Архитектура вычислительных систем

Распределенные приложения

Романюта Алексей Андреевич

alexey-r.98@yandex.ru

Кафедра вычислительных систем
Сибирский государственный университет телекоммуникаций и информатики

Распределенные приложения

- В чем вообще проблемы?
  - Гонки данных, race condition
  - Concurrency – возникновение одновременных событий в распределенной системе
  - Время? У каждого сервера своё и его нужно как-то синхронизировать
  - Периодические отказы серверов/экземпляров приложения

Распределенные приложения

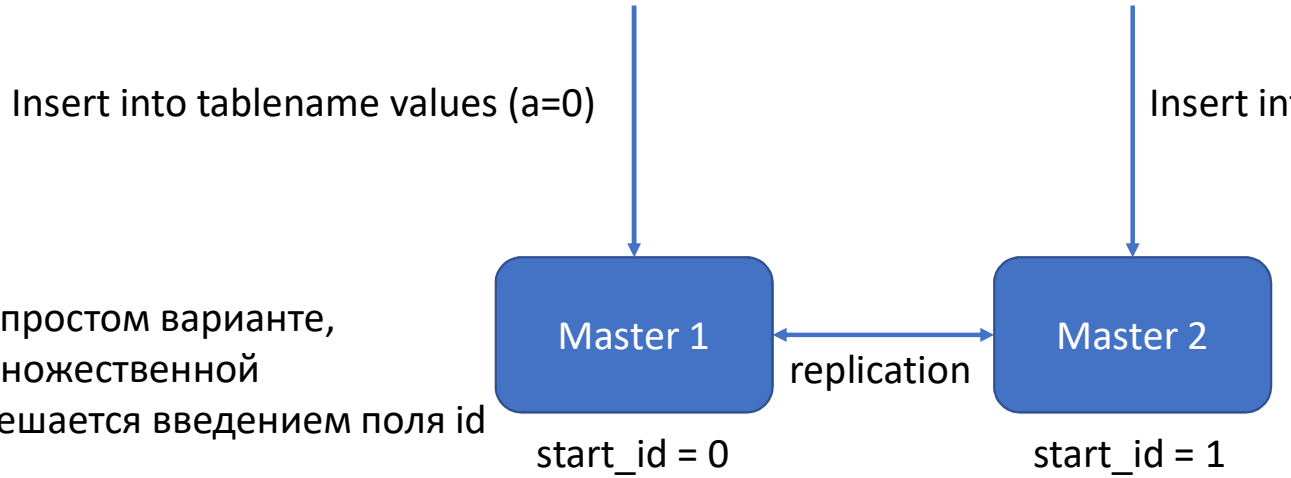
- Каким приложениям нужно друг с другом договариваться?
  - Базы данных
  - Очереди сообщений
  - Координаторы

Распределенные приложения

- Базы данных
  - Распространенный подход - Master-slave, просто пишем в мастер, реплики сами прочитают все изменения. Реплики read only!
  - Multimaster (Postgres, Cassandra, mariadb galera, etc…) – деградация производительности при интенсивной записи
  - Master-slave + coordinator (Patroni)

Распределенные приложения

- Базы данных – multimaster (mariadb/mysql plain)



