МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра компьютерных систем

Направление подготовки: 230100 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ БАКАЛАВРСКАЯ РАБОТА

Разработка менеджера блокировок для распределённых систем

Ильин Александр Владимирович

«К защите допущена»

Заведующий кафедрой,

к. т. н, с. н. с. КТИ ВТ СО РАН

Пищик Б. Н. /..……..

(фамилия , И., О.) / (подпись, МП)

«……»……2014 г.

Научный руководитель

доцент, НГУ,

к. ф.-м. н.

Кренделев С. Ф. /……...

(фамилия , И., О.) / (подпись, МП)

«……»……2014 г.

Дата защиты: «……»……2014 г.

Автор Ильин А. В. /…….....…...

(фамилия , И., О.) / (подпись)

Новосибирск, 2014 г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра систем информатики

Направление подготовки: 230100 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

УТВЕРЖДАЮ
Зав. Кафедрой Пищик Б.Н.
(фамилия, И., О.)
……
(подпись, МП)
«……»……20…г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ БАКАЛАВРСКУЮ РАБОТУ

Студенту Ильину Александру Владимировичу

Тема: «Разработка менеджера блокировок для распределённых систем»

Исходные данные (или цель работы): разработка менеджера блокировок для применения в распределённых системах.

Структурные части работы: анализ необходимого научного материала; формулирование требований к разрабатываемой системе; исследование существующих реализаций менеджеров блокировок, выявление их достоинств и недостатков; проектирование архитектуры менеджера блокировок; реализация менеджера блокировок; тестирование получившегося продукта.

Научный руководитель
доцент, НГУ,
к. ф.-м. н.
Кренделев С. Ф. /…….
(фамилия , И., О.) / (подпись)
«…»……20…г.

Задание принял к исполнению
Ильин А.В. /……..
(ФИО студента) / (подпись)
«…»……20…г.

Содержание

ВВЕДЕНИЕ

В настоящий момент приложениям все чаще приходится работать в распределенных системах. **Распределенная система** – это набор независимых компьютеров, представляющийся их пользователям единой объединенной системой [1]. В связи с тем, что компьютеры объединяются сетью, также будем называть их узлами распределенной системы. Объединение компьютеров в распределённую систему позволяет сосредоточить большое количество вычислительных ресурсов для работы над одной задачей. Тем не менее, необходимо грамотно организовать работу с разделяемыми ресурсами и взаимодействие с другими узлами. **Разделяемым** назовем ресурс, который используется несколькими процессами. Однако не всегда программное обеспечение разрабатывается с расчётом на дальнейшее использование в распределённых системах. С ростом потребительских возможностей и потребностей может возникнуть необходимость объединения компьютеров с некоторым программным обеспечением в распределённую систему, что может потребовать больших изменений в коде при неудачном подходе.

Системы, состоящие из множества процессов, и намеревающиеся использовать разделяемые ресурсы, проще всего программировать с использованием критических секций. **Критическая секция** – часть программы, в которой используется доступ к разделяемому ресурсу [2]. Отсутствие какого-либо контроля над порядком исполнения процессов внутри критической секции может привести к неправильной и непредсказуемой работе программы. Для того, чтобы этого избежать, используется механизм взаимного исключения: каждый процесс, намеревающийся использовать разделяемый ресурс, сначала входит в критическую секцию, чтобы путём взаимного исключения убедиться, что никакой другой процесс не использует этот ресурс одновременно с ним. В централизованных системах (состоящих из одного компьютера) для защиты критических секций используются семафоры, мьютексы и другие примитивы синхронизации. Но данным решением нельзя воспользоваться, если процессы разнесены по нескольким компьютерам, каждый из которых находится под управлением собственной операционной системы.

Для реализации алгоритма взаимного исключения в случае, когда невозможно использовать встроенные механизмы операционной системы, используется менеджер блокировок. Менеджер блокировок – это модуль, который обеспечивает синхронизацию

доступа к разделяемым ресурсам. Менеджер блокировок может быть встроен в код приложения, которому необходимо контролировать доступ к ресурсам, например, в виде библиотеки, а может представлять отдельный компонент – веб-сервис.

Данная работа является частью проекта «Производство и тестирование платформы для организации облачного хостинга веб приложений и пользовательского контента» и выполнена при финансовой поддержке Минобрнауки РФ (договор № 02.G25.31.0054).

Глава 1 Цели и задачи

Целью данной работы является разработка менеджера блокировок для применения в распределённых системах.

В рамках намеченной цели были поставлены следующие задачи:

1) Детально изучить уже существующие реализации менеджеров блокировок и провести анализ задач, которые перед ними возникают.
2) Сформулировать требования к менеджеру блокировок.
3) Разработать архитектуру собственного менеджера блокировок, удовлетворяющую всем поставленным требованиям.
4) Реализовать менеджер блокировок в соответствии с полученной архитектурой
5) Провести тестирование получившегося продукта.

Глава 2 Менеджеры блокировок

С точки зрения менеджера блокировок ресурс – это некоторая сущность, которой соответствует своя блокировка [3]. Каждый ресурс имеет уникальный идентификатор, по которому к нему осуществляется доступ. Ресурс создаётся внутри менеджера блокировок при первом обращении приложения к нему. Далее по тексту понятия ресурса и блокировки будут отождествляться.

2.1 Схемы расположения

При проектировании менеджера блокировок важно учитывать схему расположения менеджера относительно распределённой системы. Можно выделить три общих схемы расположения менеджера блокировок [1]:

1) Централизованная
2) Распределенная
3) Распределенная схема с передачей маркера

Централизованная схема характеризуется тем, что вся логика менеджера блокировок находится на одном узле. Этот узел может быть, как выделенный только для менеджера блокировок, так и один из рабочих узлов распределенной системы. В данном случае все узлы, использующие менеджер блокировок, являются клиентами для него. Централизованная схема не мешает расположить дополнительный код для работы с программным интерфейсом менеджера блокировок на всех узлах распределенной системы. Для получения блокировки все узлы распределенной системы обращаются к выделенному узлу менеджера. Менеджер блокировок отправляет клиенту на его запрос ответное сообщение, когда блокировка получена.

Распределенная схема предполагает, что основная логика менеджера блокировок находится на каждом узле кластера и эти узлы совместно, после посылки сообщений между собой приходят к единому решению по поводу захвата и освобождения блокировок.

