размер больше одного, а значит нам нужно найти количество взятий операции корня к числу N для того, чтобы оно обратилось в единицу. Количество таких операций равно $O(\log\log n)$. Теперь у нас есть всё, чтобы оценить необходимые ресурсы для работы структуры данных:

- Время работы предпроцесса: $O(n\log\log n)$.
- Необходимая память: $O(n\log\log n)$.
- Время ответа на один запрос: $O(\log\log n)$ или $O(1)$, при использовании битовых операций для определения нужного слоя.
- Время модификации элемента: $O(\sqrt{n})$.

В данной структуре без проблем можно использовать любую операцию, для которой выполняется свойство ассоциативности. Для примера, такими операциями являются: сложение, умножение, а также все битовые операции.

Немного о скорости относительно других известных структур данных на бенчмарке состоящем из 10^7 запросов и 10^5 элементов.

алгоритм\ количество модификаций	0	100	10^5	10^7
Segment tree	10.11 сек.	10.28 сек.	11.78 сек.	12.54сек.
Sqrt tree	2.01 сек.	2.28сек.	4.92сек.	13.28сек.
Sqrt decomposition	138.17 сек.	138.75сек.	140.32сек.	162.83сек.

Как видно структура работает в несколько раз быстрее, чем другие структуры данных, пока дело не доходит до большого числа изменений.

Список использованных источников:
1. N Alon, B Schieber, Optimal preprocessing for answering on-line product queries
2. Т. Кормен, Алгоритмы. Построение и анализ
3. Sqrt Decomposition, e-maxx.ru

ГРАФОВЫЕ БАЗЫ ДАННЫХ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Чурин А.П., Владыко В.Д.

Теслюк В.Н. – к.ф-м.н., доцент

В настоящее время есть потребность для обработки разноордных данных, которые имеют непредсказуемую структуру. Которая может превратиться либо в BigData, либо в сложную семантическую сеть, и зачастую мы не можем предвидеть, какой она будет.

Как альтернатива базам данных SQL с 2000-х годов развивается направление NoSQL. В эту категорию попадают – от иерархических и сетевых БД до упрощённых БД, которые имеют подобие хэш-таблиц, и документарные БД. Причиной эволюции базы данных заключается в следующем: если программный продукт оперирует однотипными и примитивными данными, то можно использовать классический подход к разработке БД. Но если нужно обратиться к 10, 100, 1000 таблицам, чтобы получить данные, то реляционная база данных начинает работать медленно, а написание такого запроса займет довольно много времени.

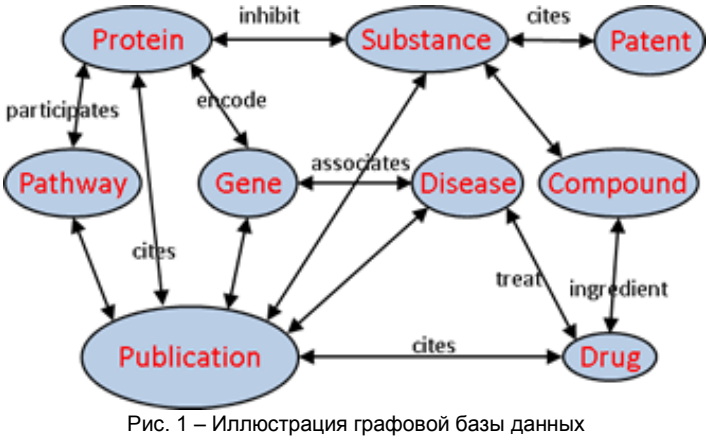

Рис. 1 – Иллюстрация графовой базы данных

Наконец отдельным классом стоят NoSQL – к ним относят графовые базы данных. Они располагают функционалом для естественного предоставление информации. Прямая альтернатива SQL – документарные базы данных, их главная особенность –это отсутствие схемы, присущие реляционным базам данных.