В самом простом варианте, задача множественной записи решается введением поля id

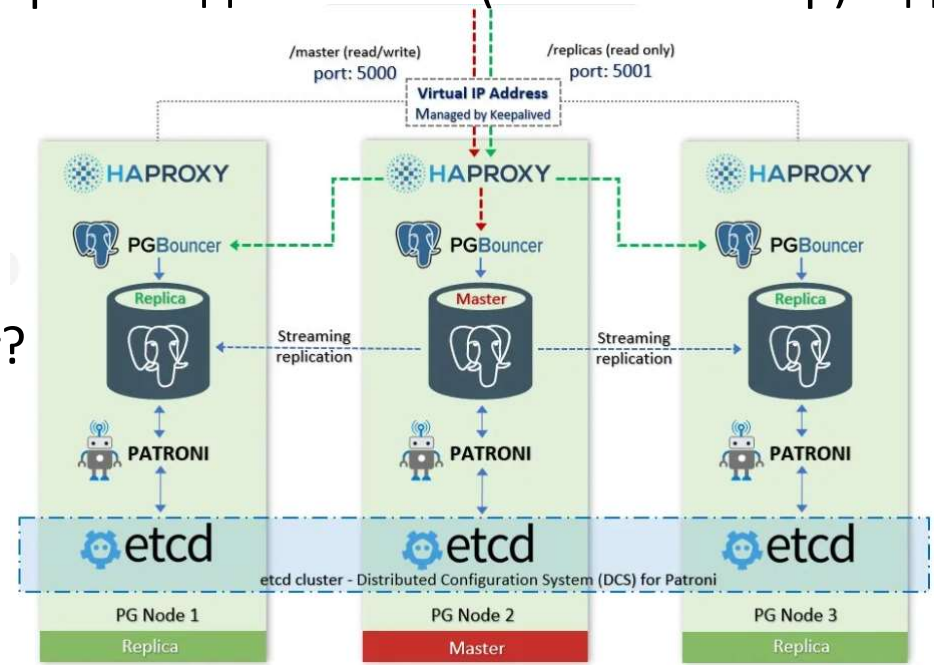
Шаг поля id задается равным количеству мастеров (step = 2)

tablename:

id	a
0	0
1	2

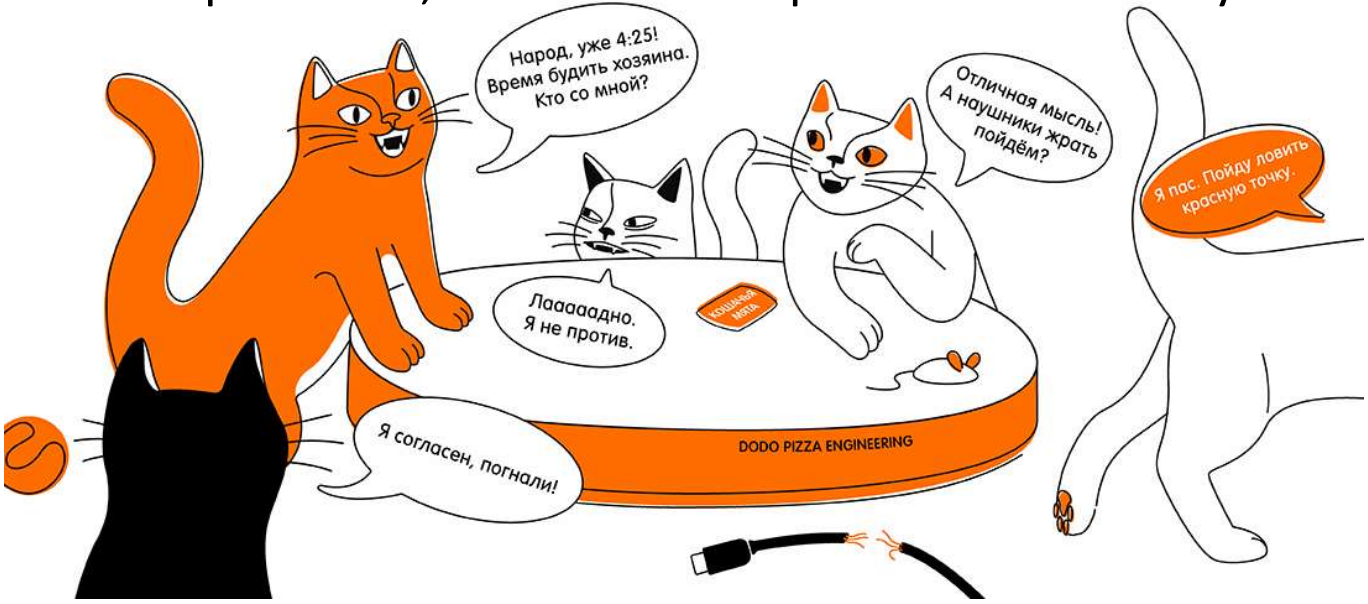
Распределенные приложения

- Базы данных
  - Master-slave + coordinator (Patroni)
  - Добавляется внешний балансировщик и координатор
  - Балансировщик имет два endpoint – для записи (только мастер) и для чтения данных
  - Координатор определяет какой экземпляр будет мастером
  - Как координатор это решает?



