Принципы построения распределенных баз данных

Алексей Никитин, старший заднеконцевик в Bookmate

19.06.2015

Определения

- *Распределенная система* - система, компоненты которой работают на разных хостах и общаются между собой по сети

- *Нода* - узел информационно-вычислительной сети, на котором работает распределенная система

Восемь заблуждений о распределенных вычислениях

1. Сеть надежна
2. Латентность нулевая
3. Пропускная способность бесконечна
4. Сеть безопасна
5. Топология никогда не меняется
6. Администратор всегда только один
7. Цена передачи данных нулевая
8. Сеть однородна

Виды распределенных систем

- Statless (без состояния, сервисы)
- Statefull (с состоянием, БД или ведет себя как БД)

Примеры распределенных statefull систем

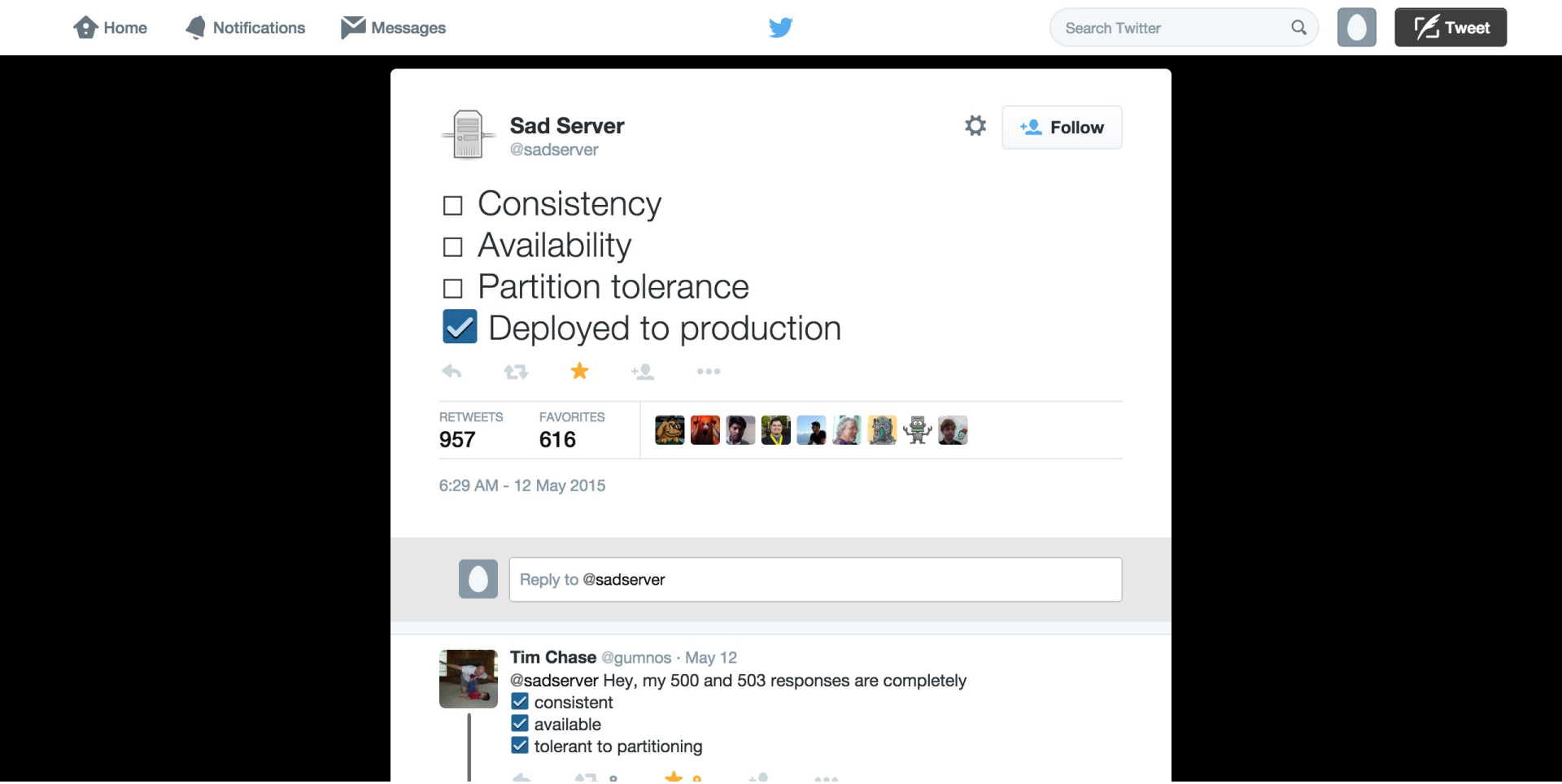
- РСУБД с репликацией и/или шардингом
- NoSQL (riak, cassandra)
- Service discovery (etcd, consul.io, zookeeper)
- РСКВ (git, mercurial)

