Оптимизация процесса дедупликации на основе баз данных

Н.П. Плотникова, В.А. Кевбрин

Мордовский государственный университет имени Н.П. Огарёва

Аннотация: Нельзя представить настоящее время без программного обеспечения. Огромные потоки информации проходят через компьютерные вычислительные системы. Совершенно невозможно обработать неструктурированные бесконечно поступающие данные, поэтому необходимо выделять конкретные задачи и подготавливать информацию к обработке. Одним из подобных действий является дедупликация. В данной статье рассматриваются возможные оптимизации метода удаления дубликатов с использованием баз данных.

Ключевые слова: дедупликация, база данных, поле, строка, текстовые данные, запрос, программное обеспечение, неструктурированные данные.

Введение

Вопрос дедупликации[1] данных уже рассматривался в статье [2]. В текущей статье анализируются возможные варианты и подходы к оптимизации данного процесса. Стоит сразу отметить основное отличие, в предыдущем варианте алгоритм подходил для обработки потоков данных, которые будут поступать постоянно. Сейчас анализируется вариант, когда все данные уже имеются сразу и сохранены в некотором виде.

Описание тестового стенда и исходных данных

Имеется текстовый файл, который хранит в себе строки в формате json [3, 4], показано на рис.1.


```
{"id":17,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000013172","part_91":"","part_92":""}CRLF
{"id":18,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000017821","part_91":"","part_92":""}CRLF
{"id":19,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000019670","part_91":"","part_92":""}CRLF
{"id":20,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000022386","part_91":"","part_92":""}CRLF
{"id":21,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000031054","part_91":"","part_92":""}CRLF
{"id":22,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000034245","part_91":"","part_92":""}CRLF
{"id":23,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000041157","part_91":"","part_92":""}CRLF
{"id":24,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000041803","part_91":"","part_92":""}CRLF
{"id":24,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000041803","part_91":"","part_92":""}CRLF
{"id":25,"gtin_id":11203,"batch_id":11204,"part_01":"04605949001871","part_21":"0000000048308","part_91":"","part_92":""}CRLF
```


Рис. 1. – Пример исходных данных

Объём файла составляет 17.3 гигабайта (далее Гб). Количество строк – 133327409, из них 33327409 дубликаты. Все описанные дальше тесты были

реализованы на языке программирования Python 3.9 в среде разработки PyCharm. В качестве СУБД протестированы Clickhouse [5, 6], Postgres [7, 8]. Для удобства и анализа использованы: wsl 2, docker [9].

Основной алгоритм

Основная последовательность действий не будет изменяться во время тестирования. Изменениям подлежат параметры работы, настройки баз данных (далее БД). Алгоритм дедупликации:

1) Построковое чтение текстового файла;

2) Накопление определенного количества данных (работа осуществляется с блоками batch_size = 10^6 строк);

3) Сохранение блока данных в БД;

4) Если текстовый файл не прочитан полностью, возвращение к пункту 1;

5) Дедупликация данных – может происходить автоматически во время записи данных в БД или при получении данных из БД;

6) Сохранение результата дедупликации блоками в множество текстовых файлов, разбитых по размеру batch_size.

Описание предыдущих результатов и показателей новых тестов

Оптимизация происходит в несколько этапов, первым из которых является установление точки отсчета. За такую точку были взяты результаты дедупликации из статьи [2], показаны в таблице 1, которые имели наилучшие показатели. Основными метриками являются:

1) Средние затраты мощностей CPU;
2) Средние затраты оперативной памяти;
3) Время выполнения.

Так как в рамках данной статьирассматривается поэтапная обработка всего объема данных, то текущие тесты стоит дополнительно разделить на этапы записи и чтения БД.

Таблица № 1

Результаты алгоритма дедупликации из статьи 1

Параметр / Тип базы данных	Clickhouse	SSDB	Postgres №1	Postgres №2
БД CPU (среднее), %	689.7	38.61	44.34	93.49
БД RAM (среднее), Гб	5.314	1.981	7.65	10.51
Время выполнения, с	3046.6	5815.6	6762	8550.1
Занимаемый объем физической памяти, Гб	3.1	5.1	10.8	14.6

Оптимизация

В первую очередь стоит попробовать работу со строками напрямую. Работа в БД осуществляется с таблицей, состоящей из одного текстового поля.