Распределенная схема менеджера блокировок с передачей маркера является вариацией распределенной схемы, когда логика контроля над блокировками находится на каждом узле системы, однако, выбор узла, который сможет заблокировать ресурс, основывается на круговой передаче некоторого маркера-сообщения. Узел, у которого в данный момент есть маркер и который в это же время хочет захватить ресурс, получает доступ, остальные узлы ожидают получения маркера.

Согласно Таненбауму [1] централизованный алгоритм наиболее прост и наиболее эффективен. Для того, чтобы войти в критическую секцию, ему достаточно всего трех сообщений — запрос, разрешение на вход и сообщение о выходе. Распределенному алгоритму в самой простой реализации требуется для запроса n - 1 сообщений, по одному на каждый процесс, и дополнительно n - 1 сообщений на разрешение, всего 2(n - 1), где n - количество процессов. В алгоритме маркерного кольца число сообщений различно (табл. 1).

Таблица 1. Сравнение схем расположения менеджера блокировок

Алгоритм	**Число сообщений на вход-выход**	**Задержка перед входом в числе сообщений**	**Возможные проблемы**
Централизованный	3	2	Крах координатора
Распределённый	2(n-1)	2(n-1)	Сбой в одном из процессов
Маркерного кольца	От 1 до ∞	От 0 до n-1	Потеря маркера, сбой в одном из процессов

Задержка с момента, когда процессу понадобилось войти в критическую секцию, до момента входа для этих трех алгоритмов также различна. Если критические секции малы и используются редко, основным фактором задержки является механизм входа в критическую секцию. Если критические секции используются постоянно, основным фактором задержки является ожидание своего хода. В случае централизованного алгоритма, для того чтобы войти в критическую секцию, требуется всего два сообщения. В случае распределенного алгоритма требуется 2(n - 1) сообщений, с учетом того, что они посылаются одно за другим. Для маркерного кольца время варьируется от 0 до n - 1.

Все три алгоритма тяжело реагируют на сбои. Чтобы сбой не привел к полному краху системы, приходится предпринимать специальные меры и дополнительно усложнять алгоритм. Распределенный алгоритм в данной реализации более чувствителен к сбоям, чем централизованный.

Распределенная схема менеджера блокировок устраняет единую точку отказа, однако, при выходе из строя одного из узлов нужно уметь это отследить, потому, что в этой реализации узел, запрашивающий ресурс будет ждать ответа от погибшего узла. В общем случае, если количество узлов в распределенной системе велико, то использование единственного сервера в качестве менеджера блокировок может исчерпать входной канал по запросам, однако, и в распределенном менеджере возникает схожая проблема. Централизованный менеджер также

удобен тем, что можно вынести всю нагрузку, связанную с блокировками на один выделенный узел, в то время как в распределенном варианте часть нагрузки легла бы на каждый узел.

2.2 Структура блокировок

Также при выборе менеджера блокировок важно учесть структуру блокируемых ресурсов и возможности, которые предоставляет менеджер блокировок для одновременной и эксклюзивной работы с ресурсами. Так, например, если ресурсы образуют иерархию, удобно осуществлять работу с блокировками, если менеджер поддерживает иерархическую структуру блокировок, основанную, допустим, на файловой системе. Немаловажно учесть гранулярность блокировок, которая указывает, насколько большие или мелкие ресурсы блокируются. Например, важный аспект работы СУБД заключается в том, блокируется ли вся таблица или только строка таблицы при изменении значения таблицы.

2.3 Реализации менеджеров блокировок

Существующие менеджеры блокировок служат не только для того, чтобы контролировать доступ к разделяемым ресурсам. Среди поддерживаемых ими функций также можно встретить предоставление услуг наименования, хранения метаданных, выбор лидера среди узлов для проведения операций [3-8].

На основе вышеперечисленных особенностей менеджеров блокировок можно сравнить имеющиеся реализации: ZooKeeper, Chubby, OpenDLM (табл. 2).

Таблица 2. Обзор существующих реализаций менеджеров блокировок.

	ZooKeeper	**Chubby**	**OpenDLM**
Синхронизация ресурсов	+	+	+
Служба имен	+	+	-
Выбор лидера	+	+	-
Хранение метаданных	+	+	-
Управление конфигурацией	+	-	-

Система оповещений	+	-	-
Язык реализации	Java	C++	C/C++
Схема менеджера	Распределенный с выделенным лидером	Распределенный с выделенным лидером	Распределенный
На чем основаны блокировки	Иерархическое key/value хранилище в виде файловой системы	Файловая система. Для каждого файла и директории можно захватить «замок» на чтение и на запись	6 режимов захвата.
Условия распространения (тип лицензии)	Apache License Version 2.0	Не распространяется	GNU LGPL Version 2.1

Таким образом, данные реализации подходят для большинства общих задач. Тем не менее, важно учитывать требования к менеджеру блокировок, чтобы выбрать необходимое решение. Возможно, существует ограничение на занимаемый объем оперативной памяти, на быстродействие и прочее.

2.4 Требования к менеджеру блокировок

В рамках поставленной задачи были выдвинуты следующие требования:

- Ограничение по потребляемым ресурсам
- Возможность использования менеджера блокировок как в виде библиотеки, так и в виде веб-сервиса
- Обеспечение одновременного выполнения операций
- Отсутствие лицензионных ограничений
- Удобство внедрения и поддержки
- Расширяемость

Наиболее простой реализацией для внедрения и контроля является централизованная схема расположения менеджера блокировок. Недопустимы реализации, использующие большое количество ресурсов, например, использующие виртуальную машину Java с большим потреблением памяти. А из того, что есть требование на использование программного решения в проприетарном программном обеспечении, можно сделать вывод, что ни один их существующих менеджеров блокировок не удовлетворяет необходимым требованиям и возникает необходимость разработки своего решения.

Глава 3 Архитектура менеджера блокировок

Прежде чем приступать к реализации какого-либо программного продукта, прежде всего, нужно проработать его архитектуру. В данной главе будут рассмотрены сценарии использования менеджера блокировок, а также будет приведено его разбиение на составляющие части.

3.1 Менеджер блокировок с точки зрения клиента

Для клиента работа с менеджером блокировок должна быть похожа на работу с обычными мьютексами. Под **клиентом** понимается любой процесс, использующий в своей работе менеджер блокировок. Клиент делает запрос на блокировку некоторого ресурса с помощью определённой функции. Возврат из функции не происходит до тех пор, пока не будет захвачена запрашиваемая блокировка. Всё это время клиентский процесс находится в ожидании ответа и не потребляет процессорное время. После успешного выхода из функции считается, что клиент получил запрашиваемую блокировку. Стоит упомянуть про try-модификацию захвата блокировки, в которой возврат из функции происходит сразу же, независимо от того, была захвачена блокировка или нет.

