МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ОТКАЗОУСТОЙЧИВОЙ РАСПРЕДЕЛЁННОЙ СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 5 курса 551 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Ахмановой Элины Дамировны

Научный руководитель
зав. каф., к. ф.-м. н., доцент _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2021

ВВЕДЕНИЕ

Для выполнения бакалаврской работы была выбрана одна из наиболее востребованных тем: «Разработка отказоустойчивой распределённой системы хранения данных» для реализации архитектуры хранения данных по типу ключ-значение [1] на базе алгоритма Raft.

Актуальность темы обусловлена последними изменениями в технологиях, компьютерном обеспечении и требованиях пользователей к хранению данных [2], а также связи с такими видами разработки, которые поддерживают параллелизм и кооперативную многозадачность (от англ. «concurrency»).

Окончание действия закона Мура (наблюдение о скорости развития микропроцессоров, согласно которому количество транзисторов на кристалле микропроцессора должно удваиваться каждые два года) повлияло на распространение многоядерных вычислений, [3]. Для работы с многоядерными компьютерами разработчики всё чаще обращаются к параллельности и кооперативной многозадачности.

Веб-сервисы стали более доступными по всему миру из-за роста средней скорости доступа к сети Интернет. Пользователи таких сервисов ожидают устойчивости при высоких нагрузках и надёжности хранения данных (в том числе минимизации случаев потери данных). Вместе с этим многие интернет-компании (Google, Amazon, Red Hat и другие) всё чаще работают с большими данными (от анг. «Big Data») или облачными хранилищами данных (от англ. «cloud data-storage»). В основе этих технологий используются алгоритмы консенсуса и распределённого хранения данных.

Цель бакалаврской работы: изучить основы архитектуры отказоустойчивых распределённых систем и разработать отказоустойчивое распределённое хранилище на основе алгоритма консенсуса Raft.

Задачи работы:

— описать понятие и методы обеспечения отказоусточивости;
— описать понятие консенсуса и распределённой системы;
— систематизировать способы учёта времени и очерёдности действий в распределённых системах;
— рассмотреть виды отказоустойчивых распределённых систем;
— определить наиболее актуальные алгоритмы и языки программирования для реализации распределённых систем;

— разработать библиотеку с алгоритмом Raft;
— разработать отказоустойчивую распределённую системы хранения данных;
— протестировать систему на возможные сценарии отказа программного и аппаратного обеспечения, ошибки сети.

Теоретической основой работы послужили труды (статьи и курсы):

— Л. Лампорта (работа о проблеме византийских генералов [4], теория времени в распределённых системах [5], а также работа о его первом алгоритме консенсуса Paxos [6]);
— лекции Массачусетского технологического института по распределённым системам [7];
— лекции Калифорнийского университета в Ирвайне о кооперативной многозадачности в языке Golang [8], [9];
— и прочее.

Практическая часть работы использует подходы других практических трудов:

— статью об особенностях реализации алгоритма Raft [10];
— тестовую систему практических занятий от курса Массачусетского технологического института по распределённым системам [7];
— статью об архитектуре одной из первых распределённых отказоустойчивых распределённых систем хранения данных — Bigtable [11];
— книгу о практических подходах к построению высоконагруженных приложений [2];
— и прочее.

Теоретическая значимость бакалаврской работы обусловлена рассмотрением современных отказоустойчивых распределённых систем и сопутствующих понятий (отказоустойчивости, алгоритмов консенсуса, теории времени Лампорта).

Практическая значимость бакалаврской работы заключается в возможности дальнейшего использования:

— модульной библиотеки алгоритма консенсуса Raft (может быть использована для любой распределённой системы, вне зависимости от языка программирования);
— распределённой отказоустойчивой системы хранения данных по типу

ключ-значение (может быть использована как распределённая база данных или сервис по синхронизации данных между маломощными устройствами).

Структура и объём работы. Бакалаврская работа состоит из определений, обозначений и сокращений, введения, 3 разделов, заключения, списка использованных источников и 3 приложений.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

В первом разделе представлены теоретические аспекты рспределённых систем.

