Вопросы дедупликации и сжатия данных в ClickHouse

Ответы на вопросы

1. Есть ли в кликхаусе дедупликация?
   - В ClickHouse под дедупликацией понимается процесс удаления дублирующих строк из набора данных. Других вариантов дедупликаци нет.
2. Будут ли дедуплицироваться два значения разных типов, например значение uint64 12345 и uint32 12345 будут дедуплицированы? Т.е. происходит ли дедупликация по значению или по значению и типу?
   - Таких вариантов дедупликации нет
   - Для оптимизации хранения данных в пределах столбца можно настроить сжатие данных
3. Происходит ли глобальная дедупликация значений между разными таблицами, разными нодами?
   - Не происходит
4. Получим ли мы увеличение коэффициента сжатия, если выделим под кликхаус не диск, а сторадж у которго будет включено свое сжатие и своя дедупликация?
   - Сжатие на уровне ClickHouse влияет на объем данных, которые сохраняются на диск узла кластера. Влияние сжатия или дедупликации на уровне дисковой подсистемы Вам необходимо исследовать самостоятельно. В Yandex Cloud нет технической возможности исследовать этот вопрос.
5. Какого размера должен быть блок данных на сторадже, чтобы получить максимальное сжатие и максимальную дедупликацию?
   - Вам необходимо исследовать самостоятельно. В Yandex Cloud нет технической возможности исследовать этот вопрос.
6. Сколько будет стоит сохранить значение 12345 1000 раз, размещенное в разных строках и в полях с разными типами в разных таблицах на разных нодах и разных шардах? Вы возьмете 1 раз за хранение 12345 или умножите на 1000 ?
   - Если предположить, что это значение не повторяется в одном столбце, где есть возможность оптимизации хранения за счет сжатия, то на дисковой подсистеме Yandex Cloud значение будет сохранено 1000 раз.
7. Давайте разберем вариант когда у вас канал гарантированно не теряющий пакеты с шириной скажем 400Гб/сек и сторадж со сжатием и дедупликацией. Вопрос, мне зачем сжимать и тратить CPU на ноде, если это так и так произойдет на сторадже? И лучше не сжимать чтобы дедупликация была более эффективна.
   - Цитата: *Меньше данных на диске означает меньше I/O и более быстрые запросы и вставки. В большинстве случаев накладные расходы любого алгоритма сжатия по отношению к CPU перевешиваются сокращением операций ввода-вывода. Следовательно, улучшение сжатия данных должно быть первым приоритетом при обеспечении быстрой работы запросов ClickHouse.*

О сжатии данных в ClickHouse

- Данные для разных столбцов изолируются и сжимаются независимо друг от друга. Повторяющиеся значения в пределах одного столбца эффективно сжимаются, но повторяющиеся значения в разных столбцах не оптимизируются.

- Тесты показали, что на размер хранения данных не влияет повторяемость значений в разных столбцах или разных таблицах. Суммарный объем хранения на диске одинаков в тестах с одинаковыми и разными значениями.
- Выбор типа данных для столбцов влияет на размер хранения. Но с учетом сжатия разница в объеме хранения тестовых данных оказалась незначительная для разных типов данных.
- Сжатие в ClickHouse является основным методом оптимизации хранения данных и повышения производительности выполнения запросов. Цитата: *Меньше данных на диске означает меньше I/O и более быстрые запросы и вставки. В большинстве случаев накладные расходы любого алгоритма сжатия по отношению к CPU перевешиваются сокращением операций ввода-вывода. Следовательно, улучшение сжатия данных должно быть первым приоритетом при обеспечении быстрой работы запросов ClickHouse.*

Результаты исследования хранения в ClickHouse одинаковых значений в разных столбцах разных типов и разных таблицах

1. Хранение одинаковых значений в разных столбцах разного типа

Тестовая таблица

```
CREATE TABLE test
(
    id UInt64,
    data1_uint32 UInt32,
    data1_uint32_zstd UInt32 CODEC(ZSTD),
    data2_uint64 UInt64,
    data2_uint64_zstd UInt64 CODEC(ZSTD)
)
ENGINE = MergeTree()
ORDER BY (id)
```

