Программирование распределенных систем: DKVS - Курсовая Работа

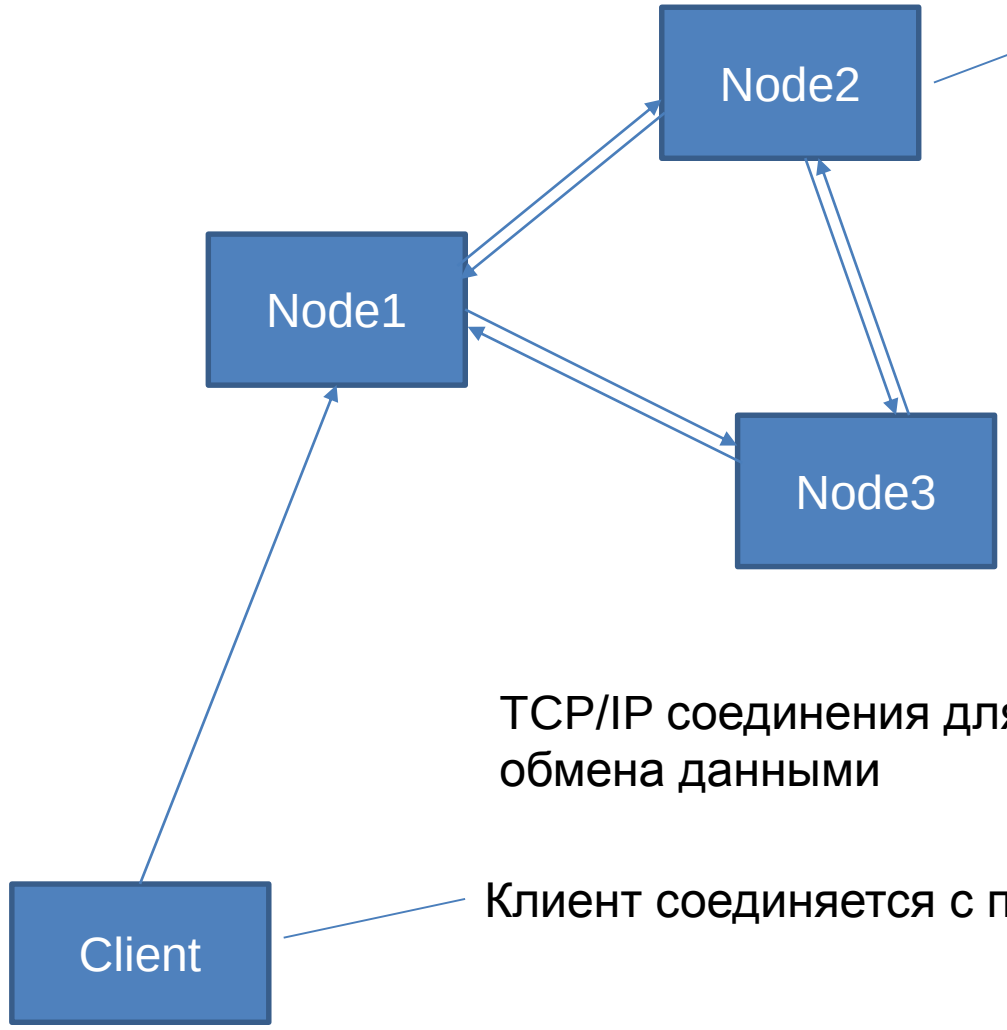
Роман Елизаров, 2016

elizarov@devexperts.com

Distributed Key Value Storage

- Хранит набор key = value, поддерживает операции get/set/delete
- Распределенное хранилище с 2f +1 узлами
  - Отказ f узлов не останавливает работу системы (может только затормозить на какое-то время)
- Каждый узел ведет журнал на диске
  - При останове всей системы и подъеме снова все данные сохраняются
- Согласованность данных
  - Каждый узел видит одинаковую картину мира (консенсус насчет текущего значения для каждого ключа)

Компоненты



Для f=1

Клиентский протокол

- Простой текстовый протокол (вдохновлен протоколом memcached)
  - Ключи текстовые без пробелов, данные без переводов строк
  - Должен понимать любой перевод строки для простоты отладки