Подраздел «Распределённые системы» включает в себя понятие распределённой системы, как парадигмы достижения соглашения между несколькими компьютерами, причины перехода от одного компьютера к распределённой сети, среди которых выделяют возможности масштабируемости, отказоустойчивости, высокой доступности и особенности географического расположения данных.

На основе понятия распределённой системы описаны способы определения надёжности узлов системы, а также понятия византийского поведения и византийского узла в таких системах.

Подраздел «Построение распределённых систем на базе параллелизма или кооперативной многозадачности» включает основные вопросы, возникающие при построении распределённых систем:

— как узлы будут общаться с пользователем;

— как обеспечить разделение ресурсов между узлами;

Ответ на эти вопросы привёл к двум фундаментально разным подходам построения систем: *параллелизм* и *кооперативная многозадачность*.

Основное требование параллелизма — это оборудование с несколькими процессорами. В одноядерном процессоре мы не можем получить параллелизм, потому что в одном ядре может запускаться только один поток задач (одна задача в одну единицу времени).

Кооперативная многозадачность означает возможность выполнять несколько задач вместе, но не одновременно. Такой способ разбиения позволяет выполнять задачи, повышая скорость работы системы за счёт того, что во время ожидания одним процессом ресурсов, другой процесс может приступить к выполнению, не влияя на состояние первого.

Для более качественного понимания систем было выделено основное отличие кооперативной многозадачности от параллелизма.

Система выполняется в режиме многозадачности, если она может *поддерживать* как минимум две задачи одновременно. Система называется параллельной, если две или более задачи *выполняются* одновременно.

Следующий подраздел «Понятие консенсуса» описывает проблему консенсуса между несколькими узлами в распределённой системе.

Условиями выполнения алгоритма консенсуса считаются [12]:

— соглашение;
— результативность;
— целостность.

Проблема консенсуса также включает в себя задачу о византийских генералах, которая была описана в том же подразделе.

В следующем подразделе описана теория часов Лампорта для узлов распределённых систем, а также условие для работы этих часов.

На базе этих понятий описаны правила реализации часов Лампорта.

С помощью условий и правил реализации логических часов получилось понять концепцию словосочетания «произошло прежде», которое определяет инвариант порядка сообщений в распределённой системе с несколькими процессами. С точки зрения распределённых алгоритмов наиболее важным является факт относительной упорядоченности событий.

Один из наиболее важных подразделов этой главы *описывает понятие отказоустойчивости*, а также основные требования к отказоустойчивым системам:

1. Отсутствие единой точки отказа.
2. Изоляция вышедшего из строя компонента.
3. Идентификация типа неисправности.
4. Сдерживание выхода из строя отдельных компонентов для предотвращения отказа всей системы.
5. Доступность

Отказоустойчивые системы обычно основаны на концепции избыточности. Однако не все системы удовлетворяют сразу всем требованиям.

Следующий подраздел *«Методы обеспечения отказоустойчивости»* рассматривает уровни взаимодействия в система, а на их базе описывает ме-

тоды обеспечения отказоустойчивости:

1. Репликация.
2. Терпимость к сбоям.
3. Восстановительные работы.

Второй раздел «Разработка модуля с алгоритмом консенсуса Raft» посвящен разработке библиотеки с помощью языка Golang. Эта библиотека необходима для обеспечения консенсуса в распределённых системах.

Первый подраздел *«Обоснование выбора языка программирования»* содержит описание процесса выбора языка программирования. Выбор осуществялся между языками Golang, Python и C++. Каждый язык имеет развитую стандартную библиотеку, состоящую из множества стандартных решений и алгоритмов. Однако именно Golang является компромиссом между скоростью работы программ и простотой их написания.

В дополнение были описаны горутины языка, которые позволяют работать в режиме кооперативной многозадачности, а также их планировщик.

Во втором подразделе был осуществлён сравнительный анализ алгоритмов консенсуса. Был выбран алгоритм Raft вместо более старого алгоритма Paxos, так как создатели алгоритма Raft были нацелены на сохранение эффективности Paxos и на упрощение понимания, что положительно влияет на построение всей системы.

Следующий подраздел «Архитектура алгоритма Raft» содержит описание работы алгоритма, особенности архитектуры узлов. Описаны также возможные способы решения разрыва соединения между узлами.

