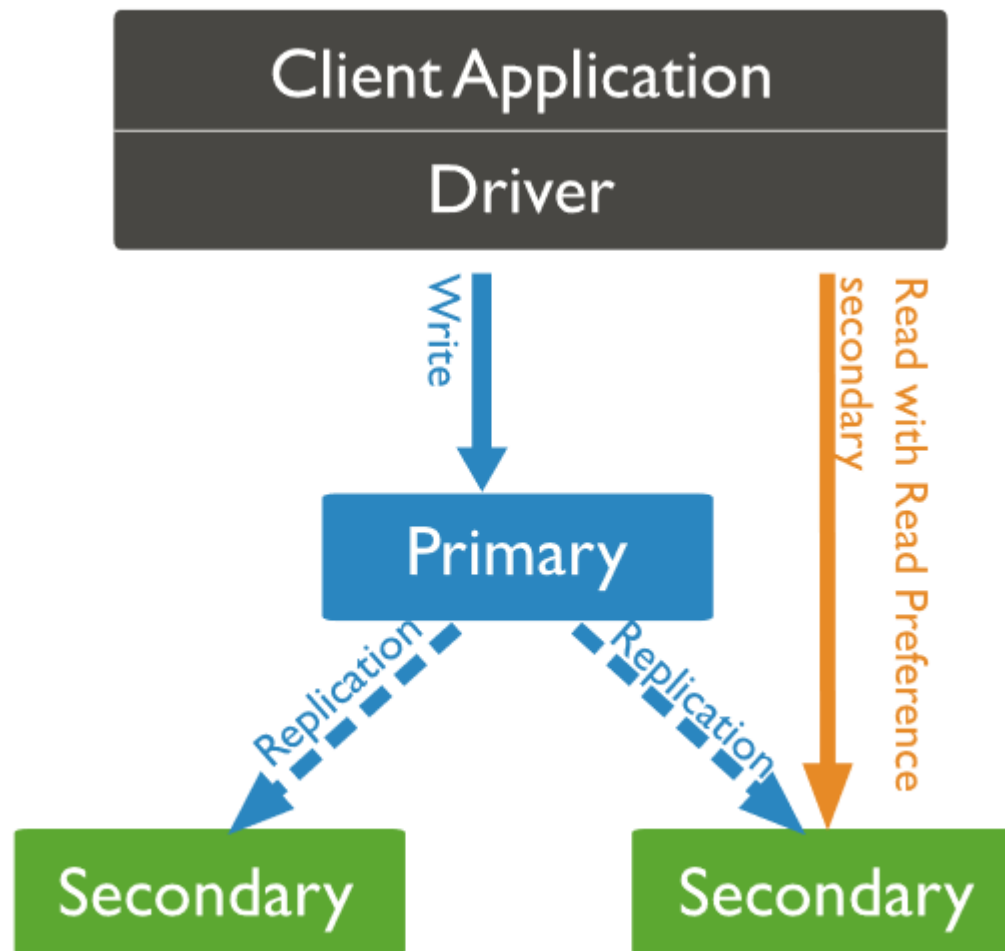
Виды репликации. Практика.

ReplicaSet

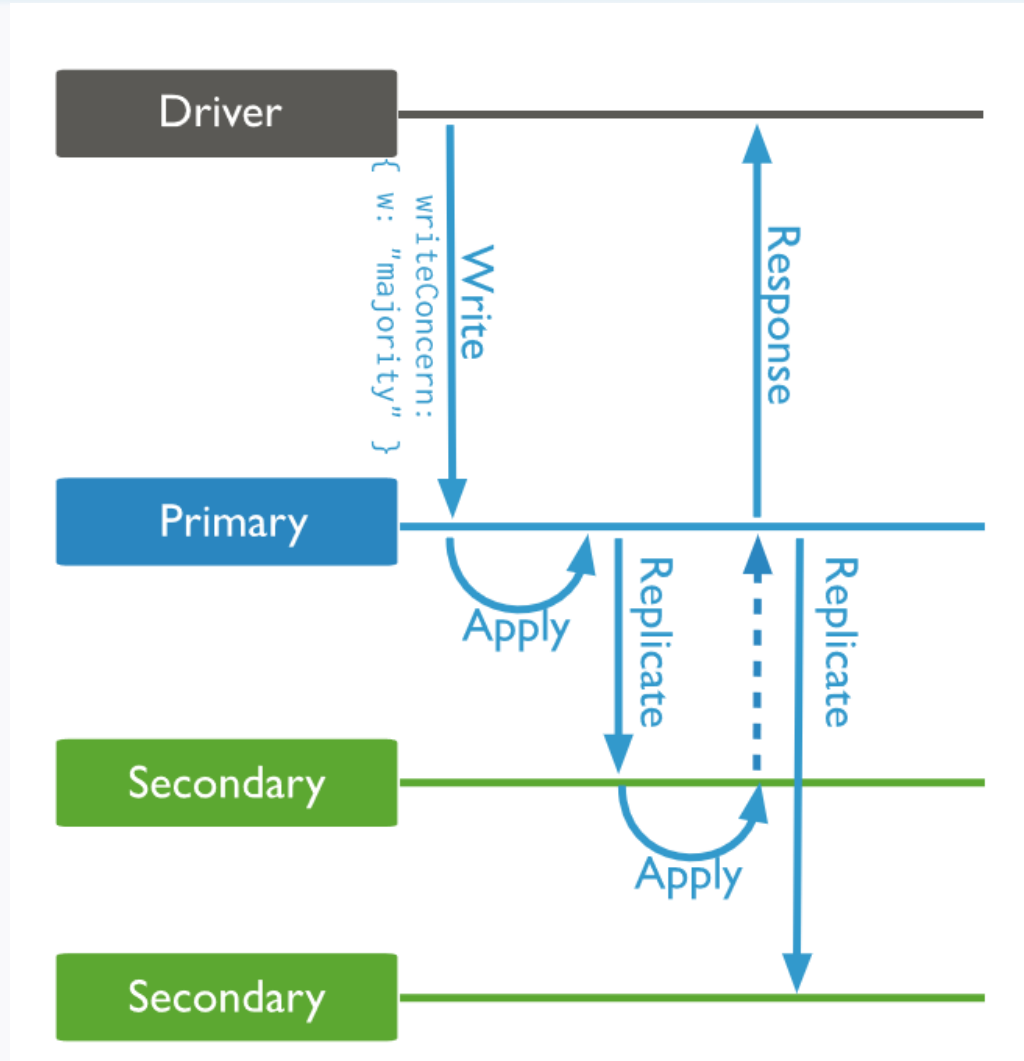
The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band that contains a white network pattern of interconnected dots and lines. The word "Вопросы?" is centered within this band in a large, white, sans-serif font.