Для обеспечения одновременной работы клиентов там, где это возможно, в менеджере блокировок поддерживаются два вида блокировок: на чтение и на запись. Один и тот же ресурс может быть захвачен одновременно несколькими клиентами на чтение, но никогда на запись.

Клиентам часто приходится иметь дело с ресурсами иерархической структуры. Поэтому целесообразно было бы предоставить возможность блокирования целых групп объектов в иерархии, например, таких как «все предки ресурса», «подуровень иерархии» или просто произвольное множество узлов иерархии.

Для предоставления клиентам такой возможности было принято решение объединять все запросы к менеджеру блокировок в группы. Группа блокировок состоит из множества пар «(идентификатор блокировки) – (вид блокировки)».

Также было решено добавить возможность передачи права владения всеми блокировками между клиентами без их освобождения. Для этого был реализован механизм присвоения реального и эффективного идентификатора процесса. Данная идея заимствована из [2]. В *nix системах реальный и эффективный идентификаторы процесса используются для определения прав доступа процесса. При определении владельца блокировки используется только эффективный идентификатор.

Все вышеперечисленные варианты использования менеджера блокировок приведены на рисунке 1.

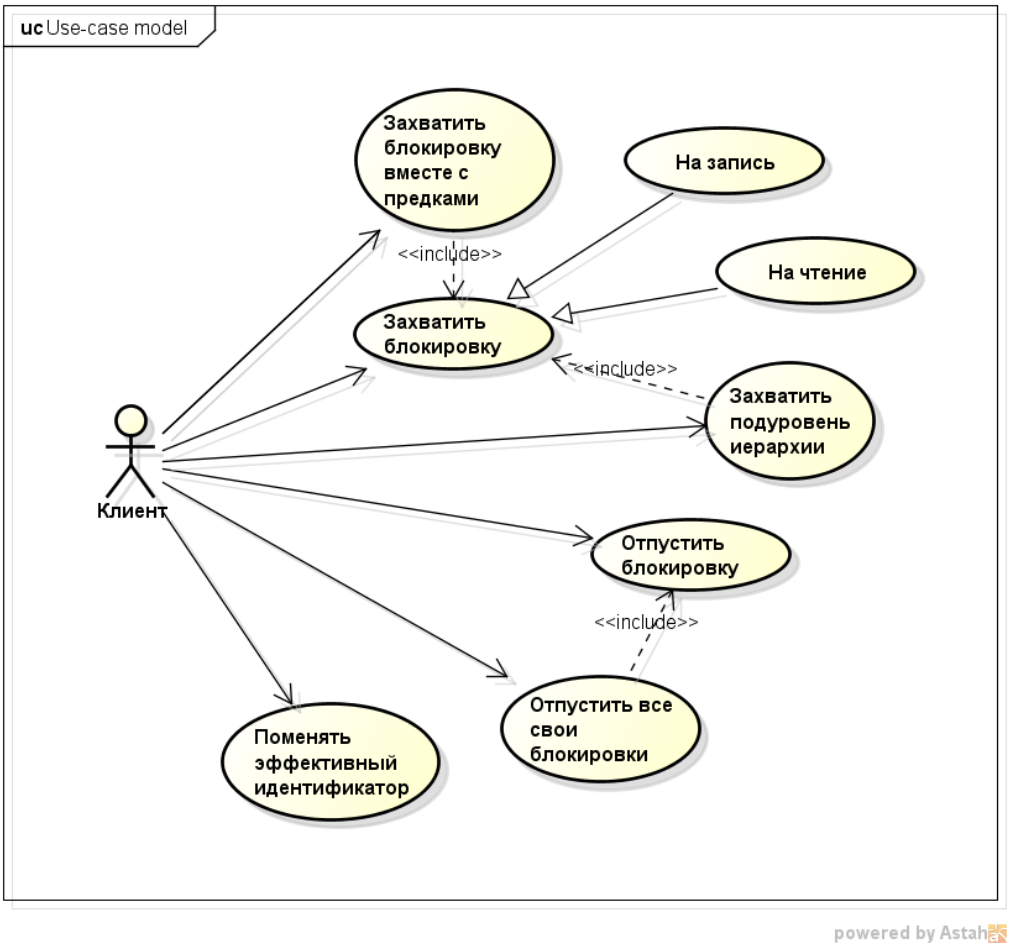

Рисунок 1. Варианты использования менеджера блокировок

3.2 Выделяемые компоненты менеджера

Общая схема компонентов менеджера блокировок представлена на рисунке 2. Клиентское приложение взаимодействует с клиентом менеджера блокировок посредством клиентского API. В приложении А представлен интерфейс работы с клиентом менеджера блокировок. Далее, клиент транслирует все запросы к серверу, а затем принимает от него ответ. Сервер и клиент физически могут располагаться на разных компьютерах, тогда они вступают в сетевое взаимодействие. В свою очередь, сервер взаимодействует с ядром, в котором расположена вся бизнес-логика менеджера блокировок.