Распределенные приложения

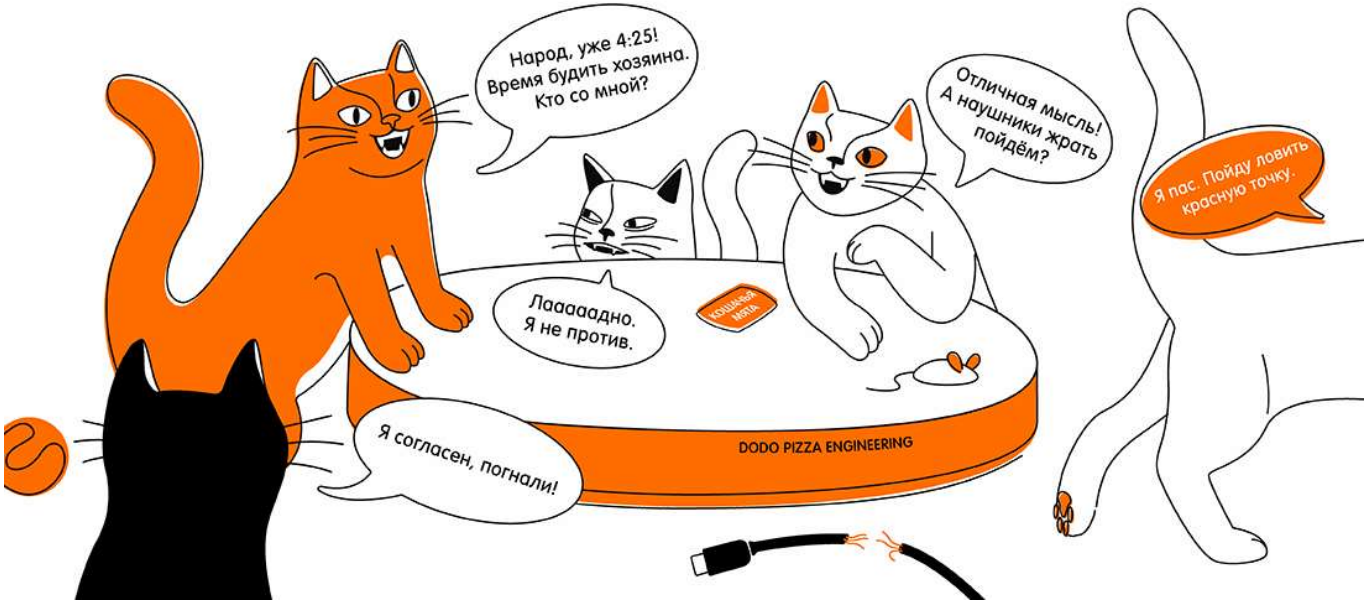
- Что будет, если экземпляры приложения не будут договариваться между собой?
  «В комнате заперты 5 котов, и чтобы пойти разбудить хозяина им необходимо всем вместе договориться между собой об этом, ведь дверь они могут открыть только впятером навалившись на неё. Если один из котов – кот Шрёдингера, а остальные коты не знают о его решении, возникает вопрос: «Как они могут это сделать?»