Вопросы?

Виды репликации. Read from secondary



Виды репликации. WriteConcern/ Majority



Виды репликации.

Проблемы, которые могут возникнуть при чтении с реплики:

Виды репликации.

Проблемы, которые могут возникнуть при чтении с реплики:

1. так как запись асинхронная, она может быть уже сделана на primary, но не доехать до secondary и мы прочитаем старые данные с secondary

Виды репликации.

Проблемы, которые могут возникнуть при чтении с реплики:

1. так как запись асинхронная, она может быть уже сделана на primary, но не доехать до secondary и мы прочитаем старые данные с secondary
2. отсюда же вытекает что записав данные на основной сервер, мы не можем быть уверены, когда остальные получат эти данные

Виды репликации.

Проблемы, которые могут возникнуть при чтении с реплики:

1. так как запись асинхронная, она может быть уже сделана на primary, но не доехать до secondary и мы прочитаем старые данные с secondary
2. отсюда же вытекает что записав данные на основной сервер, мы не можем быть уверены, когда остальные получат эти данные
3. опять же мы можем установить приоритет на запись или чтение...

Виды репликации.

Проблемы, которые могут возникнуть при чтении с реплики:

1. так как запись асинхронная, она может быть уже сделана на primary, но не доехать до secondary и мы читаем старые данные с secondary
2. отсюда же вытекает что записав данные на основной сервер, мы не можем быть уверены, когда остальные получат эти данные
3. опять же мы может установить приоритет на запись или чтение...
4. ну и самое веселое, когда падает основной сервер и происходят выборы %)

Виды репликации.

Варианты сделать доступной реплику для чтения:

1. `db.slaveOk()`
2. указывать в строке подключения драйвера нужные параметры
<https://docs.mongodb.com/manual/reference/connection-string/>
3. Более точечное использование
`db.collection.find({}).readPref("secondary", [{ "region": "South" }])`

при создании реплики указываем облако тегов для каждого экземпляра,
потом обращаемся по нему

<https://docs.mongodb.com/manual/reference/method/cursor.readPref/>

Виды репликации.

Read Preference Mode / Description

primary

Default mode. All operations read from the current replica set primary.

Multi-document transactions that contain read operations must use read preference primary. All operations in a given transaction must route to the same member.

primaryPreferred

In most situations, operations read from the primary but if it is unavailable, operations read from secondary members.

secondary

All operations read from the secondary members of the replica set.

secondaryPreferred

In most situations, operations read from secondary members but if no secondary members are available, operations read from the primary.

nearest

Operations read from member of the replica set with the least network latency, irrespective of the member's type.

Виды репликации.

Tip

When possible, use a logical DNS hostname instead of an ip address, particularly when configuring replica set members or sharded cluster members. The use of logical DNS hostnames avoids configuration changes due to ip address changes.

Виды репликации.

Вопросы:

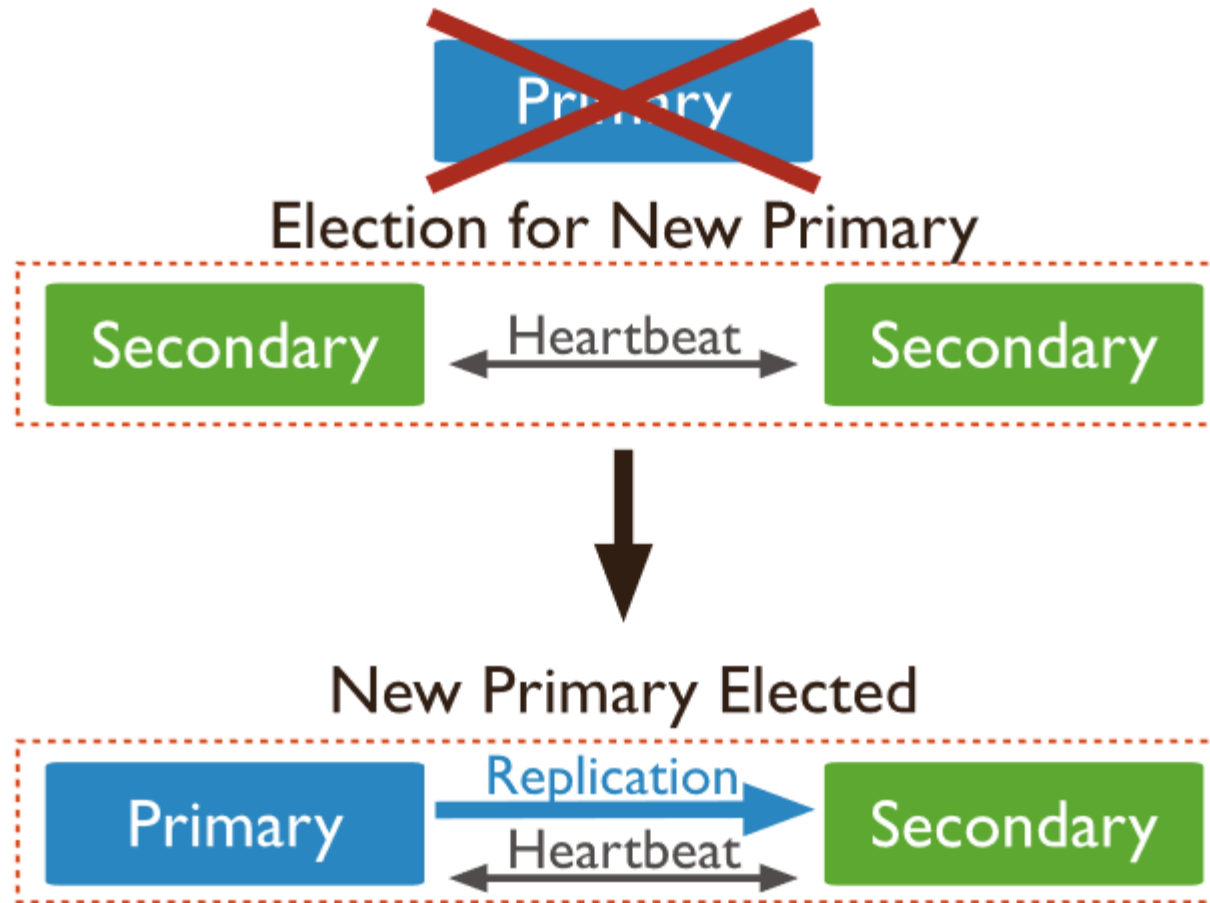
Почему при возвращении 1 сервера в строй происходит его переход из secondary в primary?