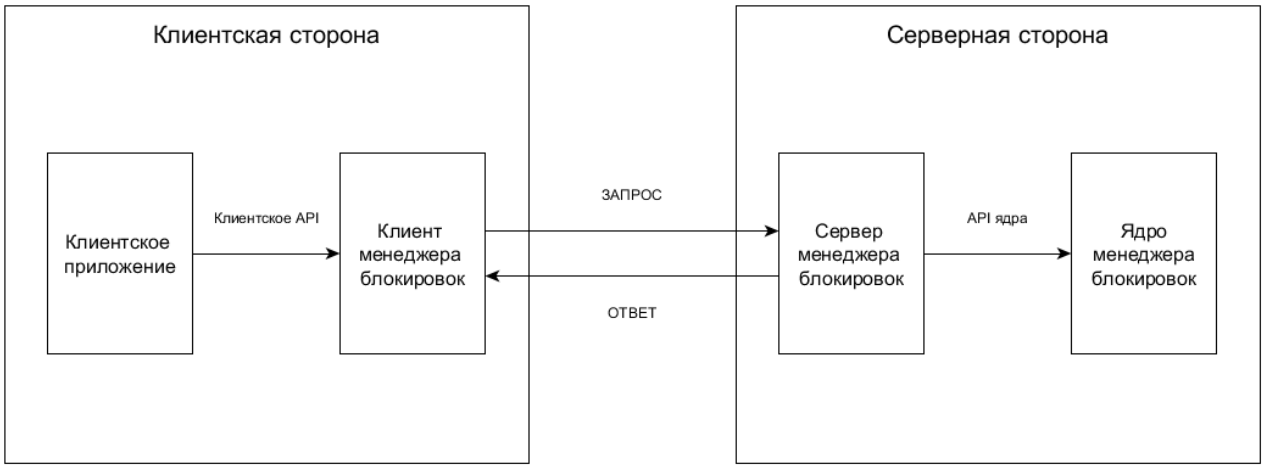

Рисунок 2. Схема компонентов менеджера блокировок

Ядро менеджера блокировок является основой менеджера блокировок. Оно определяет, какими блокировками в данный момент обладает любой клиент, может ли быть захвачена следующая блокировка клиентом, какие клиенты находятся в ожидании освобождения блокировки. Все вызовы ядра менеджера блокировок являются неблокирующими, то есть ответ на запрос приходит сразу после его обработки. Внутренние структуры ядра менеджера блокировок должны существовать вне времени жизни процессов, работающих с ним. Поэтому они размещаются в персистентном хранилище, например, таком как разделяемая память операционной системы.

Ядро, в свою очередь, состоит из таких компонент, как хранилище блокировок, модуль идентификации клиентов, модуль обнаружения дедлоков.

Хранилище блокировок представляет собой таблицу ключ-значение, ключом в которой является уникальный идентификатор ресурса, а значением – структура, содержащая все необходимые сведения о ресурсе.

Модуль идентификации клиентов служит для сопоставления реальных идентификаторов клиентов эффективным идентификаторам. Клиент в любой момент может поменять свой эффективный идентификатор на идентификатор другого клиента.

Модуль обнаружения дедлоков необходим для обнаружения ошибок программистов клиентских приложений. Со стороны менеджера блокировок гарантировать отсутствие дедлоков невозможно, так как клиент сам управляет захватом блокировок, и невозможно нарушить ни одно необходимое условие возникновения взаимных блокировок [2]. Известно, что система находится в состоянии взаимной блокировки тогда и только тогда, когда

существует цикл в графе ресурсов. Данный модуль строит граф ресурсов и пытается обнаружить в нём циклы. При нахождении цикла в графе возбуждается исключение.

3.3 Блокирование клиентских процессов

Одним из архитектурных решений является отделение логики работы с ресурсами от механизма блокирования клиентских процессов. Это сделано для поддержки различных способов блокирования. Как уже было сказано ранее, менеджер блокировок должен уметь работать в двух конфигурациях: в виде библиотеки, и в виде веб-сервиса. Блокирование клиентских процессов в них реализовано по-разному.

Конфигурация в виде веб-сервиса (также будем называть эту конфигурацию многосерверной, так как она позволяет взаимодействовать нескольким клиентам в разных узлах сети) подразумевает, что на выделенном узле распределённой системы будет запущен *сервер* менеджера блокировок. Каждый клиент, желающий обратиться к менеджеру блокировок, должен будет установить сетевое соединение с сервером и передавать свои сообщения через это соединение. После отправки запроса, клиент обязан дождаться ответа на сервере. В случае, если в данный момент его запрос не может быть удовлетворён, клиент блокируется – сервер просто не отправляет свой ответ. Причём клиентский процесс блокируется на системном вызове read, и в таком случае операционная система поймёт, что на данный процесс не нужно расходовать процессорное время.

При конфигурации менеджера блокировок в виде библиотеки (также будем называть эту конфигурацию односерверной) отсутствует такая сущность как сервер. Функции ядра менеджера блокировок вызываются напрямую из клиентского процесса. В таком случае, нам необходимо каким-то образом организовать межпроцессное взаимодействие, чтобы процессы уведомляли друг друга о возможности захвата блокировки. Было принято решение использовать разделяемую память и хранить в ней мьютексы и условные переменные, использующиеся для того, чтобы избежать активного ожидания процессов. Таким образом, в односерверной конфигурации блокировка клиентов происходит с помощью мьютексов, хранящихся в разделяемой памяти.

3.4 Конфигурации менеджера блокировок

На рисунке 3 представлена схема работы менеджера блокировок в односерверной конфигурации. *Синхронный исполнитель* предоставляет клиентский интерфейс для работы с менеджером блокировок. Клиенты посредством этого интерфейса производят запросы на

блокирование ресурсов. Каждая стрелка на рисунке 3 подразумевает отдельный клиентский процесс.

