

# Содержание

1	Введение
---	----------

2
---

# 1 Введение

Я работаю разработчиком программного обеспечения в компании «СКБ Контур», и наша команда занимается разработкой веб-сервиса для электронной отчетности в контролирующие органы. В периоды отчетности нагрузка на сервис растет по сравнению с межотчетными периодами, пик посещений достигается за день до конца отчетности и продолжается вплоть до ее окончания. Если наш продукт станет недоступным в последний день отчетности — тысячи организаций не смогут отчитаться в регламентированный срок, что чревато большими штрафами от контролирующего органа и последующим оттоком клиентов из нашего сервиса. Поэтому сервис должен быть устойчив к высоким нагрузкам.

Отказ системы может произойти по разным причинам, например из-за сетевых неполадок или аварийного завершения работы одной из компонент. Например, если сервер, на котором запущен отвечающий за индексацию накладных сервис, пользователи потеряют доступ к своим данным до устранения проблемы. Эту проблему можно решить с помощью запуска сервиса на нескольких машинах: в таком случае если одна из них выйдет из строя, остальные смогут продолжить обработку запросов. Также данное решение может избавить нас от проблемы с недоступностью из-за сетевых неполадок: выпадение одной реплики сервиса из сети не остановит работу системы.

По тем же причинам необходимо использовать отказоустойчивую базу данных. По «историческим» причинам в нашем сервисе используется Cassandra от Apache: наш отдел к моменту начала разработки нового сервиса имел достаточно большой опыт в использовании и администрировании этого хранилища.

При разработке многопоточных приложений часто встает задача предоставления исключительного доступа к какому-либо разделяемому ресурсу. В большинстве современных языков программирования эта задача решается на уровне синтаксиса (конструкции `lock` в C#, `synchronized` в Java), но только в рамках одного приложения на одной машине. В нашем же случае

мы имеем дело с распределенной системой, где нескольким компонентам может потребоваться доступ к одному и тому же ресурсу. Отсюда возникает задача распределенной блокировки — необходим механизм, позволяющий решать проблему исключительного доступа к ресурсу из разных приложений.

Существуют готовые решения данной задачи — например Chubby от Google и получившийся из него ZooKeeper. Однако внедрение нового стороннего решения влечет за собой множество других задач. Разворачивание, настройка, администрирование — все это занимает достаточно большое количество времени, если подходить к задаче ответственно. В магистерской работе Федора Фоминых данная задача всплыла в ходе построения распределенной очереди задач и было предложено ее решение. Данная работа посвящена выявлению проблем описанного алгоритма и разработке нового решения, в котором исправлены эти проблемы и улучшена эффективность.