Таблица № 2

Результаты тестирования с простой таблицей из одного поля

Процесс	Параметр	Clickhouse	Postgres
Чтение текстового файла, сохранение данных в БД	БД CPU (среднее), %	35	24.29
	БД RAM (среднее), Гб	0.576	3.9
	Время выполнения, с	683	2889
Извлечение результата из БД, сохранение в текстовые файлы	БД CPU (среднее), %	1000	300
	БД RAM (среднее), Гб	17.12	21.17
	Время выполнения, с	1928	22798
Оба процесса	Время выполнения, с	2611	25687

Анализируя полученные результаты, можно сделать следующие выводы:

1) Clickhouse отрабатывает на вставку гораздо быстрее чем PostgreSQL;

2) Время отработки процесса меньше, чем результаты, полученные в предыдущей статье;

3) Принцип записи данных в итоговые текстовые файлы немного различался, поэтому далее в тестах был использован оптимизированный вариант кода save_batch_2, показанный на рис.2.

```
192     def save_batch(self, data_batch: list, file_path: Path):
193         start_time = time.time()
194         with open(Path(workdir, file_path), 'wb') as out_f:
195             for data in data_batch:
196                 out_f.write(json.dumps(data))
197                 out_f.write(b'\n')
198         self.logger.info(f"Save batch time: '{time.time() - start_time}'")
199
            5 usages
200     def save_batch_2(self, data_batch: list, file_path: Path):
201         start_time = time.time()
202         open(Path(workdir, file_path), 'wb').write(b"\n".join([json.dumps(row) for row in data_batch]))
203         self.logger.info(f"Save batch time: '{time.time() - start_time}'")
```

Рис. 2. – Код сохранения блока дедуплицированных данных в файл

Как уже упоминалось выше, принцип сохранения результатов обработки в текстовые файлы в таблице 1 отличается. В тесте с Postgres использовался более медленный вариант, время сохранения одного блока которого было равно 240 секундам. Оптимизация этого участка кода позволила уменьшить это время до 15 секунд. Во всех приведенных далее тестах будет использоваться оптимизированный вариант сохранения данных.

Стоит обратить внимание на индексы в БД и провести следующую серию тестов. В Clickhouse по умолчанию уже использовался engine=MergeTree. Сейчас возьмем ReplacingMergeTree, а для PostgreSQL наложим UNIQUE. Результаты продемонстрированы в таблице 3.

Таблица № 3

Результаты тестирования с простой таблицей с индексацией

Процесс	Параметр	Clickhouse	Postgres
Чтение текстового файла, сохранение данных в БД	БД CPU (среднее), %	100	57
	БД RAM (среднее), Гб	0.9	10.1
	Время выполнения, с	677	4751
Извлечение результата из БД, сохранение в текстовые файлы	БД CPU (среднее), %	1000	100
	БД RAM (среднее), Гб	10.32	15.71
	Время выполнения, с	1912	1012
Оба процесса	Время выполнения, с	2589	5763

По полученным выше результатам можно сделать следующие выводы:

1) Clickhouse при работе с одним текстовым полем не сильно зависит от выбранного engine, результаты очень похожие (в пределах погрешности);

2) PostgreSQL при включении индексации сильно увеличил время вставки одного блока данных в БД, чуть менее чем в два раза.

Выбранный подход расходует очень много физической памяти, так как мы копируем весь объем данных в БД. Стоит попробовать устранить этот недостаток. Для этого изменим структуру таблицы в БД. Вместо одного текстового поля создадим два поля:

1) Id – номер строки в исходном текстовом файле;

2) Hash_line – строка преобразованная в хеш [10].

Теперь вместо хранения непосредственно строки, будем высчитывать ее хеш. Данный подход подразумевает неоднократное прохождение по исходным данным. В первый раз, чтобы рассчитать и сохранить уникальные хеши. Во второй раз, чтобы отобрать строки по уникальными хешам, доставая номер строки из поля id. Это необходимо производить в порядке

возрастания номера, поэтому в запрос обязательно вносим ORDER BY. Результаты теста приведены в таблице 4.

Таблица № 4

Результаты тестирования на основе хешей

Процесс	Параметр	Clickhouse	Postgres
Чтение текстового файла, сохранение данных в БД	БД CPU (среднее), %	4.42	53
	БД RAM (среднее), Гб	8.161	8.5
	Время выполнения, с	856	6533
Извлечение результата из БД, сохранение в текстовые файлы	БД CPU (среднее), %	1000	300
	БД RAM (среднее), Гб	22.2	15.1
	Время выполнения, с	4544	2347
Оба процесса	Время выполнения, с	5400	8880

Из приведенных выше результатов следуют следующие выводы:

1) Экономия физической памяти привела к увеличению затрат времени в обеих БД;

2) Замедление произошло как в процессах вставки блоков данных, так и извлечения.

3) Далее этот принцип использовать не следует.

Последним этапом тестов было использование многопоточности. Следует сразу отметить, что Clickhouse не подходит для этого. Можно обратить внимание на то, что во всех тестах Clickhouse использовал 1000% ресурсов CPU, что эквивалентно десяти ядрам процессора – это вся мощность, что может использовать docker на данном тестовом стенде. Тем не менее были предприняты попытки организовать мультипоточную работу с библиотеками clickhouse_driver и clickhouse_connect. В результате

сохранение данных всё равно происходило последовательно, а чтение зависало. В таблице 5 представлены результаты тестов с Postgres.