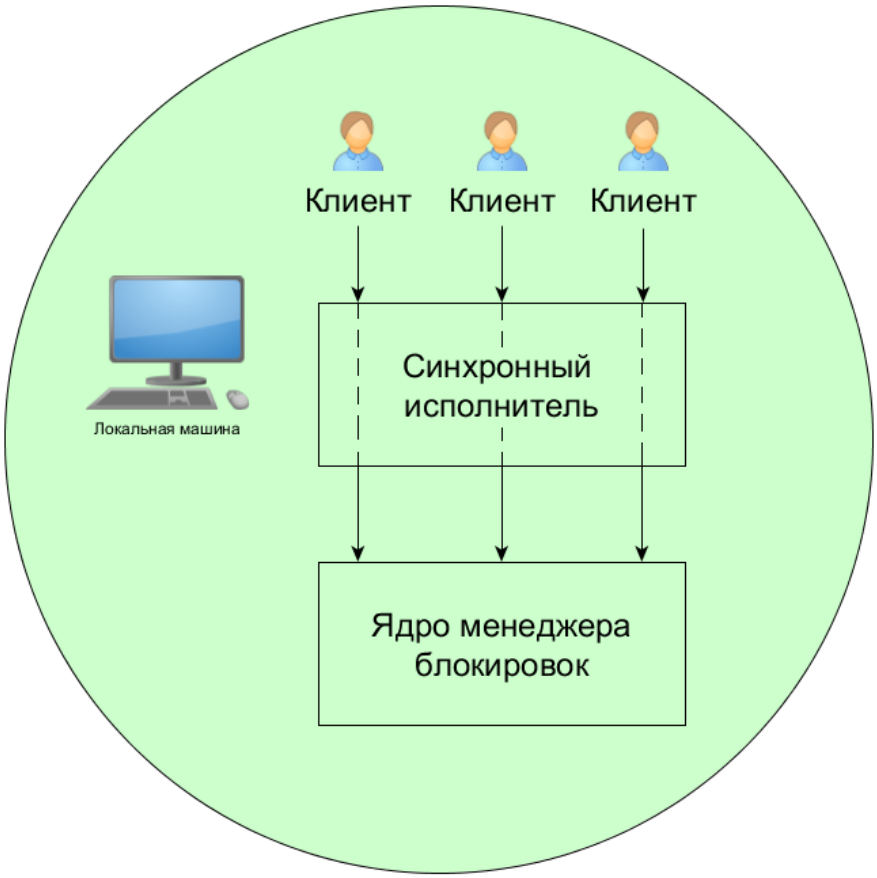

Рисунок 3. Односерверная конфигурация менеджера блокировок

Рассмотрим подробнее, как синхронный исполнитель обеспечивает блокировку процессов. Допустим, Клиент1 и Клиент2 пытаются захватить ресурс A в эксклюзивное пользование. Первым был обработан запрос Клиента1, и он становится текущим владельцем ресурса A. Клиент2 должен перейти в состояние ожидания освобождения ресурса A. Для этого синхронный исполнитель создаёт в разделяемой памяти мьютекс и условную переменную, ассоциированную с идентификатором Клиента2 (ИД2). Вызывается wait на только что созданной условной переменной. В то же время внутри ядра появляется запись о том, что ИД2 ожидает освобождения ресурса A. Теперь, когда Клиент1 освободит ресурс A, из ядра вернётся информация о том, что клиент и идентификатором ИД2 находится в ожидании. Клиент1 вызовет notify на условной переменной Клиента2, и они оба продолжат работу.

Схема многосерверной конфигурации менеджера блокировок приведена на рисунке 4. В данном случае работа с ядром менеджера блокировок производится в пуле потоков *асинхронного исполнителя*. Здесь мы получаем большое преимущество из-за того, что вызовы

ядра менеджера блокировок являются неблокирующими. В случае блокирующих вызовов нельзя было бы использовать пул потоков для работы с ядром, так как все потоки из пула могли бы заблокироваться. Пришлось бы создавать по одному процессу на клиентское соединение, что неэффективно.

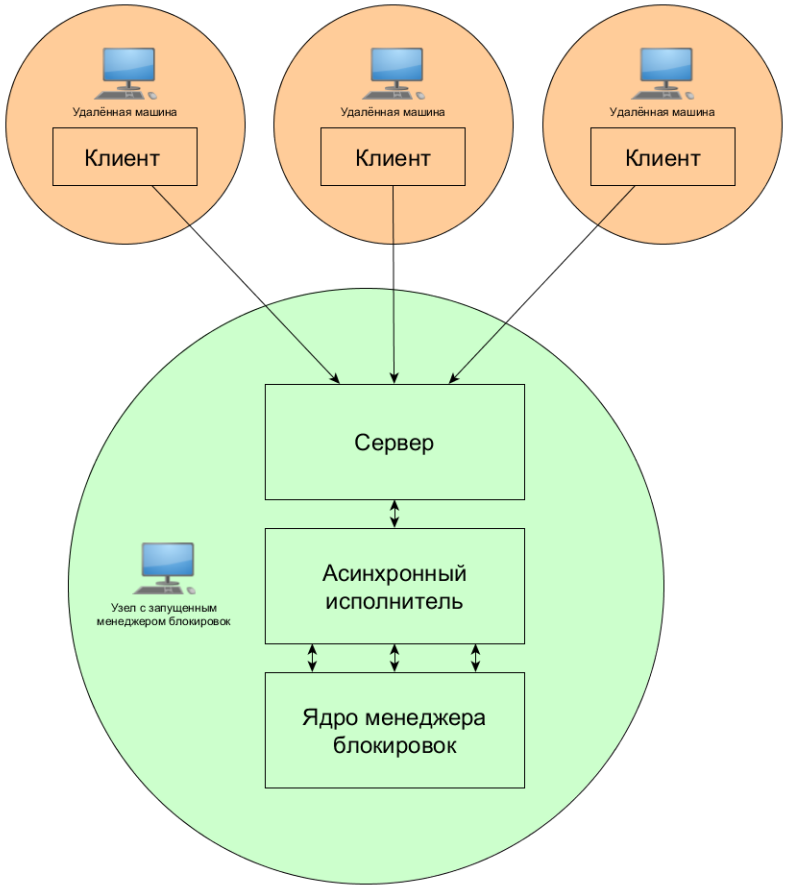

Рисунок 4. Многосерверная конфигурация менеджера блокировок.

Ответы от ядра менеджера блокировок в данной конфигурации обрабатываются следующим образом. Каждый запрос на блокирование ресурса представляется в виде задачи, исполняемой в пуле потоков. Асинхронный исполнитель хранит две очереди задач. Первая очередь – на исполнение, вторая – очередь неактивных задач. Если запрос на блокирование ресурса должен привести к ожиданию клиента, то задача помещается в очередь неактивных задач. Когда требуемый ресурс будет освобождён, задача переместится в очередь на исполнение.

Глава 4 Тестирование менеджера блокировок

Протестировать многопоточное или выполняющееся в нескольких процессах приложение на порядок сложнее, чем однопоточное. Проблема заключается в недетерминированном порядке исполнения процессов. Этот порядок определяется операционной системой, и не может контролироваться вручную. Несмотря на то, что можно задать некоторые дополнительные точки синхронизации процессов, добиться одновременного исполнения одной команды в двух различных процессах невозможно. Тем не менее, для тестирования менеджера блокировок реализовано несколько классов тестов.

4.1 Модульное тестирование

Для написания модульных тестов использовался Google C++ Testing Framework, распространяющийся под New BSD License. Покрытие кода модульными тестами было измерено с помощью утилиты gcov, которая является частью пакета GCC.