[картинка отсюда] https://habr.com/ru/companies/dododev/articles/463469/

Распределенные приложения

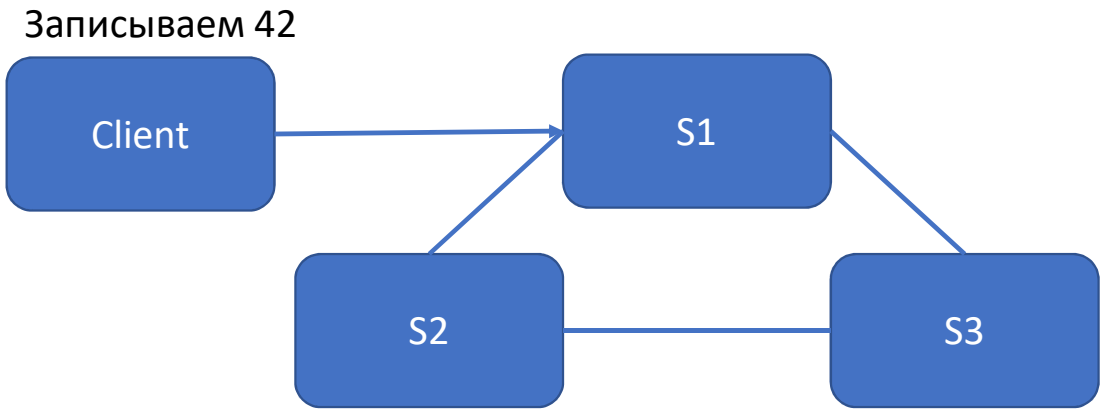
- Каждый экземпляр должен принять для конкретного события одно и то же решение