Каким образом выбирается новый primary?

The background of the slide features an aerial photograph of a dense urban skyline, likely New York City, with numerous skyscrapers. A semi-transparent blue overlay covers the entire image. Overlaid on this blue background is a white network diagram consisting of numerous small dots connected by thin lines, creating a web-like pattern that suggests connectivity or a digital network.

Концепция кворума

Концепция кворума



<https://docs.mongodb.com/manual/replication/#replication-auto-failover>

Концепция кворума

Особенность для 3 нод:

при конфигурации Primary-Secondary-Arbiter (P-S-A) и writeConcern : **majority** необходимо 2 ноды для согласования записи данных, соответственно при падении Primary или Secondary у нас останется только чтение и неконсистентная запись с “w:1” - подтверждение еще с 1 ноды нужно %)

при конфигурации Primary-Secondary-Secondary (P-S-S) и writeConcern : **majority** при падении 1 ноды запись останется работать

<https://docs.mongodb.com/manual/reference/write-concern/#calculating-majority-count>

Концепция кворума

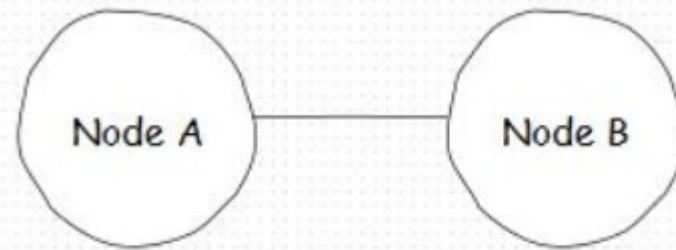
<https://raft.github.io/>

Концепция кворума

При голосовании за новую ноду, должно проголосовать округленное значение $N/2 + 1$, где N , это количество хостов, которым разрешено голосовать

Data Nodes	$N/2$	Voting Minimum	Fault Tolerance	Fault Tolerance w/ Arbiter	Split Brain Possible
2	1	2	0	1	True
3	1.5	2	1	2	
4	2	3	1	2	True
5	2.5	3	2	3	
6	3	4	2	3	True
7	3.5	4	3	4	

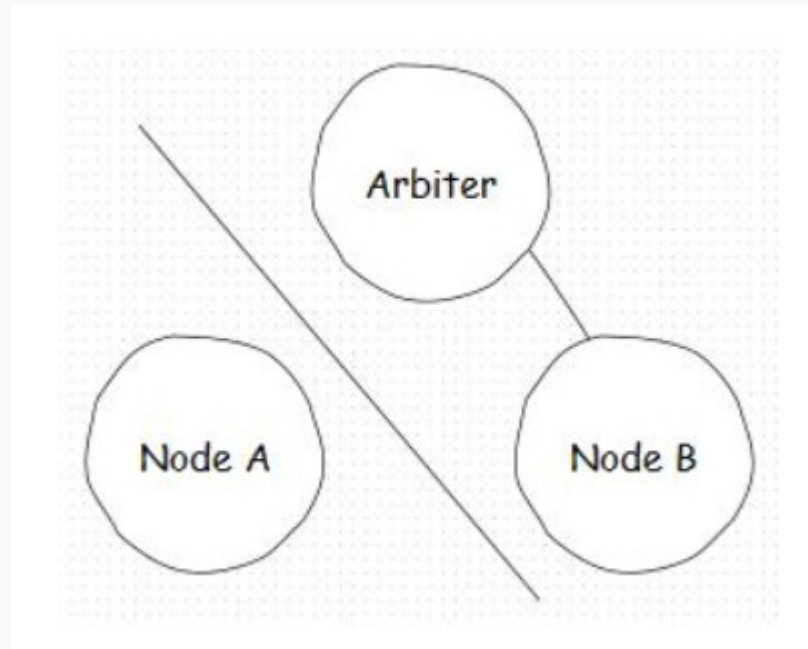
Split brain



If they lose communication, the outcome is symmetrical:



Split brain



Концепция кворума

Голосовать могут только ноды со статусами:

- [PRIMARY](#)
- [SECONDARY](#)
- [STARTUP2](#)
- [RECOVERING](#)
- [ARBITER](#)
- [ROLLBACK](#)

Концепция кворума

members[n].priority

int from 0 до 1000 for primary/secondary; 0 or 1 for arbiters.