`get <key>`	⇒	`VALUE <key> <data>`
		`NOT_FOUND`
`set <key> <value>`	⇒	`STORED`
`delete <key>`	⇒	`DELETED`
		`NOT_FOUND`
`ping`	⇒	`PONG`

Один порт

- Каждый узел слушает на одном порту (указанном в конфигурационном файле)
- Узлы подключаются друг к другу через тот же порт, что и клиенты

```
node <i>
```

```
ACCEPTED
```

- Формат дальнейших сообщений на ваше усмотрение, но должен быть текстовым
- При двукратном подключении одного и того же узла – разрыв *более старого* соединения
- Между каждой парой узлов два соединения, их использование на ваше усмотрение

Конфигурация и запуск узла

- Файл **dkvs.properties** (одинаковый для всех узлов)

```
node.1=<ip>:<port>
node.2=<ip>:<port>
…
node.<n>=<ip>:<port>
timeout=<millis>
```

- При запуске узла в командной строке указывается его номер

```
dkvs_node <i>
```

- Для удобства отладки запуск без аргументов – запуск всех

Журнал

- Каждый узел пишет в файл **dkvs_<i>.log** журнал, необходимый ему для восстановления своего состояния при перезапуске
  - Использовать текстовый, человеко-читаемый формат
  - Не заморачиваться на сжатие журнала (только дописывать туда информацию)

Алгоритмы консенсуса (в вариантах)

- (1) Multi Paxos
  - http://www.cs.cornell.edu/courses/cs7412/2011sp/paxos.pdf
- (2) View Stamped Replication
  - http://pmg.csail.mit.edu/papers/vr-revisited.pdf
- (3) Raft
  - https://ramcloud.stanford.edu/raft.pdf

Алгоритм выбора варианта курсовой работы:

```
String s = “<фамилия-студента-большими-русскими-буквами>”;
int x = (s.hashCode() & 0x7fffffff) % 3 + 1;
```

Пример ЕЛИЗАРОВ → 2

Консенсус

- Каждый узел знает кто сейчас лидер и использует соответствующий алгоритм консенсуса для операций set/delete
- Операцию get отрабатывать всегда из локального состояния (без консенсуса)
  - Локальное состояние для get должно отражать результат только тех операций, которые не могут быть отменены
  - Из-за этого может нарушаться причинная согласованности между разными клиентами (один видит новое значение, а другой – еще старое), но и не ставим себе такую цель
- Не стоит задача поддерживать изменение конфигурации системы (добавление или удаление узлов)

Другие замечания

- В алгоритмах не должно быть рандомизации
  - В качестве первого лидера пытаться использовать node.1
- Все узлы каждый timeout шлют друг-другу ping, если ничего другого не происходит
- Узлы считают соседей «мертвыми» и рвут соединение, если в течение timeout нет ответа (в т.ч. нет ответа на ping) или ошибка соединения
  - В любом случае пытаемся соединиться заново
- Каждый узел должен на консоль писать всё, что происходит
  - Комментировать все фазы алгоритма (получение сообщений и посылки ответов)