Преимущества графовых баз данных:

- База может сохранять любой объект, например объект с большим коллиствам полей.
- Гибкая система расширения и модернизация в любой момент времени.

Недостатки графовых баз данных:

- Проблемы с работой в параллельных архитектурах.
- Низкая производительность при большом колличестве связей и больших объемах.

Графовые базы данных оказываются универсальном вариантом, которые помогают подстраховаться на случай изменения требований и увеличением либо уменьшением функционала. Добавление новых связей делает непригодной для использования документарную базу данных, а рост количества сырых объектов с несвзяной структурой снижает производительность реляционной БД. Сегодня ведется активная разработка и доработка RDF – основного стандарта, согласно которому работают графовые базы данных. Именно стандартизация SQL делала популярным реляционные БД. При этом ряд проектов демонстрируют поддержку OData, для создания классических веб-запросов через HTTP. Язык SPARKQL, обладающий возможностями для работы с вариативными видами запросов и данных. За счет развития архитектуры производительность графовых баз данных тоже растет, возможно в будущем за счет развития окажется выше реляционной даже при небольшом количестве связей.

Список использованных источников:

1. Ян Робинсон, Джим Вебер. Грфовые базы данных. Новые возможности для работы со связными данными. 2016. – 256с.

КЛАСТЕРИЗАЦИЯ В HADOOP

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Владыко В.Д., Чурин А.П.

Теслюк В.Н. – к.ф.-м.н., доцент

Непрерывный рост данных и увеличение скорости их генерации порождают проблему их обработки и хранения. Неудивительно, чтотема «больших данных» (Big Data) является одной из самых обсуждаемых. Одним из самых известных проектов в области распределенных вычислений является Hadoop — набор из утилит, библиотек и фреймворк для разработки и выполнения программ распределенных вычислений. Кластеризация один из важных моментов работы с большими данными.

Hadoop – это проект с открытым исходным кодом, находящийся под управлением Apache Software Foundation. Hadoop используется для надежных, масштабируемых и распределенных вычислений, но может также применяться и как хранилище файлов общего назначения, способное вместить петабайты данных.

В состав проекта Hadoop входят следующие подпроекты:

Common — набор компонентов и интерфейсов для распределенных файловых систем и общего ввода-вывода;

MapReduce — модель распределённых вычислений, предназначенная для параллельных вычислений над очень большими (до нескольких петабайт) объемами данных;

HDFS — распределенная файловая система, работающая на больших кластерах типовых машин.

Классическая конфигурация кластера Hadoop состоит из одного сервера имён (NameNode), одного мастера MapReduce (т.н. JobTracker) и набора рабочих машин, на каждой из которых одновременно крутится сервер данных (DataNode) и обработчик (TaskTracker).

Namenode — сервер имён. Как правило, один узел на кластер. Хранит в себе все метаданные системы, сами файлы на этом узле не хранятся.

JobTracker — узел, который координирует параллельную обработку данных используя MapReduce.

TaskTracker — узел, который принимает задачи по обработке данных от JobTracker.

DataNode — таких узлов в кластере очень много. Они хранят непосредственно блоки файлов. Узел регулярно отправляет NameNode свой статус и ежечасно — информацию обо всех хранимых на этом узле блоках. Это необходимо для поддержания нужного уровня репликации.

При правильной архитектуре приложения, с помощью информации о том, на каких машинах (узлах) расположены блоки данных, позволяет запустить на них же вычислительные процессы и выполнить большую часть вычислений локально, т.е. не передавая данные по сети.

Кластеры в Hadoop позволяют ускорить анализ данных для приложений и улучшить их масштабируемость. Если при растущих объемах данных, кластер начинает не справляться, то можно добавить дополнительные узлы для увеличения пропускной способности. Также кластер обладает высокой отказоустойчивостью, поскольку каждый блок данных копируется на другие узлы, гарантируя, что данные не будут потеряны, даже если один из узлов выдаст ошибку.