Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Тюшев Максим Владимирович

Сравнительный анализ эффективности графовых баз данных для маршрутизации в ПКС сетях.

Курсовая работа

Научный руководитель:
к.ф.-м.н.
Смелянский Руслан Леонидович

Москва, 2023

Содержание

1 Введение

Программно конфигурируемые сети (ПКС)[1] – это сети с централизованным программным управлением. Основная причина этой лавины технологий - изменения требований со стороны приложений к инфраструктуре вычислений. Именно приложения, точнее их требования, являются основной движущей силой развития инфраструктуры вычислений. Программное управление этой инфраструктурой позволяет полнее и эффективнее удовлетворять эти требования. Данные аналитической компании MarketsandMarkets свидетельствуют, что в 2019 году объём мирового рынка программно-конфигурируемых сетей и дата-центров достиг $51,7 млрд.[2] Эксперты утверждают, что рынок является растущим и останется таковым. Ожидается, что продажи расходы на программно-конфигурируемые решения в глобальном масштабе будут увеличиваться на 25,5% ежегодно, а к 2024 году они достигнут $160,8 млрд.[2] Эксперты IDC также подтверждают растущий спрос на программно-конфигурируемые технологии.[2]

Процесс внедрения в практику ПКС сетей естественно заставляет переосмысливать решения проблем, которые были разработаны для традиционных сетей. Одной из таких проблем является проблема маршрутизации. Традиционные алгоритмы сетевой маршрутизации модифицируются и применяются к ПКС сетям для решения проблем с маршрутизацией трафика. Однако одним из критических параметров таких алгоритмов является сложность, которая определяет время сходимости сети. В ПКС сети на этот параметр существенное влияние оказывают количество сетевых устройств в контуре данных и их максимальная удаленность от контура управления. В традиционных TCP/IP сетях данная проблема решается путём разбиения сети на локальные зоны (area), размер области в топологии автономной системы выбирают так, чтобы ее можно было бы представить структурой данных с операциями поиска пути в графе приемлемой сложности. Для ПКС сетей такой подход не годится. Одна из причин – сложность контура управления – для каждой области свой контур управления.

Одним из возможных решений проблемы маршрутизации в ПКС сетях могут стать графовые базы данных (ГБД)[3]. ГБД - это специализированная платформа для построения графов и управления ими. Данные в графовых базах данных хранятся в виде вершин и ребер, которые представляют объекты и связи между ними. Узлы содержат информацию о каждом объекте, такую как его идентификатор, тип и произвольный набор атрибутов. Ребра представляют связи между узлами и содержат идентификатор,

информацию о типе связи, ее направлении и произвольный набор атрибутов. Графовые базы данных также предоставляют эффективно реализованные графовые алгоритмы, например поиск маршрутов.

2 Постановка задачи

Пусть $N =< G, f >$ - транспортная сеть с топологией, представленной графом $G =< V, E >$, где

- V - множество вершин,
- E - множество ребер, на котором задана функция разметки f
- $f : e \to d(e), e \in E, d(e) \in \mathbb{R}^+$ - вес ребра.

Пусть n - количество вершин в графе, m - количество вершин, $p = \frac{m}{n(n-1)}$ - плотность матрицы смежности графа.

Пусть задан поток отказов $R(t_i) = (r_1(t_i), \ldots, r_n(t_i))^T$ в дискретный момент времени $t_i = t_{i-1} + \Delta t$, где Δt - заданный интервал, $r_j(t_i)$ - вероятность отказа j-ого ребра в момент времени t_i.

Пусть $C(N, R, p)$ - функция стоимости стандартного алгоритма маршрутизации, а $G(N, R, p)$ - функция стоимости алгоритма маршрутизации, использующего графовую базу данных. Функции стоимости определяются как время, затраченное на расчёт маршрутов. Пусть T - время сходимости сети (время, требуемое на приведение таблиц в состояние, отражающее актуальное состояние сети). Таким образом, задача минимизации заключается в выборе алгоритма маршрутизации при заданных (N, R, p).

$$T = \min_{(N,R,p} \{C(N, R, p), G(N, R, p)\} \tag{1}$$

Дополнительной задачей является определение границ применимости алгоритмов $C(N, R, p)$ и $G(N, R, p)$, т. е. допустимых размеров топологии N и плотности матрицы смежности p при заданном потоке отказов R.

3 Обзор существующих решений

Графовые базы данных (ГБД) представляют собой специальный тип баз данных, ориентированных на хранение и обработку графовых структур данных. Графовые базы данных - это перспективный инструмент для анализа и обработки информации, структура которой может быть представлена графом.

