Анализ производительности десериализации данных XML в сравнении с MS SQL Server

Название курса

Меня хорошо видно & слышно?

Защита проекта
Тема:

Сергей Игнатов

Ведущий разработчик баз данных концерна Русэлпром

План защиты

- Цель и задачи проекта
- Какие технологии использовались
- Что получилось
- Выводы
- Вопросы и рекомендации

Цель и задачи проекта

Цель проекта:
протестировать десериализацию xml-данных в PostgreSQL, выяснить какие аппаратные ресурсы для этого потребуется, и какое время будет потрачено в сравнении с MS SQL Server

1. Подготовить xml-данные для тестирования и загрузить их в MS SQL Server (дальше буду сокращать: MSSQL) и в PostgreSQL
2. Провести десериализацию xml-данных в MSSQL и в PostgreSQL
3. Сопоставить время, затраченное на десериализацию в PostgreSQL с временем в MSSQL
4. Выявить нюансы десериализации в PostgreSQL (если они есть)

Какие технологии использовались

1. Hyper-V Server
2. MS Windows 10, MS SQL Server 2014, SQL Server Management Studio 21
3. Debian 12, PostgreSQL 17, DBeaver 25
4. Штатные средства мониторинга ресурсов

Почему именно xml?

В концерне с 30-летней историей IT и огромной средой интеграции информационных систем, именно XML является базовым форматом. Да, DBF уже практически исчез, а JSON набирает силу. Но, при сохранении, MS SQL Server 2014, Oracle 9.2, 1C 7.7, Microsoft Dynamics CRM 2011, PTC Windchill PDM и прочих “старичков”, альтернативы XML по гибкости и универсальности промежуточных данных нет. Более того, даже при переезде, например, на 1C 8.3, проще сохранить предыдущий формат обмена, чем переписывать всю систему интеграции.

Варианты обмена данными

При обмене данными между системами, обычно встречаются два варианта:

- сильно связанная система,
  когда система **B** забирает данные из системы **A** напрямую;
- слабо связанная система,
  когда между получателем **B** и источником **A** определён промежуточный формат, в нашем случае это xml; при этом сама передача xml от **A** к **B** возможна множеством различных способов (вручную через файл и нажатием кнопочек, роботом через брокер сообщений и т.д.)

Хотя xml является тяжеловесным форматом, он вполне годится не только для текущего процесса обмена новостями, но и для начальной инициализации данными (справочники, журналы прошлых периодов).

Формат тестовых данных

Для примера начальной инициализации справочника, сформируем xml *справочника номенклатуры* - 254_898 записей. Чтобы файл имел минимальный размер, значение полей выведем в атрибуты, а имена колонок пусть будут в кириллице, как это любят программисты 1С…)))

Пример данных приведён справа. Итоговый файл имеет размер - 63.8 MB

```
<Сообщение
   Источник="Ruselprom.Branch.Otus"
   Агент="Филиал Русэлпром - Отус"
   ВерсияАгента="1"
   Количество="254898">
  <Номенклатура
   Группа="-"
   Подгруппа="-"
   Код="0"
   Наименование="-"
   ЕдИзмБазовая="шт" />
  <Номенклатура
   Группа="Электродвигатель (низковольтный)"
   Подгруппа="Электродвигатель ВЭМЗ"
   Код="13"
   Наименование="4АМИ180М6У3 IM1081 220/380В К3-1"
   ЕдИзмБазовая="шт"
   Чертёж="ИАФШ 5262220640103" />
  …
</Сообщение>
```

Структура таблицы с данными xml

Тестовые таблицы с данными в MSSQL и в PostgreSQL практически идентичны.

MSSQL

```
create table dbo.file_cache (
  id int not null primary key,
  file_nm nvarchar(255) not null,
  file_data varbinary(max) not null
)
```

PostgreSQL

```
create table public.file_cache (
  id int not null primary key,
  file_nm text not null,
  file_data bytea not null
)
```

Почему колонка **file_data** не сразу xml? По множеству причин:

- xml может быть битый, тогда процедура регистрации записей упадёт в exception
- текстовые данные могут поступить с разной кодировкой от разных источников
- рано или поздно xml уступит данным в формате json
- …