[картинка отсюда] https://habr.com/ru/companies/dododev/articles/463469/

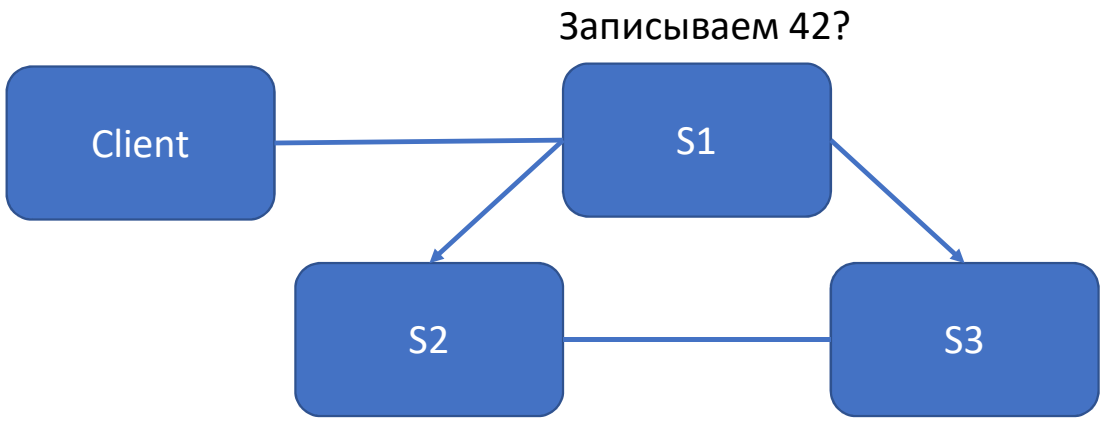
Распределенные приложения

- Каждый экземпляр должен принять для конкретного события одно и то же решение



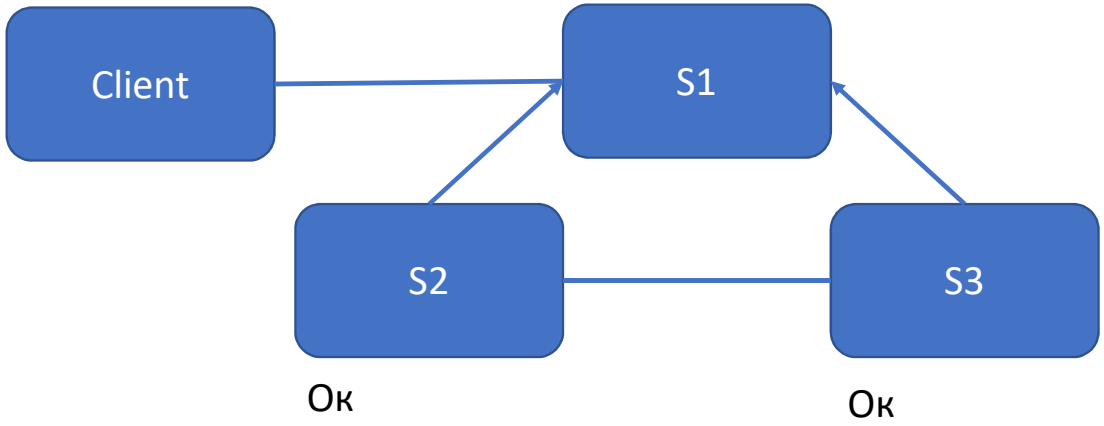
Распределенные приложения

- Каждый экземпляр должен принять для конкретного события одно и то же решение



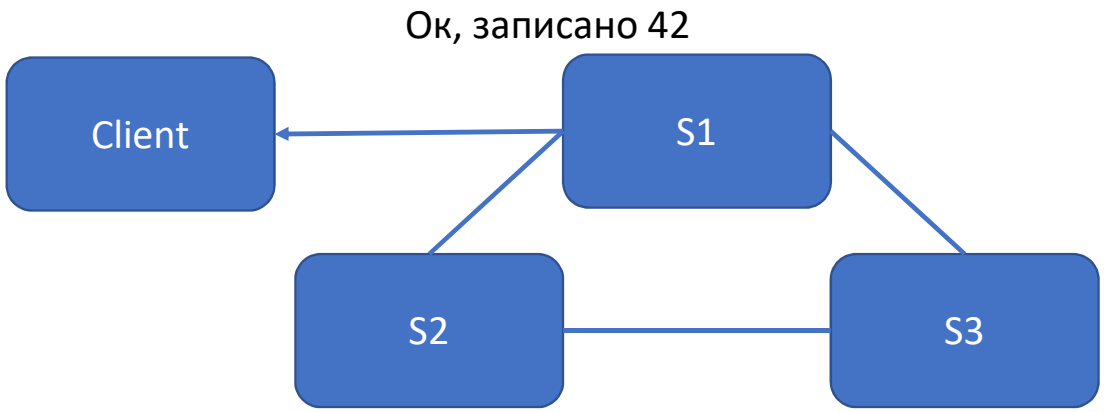
Распределенные приложения

- Каждый экземпляр должен принять для конкретного события одно и то же решение



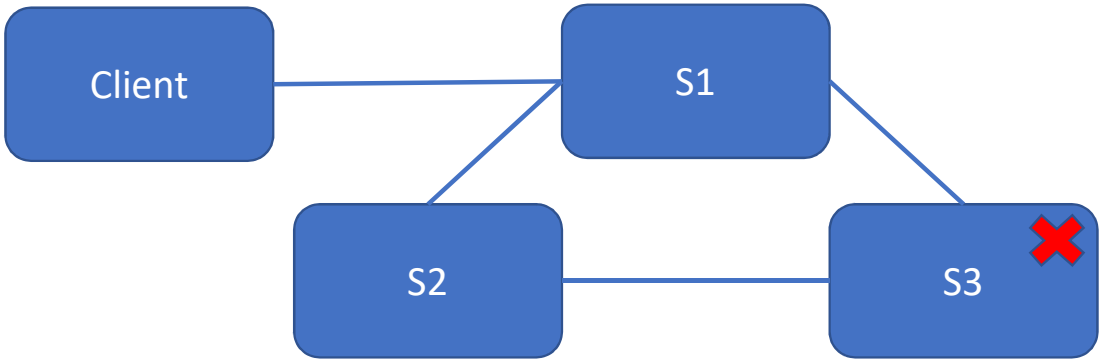
Распределенные приложения

- Каждый экземпляр должен принять для конкретного события одно и то же решение



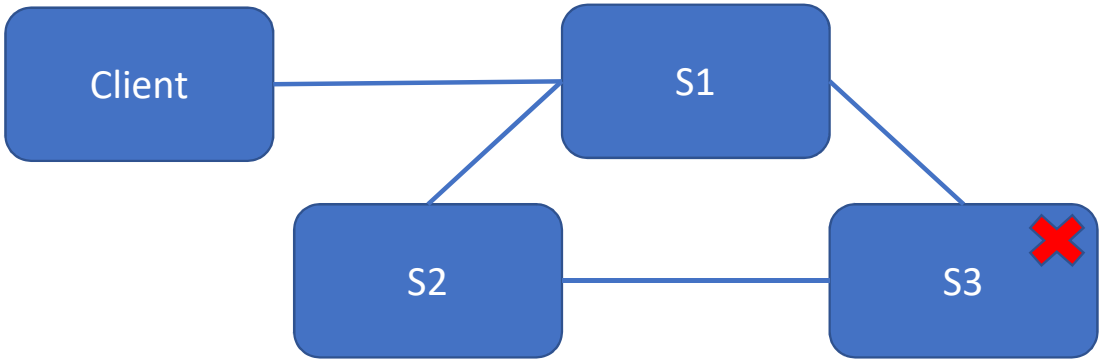
Распределенные приложения