4.2 Функциональное тестирование

В функциональных тестах эмулируется поведение клиентов, за счет чего проверяется корректность работы менеджера блокировок. Для большей гибкости тесты написаны на языке PHP. Было рассмотрено несколько сценариев работы клиентов, которые повторяют возможное поведение клиентов на практике. Приведём описание некоторых сценариев:

Сценарий 1. Проверка обеспечения консистентности изменений при одновременном доступе

1) Создается файл, в котором записано число 0.
2) Создается N клиентов (N = 1...40).
3) Каждый клиент работает следующим образом:
    a) Захватывает блокировку, связанную с файлом на запись.
    b) Считывает число из файла.
    c) Увеличивает число на 1.
    d) Записывает число обратно в файл.
    e) Отпускает блокировку, связанную с файлом.
4) Операция 3 повторяется K раз.
5) После окончания работы всех клиентов, проверяется число в файле. Оно должно быть равно произведению N*K.

Сценарий 2. Проверка обеспечения целостности связей объектов

1) Создаются вложенные друг в друга каталоги, вложенность варьируется 1...5.
2) Каждому каталогу сопоставляется одноименная блокировка.
3) Создается 2 клиента, которые выполняют следующие действия:
    a) Клиент1 пытается создать файл в каталоге на нижнем уровне вложенности, Клиент2 пытается удалить каталог нижнего уровня вложенности.
    b) Восстанавливается система вложенных каталогов.
    c) Клиент1 пытается создать файл в каталоге на нижнем уровне вложенности, Клиент2 пытается удалить каталог более высокого уровня вложенности.
    d) Восстанавливается система вложенных каталогов.
    e) Повторяются операции b.-d. до тех пор, пока не будет проведена проверка для всех уровней вложенности.
    f) Повторяются операции a.-e., но теперь вначале Клиент2 производит свое действие, а потом Клиент1.

Сценарий 3. Рекурсивный захват блокировки

1) Создается клиент.
2) Клиент захватывает 2 раза одну и ту же блокировку на запись.
3) Клиент отпускает 2 раза блокировку.
4) Клиент захватывает 2 раза одну и ту же блокировку на чтение.
5) Клиент отпускает 2 раза блокировку.
6) Ожидаемое поведение: дедлок не будет обнаружен, блокировки будут захвачены и освобождены без ошибок и исключений.

4.3 Тестирование производительности

В процессе развития проекта происходит постоянная модификация исходного кода. Необходимо обеспечить гарантию того, что последние изменения в коде не повлекли за собой резкого и недопустимого спада производительности. Поэтому тесты на производительность проводятся регулярно, чтобы оперативно обнаруживать возможные проблемы. Набор тестов на производительность также написан на PHP и состоит из нескольких сценариев:

Сценарий 1. Измерение средней скорости захвата непересекающихся блокировок

1) Создается N клиентов.
2) В теле каждого клиента совершаются следующие действия:
    a) Создается запрос на M блокировок, каждая из которых имеет глубину L. Для каждого клиента блокировки имеют уникальное имя.

   b) Замеряется время перед запросом на блокировку time1.
   c) Производится захват блокировок.
   d) Замеряется время после выполнения запроса на блокировку time2.
   e) Вычисляется скорость захвата V = M / (time2 - time1).
   f) Операции b.-e. выполняются K раз.
   g) Вычисляется средняя скорость захвата блокировки V_{avg} = V / K.
3) Вычисляется средняя скорость захвата блокировки одним клиентом.
4) Операция 2. повторяется для глубины L = 1...5.
5) Операции 1.-4. повторяются для N = 1...40.

Сценарий 2. Измерение среднего времени захвата и освобождения непересекающихся блокировок

1) Создается N клиентов.
2) Отдельным клиентом захватывается M блокировок.
3) В теле каждого клиента совершаются следующие действия:
   a) Создается запрос на 1 блокировку глубины L. Для каждого клиента блокировка имеет уникальное имя.
   b) Замеряется время перед захватом блокировки time1.
   c) Производится захват блокировки.
   d) Замеряется время после захвата блокировки time2.
   e) Вычисляется время выполнения операции захвата T_L = time2 - time1.
   f) Замеряется время перед освобождением блокировки time3.
   g) Производится освобождение блокировки.
   h) Замеряется время после освобождения блокировки time4.
   i) Вычисляется время выполнения операции освобождения T_U = time4 - time3.
   j) Операции b.-i. выполняются K раз.
   k) Вычисляется среднее время захвата и освобождения блокировки и потребляемая память.
4) Освобождается M блокировок.
5) Вычисляется среднее время захвата и освобождения блокировки одним клиентом и суммарный объём потребляемой памяти.
6) Операции 2.-4. повторяются для глубины L = 1...5.

7) Операции 1.-6. повторяются для N = 1...40.

Сценарий 3. Измерение средней скорости захвата пересекающихся блокировок

1) Создается N клиентов.
2) В теле каждого клиента совершаются следующие действия:
    a) Создается запрос на M блокировок, каждая глубины L. *P* процентов блокировок имеют одинаковое имя для всех клиентов, *100 - P* процентов имеют уникальное имя для каждого клиента.
    b) Замеряется время перед блокировкой time1.
    c) Производится захват блокировки.
    d) Замеряется время после захвата блокировки time2.
    e) Вычисляется скорость захвата V = M / (time2 - time1).
    f) Операции b.-e. выполняются K раз.
    g) Вычисляется средняя скорость захвата блокировки V_{avg} = V / K.
    h) Операции a.-g. выполняются для P = 10...100 с шагом 10.
3) Вычисляется средняя скорость захвата блокировки одним клиентом.
4) Операция 2. повторяется для глубины L = 1...5.
5) Операции 1.-3. повторяются для N = 1...40.

Пример результатов тестирования представлен в приложении Б. На основе данных результатов можно сделать вывод, что менеджер блокировок обладает приемлемыми характеристиками по производительности.

Заключение

В ходе работы над данным проектом были достигнуты следующие результаты:

1) Произведён анализ необходимого научного материала для выполнения поставленной задачи.
2) Сформулированы требования к разрабатываемой системе.
3) Проведены исследования уже существующих реализаций менеджеров блокировок, выявлены их достоинства и недостатки.
4) Спроектирована архитектура менеджера блокировок, отвечающая всем поставленным требованиям
5) Написан код для конфигураций менеджера блокировок в виде библиотеки и веб-сервиса на языке C++.
6) Написан PHP extension для использования менеджера блокировок из языка PHP.
7) Составлены тесты на производительность и функциональные тесты.
8) Работа была представлена на конференции на 52-ой Международной студенческой конференции «Студент и научно технический прогресс» и опубликована в сборнике материалов конференции [9].
9) Работа была представлена на всероссийской научной конференции молодых учёных «Наука. Технологии. Инновации» и опубликована в сборнике материалов конференции [10].