На базе этой архитектуры в *следующем разделе представлена реализация алгоритма Raft*. Были описаны основные переменные и константы каждого из узлов.

Каждый лидер должен инициировать добавление новых записей в хранилище узлов-последователей.

Для этого была использована структура данных $AppendEntriesArgs$.

В базовой модели Raft присутствует только один узел. Каждый узел может быть в трёх состояниях: последователь, лидер, кандидат. Все узлы начинают с одинакового состояния последователей, без лидеров и кандидатов.

Во время запуски функции $Start(..)$, сам узел предполагает, что этот узел является лидером, если не докажет обратного.

Для получения состояния узла функция *GetState* возвращает последнюю запись и статус узла (лидер ли он).

Если узел был в состоянии последователя, лидер не отвечает ему в рамках времени таймера, то последователь превратится в кандидата и инициирует голосование за лидера.

Как только лидер выбран, он начинает обслуживать запросы клиентов.

Если последователи выходят из строя или работают медленно, или если сетевые пакеты потеряны, лидер повторяет запросы *RPC AppendEntries* бесконечно (даже после того, как он ответил клиенту), пока все последователи в конечном итоге не сохранят все записи журнала.

Если в системе произошёл некритичный сбой (всё ещё доступно большинство узлов), то она будет продолжать работу. Даже если узел лидера подвергся сбою, через какое-то время система вернётся к начальному состоянию, в котором в ней нет ни одного лидера, и продолжит работу в штатном режиме.

В следующем подразделе было произведено тестирование модуля.

Были описаны тесты:

1. Базовый тест.
2. Тест с перевыбором лидера после отказа.
3. Тесты на проверку переданных сообщений, индексов сохранённых системой значений и то, что количество переданных сообщений разумно для принятия одного решения.
4. Тесты на бесконечные отказы (с некоторым перерывом).
5. Тест на то, что ни одно решение не будет принято, если в системе слишком много сетевых разделений или отказов в узлах.
6. Тест на верное заполнение журналов и другие.

На базе тестов был представлен вывод.

На рисунке 1 видно, что наименее производителен модуль не при сбоях сети/узлов, а при сбоях в записях журнала и записи неверного журнала. Потому что имеено это влияет на быстрое восстановление системы после сбоя. Остальные тесты модуль прошёл на высоких показателях скорости (меньше 10 миллисекунд на одно решение).

Третий раздел «Разработка отказоустойчивой распределённой системы» содержит в себе описание процесса разработки системы для хра-

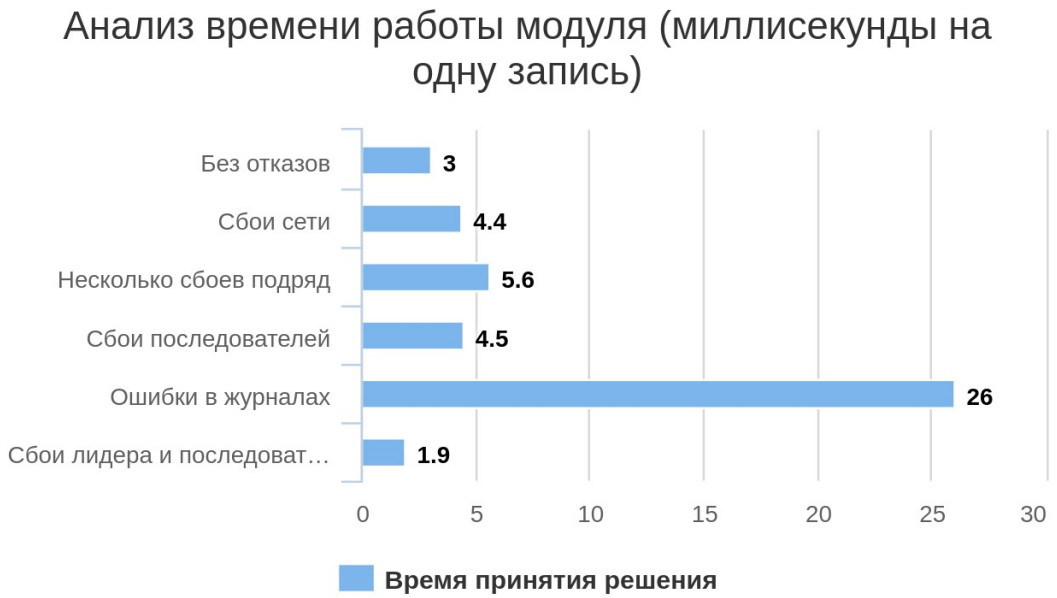

Рисунок 1 – Тестирование модуля Raft. Скорость работы

нения данных и тестирования её на возможные сбои сети и узлов.