Цель настоящего обзора заключается в выборе реализации графовой базы данных для проведения экспериментального исследования, основываясь следующих критериях: открытость СУБД, наличие реализаций графовых алгоритмов, масштрабируемость, производительность запросов, надежность и сложность использования. В данной работе рассматривается несколько самых популярных реализаций графовых баз данных, а именно: Neo4j[3], ArangoDB[4], MongoDB[5] и Titan[6].

Neo4j — это графовая система управления базами данных с открытым исходным кодом, реализованная на Java. Она является ведущей графовой СУБД в мире. С помощью Neo4j можно легко создавать и обрабатывать графы, а также использовать высокоуровневые инструменты, которые предоставляет система. Neo4j обладает простым и интуитивно понятным языком запросов Cypher. Также важно отметить, что СУБД Neo4j включает Graph Data Science (GDS) — это библиотека предоставляющая набор инструментов для анализа графовых данных в Neo4j. GDS разработана специально для Neo4j и обеспечивает мощные возможности для анализа графов, включая такие операции, как поиск маршрутов, анализ центральности, поиск сообществ, анализ схожести и другие. GDS предоставляет высокоуровневый API для выполнения запросов и операций над графом, что делает работу с графом более удобной и эффективной. Эта библиотека позволяет быстро и легко проводить сложные анализы на масштабных графах, что делает ее важной составляющей в экосистеме Neo4j.

ArangoDB — мультимодельная база данных с открытым исходным кодом. Она представляет собой комбинацию графовой базы данных, базы данных документов[7] и хранилище с доступом по ключу[8] в одной системе. ArangoDB обеспечивает высокую производительность и масштабируемость, а также поддерживает различные языки программирования. ArangoDB обладает универсальным языком запросов AQL, который поддерживает не только графовые запросы, но и запросы к документным и ключ-значение хранилищам. Это делает его более гибким в использовании, особенно в случаях, когда база данных содержит не только графовые данные.

MongoDB — мультимодельная база данных, с открытым исходным кодом. MongoDB – документоориентированная[7] СУБД, однако в последнее время ее функциональность была расширена до графовых структур данных. MongoDB обеспечивает высокую производительность и масштабируемость. Для хранения графовых данных в MongoDB можно использовать набор документов, которые представляют вершины графа, и другой набор документов, который представляет связи между вершинами. Однако, следует отметить, что MongoDB не имеет нативных инструментов для работы с графовыми данными, например таких как Cypher или AQL. В следствие чего работа с более сложными запросами к графам требует больше усилий и ресурсов для реализации операций, связанных с графовой обработкой.

Titan – графовая база данных, созданная для хранения большого объема данных. Она поддерживает Cassandra[6] в качестве внутреннего хранилища. Cassandra - это распределенная колоночно-ориентированная[7] база данных с открытым исходным кодом. Колоночно-ориентированные базы данных отличаются от традиционных СУБД тем, что они хранят данные не по строкам, а по столбцам. В Колоночно-ориентированных СУБД данные представлены в виде таблиц, как и в реляционных СУБД, однако физически эти таблицы являются совокупностью колонок, каждая из которых представляет собой таблицу из одного поля. Cassandra была создана, чтобы обеспечить масштабируемость и высокую отказоустойчивость при работе с большими объемами данных. Cassandra имеет различные преимущества для хранения графовых данных - она обладает высокой производительностью и возможностью масштабирования, что позволяет работать с большими графами. Кроме того, Cassandra поддерживает горизонтальное масштабировани и автоматическое реплицирование, т.е создания точных копий данных в одной или нескольких базах данных. Однако, Cassandra не является специализированной графовой базой данных, и не имеет нативной поддержки работы с графами, например такой как Cypher или AQL. Поэтому работа с графовыми данными в Cassandra может быть более сложной, чем в специализированных графовых базах данных.

Из рассмотренных реализаций графовых баз данных, Neo4j и ArangoDB являются узкоспециализированными системами управления графами. Они используют оптимизированные алгоритмы и структуры данных для хранения и обработки графов, поэтому Neo4j и ArangoDB имеют высокую производительность при работе с данными, представленными в виде графовой структуры, что является основным преимуществом для целей

маршрутизации в сети. Эти реализации также предоствляют удобные языки запросов и имеют высокую надежность. Neo4j и ArangoDB могут быть эффективно использованы для решения задачи маршрутизации в программно-конфигурируемых сетях. Однако главным преимуществом Neo4j является наличие GDS, что делает работу с графами более удобной и эффективной, поэтому Neo4j была выбрана в качестве исследуемой ГБД.