- Каждый экземпляр должен принять для конкретного события одно и то же решение
- В системе отказал один из узлов – конечного решения система принять не сможет. Что делать?



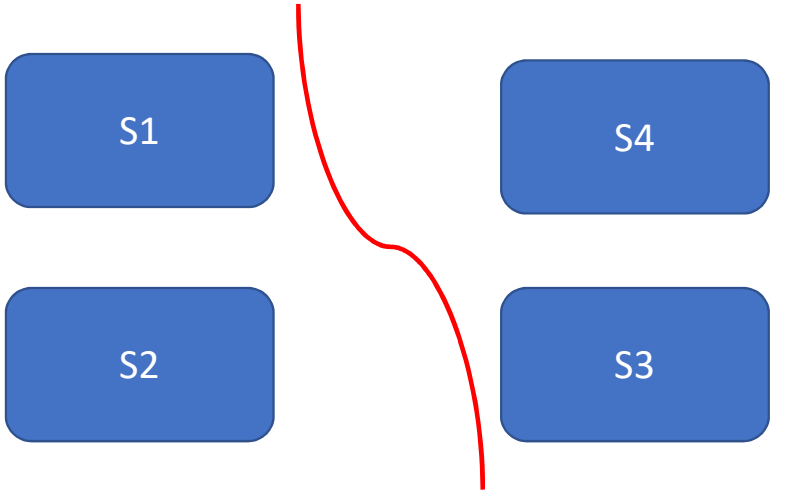
Распределенные приложения

- Каждый экземпляр должен принять для конкретного события одно и то же решение
- В системе отказал один из узлов – конечного решения система принять не сможет. Что делать? Достаточное условие – подтверждение большинства, *кворум*



Распределенные приложения

- Кворум в системе из N экземпляров требует наличие хотя бы (N+1)/2 работоспособных голосов. Отсюда следует, что N должно быть нечетным
- Если N четное, то при отказе половины узлов все еще будет кворум.
- Если узлы на самом деле не отказали, а продолжают работать?