Создаем 1 млн. строк вида

```
INSERT INTO db1.test (id, data1_uint32, data1_uint32_zstd, data2_uint64,
data2_uint64_zstd) VALUES
(1, 12345, 12345, 12345, 12345)
(2, 12345, 12345, 12345, 12345)
...
```

Занятое место

```
SELECT
    name,
    formatReadableSize(sum(data_compressed_bytes)) AS compressed_size,
    formatReadableSize(sum(data_uncompressed_bytes)) AS uncompressed_size,
    round(sum(data_uncompressed_bytes) / sum(data_compressed_bytes), 2) AS ratio
FROM system.columns
WHERE `table` = 'test'
GROUP BY name
ORDER BY name
```

name	compressed_size	uncompressed_size	ratio
data1_uint32	17.62 KiB	3.81 MiB	221.7
data1_uint32_zstd	2.84 KiB	3.81 MiB	1373.15
data2_uint64	35.44 KiB	7.63 MiB	220.43
data2_uint64_zstd	5.76 KiB	7.63 MiB	1355.24
id	3.82 MiB	7.63 MiB	2

```
SELECT
    table,
    formatReadableSize(bytes_on_disk) as table_size_on_disk
FROM system.parts
WHERE table = 'test'
```

table	table_size_on_disk
"test"	"3.88 MiB"

2. Хранение разных значений в разных столбцах разного типа

Тестовая таблица

```
CREATE TABLE test
(
    id UInt64,
    data1_uint32 UInt32,
    data1_uint32_zstd UInt32 CODEC(ZSTD),
    data2_uint64 UInt64,
    data2_uint64_zstd UInt64 CODEC(ZSTD)
)
ENGINE = MergeTree()
ORDER BY (id)
```

Создаем 1 млн. строк вида

```
INSERT INTO db1.test (id, data1_uint32, data1_uint32_zstd, data2_uint64,
data2_uint64_zstd) VALUES
(1, 12345, 12345, 54321, 54321)
(2, 12345, 12345, 54321, 54321)
...
```

Занятое место

```
SELECT
    name,
    formatReadableSize(sum(data_compressed_bytes)) AS compressed_size,
    formatReadableSize(sum(data_uncompressed_bytes)) AS uncompressed_size,
    round(sum(data_uncompressed_bytes) / sum(data_compressed_bytes), 2) AS ratio
FROM system.columns
WHERE `table` = 'test'
```

```
GROUP BY name
ORDER BY name
```

name	compressed_size	uncompressed_size	ratio
data1_uint32	17.62 KiB	3.81 MiB	221.7
data1_uint32_zstd	2.84 KiB	3.81 MiB	1373.15
data2_uint64	35.44 KiB	7.63 MiB	220.43
data2_uint64_zstd	5.76 KiB	7.63 MiB	1355.24
id	3.82 MiB	7.63 MiB	2

```
SELECT
    table,
    formatReadableSize(bytes_on_disk) as table_size_on_disk
FROM system.parts
WHERE table = 'test'
```

table	table_size_on_disk
"test"	"3.88 MiB"

3. Хранение одинаковых значений в разных столбцах одинакового типа

Тестовая таблица

```
CREATE TABLE test
(
    id UInt64,
    data1_uint32 UInt32,
    data1_uint32_zstd UInt32 CODEC(ZSTD),
    data2_uint32 UInt32,
    data2_uint32_zstd UInt32 CODEC(ZSTD)
)
ENGINE = MergeTree()
ORDER BY (id)
```

Создаем 1 млн. строк вида

```
INSERT INTO db1.test (id, data1_uint32, data1_uint32_zstd, data2_uint32,
data2_uint32_zstd) VALUES
(1, 12345, 12345, 12345, 12345)
(2, 12345, 12345, 12345, 12345)
...
```

Занятое место

```
SELECT
    name,
    formatReadableSize(sum(data_compressed_bytes)) AS compressed_size,
    formatReadableSize(sum(data_uncompressed_bytes)) AS uncompressed_size,
    round(sum(data_uncompressed_bytes) / sum(data_compressed_bytes), 2) AS ratio
FROM system.columns
```

```
WHERE `table` = 'test'
GROUP BY name
ORDER BY name
```

name	compressed_size	uncompressed_size	ratio
data1_uint32	17.62 KiB	3.81 MiB	221.7
data1_uint32_zstd	2.84 KiB	3.81 MiB	1373.15
data2_uint32	17.62 KiB	3.81 MiB	221.7
data2_uint32_zstd	2.84 KiB	3.81 MiB	1373.15
id	3.82 MiB	7.63 MiB	2

```
SELECT
    table,
    formatReadableSize(bytes_on_disk) as table_size_on_disk
FROM system.parts
WHERE table = 'test'
```

table	table_size_on_disk
"test"	"3.86 MiB"