Default: 1.0 for primary/secondary; 0 for arbiters.

Changing the balance of priority in a replica set will trigger one or more elections. If a lower priority secondary is elected over a higher priority secondary, replica set members will continue to call elections until the highest priority available member becomes primary.

```
> cfg = rs.conf()  
> cfg.members[3].priority = 2  
> rs.reconfig(cfg)
```

members[n].votes - максимум 7 реплик с правом голоса из 50

The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band that contains a white network diagram. This diagram consists of numerous small dots connected by thin white lines, creating a web-like structure that spans the width of the slide. Centered within this band is the Russian word "Вопросы?" in a large, bold, white sans-serif font.

Вопросы?

The background of the slide is a blue-tinted aerial photograph of a dense city skyline, likely New York City. Overlaid on this image is a semi-transparent network pattern consisting of white dots connected by thin white lines, creating a web-like structure across the center of the slide.

Масштабирование

Масштабирование

Проблема - все тормозит, репликасет не спасает?

Масштабирование

Проблема - все тормозит, репликасет не спасает?

1. отключаем подтверждение записи primary большинством голосов {w:1, j:true}

Масштабирование

Проблема - все тормозит, репликасет не спасает?

- 1. отключаем подтверждение записи primary большинством голосов {w:1, j:true}**
- 2. отключаем журналирование {w:1, j:false}**

Масштабирование

Проблема - все тормозит, репликасет не спасает?

- 1. отключаем подтверждение записи primary большинством голосов {w:1, j:true}**
- 2. отключаем журналирование {w:1, j:false}**
- 3. отключаем любые проверки {w:0, j:false}**

Масштабирование

Проблема - все тормозит, репликасет не спасает?

- 1. отключаем подтверждение записи primary большинством голосов {w:1, j:true}**
- 2. отключаем журналирование {w:1, j:false}**
- 3. отключаем любые проверки {w:0, j:false}**
- 4. что дальше?**

<https://docs.mongodb.com/manual/reference/write-concern/>

<https://habr.com/ru/post/335698/>

The background of the image is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue gradient. A network of white lines and dots, resembling a digital or social network, is superimposed over the blue area, particularly concentrated around the central text.

Шардирование

Шардирование

Есть коллекция, мы ее хотим как-то раскидать по шардам.

Для этого **MongoDB делит коллекцию на чанки** с помощью **ключа шардирования**, пытаюсь поделить их в пространстве шард-ключа на равные кусочки.

Дальше включается балансировщик, который старательно **раскладывает эти чанки по шардам в кластере**. Причем балансировщику все равно, сколько эти чанки весят и сколько в них документов, так как балансировка идет в чанках поштучно.

Шардирование

Минусы:

- **С шардированием обязательно будет усложняться код** — во многие места придется протащить ключ шардирования. Это не всегда удобно, и не всегда возможно. Некоторые запросы пойдут либо broadcast, либо multicast, что тоже не добавляет масштабируемости. Подходите к выбору ключа по которому пойдет шардирование аккуратнее.
- **В шардированных коллекциях ломается операция count**. Она начинает возвращать число больше, чем в действительности — может соврать в 2 раза. Причина лежит в процессе балансировки, когда документы переливаются с одного шарда на другой. Когда документы перелились на соседний шард, а на исходном еще не удалились — count их все равно посчитает.
- **Шардированный кластер гораздо тяжелее в администрировании**. Девопсы перестанут с вами здороваться, потому что процедура снятия бэкапа становится радикально сложнее. Необходимо большая автоматизация работы.

Шардирование

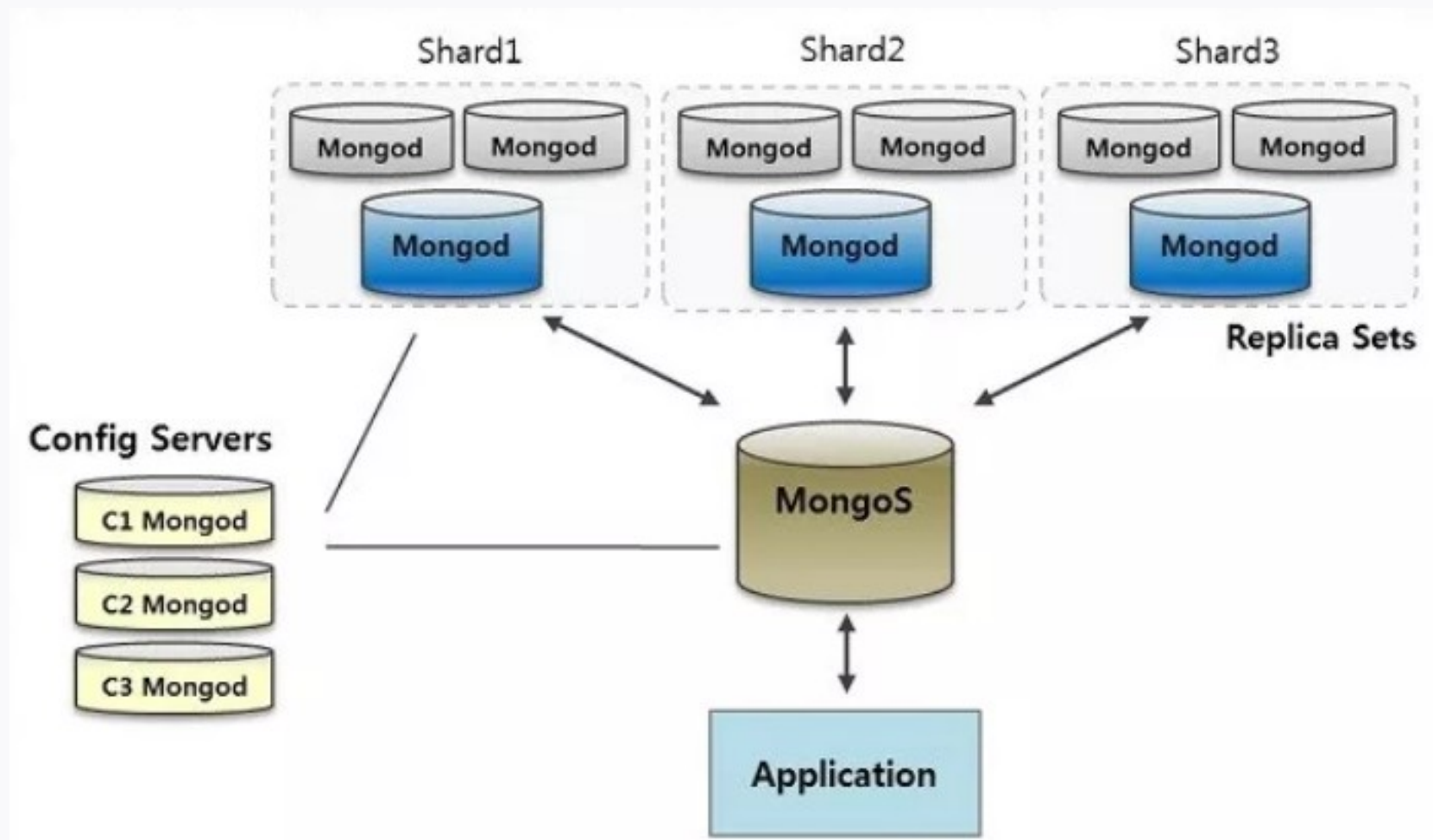
Проблемы:

При включении шардинга у нас есть 2 основные группы проблем.

1. Шардинг коллекции не возможен, если:
 - требуется обновление полей, входящих в ключ шардирования;
 - на коллекции есть несколько уникальных ключей;
 - под результат запроса `findAndModify` попадают данные на разных шардах.

1. Несбалансированная нагрузка по шардам, если:
 - слабая селективность ключа шардирования;
 - запросы без значений ключа шардинга.

Шардирование



Шардирование

В схеме мы можем видеть config сервер, его отличие от mongod в том, что он не обрабатывает клиентские запросы, а является хранилищем метаданных — он знает физические адреса всех chunk-ов, знает какой chunk, на каком шарде искать и какой диапазон у того или иного шарда. Все эти данные он хранит в специально отведенном месте — config database.

В данной схеме также присутствует роутер запросов — mongos, на него возлагаются следующие задачи:

1. Кэширование данных, хранимых на config сервере
2. Роутинг запросов чтения и записи от драйвера — маршрутизация запросов от приложений на нужные шарды, mongos точно знает где физически находится тот или иной чанк
3. Запуск фонового процесса “Балансер”

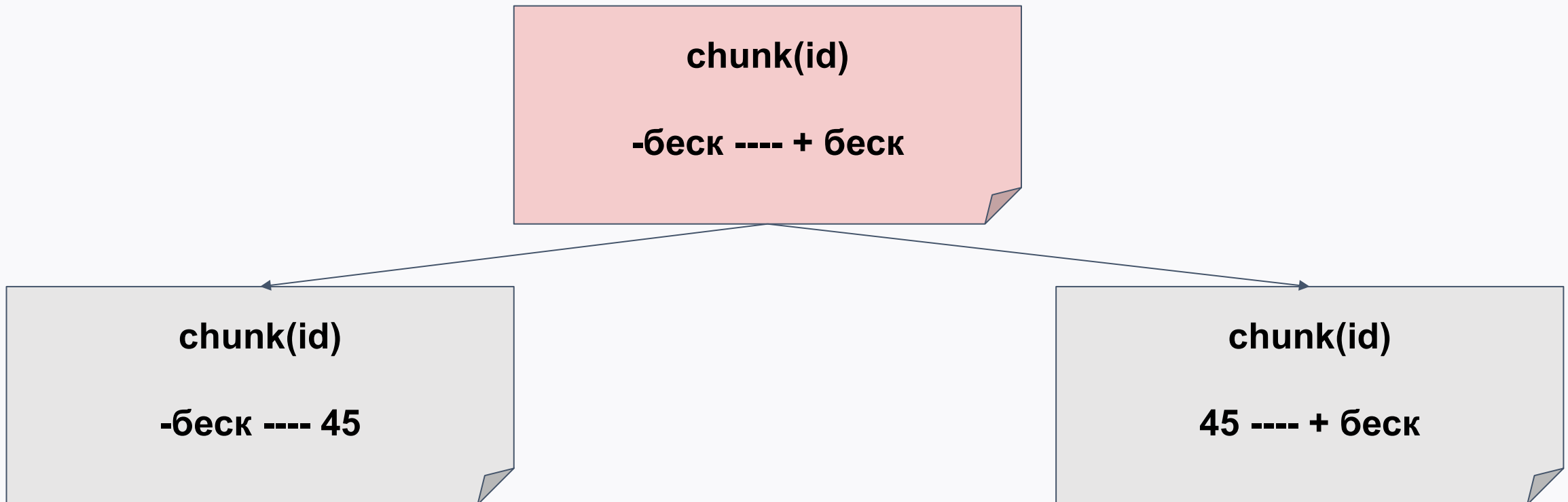
Шардирование

chunk(id)

-беск ---- + беск

Шардирование

{“name”: “Max”, “age”: 23, “id”: 23}
{“name”: “John”, “age”: 28, “id”: 15}
{“name”: “Nick”, “age”: 19, “id”: 56}
{“name”: “Carl”, “age”: 19, “id”: 78}