Сетевой связности между s1/s2 и s3/s4 нет,
но попарно узлы друг друга видят

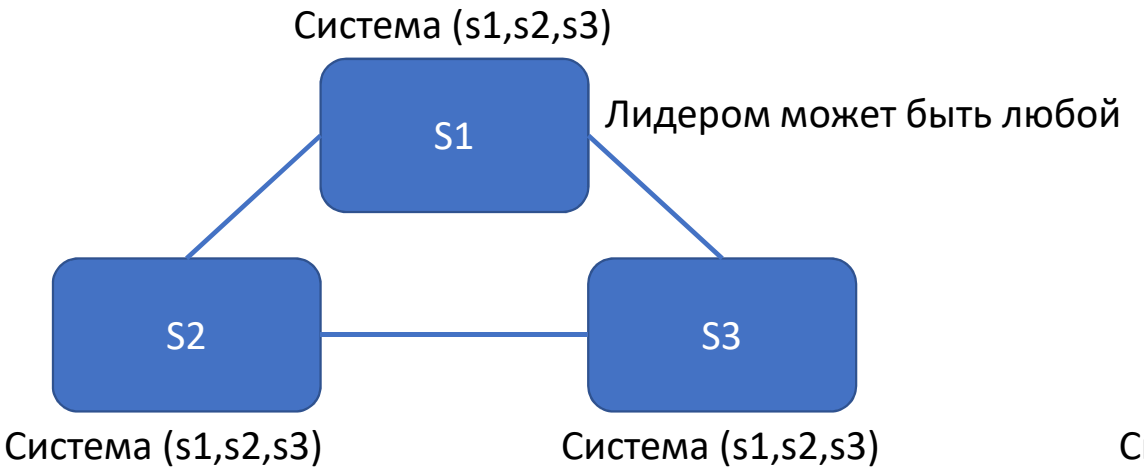
Распределенные приложения

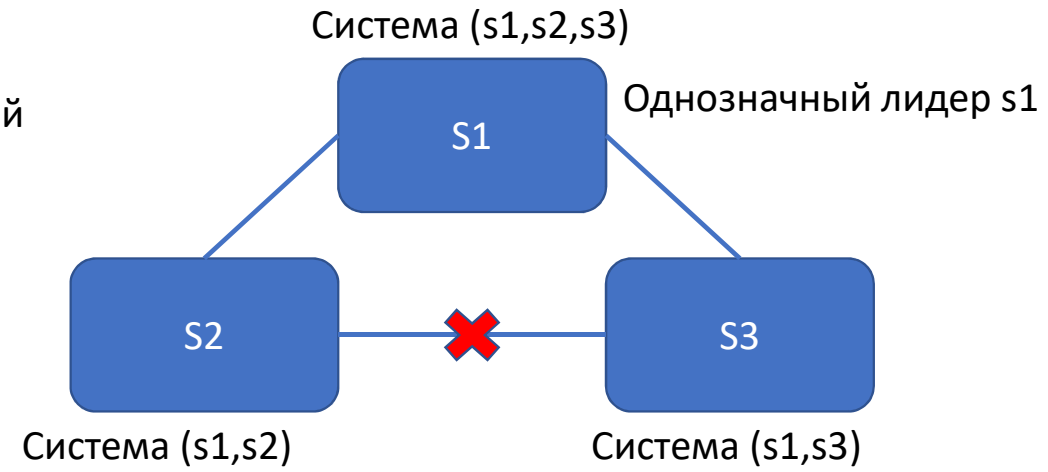
- RAFT – алгоритм для решения задач консенсуса в сети ненадёжных вычислений
- Появляется в системе *лидер* – экземпляр, который видит большую часть системы

[картинка отсюда] https://habr.com/ru/companies/dododev/articles/469999/

Распределенные приложения

- RAFT – алгоритм для решения задач консенсуса в сети ненадёжных вычислений
- Появляется в системе *лидер* – экземпляр, который видит большую часть системы