В дальнейшем возможно следующее развитие данной работы:

1) Математическое доказательство корректности используемого алгоритма.
2) Обеспечение отказоустойчивости в многосерверной конфигурации.

Литература

1. Э. Таненбаум, М. ван Стеен. Распределенные системы. Принципы и парадигмы. СПб.: Питер, 2003. - 877с.
2. Э. Таненбаум. Современные операционные системы. 3-е изд. – СПб.: Питер, 2010. – 1120 с.
3. Programming Locking Applications [Электронный ресурс] – Режим доступа: http://opendlm.sourceforge.net/cvsmirror/opendlm/docs/dlmbook_final.pdf, свободный
4. ZooKeeper [Электронный ресурс] – Режим доступа: http://zookeeper.apache.org/doc/trunk/zookeeperOver.pdf, свободный
5. ZooKeeper Internals [Электронный ресурс] – Режим доступа: http://zookeeper.apache.org/doc/trunk/zookeeperInternals.pdf, свободный
6. ZooKeeper Administrator's Guide [Электронный ресурс] – Режим доступа: http://zookeeper.apache.org/doc/trunk/zookeeperAdmin.pdf, свободный
7. ZooKeeper Java Example [Электронный ресурс] – Режим доступа: http://zookeeper.apache.org/doc/r3.1.2/javaExample.pdf, свободный
8. The Chubby lock service for loosely-coupled distributed systems [Электронный ресурс] – Режим доступа: http://research.google.com/archive/chubby-osdi06.pdf, свободный
9. Ильин А.В., Бобренок С.И. Разработка менеджера блокировок для распределённых систем // Материалы 52-й Международной научной студенческой конференции МНСК-2014: Информационные технологии/ Новосиб. гос. ун-т. Новосибирск, 2014. 265 с.
10. Ильин А.В., Арыков Н.Е., Бобренок С.В., Разработка менеджера блокировок для распределённых систем // НАУКА. ТЕХНОЛОГИИ. ИННОВАЦИИ Материалы всероссийской научной конференции молодых ученых в 10 ч. – Новосибирск: Изд-во НГТУ, 2013. – Часть 2. – 241 с.

Приложение А

(обязательное)

Интерфейс взаимодействия с менеджером блокировок на языке PHP

```
class LockManagerClient {
    const HIERARCHY = 1;
    const MASK = 2;
    /**
     * Factory method for LockQuery.
     *
     * @param array $query associative array of
     *        resource name (string) and lock type (READ\WRITE constant from
LockQuery) for it.
     * @param int $flags HIERARCHY\MASK constant from LockManagerClient.
     * @return LockQuery Query contained resources and it's lock types.
     */
    public function getLockQuery($query, $flags = 0) {}

    /**
     * @return string Current identifier.
     * @throws ConnectionException
     */
    public function getToken() {}

    /**
     * Changes client identifier on parameter token.
     * If the client has locks and it is the last client with that identifier,
the lock "released".
     *
     * @param string $token
     * @return void
     */
    public function setToken($token) {}

    /**
     * Unlocks all resources associated with the current identifier (token).
     *
     * @return void
     * @throws ConnectionException
     */
    public function unlockAll() {}
}
class LockQuery {
  public const READ = 2;
  public const WRITE = 3;

  /**
   * Lock resources. Blocking call.
   * If the resource can't be obtained, execution is blocked.
   *
   * @param string $comment Optional argument is used in the detection of
deadlocks.
   * @return int which equals LOCKED or ALREADY_LOCKED - constant from Lock.
   * @throws DeadlockException
   * @throws ConnectionException
   */
  public function lock($comment = '') {}
```

```
  /**
   * Lock resources. Non-blocking call.
   *
   * @param string $comment Optional argument is used in the detection of
deadlocks.
   * @return int which equals LOCKED or ALREADY_LOCKED or CANNOT_LOCK - constant
from Lock.
   * @throws DeadlockException
   * @throws ConnectionException
   */
  public function tryLock($comment = '') {}

  /**
   * Unlocks resources.
   *
   * @return int which equals UNLOCKED or NOT_LOCKED - constant from Lock.
   * @throws ConnectionException
   */
  public function unlock() {}
}
```

Приложение Б

(обязательное)

Результаты тестирования односерверной конфигурации менеджера блокировок на производительность.