4 Методика и инструментарий сравнительного анализа

Для проведения сравнительного анализа эффективности использования графовой базы данных и традиционных подходов маршрутизации на ПКС контроллере по критерию минимизации времени сходимости в сети в зависимости от её топологии и потока отказов, была разработана следующая методика эксперимента.

Исследование проводится методом статистических испытаний на стенде, который включает имитационную модель сети (контура данных) Mininet[9], сетевую операционную систему RunOS[1] и ее приложения «маршрутизация» и приложение маршрутизации, использующее графовую базу данных Neo4j.

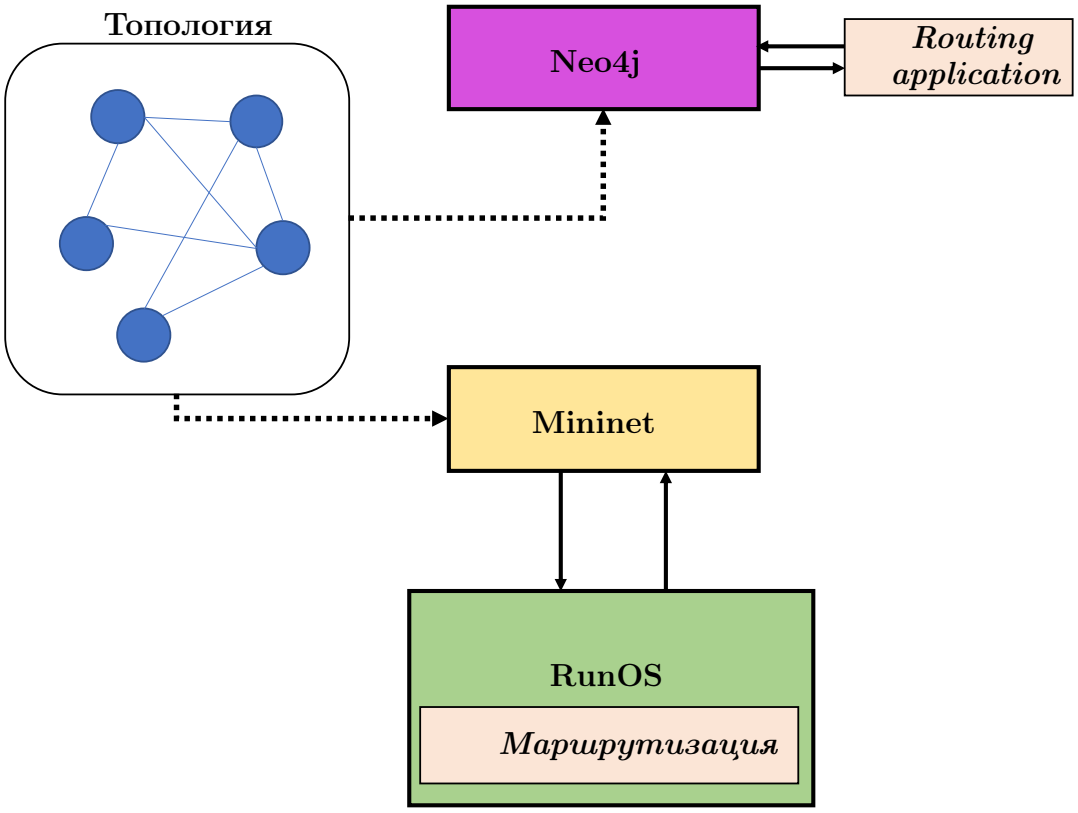

Рис. 1: Экспериментальный стенд

Методика проведения экспериментального исследования:

1. **RunOS**

   (a) Эмулирование тологии сети (контура данных) с помощью Mininet.

   (b) Измерение времени сходимости сети приложением *маршрутизация*, используемым в RunOS, при заданной топологии и потоке отказов.

2. **Neo4j**

   (a) Замена приложения *маршрутизация* на приложение, использующее для построения маршрутов графовую базу данных Neo4j.

   (b) Повторение измерения времени сходимости сети при использовании нового приложения на тех же самых входных топологиях и потоках отказов.