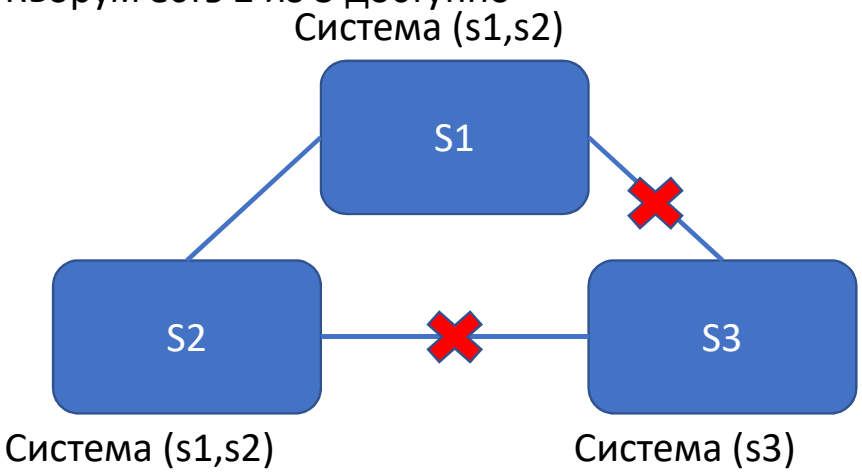


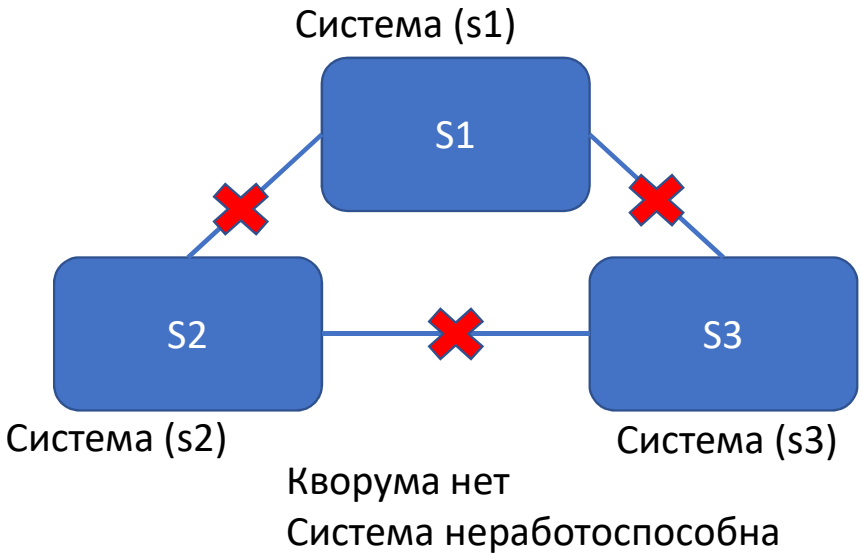
Могут начаться «качели» - система поочередно будет собирать кворум из (s1,s2) и (s1,s3)

Распределенные приложения

- RAFT – алгоритм для решения задач консенсуса в сети ненадёжных вычислений
- Появляется в системе *лидер* – экземпляр, который видит большую часть системы

Лидер s1 или s2
Кворум есть 2 из 3 доступно





Кворума нет
Система неработоспособна

Распределенные приложения

- Где применяется?
  - Системы очередей – kafka (Начиная с v3, v2 работает в связке с zookeeper)
  - Etcd – распределенная key-value база данных. Используется в k8s для хранения состояния кластера и примененных манифестов
  - InfluixDB в режиме кластера использует raft

- Аналоги?
  - Zookeeper использует свою реализацию протокола достижения консенсуса – ZAB (Zookeeper Atomic Broadcast) (Zk использует например clickhouse, kafka v2)
  - Paxos – один из первых протоколов

- Есть визуализация протокола RAFT - https://raft.github.io/

Распределенные приложения

- В георасперделенных системах ключевым показателем являются параметры канала связи – пропускная способность, лаг времени, скорость доступа
- Для георасперделенной системы может применяться подход с кластеризацией систем. В каждом регионе отказоустойчивая система, а между регионами синхронизация достигается путем систем очередей и внешних систем

- В случаях с etcd и zookeeper в требованиях к скорости доступа есть в т.ч. требование к скорости записи на диск и количеству IOps

Live section

Романюта Алексей Андреевич

alexey-r.98@yandex.ru

Кафедра вычислительных систем
Сибирский государственный университет телекоммуникаций и информатики