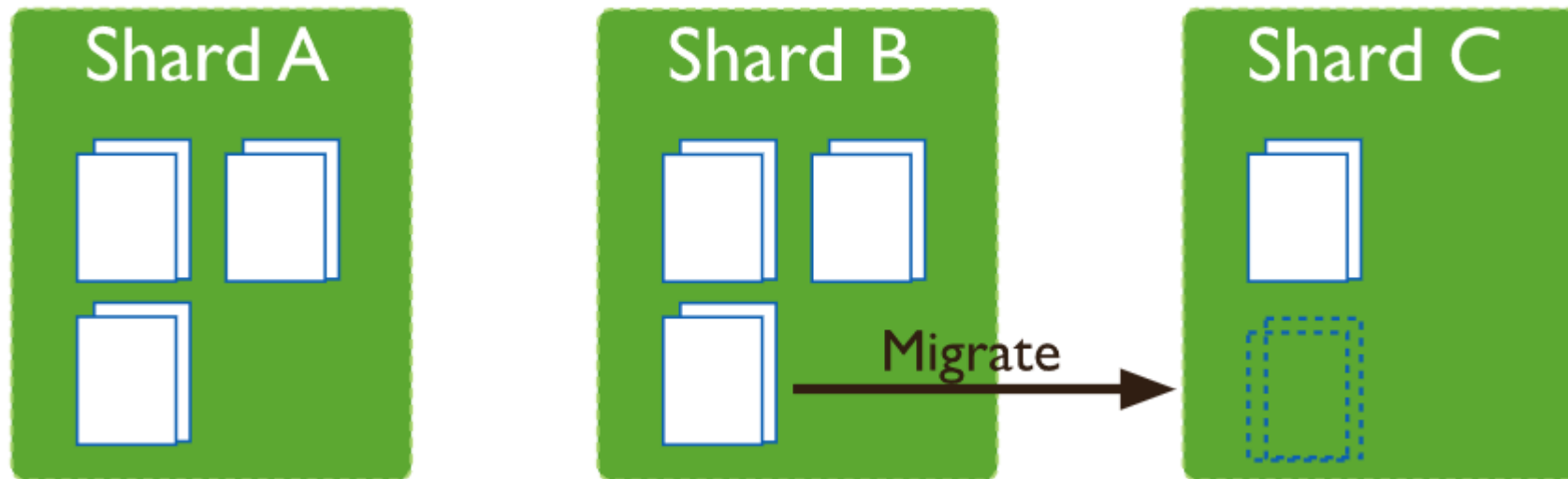


Практика

```
bank.tickets
  shard key: { "amount" : 1 }
  chunks:
    shard0000      7
    { "amount" : { "$minKey" : 1 } } --> { "amount" : 0.000011292286217212677 } on : shard0000 Timestamp(1, 6)
    { "amount" : 0.000011292286217212677 } --> { "amount" : 19.9713398469612 } on : shard0000 Timestamp(1, 7)
    { "amount" : 19.9713398469612 } --> { "amount" : 39.94111982174218 } on : shard0000 Timestamp(1, 1)
    { "amount" : 39.94111982174218 } --> { "amount" : 59.95003548450768 } on : shard0000 Timestamp(1, 2)
    { "amount" : 59.95003548450768 } --> { "amount" : 79.90587004460394 } on : shard0000 Timestamp(1, 3)
    { "amount" : 79.90587004460394 } --> { "amount" : 99.86508572474122 } on : shard0000 Timestamp(1, 4)
    { "amount" : 99.86508572474122 } --> { "amount" : { "$maxKey" : 1 } } on : shard0000 Timestamp(1, 5)
```

```
bank.tickets
  shard key: { "amount" : 1 }
  chunks:
    shard0001      5
    shard0000      5
    { "amount" : { "$minKey" : 1 } } --> { "amount" : 0.000011292286217212677 } on : shard0001 Timestamp(2, 0)
    { "amount" : 0.000011292286217212677 } --> { "amount" : 4.973471583798528 } on : shard0001 Timestamp(3, 0)
    { "amount" : 4.973471583798528 } --> { "amount" : 9.985880833119154 } on : shard0001 Timestamp(4, 0)
    { "amount" : 9.985880833119154 } --> { "amount" : 14.98560153413564 } on : shard0001 Timestamp(5, 0)
    { "amount" : 14.98560153413564 } --> { "amount" : 19.9713398469612 } on : shard0001 Timestamp(6, 0)
    { "amount" : 19.9713398469612 } --> { "amount" : 39.94111982174218 } on : shard0000 Timestamp(6, 1)
    { "amount" : 39.94111982174218 } --> { "amount" : 59.95003548450768 } on : shard0000 Timestamp(1, 2)
    { "amount" : 59.95003548450768 } --> { "amount" : 79.90587004460394 } on : shard0000 Timestamp(1, 3)
    { "amount" : 79.90587004460394 } --> { "amount" : 99.86508572474122 } on : shard0000 Timestamp(1, 4)
    { "amount" : 99.86508572474122 } --> { "amount" : { "$maxKey" : 1 } } on : shard0000 Timestamp(1, 5)
```

Шардирование



Шардирование

Балансировка шардов

Шардируем, запускаем, и внезапно узнаем, что **балансировка не бесплатна**.

Балансировка проходит несколько стадий. Балансировщик выбирает чанки и шарды, откуда и куда будет переносить. Дальнейшая работа идет в две фазы: сначала **документы копируются** с источника в цель, а потом документы, которые были скопированы, **удаляются**.

Шард у нас перегружен, в нем лежат все коллекции, но первая часть операции для него легкая. А вот вторая — удаление — совсем неприятная, потому что уложит на лопатки шард и так страдающий под нагрузкой.