5 Описание практической части

1. **Эмулирование контура данных**

   Топология сети задается в XML файле *topology.gml*. Для разбора данного файла и извлечения информации о множествах *nodes* - множество вершин (сетевых устройств) и *edges* - множество ребер (линий связи между сетевыми устройствами), используется скрипт ***parsgml.py*** на языке Python с использованием библиотеки pygmlparser.

   (a) **RunOS**

   Для создания контура данных, представляющего топологию сети, используется эмулятор сети Mininet. По полученным из ***parsgml.py*** множествам вершин и ребер, с помощью Mininet Python API производится инициализация сетевой топологии и её подключение к ПКС контроллеру.

   (b) **Neo4j**

   По полученным из ***parsgml.py*** множествам вершин и ребер, генерируется Cypher запрос с использованием скрипта ***topology_ creator.py*** на языке Python. Далее, с помощью Neo4j Python API, осуществляется доступ к СУБД и отправка запроса на выполнение.

2. **Измерение времени расчета маршрута**

   (a) **RunOS**

   Для измерения времени расчета маршрута разработано приложение ***pcd*** на языке C++. Для построения маршрута приложение ***pcd*** использует функцию *newRoute* из стандартного приложения контроллера ***topology***.

   Для измерения времени выполнения функции построения маршрутов, была использована стандартная библиотека C++ *chrono*. Она обеспечивает высокую точность при измерении времени выполнения кода, так как в отличие от других методов измерения времени, таких как использование системных вызовов или сторонних библиотек, *chrono* работает непосредственно с высокочастотными таймерами процессора. Это позволяет ей точно определять время выполнения функций с точностью до наносекунд.

(b) **Neo4j**

Для измерения времени расчета маршрута с помощью ГБД, было разработано приложение на языке Python, которое в зависимости от выбранной метрики построения маршрутов формирует Cypher запрос для вызова функции *shortestPath*, либо функции *gds.shortestPath.dijkstra.stream*. Для коректной работы*gds.shortestPath.dijkstra.stream* необходимо сформировать проекцию графа с помощью *gds.graph.project*. Далее с помощью Neo4j Python API, осуществляется доступ к СУБД и отправка запроса на выполнение.

Для измерения времени выполения запроса используется стандратный модуль Python time. Модуль time является стандартной библиотекой Python и предоставляет функции для измерения времени выполнения операций и функций в Python. Эта библиотека является надежным инструментом для измерения времени выполнения кода в Python. Модуль time позволяет измерять время с точностью до микросекунды и меньше в зависимости от используемого аппаратного обеспечения и операционной системы. Это гарантирует высокую точность при измерении времени выполнения кода.

6 Результаты

При проведение эксперимательного исследования были сделаны следующие допущения:

1. $f \equiv 1$, то есть используется метрика хопов
2. Поток отказов R имеет равномерное распределение и в каждый момент времени t_i отказывает только одно ребро.

Далее приведены результаты экспериментального исследования.
Обозначения:

- $N-$ количество вершин в топологии
- $p-$ плотность топологии
- Горизонтальная ось - дискретный момент времени t_i