Первый подраздел «Требования к системе» описывает основные требования для использования системы.

Система должна быть реплицированным конечным автоматом, состоящим из нескольких серверов (узлов), которые будут хранить ключи и значения и использовать модуль Raft для репликации.

Система должна обрабатывать клиентские запросы до тех пор, пока большинство узлов функционируют и могут обмениваться данными несмотря на разного рода (системные, сетевые и другие) сбои.

Были описаны операции, которые должна поддерживать система:

1. $Put(,)$ заменяет или добавляет значение для определённого ключа в базе данных системы.
2. $Append(,)$ добавляет к текущему значению системы новое. Для несуществующего ключа должна работать как $Put(..)$.
3. $Get()$ получает текущее значение ключа и возвращает его. Для несуществующего ключа должен возвращать пустую строку.

Каждый пользователь общается с системой через посредника, клиента (клерк) с помощью методов Put / Append / Get. Клерк должен управлять взаимодействиями между серверами.

Второй подраздел «Архитектура отказоустойчивой распределённой системы» содержит описание интерфейса для общения с клиентом и иден-

тичные узлов системы.

Также был описан журнал ведения записей для восстановительных работ в системе.

Каждая запись журнала хранит команду конечного автомата вместе с номером сообщения, когда запись была получена лидером. Номера сообщения в записях журнала используются для обнаружения несоответствий между журналами. Каждая запись журнала также имеет целочисленный индекс, определяющий ее положение в журнале.

На базе архитектуры *в следующем подразделе «Разработка системы хранения данных»* была представлена непосредственная разработка отказоустойчивой распределённой системы хранения данных по типу ключ-значение.

Каждый клиент системы представляет из себя структуру (аналог класса в Golang) Clerk:

```
type Clerk struct {
  servers []*labrpc.ClientEnd
  leaderId      int
    clientId       int64
    lastRequestId int
}
```

Система вызывает *Make*(...) для создания узла, и начинает работу с командой *Start*(...). Получить состояние узла и его статус (является ли он лидером) можно с помощью команды *GetState*():

```
rf = Make(peer, id)
rf.Start(command interface{}) (id, term, isleader)
rf.GetState() (term, isLeader)
```

Был описан процесс использования модуля Raft клерком, а также команды, с помощью которых пользователь может запросить у клиента значение по ключу, добавить или изменить значение (что с точки зрения сервиса является одной операцией)

Так как операция удаления невозможна в чистом виде для распределённой системы, её заменяет операция изменения значения.

В последнем подразделе «Тестирование работоспособности системы в условиях отказа» были рассмотрены результаты работы системы с помощью тестов.

Тесты содержат проверки на разбиение сети и на сбои в узлах, на возможности системы к восстановлению, а также тесты производительности (на скорость принятия решений и на количество переданных сообщений).

Рисунок 2 – Тестирование распределённой системы. Скорость работы

Был представлен результат (рисунок 2), по которому видно, что наихудший результат по времени для работы системы — это одновременное наличие нескольких пользователей, ошибок сети (передача неверных данных), сбоев в последователях и лидерах, а также разделения сети на подсети.

Все тесты в соответствии с требованиями к системе были пройдены успешно.

ЗАКЛЮЧЕНИЕ

Распространение сети Интернет по всему миру, увеличение скорости доступа, а также высокий спрос на облачные сервисы сделали распределённые системы наиболее восстребованной архитектурой для разработки программного обеспечения. С каждым годом также всё важнее обеспечить отказоустойчивость систем во избежание потери критические важных данных. Именно поэтому тема бакалаврской работы была направлена на построение отказоустойчивой распределённой системы.