Таблица № 5

Результаты тестирования PostgreSQL в многопоточном режиме

Процесс	Параметр	Postgres 4 потока	Postgres 10 потоков
Чтение текстового файла, сохранение данных в БД	БД CPU (среднее), %	719	701
	БД RAM (среднее), Гб	15.7	16
	Время выполнения, с	1057	1007
Извлечение результата из БД, сохранение в текстовые файлы	БД CPU (среднее), %	150	164.49
	БД RAM (среднее), Гб	16.06	17.1
	Время выполнения, с	517	559
Оба процесса	Время выполнения, с	1574	1566

По полученным результатам можно сделать следующие выводы:

1) Увеличение количества потоков целесообразно только на небольшое количество;
2) Clickhouse плохо работает с многопоточностью.

Заключение

По итогу многочисленных экспериментов с дедупликацией на основе баз данных был улучшен результат работы с 3046.6 секунд до 1574 секунд. Стоит отметить следующие особенности:

1) Clickhouse показал непревзойденные результаты при вставке блоков данных;

2) При оптимизации работы с PostgreSQL, разница в скорости выборки данных стала превосходить потери при сохранении.

Таким образом, после оптимизации программного обеспечения предпочтительным вариантом стала работа с PostgreSQL в многопоточном режиме.

Литература

1. Жуков М.А. Афанасьев Д.Б. Анализ и оценка минимального уровня префиксного дерева в системе бесхешевой дедупликации // Научно-технический вестник информационных технологий, механики и оптики. 2015. №3 С. 470-475.

2. Плотникова Н.П., Кевбрин В.А., Болохов Д.А. Дедупликация больших объемов данных при помощи баз данных // Инженерный вестник Дона. 2023. №9 URL: ivdon.ru/ru/magazine/archive/n9y2023/8676

3. Crockford D. JSON: The Fat-Free Alternative to XML // Boston 2006. URL: json.org/fatfree.html#:~:text=JSON%3A%20The%20Fat%2DFree%20Alternative%20to%20XML

4. Наумов Р.К., Железков Н.Э. Сравнительный анализ форматов хранения текстовых данных для дальнейшей обработки методами машинного обучения // Научный результат. Информационные технологии. 2021. №1. С. 40-47.

5. Mostafa J., Chilingaryan S., Wehbi S., Kopmann A. SciTS: A Benchmark for Time-Series Databases in Scientific Experiments and Industrial Internet of Things // Proceedings of the 34th International Conference on Scientific and Statistical Database Management. 2022. P. 12.

6. Терских М.Г. Развертывание кластера для хранения и обработки статистики с помощью Yandex Clickhouse // Теория и практика современной науки. 2017. №11. С. 540-547.

7. Плотникова Н.П., Мартынов В.А. Применение дерева отрезков в PostgreSQL // Инженерный вестник Дона. 2023. №9 URL: ivdon.ru/ru/magazine/archive/n9y2023/8684

8. Hellerstein J.H. Looking back at Postgres // Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker. 2018. P. 205–224.

9. Karim Z. Take the confusion out of Docker, VMs, and microservices // IBM Developer 2019 URL: developer.ibm.com/articles/breaking-down-docker-and-microservices (дата обращения: 15.08.2023).

10. Rivest R. The MD5 Message-Digest Algorithm // RFC 1321. MIT Laboratory for Computer Science and RSA Data Security: 1991. URL: ietf.org/rfc/rfc1321.txt

References

1. Zhukov M.A. Afanas'ev D.B. Nauchno-tehnicheskij vestnik informacionnyh tehnologij, mehaniki i optiki. 2015. №3. pp. 470-475.

2. Plotnikova N.P., Kevbrin V.A., Bolohov D.A. Inzhenernyj vestnik Dona. 2023. №9. URL: ivdon.ru/ru/magazine/archive/n9y2023/8676

3. Crockford D. JSON: The Fat-Free Alternative to XML. Boston: 2006. URL: json.org/fatfree.html#:~:text=JSON%3A%20The%20Fat%2DFree%20Alternative%20to%20XML

4. Naumov R.K., Zhelezkov N.E. Nauchnyj rezul'tat. Informatsionnye tekhnologii. 2021. №1. pp. 40-47.

5. Mostafa J., Chilingaryan S., Wehbi S., Kopmann A. SciTS: A Benchmark for Time-Series Databases in Scientific Experiments and Industrial Internet of Things. Proceedings of the 34th International Conference on Scientific and Statistical Database Management. 2022. P. 12.

6. Terskih M.G. Teorija i praktika sovremennoj nauki. 2017. №11. pp. 540-547.

7. Plotnikova N.P., Martynov V.A. Inzhenernyj vestnik Dona. 2023. №9. URL: ivdon.ru/ru/magazine/archive/n9y2023/8684

8. Hellerstein J.H. Looking back at Postgres. Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker. 2018. pp. 205–224.

9. Karim Z. Take the confsion out of Docker, VMs, and microservices. IBM Developer 2019 URL: developer.ibm.com/articles/breaking-down-docker-and-microservices (date assessed: 15.08.2023).

10. Rivest R. The MD5 Message-Digest Algorithm. RFC 1321. MIT Laboratory for Computer Science and RSA Data Security: 1991. URL: ietf.org/rfc/rfc1321.txt

Дата поступления: 30.12.2023

Дата публикации: 3.02.2024