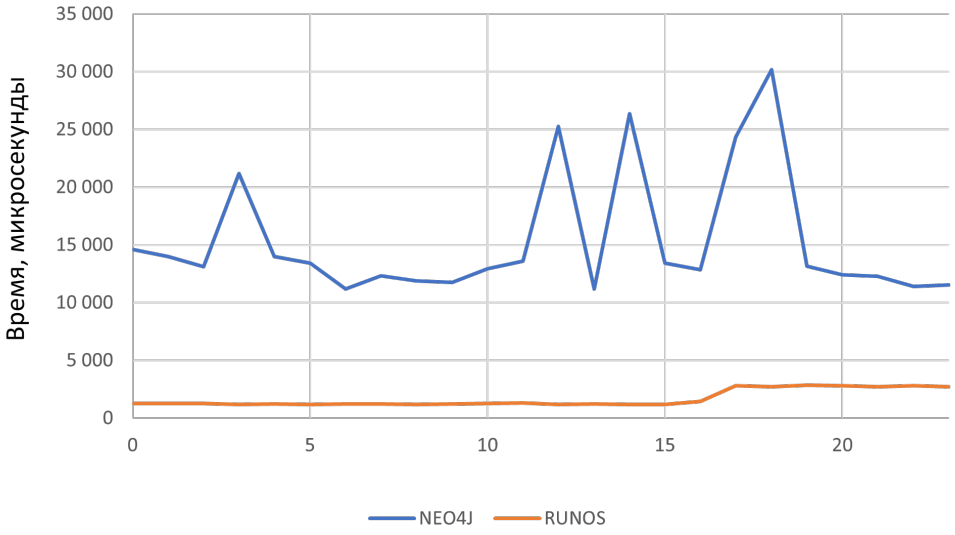

Рис. 2: $N = 34$, $p = 0.08$

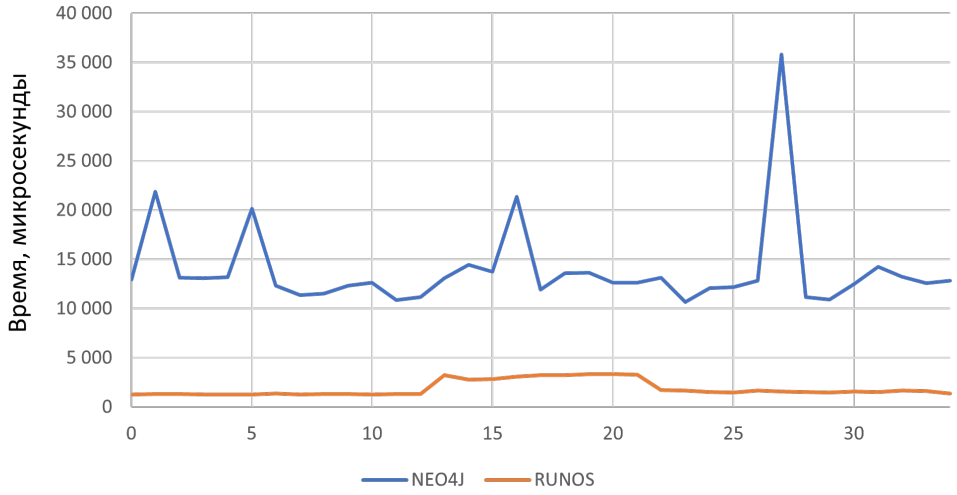

Рис. 3: $N = 54$, $p = 0.04$

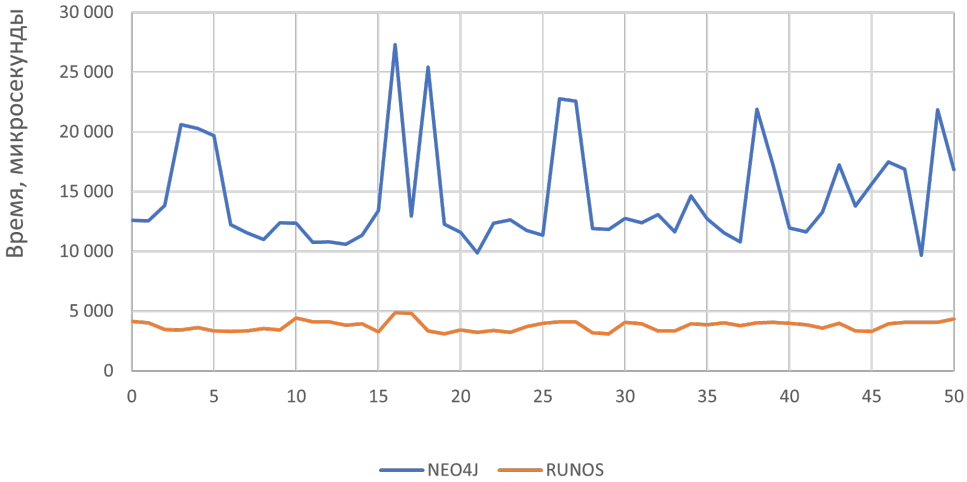

Рис. 4: $N = 153$, $p = 0.016$

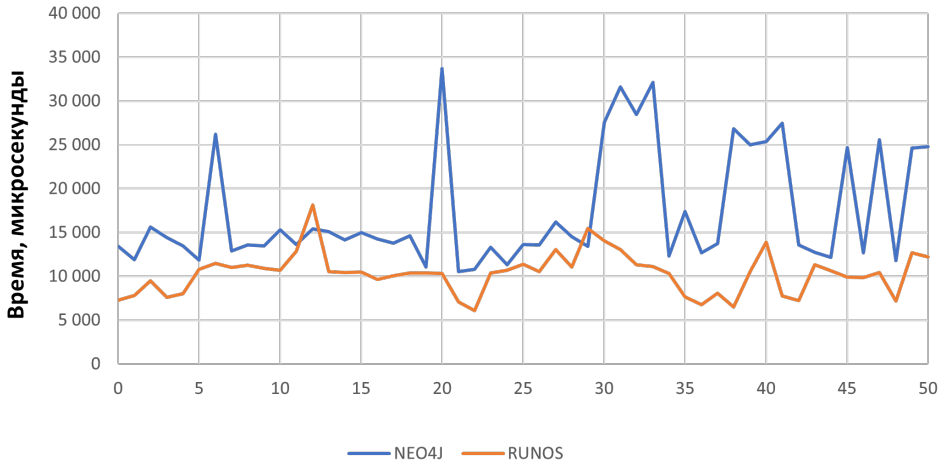

Рис. 5: $N = 258$, $p = 0.013$

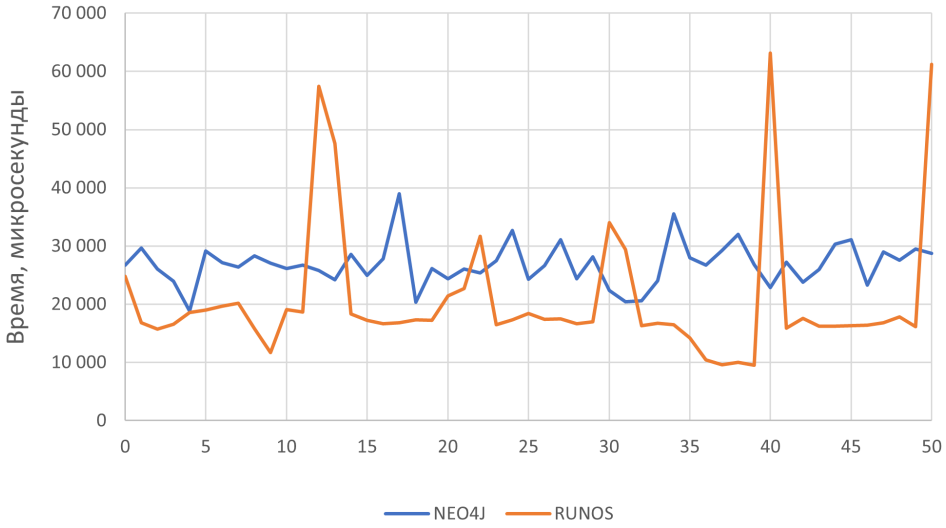

Рис. 6: $N = 500$, $p = 0.008$

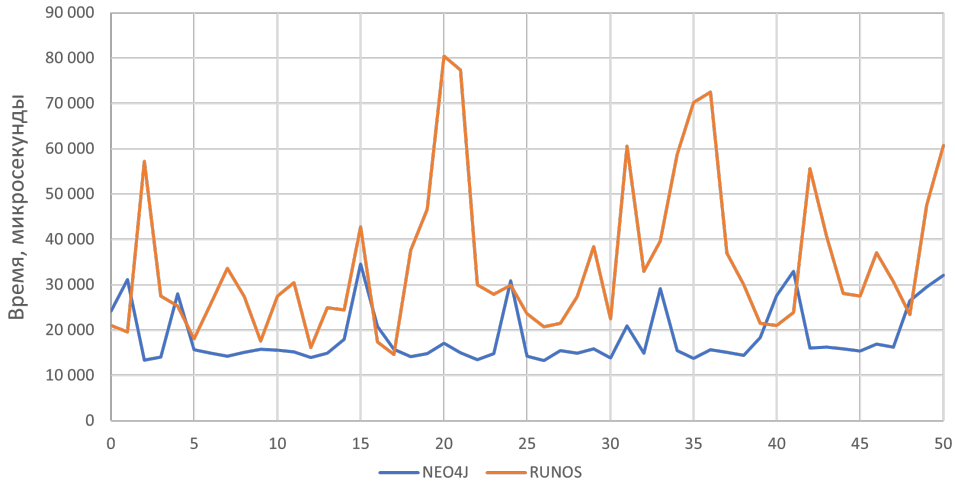

Рис. 7: $N = 730$, $p = 0.005$

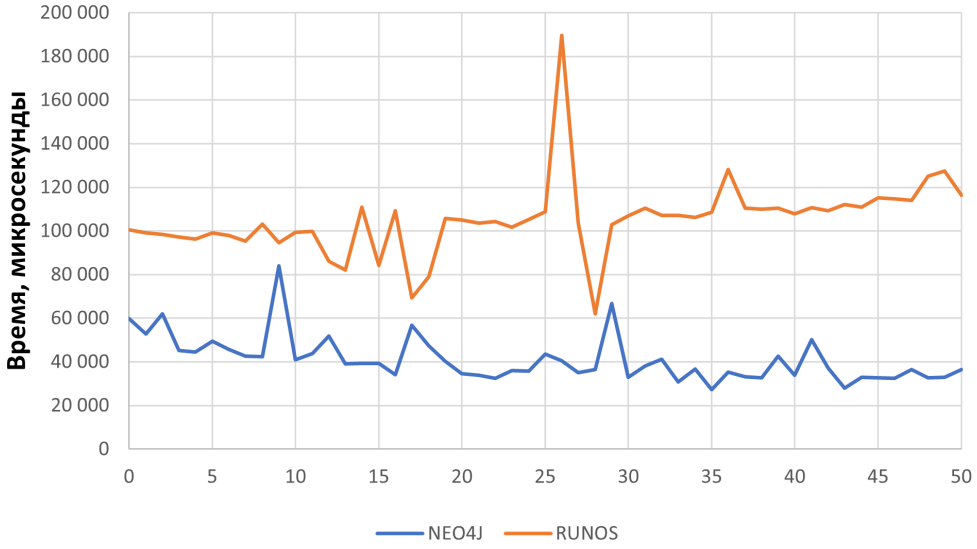

Рис. 8: $N = 1025$, $p = 0.003$

Когда количество вершин в топологии сети растет, становится заметно, что время, необходимое для расчета маршрутов при использовании