В ходе работы были описаны основные понятия распределённых и отказоустойчивых систем. Большую роль имеет понятие времени для узлов распределённой системы, введённое впервые Л. Лампорт в его работе «Time, Clocks, and the Ordering of Events in a Distributed System» [5]. Автор статьи как никто повлиял на развитие алгоритмов консенсуса, впервые описав алгоритм Paxos [6]. На базе именно этого алгоритма появились остальные, в том числе, алгоритм Raft. В первой части работы было произведено сравнение и систематизация особенностей алгоритмов консенсуса Paxos и Raft. На её основе для реализации был выбрал алгоритм Raft.

Практическая часть работы была направлена на построение распределённой отказоустойчивой системы хранения данных по типу ключ-значение. В основе была положена разработка библиотеки с алгоритмом консенсуса Raft без использования существующих реализаций. Алгоритм оказался простым для понимания и реализации. Основную сложность добавило наличие журнала для восстановления системы после сбоя.

При тестировании библиотеки стало понятно, что наиболее слабым местом являются не сбои, а ошибки в журналах. Потому что при наличии верного журнала система может легко восстановиться при сбое, в противоположном случае ей нужно повторить сбор журнала для каждого неверного изменения данных. В рамках будущих изменений желательно рассмотреть возможность защиты журнала хеш-суммами.

На базе библиотеки Raft была разработана система, описанная выше. Основная часть системы — работа клиента, который должен быть промежуточной точкой между пользовательскими запросами и узлами Raft. Во время тестирования были выявлены наиболее слабые места системы: совпадение нескольких помех одновременно (ошибки сети, перезапуск узлов, ненадёжное

соединение, разделение сети) влияет на производительность системы наибольшим образом, а наличие большого количества запросов и пользователей влияет на размер журнала и количество RPC сообщений. Может оказаться полезным в будущем разделять журнал на несколько частей, чтобы работать только с нужной в каждый отдельный момент времени.

Работу можно продолжить в будущем, чтобы использовать систему для хранения данных (файлов или элементов базы данных) на распределённой сети компьютеров. При чём, согласно данным тестирования системы, она подходит даже для маломощных устройств.

Таким образом, цель работы была достигнута. Все задачи были выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *DeCandia, G.* Dynamo: Amazon's highly available key-value store / G. DeCandia, D. Hastorun, J. Madan // *21st ACM Symposium on Operating Systems Principles (SOSP).* — 2007.

2 *Kleppmann, M.* Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems / M. Kleppmann. — O'Reilly Media, 2017. — 616 pp.

3 *Moore, G. E.* Cramming more components onto integrated circuits, reprinted from electronics / G. E. Moore // *Electronics Magazine.* — 1965. — Vol. 38. — Pp. 114–118.

4 *Lamport, L.* Byzantine generals problem / L. Lamport, R. Shostak, M. Pease // *ACM Transactions on Programming Languages and Systems.* — 1982. — Pp. 382–401.

5 *Lamport, L.* Time, clocks, and the ordering of events in a distributed system / L. Lamport // *Massachusetts Computer Associates, Inc.* — 1978. — Pp. 1–8.

6 *Lamport, L.* Generalized consensus and paxos / L. Lamport // *Microsoft Research Technical Report.* — 2004. — Vol. 33. — P. 63.

7 Mit 6.824: Distributed systems. — URL: http://nil.csail.mit.edu/6.824/2020/index.html (Дата обращения 30.05.2021). Загл. с экр. Яз. англ.

8 Golang official website. — URL: https://golang.org/ (Дата обращения 30.05.2021). Загл. с экр. Яз. англ.

9 Concurrency in go. — URL: https://www.coursera.org/learn/golang-concurrency (Дата обращения 30.05.2021). Загл. с экр. Яз. англ.

10 *Ongaro, D.* In search of an understandable consensus algorithm / D. Ongaro, J. Ousterhout // *Stanford University.* — 2014.

11 Bigtable: A distributed storage system for structured data / F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach Mike Burrows, T. Chandra, A. Fikes, R. E. Gruber // *Google, Inc.* — 2006. — Pp. 1–14.

12 *Coulouris, G.* Distributed Systems: Concepts and Design (3rd Edition) / G. Coulouris, J. Dollimore, T. Kindberg. — Addison-Wesley, 2001. — 452 pp.