CAP-теорема

Эрик Брюер в 2000 году на симпозиуме по распределенным вычислениям сформулировал знаменитую CAP-теорему

- consistency (согласованность)
- availability (доступность)
- partition tolerance (устойчивость к разделению сети)

Можно выбрать любые два свойства


Home
Notifications
Messages
Search Twitter
Tweet
Sad Server
@sadserver
Follow
☐ Consistency
☐ Availability
☐ Partition tolerance
☑ Deployed to production
RETWEETS
957
FAVORITES
616
6:29 AM - 12 May 2015
Reply to @sadserver
Tim Chase @gumnos · May 12
@sadserver Hey, my 500 and 503 responses are completely
☑ consistent
☑ available
☑ tolerant to partitioning

DevConf

Home Notifications Messages Search Twitter Tweet

Stuart Sierra
@stuartsierra

Follow

Misunderstand CAP, misunderstand ACID: pick 2

RETWEETS 15 FAVORITES 9

1:56 PM - 21 May 2015

Reply to @stuartsierra

Trends

#EAE3 #ЭтимЛетомЯ Jeb Bush #ДжонСноуВернись #XboxE3 FIFA 16 Bud Black
No Type Speed High School Musical 4

CAP-теорема

- не теорема, а "эвристическое утверждение"
- consistency (ничего общего с буквой `C` в `ACID`, скорее `A`)
- нельзя отказаться от partition tolerance в реальном мире (см. заблуждение 1)
- в 2002 Гилберт и Линч сформулировали ее более строго, чтобы было больше похоже на теорему

CAP-теорема (Gilbert and Lynch)

- согласованность, как атомарность и линеаризумость; эквивалентно упорядоченному выполнению всех запросов на одной ноде
- доступность: каждый запрос полученный неотказавшей нодой должен отдать ответ
- отказ ноды приравнивается к разделению сети; невозможно построить распределенную систему, которая будет "устойчива к разделению сети"

CP/AP дихотомия

поэтому остается выбирать между CP и AP (то есть в условиях разделения сети система должна решить: либо быть доступной, либо согласованной)

- в случае разделения сети (или отказа нод(ы)) система перестает отвечать на запросы чтобы сохранить согласованность
- в случае разделения сети (или отказа нод(ы)) система продолжает обрабатывать запросы на запись, при этом теряя согласованность, в надежде что-то с этим сделать после восстановлении сети

Eventual consistency

- last write wins (Cassandra)
- слияние результатов с ручным разрешением конфликтов (git)
- разрешение конфликтов на уровне бизнес-логики приложения aka semantic conflict resolution (Amazon's Dynamo)
- Lamport timestamps, vector clocks и подобные алгоритмы aka частичное упорядочивание ивентов (Riak)

Lamport timestamps

Логические часы, инкрементальный программный счетчик.

- актор делает инкремент перед каждой отправкой изменений
- актор посылает сообщения вместе со значением счетчика
- актор обновляет свой счетчик на значение из полученного сообщения, если он больше текущего

где актор - это часть системы, которая может вносить изменения в состояние

Vector clocks

- Parent and child

```
{a: 1, b: 1, c: 3, d: 1}
{a: 1, b: 2, c: 7, d: 2}
```

- Siblings:

```
{a: 1, b: 3, c: 3, d: 1}
{a: 1, b: 1, c: 4, d: 2}
```

Strong eventual consistency

Conflict-free replicated data types (CRDT)

- Операция изменения монотонно увеличивает значение
- Операция мерджа коммутативна, ассоциативна и идемпотентна