приложения *маршрутизация* в RunOS, начинает превышать время, затрачиваемое на расчеты с помощью приложения, использующего графовую базу данных.

Рис. 9

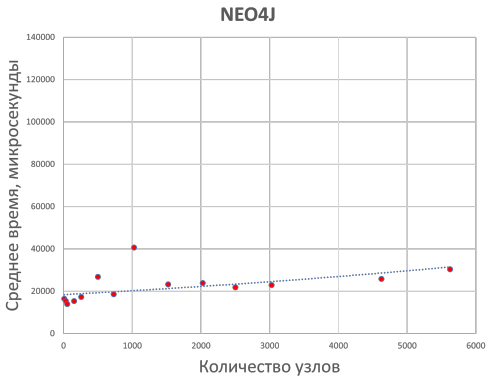

Рис. 10

Ниже приведены результаты исследования, которое показывает зависимость времени расчета маршрутов от плотности топологии при разном количестве вершин в топологии.

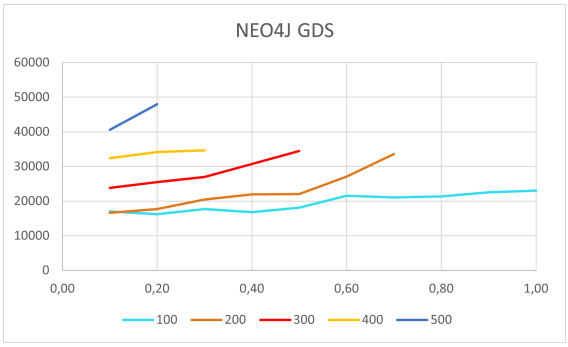

Рис. 11

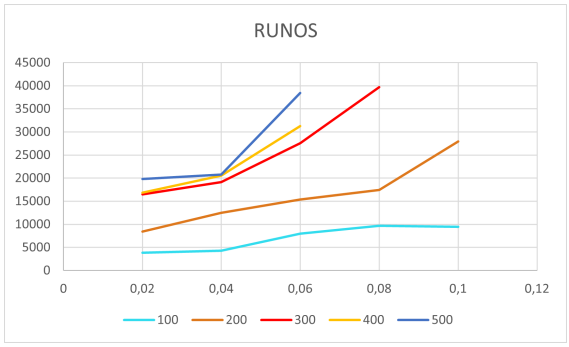

Рис. 12

Neo4j легко масштабируется и сохраняет высокую производительность при топологиях с большой плотностью. В отличие от RunOS, где информация о топологии хранится в оперативной памяти и имеет ограничения, данные в Neo4j хранятся на диске в формате, который оптимизирован для эффективного поиска и обновления графовой структуры. Использование индексации и кэширования в Neo4j позволяет

быстро находить узлы и связи, а поддержка параллельного выполнения запросов ускоряет работу с графами больших размеров.
Neo4j предоставляет еще один инструмент для поиска маршрутов - функцию *shortestPath*, но она применима только к метрике хопов. Эта функция работает быстрее, чем функции расчета маршрутов из GDS, потому что она не требует дополнительных затрат на создание проекции графа и других операций, а работает напрямую со структурой графа.