4. Хранение разных значений в разных столбцах одинакового типа

Тестовая таблица

```
CREATE TABLE test
(
    id UInt64,
    data1_uint32 UInt32,
    data1_uint32_zstd UInt32 CODEC(ZSTD),
    data2_uint32 UInt32,
    data2_uint32_zstd UInt32 CODEC(ZSTD)
)
ENGINE = MergeTree()
ORDER BY (id)
```

Создаем 1 млн. строк вида

```
INSERT INTO db1.test (id, data1_uint32, data1_uint32_zstd, data2_uint32,
data2_uint32_zstd) VALUES
(1, 12345, 12345, 54321, 54321)
(2, 12345, 12345, 54321, 54321)
...
```

Занятое место

```
SELECT
    name,
    formatReadableSize(sum(data_compressed_bytes)) AS compressed_size,
    formatReadableSize(sum(data_uncompressed_bytes)) AS uncompressed_size,
    round(sum(data_uncompressed_bytes) / sum(data_compressed_bytes), 2) AS ratio
```

```
FROM system.columns
WHERE `table` = 'test'
GROUP BY name
ORDER BY name
```

name	compressed_size	uncompressed_size	ratio
data1_uint32	17.62 KiB	3.81 MiB	221.7
data1_uint32_zstd	2.84 KiB	3.81 MiB	1373.15
data2_uint32	17.62 KiB	3.81 MiB	221.7
data2_uint32_zstd	2.84 KiB	3.81 MiB	1373.15
id	3.82 MiB	7.63 MiB	2

```
SELECT
    table,
    formatReadableSize(bytes_on_disk) as table_size_on_disk
FROM system.parts
WHERE table = 'test'
```

table	table_size_on_disk
"test"	"3.86 MiB"

5. Хранение одинаковых значений в разных таблицах

Тестовые таблицы

```
CREATE TABLE test1
(
    id UInt64
)
ENGINE = MergeTree()
ORDER BY (id);

CREATE TABLE test2
(
    id UInt64
)
ENGINE = MergeTree()
ORDER BY (id)
```

Создаем строки вида

```
INSERT INTO db1.test1 (id) VALUES
(12345);

INSERT INTO db1.test2 (id) VALUES
(12345)
```

Занятое место

```
SELECT
    table,
    formatReadableSize(bytes_on_disk) as table_size_on_disk,
    path
FROM system.parts
WHERE table = 'test1' or table = 'test2'
```

table	table_size_on_disk	path
"test1"	"219.00 B"	"/var/lib/clickhouse/store/e7f/e7f5bee5-2164-4b9f-8dec-fb4045f1626d/all_1_1_0/"
"test2"	"219.00 B"	"/var/lib/clickhouse/store/813/813cb4d1-4ca3-4640-b7e0-099e8bc80f9c/all_1_1_0/"

6. Хранение разных значений в разных таблицах

Тестовые таблицы

```
CREATE TABLE test1
(
    id UInt64
)
ENGINE = MergeTree()
ORDER BY (id);

CREATE TABLE test2
(
    id UInt64
)
ENGINE = MergeTree()
ORDER BY (id)
```

Создаем строки вида

```
INSERT INTO db1.test1 (id) VALUES
(12345);

INSERT INTO db1.test2 (id) VALUES
(54321)
```

Занятое место

```
SELECT
    table,
    formatReadableSize(bytes_on_disk) as table_size_on_disk,
    path
FROM system.parts
WHERE table = 'test1' or table = 'test2'
```

table	table_size_on_disk	path
"test1"	"219.00 B"	"/var/lib/clickhouse/store/f14/f14416fe-61f2-417d-ac7d-62d1515cec0f/all_1_1_0/"
"test2"	"219.00 B"	"/var/lib/clickhouse/store/fbb/fbbd38dc-44d0-4248-8663-13b6cf4365a0/all_1_1_0/"