Проблема усугубляется тем, что если мы балансируем много чанков, например, тысячи, то с настройками по умолчанию все эти чанки сначала копируются, а потом приходит удалятор и начинает их скопом удалять. В этот момент на процедуру уже не повлиять и приходится только грустно наблюдать за происходящим.

Поэтому если вы подходите к тому, чтобы шардировать перегруженный кластер, вам нужно планировать, так как **балансировка занимает время**. Желательно это время брать не в пайм-тайм, а в периоды низкой нагрузки. Балансировщик — отключаемая запчасть. Можно подойти к первичной балансировке в ручном режиме, отключать балансировщик в пайм-тайм, и включать, когда нагрузка снизилась, чтобы позволить себе больше.

Шардирование

sh.stopBalancer()

3.6 можно останавливать миграцию

4.2 также отключает автосплит

sh.getBalancerState()

sh.disableBalancing("students.grades")

поменять размер чанка в Мб

use config

config = db.getSiblingDB("config")

config.shards.updateOne({ "_id" : "<shard>" }, { \$set : { "maxSize" : 1024 } })

или поменять значение по умолчанию

db.settings.save({ _id:"chunksize", value: <sizeInMB> })

Шардирование

```
db.settings.update(  
  { _id: "balancer" },  
  { $set: { activeWindow : { start : "<start-time>", stop : "<stop-time>" } } },  
  { upsert: true }  
)
```

Replace <start-time> and <end-time> with time values using two digit hour and minute values (i.e. HH:MM) that specify the beginning and end boundaries of the balancing window.

Необходимое количество разницы чанков между нодами для старта миграции:

Fewer than 20	2
20-79	4
80 and greater	8

Шардирование

если доступ по сети:

<https://docs.mongodb.com/manual/core/security-mongodb-configuration/>

mongod --bind_ip_all

default localhost

Шардирование

не так давно появилась рекомендация:

при конфигурации конфиг сервера указываем имя, а не IP:

```
rs.initiate(  
  {  
    _id: "<replSetName>",  
    configsvr: true,  
    members: [  
      { _id : 0, host : "cfg1.example.net:27019" },  
      { _id : 1, host : "cfg2.example.net:27019" },  
      { _id : 2, host : "cfg3.example.net:27019" }  
    ]  
  }  
)
```

Шардирование

```
{numInitialChunk: 1000}
```

параметр при создании шардированной коллекции, влияющий, сколько сразу будет создано чанков на разных шардах и данные будут уже туда непосредственно заливаться, иначе будет 1 чанк, он будет наполняться данными и постоянно делиться

<https://docs.mongodb.com/manual/reference/command/shardCollection/>

The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band that contains a white network diagram. This diagram consists of numerous small dots connected by thin white lines, creating a complex web-like structure that spans the width of the slide. In the center of this band, the word "Вопросы?" is written in a large, bold, white sans-serif font.

Вопросы?



Выбор правильного ключа

Выбор правильного ключа для шардирования

У хорошего ключа несколько параметров: **высокий cardinality, немутабельность** и он **хорошо ложится в частые запросы**.

Важно заметить, что при достижении каким-либо чанком неделимого диапазона, например (44, 45], деление происходить не будет и чанк будет расти свыше chunksize. Поэтому следует внимательно выбирать shard key, чтобы это была как можно наиболее случайная величина. Например, если нам надо заполнить БД всеми людьми на планете, то удачными выборами shard key были бы номер телефона, идентификационный налоговый номер, почтовый индекс. Неудачными — имя, город проживания.

Естественный выбор ключа шардирования — это первичный ключ — поле id. Если вам подходит для шардирования поле id, то лучше прямо по нему и шардировать. Это отличный выбор — у него и cardinality хороший, он и немутабельный, но а насколько хорошо ложится в частые запросы — это ваша бизнес-специфика.

Выбор правильного ключа для шардирования

Еще **пример неудачного ключа шардирования**. Есть коллекцию переводов — translations. В ней есть поле language, которое хранит язык. Например, коллекция поддерживает 100 языков и мы шардируем по языку. Это плохо — cardinality количество возможных значений всего 100 штук, что мало. Но это не самое плохое — может быть, для этих целей cardinality достаточно. Хуже, что как только мы пошардировали по языку, мы тут же узнаем, что у нас англоязычных пользователей в 10 раз больше, чем остальных. На несчастный шард, на котором находится английский язык, приходит в десять раз больше запросов, чем на все остальные вместе взятые.

Поэтому надо учитывать, что иногда шард-ключ естественным образом тяготеет к неравномерному распределению нагрузки.

Выбор правильного ключа для шардирования

Придем к практике

Алгоритм выбора может быть следующим:

1. Еще раз вспоминаем ограничения.
2. К существующим индексам добавляем другие со всеми возможными (и осмысленными) комбинациями полей.
3. Перечисляем всех кандидатов на ключи шардинга. По каждому допустимому индексу можно выбрать ключ по его любому префиксу. Например для индекса $\{ a: 1, b: 1, c: 1 \}$ можно выбрать префиксы $\{ a: 1 \}$ и $\{ a: 1, b: 1 \}$ или полный ключ $\{ a: 1, b: 1, c: 1 \}$.
Важно! Если поле в индексе, но ключ шарда построен по префиксу без этого поля, то такое поле остается изменяемым.
4. Для каждого кандидата оцениваем:
 - Селективность, есть ли ограничения на рост минимального чанка.
 - Возможность добавить в каждый поисковый запрос конкретные значения ключа чанка.
 - Возможность исключить обновление поля, попавшего в ключ шардинга.
 - Долю запросов, которые не изолируются на одном шарде.
5. Выбираем ключ с максимальной долей изолированных запросов.

Выбор правильного ключа для шардирования

Шардинг по префиксу индекса с уникальностью

Пример выбора ключа, когда пришлось остановить выбор на менее селективном ключе шардинга.