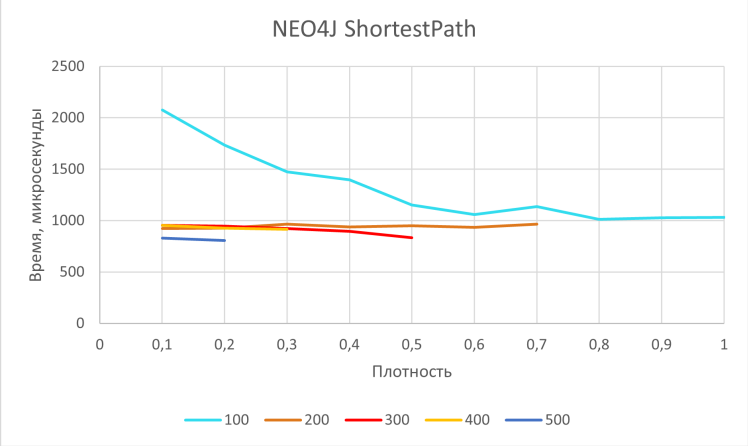

Рис. 13

При увеличении плотности топологии, то есть количества ребер, можно заметить, что время расчета маршрута сокращается, в отличие от предыдущих результатов, которые показывали увеличение времени при бóльшей плотности. Этот эффект объясняется использованием поиска в ширину в *shortestPath*, и тем, что с увеличением плотности возрастает вероятность наличия прямого пути между двумя узлами.

7 Заключение

В результате проведенной работы был выполнен сравнительный анализ различных графовых баз данных для маршрутизации в ПКС сетях. Из предложенных баз данных - Neo4j, ArangoDB, MongoDB, Titan - была выбрана Neo4j по ряду критериев, таких как открытость СУБД, наличие реализаций графовых алгоритмов, масштабируемость, производительность запросов, надежность и сложность использования.

Для проведения экспериментального исследования был разработан стенд, который включал в себя имитационную модель сети (контура данных) Mininet, сетевую операционную систему RunOS и приложение «маршрутизация», а также приложение маршрутизации, использующее графовую базу данных Neo4j.

В результате экспериментального исследования было выявлено, что графовые базы данных эффективны при работе с большими топологиями, где количество узлов превышает 500. Также было показано, что графовые базы данных лучше всего справляются с плотными топологиями.

Таким образом, можно сделать вывод, что графовые базы данных хорошо подходят для решения задачи маршрутизации в ПКС при больших топологиях и могут быть эффективным инструментом для управления сетью.

Список литературы

[1] *Смелянский Р. Л., Антоненко В. А.* Концепции программного управления и виртуализации сетевых сервисов в современных сетях передачи данных. / Антоненко В. А. Смелянский Р. Л. — "Москва Курс 2019.

[2] *"TAdviser"*. Программно-определяемые сети Software-Defined Network, SDN. — TAdviser - российский интернет-портал и аналитическое агентство. `https://www.tadviser.ru/a/158716`.

[3] *Robinson I. Webber J., Eifrem E.* Graph databases: new opportunities for connected data. / Eifrem E. Robinson I., Webber J. — "O'Reilly Media, Inc. 2015.

[4] *Fernandes D., Bernardino J.* Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. / Bernardino J. Fernandes D. — "Data. 2018. — Pp. 373–380.

[5] *Parmar R. R., Roy S.* MongoDB as an efficient graph database: An application of document oriented NOSQL database. / Roy S. Parmar R. R. — "Data Intensive Computing Applications for Big Data. 2018. — Pp. 331–358.

[6] *Mishra V., Mishra V.* Titan graph databases with cassandra. / Mishra V. Mishra V. — "Beginning Apache Cassandra Development. 2014. — Pp. 123–151.

[7] *Nayak A. Poriya A., Poojary D.* Type of NOSQL databases and its comparison with relational databases. / Poojary D. Nayak A., Poriya A. — "International Journal of Applied Information Systems. 2013. — Pp. 16–19.

[8] *Nguyen T. T., Nguyen M. H.* Zing database: high-performance key-value store for large-scale storage service. / Nguyen M. H. Nguyen T. T. — Vietnam Journal of Computer Science., 2015. — Pp. 13–23.

[9] *Kaur K. Singh J., Ghumman N. S.* Mininet as software defined networking testing platform. / Ghumman N. S. Kaur K., Singh J. — "International conference on communication, computing systems (ICCCS). 2014. — Pp. 139–142.