Increment-only counter

- update operation: `inc`
- merge operation: `max`

```
merge({a: 1, b: 5}, {a: 2, c: 2}) =
   {a: 2, b: 5, c: 2}

value({a: 2, b: 5, c: 2}) = 9
```

PN-counter

```
P = {a: 10, b: 2}
N = {a: 1, c: 5}

value = P − N = (10 + 2) − (1 + 5) = 6
```

G-Set

update operation: `add_element`

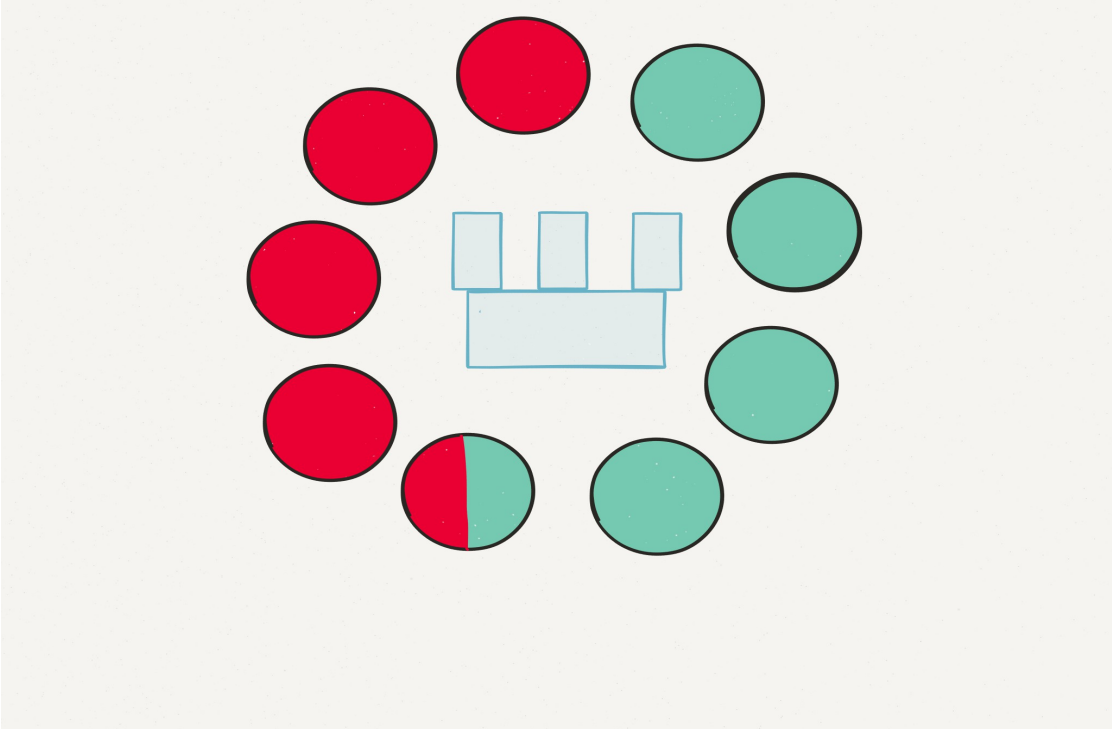
merge operation: `union`

Strong consistency

Виды ошибок в распределенных системах

- Византийские
- Невизантийские

Византийские генералы



Кворум

Лесли Лэмпорт говорит, чтобы оставаться доступными и согласованными, мы можем потерять не больше N узлов

- из 2N + 1 в условиях невизантийских ошибок (кворум N+1)
- из 3N + 1 в условиях византийских ошибок (кворум 2N+1)

Консенсус

- Задача: аккуратно и надежно вести лог репликации
- Клиенты ходят только к лидеру
- Как только получили кворум, происходит коммит и ответ клиенту
- Если лидер вывалися из кластера (ответ не укладывается в таймаут), то выбираем нового!
- Дает самые сильные гарантии, но и оверхед большой

Паксос

- Канонический. От самого Лесли Лэмпарта
- В честь греческого острова
- Есть византийская версия
- Считается сложным для понимания
- Используется в zookeeper, riak, cassandra, voldemort

Рафт

- Потому что паксос сложный
- Сильно проще, но дает такие же гарантии
- Используется в etcd, consul.io

Вопросы

twitter: @tank_bohr