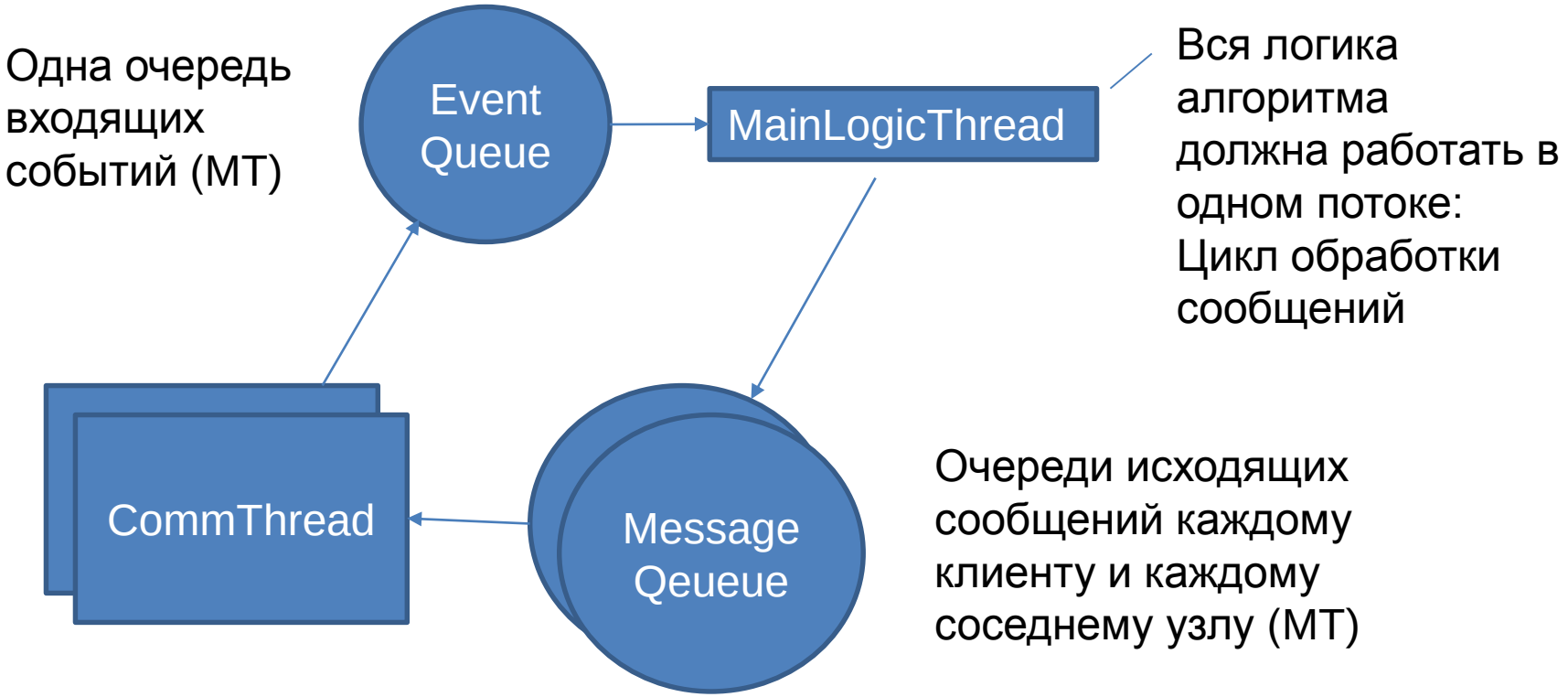
DEVEXPERTS

Дизайн кода

- Два варианта на ваш выбор
  - Многопоточный
  - Однопоточный
- Дальше идет описание дизайна
  - Это рекомендации. Так вам будет проще.

Дизайн многопоточного кода (1)

- Минимизировать и инкапсулировать всю многопоточность
  - Минимум общих структур данных (очереди, списки соединий)



Дизайн многопоточного кода (2)

- Рекомендую использовать
  - **java.lang.Thread**
  - **java.io.ServerSocket/Socket**
  - **java.util.concurrent.ConcurrentLinkedQueue**
- Инкапсулировать все события в отдельные объекты

Старт узла

- Старт узла должен производиться в таком порядке
  - Чтение журнала и восстановление текущего состояния
  - Запуск всех коммуникационных потоков:
    - Потока, который слушает соединения от клиентов и от других узлов
      - ServerSocket.accept в цикле
      - На каждой принятое соединение – новый поток
    - Потоков, устанавливающих соединение к каждому узлу-соседу
      - Socket.connect в цикле
      - После установления соединения работа до его разрыва и установление соединение заново

Дизайн однопоточного кода

- Позволяет сконцентрироваться только на аспекте того, что это распределенная система
- Использовать пакет **java.nio** для обработки всех соединений в одном потоке

Организационные моменты

- Успешная сдача – допуск к экзамену.
- Первая попытка сдачи (защиты) решения – **23 Мая**
- Если к дедайну вы сделаете *качественный рабочий* код + алгоритм распределенного лидера (разные лидеры на разные ключи) используя для этого технику “consistent hashing”
  - то автоматом 4 на экзамене
- Другой способ получит 4-ку на экзамене – успешное выступление на семинарах
  - Первый семинар – 3 доклада про алгоритмы консенсуса.
  - 30 мин выступление (обязательно со слайдами)
- Для получение «отлично» – сдача теоретического билета + доп. вопросы