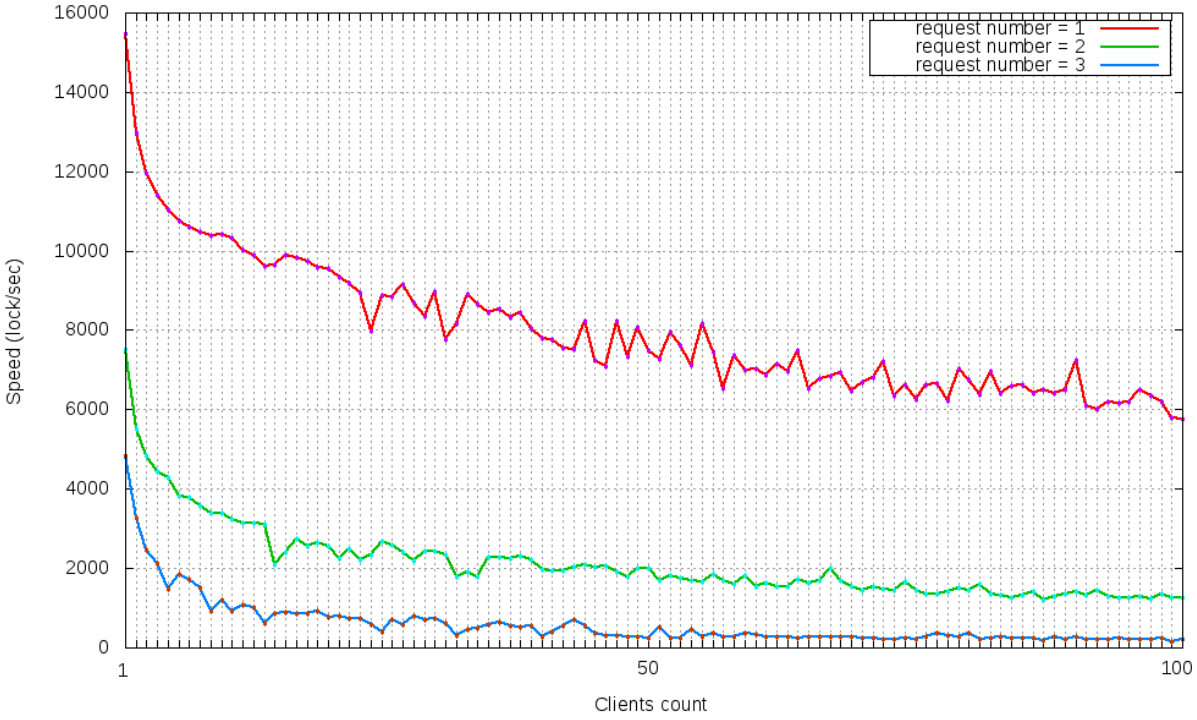

Рисунок 5. Результаты выполнения сценария 1

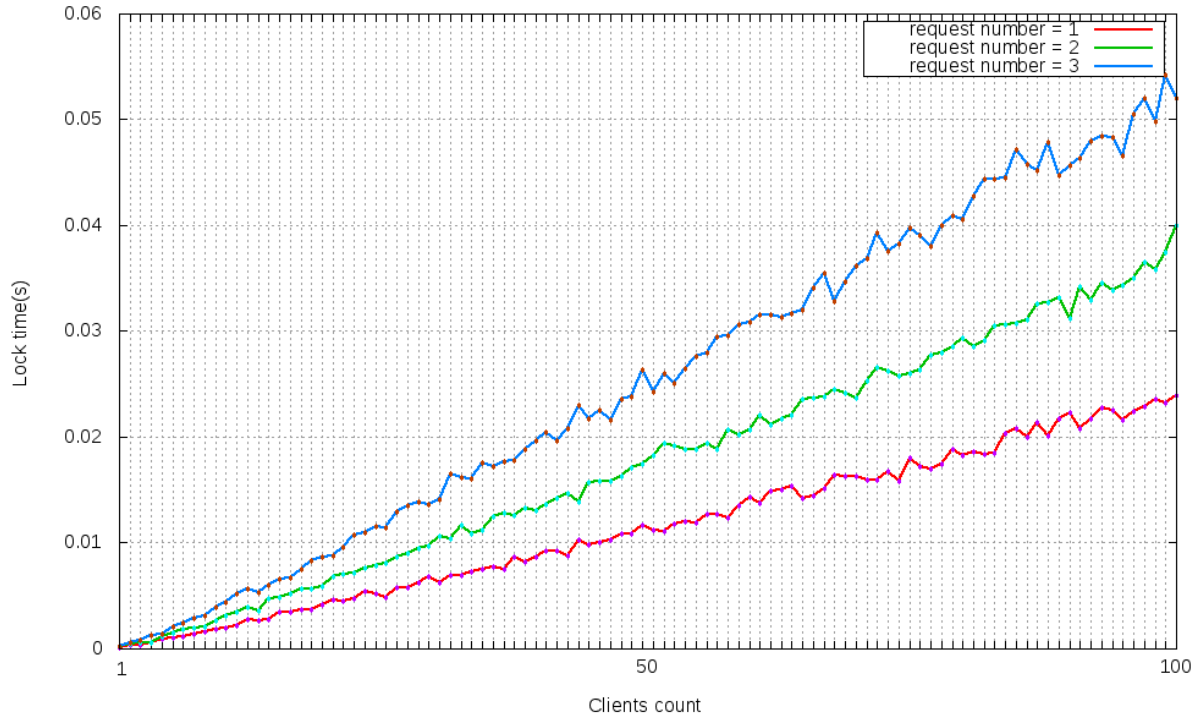

Рисунок 6. Результаты выполнения сценария 2

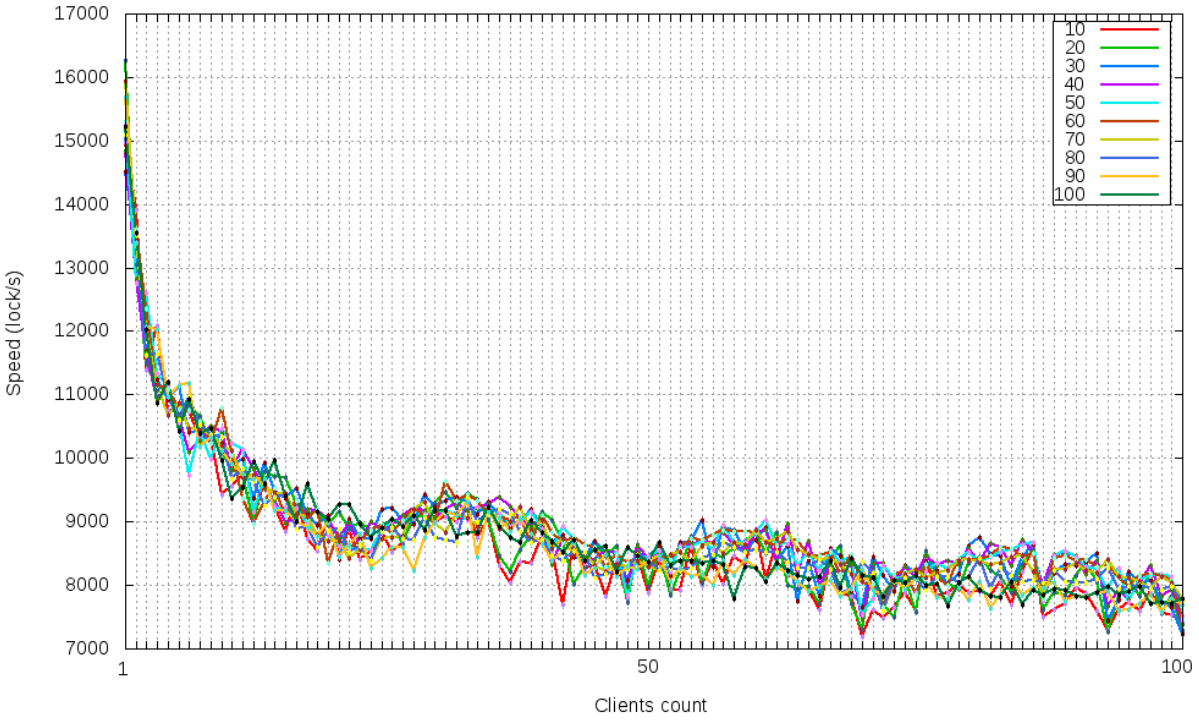

Рисунок 7. Результаты выполнения сценария 3