Сущности в гипотетическом проекте Smartcat:

- Аккаунт (Account) — владелец документов.
- Документ (Document) — документ для перевода (идентификатор глобально уникальный).
- Сегмент (Segment) — предложение в документе.

Поля сегмента:

- accountId — идентификатор аккаунта.
- idInAccount — уникальный идентификатор в рамках аккаунта.
- documentId — идентификатор документа.
- order — порядковый номер сегмента в документе.
- и другие...

Выбор правильного ключа для шардирования

Индексы:

- *{ documentId: 1, idInAccount: 1 }*
- *{ documentId: 1, order: 1 }*
- *{ accountId: 1, idInAccount: 1 }, unique = 1*

Большинство запросов включают в себя `documentId`. Значит, выбирать будем из первых двух индексов, и это исключает шардинг по `accountId`. Каждый документ имеет глобально уникальный идентификатор и не может принадлежать двум аккаунтам.

`idInAccount` в нашем коде известен только в одном поисковом запросе, т.е. при выборе полного ключа шардинга сегменты документа могут быть разделены по разным чанкам, и поисковые запросы могут обращаться к нескольким шардам.

Выбор правильного ключа для шардирования

Рассматриваем вариант шардинга по $\{ documentId: 1 \}$ — это допустимо, т.к. он является префиксом с уникальностью. Мы оцениваем максимальный объем сегментов одного документа — это примерно 50 Mb. Значит, документ отлично входит в размер одного чанка шардинга (он у нас 64 Mb).

Результат:

- Строим индекс с уникальностью по $\{ documentId: 1, idInAccount: 1 \}$
- Включаем шардинг по $\{ documentId: 1 \}$

Выбор правильного ключа для шардирования

Чек-лист при планировании схемы данных и индексов коллекции:

1. Экономим трафик репликации — минимизируем размер обновления документа.
2. Помним о rollback — бизнес-логика должна поддерживать обрыв операций записи, так же как и внезапное выключение сервера (w:majority).
3. Не препятствуем шардингу — на больших коллекциях не заводим больше одного индекса с уникальностью, но лучше вообще не включать уникальность, если это не критично для бизнес-логики.
4. Изолируем поисковые запросы одним шардом — большинство поисковых запросов должно включать в себя значения ключа шардинга.
5. Балансировка шардов — тщательно выбираем тип идентификатора, самый лучший из них — GUID.

The background of the slide is a blue-tinted aerial photograph of a dense city skyline, likely New York City, featuring numerous skyscrapers. Overlaid on this image is a semi-transparent network diagram consisting of a grid of points connected by thin lines, creating a mesh-like pattern across the center of the slide.

Primary shard

Primary shard

<https://docs.mongodb.com/manual/core/sharded-cluster-shards/>



Безопасность кластера

Безопасность кластера. Основные встроенные роли

- **read**
- **readWrite**
- **dbAdmin** - provides the ability to perform administrative tasks such as schema-related tasks, indexing, and gathering statistics. This role does not grant privileges for user and role management
- **userAdmin** - Provides the ability to create and modify roles and users on the current database. Since the userAdmin role allows users to grant any privilege to any user, including themselves, the role also indirectly provides superuser access to either the database or, if scoped to the admin database, the cluster
- **dbOwner** - combines the privileges granted by the readWrite, dbAdmin and userAdmin roles.
- **root**

Посмотрим подробнее <https://docs.mongodb.com/manual/reference/built-in-roles/>

Безопасность кластера

Backup and Restoration Roles

The admin database includes the following roles for backing up and restoring data:

backup - Provides minimal privileges needed for backing up data. This role provides sufficient privileges to use the MongoDB Cloud Manager backup agent, Ops Manager backup agent, or to use mongodump to backup an entire mongod instance.

restore - Provides the necessary privileges to restore data from backups *if* the data does not include system.profile collection data, соответственно нужны права:

- [createCollection](#)
- [createIndex](#)
- [createRole](#)
- [createUser](#)
- [etc...](#)

Безопасность кластера

Роли для кластера:

clusterAdmin - This role combines the privileges granted by the clusterManager, clusterMonitor, and hostManager roles. Additionally, the role provides the dropDatabase action.

clusterManager - Provides management and monitoring actions on the cluster. A user with this role can access the config and local databases (addShard, dbStats ...)

clusterMonitor - Provides read-only access to monitoring tools, such as the MongoDB Cloud Manager and Ops Manager monitoring agent. (dbStats, listIndexes...)

hostManager - Provides the ability to monitor and manage servers (shutdown, setParameter)

Безопасность кластера

All-Database Roles

- readAnyDatabase
- readWriteAnyDatabase
- userAdminAnyDatabase
- dbAdminAnyDatabase

Безопасность кластера

Виды ресурсов в MongoDB

- db
- collection (строка, например “any”)
- cluster (true/false)
- anyResource (true/false)

Безопасность кластера

Виды действий в MongoDB

- find
- insert
- update
- remove
- addShard
- ...
- anyAction

<https://docs.mongodb.com/manual/reference/privilege-actions/>

Безопасность кластера

практика



ДЗ

ДЗ

нет его


19.11 объемное ДЗ, пока можете потренироваться создавать шардированный реплицированный кластер

The background of the image is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this is a semi-transparent blue band that contains a white network diagram. The diagram consists of numerous small dots connected by thin white lines, creating a web-like structure that spans the width of the band. Centered within this band is the Russian word "Порефлексируем" in a large, bold, white sans-serif font.

Порефлексируем

Вопросы?

- **Какая служба отвечает за доступ к шардированному кластеру и распределяет чанки?**



Заполните, пожалуйста,
опрос о занятии по ссылке в чате
<https://otus.ru/polls/60997/>



Спасибо за внимание!
Приходите на следующие
вебинары

Курочкин Константин