Особенности работы с xml в MS SQL

В MSSQL есть странная особенность работы с xml - сначала необходимо сконвертировать двоичные данные в формат xml и куда-нибудь их сбросить, и только после этого читать. Если же всё делать на лету, то время чтения увеличивается на порядки (буквально в десятки тысяч раз)! Зная эту особенность, получаем исходный код, показанный справа. Т.к. тестируем только чтение xml, а не запись данных в таблицу, обворачиваем запрос простым подсчётом количества полученных записей.

```
declare
  @xml xml, @n int

-- предварительно загружаем и конвертируем данные
select @xml = cast(f.file_data as xml)
  from dbo.file_cache f
  where (f.id = @id)

-- читаем полученный xml
select @n = count(*) from (
  select
      ProductGrp1  = p.data.value('@Группа', 'varchar(255)'),
      ProductGrp2  = p.data.value('@Подгруппа', 'varchar(255)'),
      ProductId    = p.data.value('@Код', 'varchar(255)'),
      ProductName  = p.data.value('@Наименование', 'varchar(255)'),
      UnitNameBase = p.data.value('@ЕдИзмБазовая', 'varchar(255)'),
      DrawingNo    = p.data.value('@Чертёж', 'varchar(255)')
    from (
      select file_data = @xml
    ) f cross apply f.file_data.nodes('/Сообщение/Номенклатура') p(data)
) s
```

Результаты в MSSQL

Запрос, приведённый на предыдущем слайде, вернул данные за 0.140 сек! При этом мы сразу провалились на всю глубину дерева ('/Сообщение/Номенклатура'). Если читать данные последовательно - сначала '/Сообщение', а потом '/Номенклатура' то время чтения увеличится до 2.000 сек.

```
-- читаем полученный xml
select @n = count(*) from (
  select
      ProductGrp1  = p.data.value('@Группа', 'varchar(255)'),
      ProductGrp2  = p.data.value('@Подгруппа', 'varchar(255)'),
      ProductId    = p.data.value('@Код', 'varchar(255)'),
      ProductName  = p.data.value('@Наименование', 'varchar(255)'),
      UnitNameBase = p.data.value('@ЕдИзмБазовая', 'varchar(255)'),
      DrawingNo    = p.data.value('@Чертёж', 'varchar(255)')
    from (
      select file_data = p.data.query('./*')
        from (
          select file_data = @xml
        ) f cross apply f.file_data.nodes('/Сообщение') p(data)
    ) f cross apply f.file_data.nodes('/Номенклатура') p(data)
) s
```

Исходные коды чтения xml в PostgreSQL

Семантика чтения xml в PostgreSQL немного отличается. Код приведён справа. В отличии от MSSQL, конвертацию и чтение проведём на лету, и сразу же на всю глубину дерева ('//Сообщение/Номенклатура'). Общее время чтения составило 1.9 сек. Не плохо! Подробную информацию см.ниже.

```
explain (analyze)
select
   f2.ProductGrp1,
   f2.ProductGrp2,
   f2.ProductId,
   f2.ProductName,
   f2.UnitNameBase,
   f2.DrawingNo
 from (
   select cast(convert_from(f.file_data, 'utf-8') as xml) file_data
     from  public.file_cache f
     where (f.id = 5)
 ) f1 left join xmltable('//Сообщение/Номенклатура' passing f1.file_data columns
        ProductGrp1 text path '@Группа',
        ProductGrp2 text path '@Подгруппа',
        ProductId text path '@Код',
        ProductName text path '@Наименование',
        UnitNameBase text path '@ЕдИзмБазовая',
        DrawingNo text path '@Чертёж'
      ) f2 on true;
```

Результаты теста в PostgreSQL

```
Nested Loop Left Join  (cost=0.15..10.17 rows=100 width=196) (actual time=1850.355..1880.273 rows=254898 loops=1)
  ->  Index Scan using file_cache_pkey on file_cache f  (cost=0.15..8.17 rows=1 width=36) (actual time=0.032..0.036 rows=1 loops=1)
        Index Cond: (id = 5)
  ->  Table Function Scan on "xmltable" f2  (cost=0.01..1.00 rows=100 width=192) (actual time=1850.318..1864.227 rows=254898 loops=1)
Planning Time: 0.096 ms
Execution Time: 1887.828 ms
```

К сожалению, в PostgreSQL нельзя прочитать последовательно сначала раздел '//Сообщение', а потом '//Номенклатура', т.к. чтение '//Сообщение' теряет рутовый раздел, и в оставшейся строке получим “частичный xml” (в документации “content fragment”), который PostgreSQL читать не умеет, поэтому код справа вернёт ошибку.

```
explain (analyze)
select f1.id, f3.ProductGrp1, …
 from (
   select f.id, f.file_nm, cast(convert_from(f.file_data, 'utf-8') as xml) file_data
     from  public.file_cache f
     where (f.id = 5)
 ) f1 left join xmltable('//Сообщение' passing f1.file_data columns
        products xml path './Номенклатура'
      ) f2 on true
      left join xmltable('//Номенклатура' passing f2.products columns
        ProductGrp1 text path '@Группа',
        ProductGrp2 text path '@Подгруппа',
        ProductId text path '@Код',
        ProductName text path '@Наименование',
        UnitNameBase text path '@ЕдИзмБазовая'
```

Увеличение ресурсов PostgreSQL

Но всю эту работу на PostgreSQL мы проделали на дефолтных настройках! Исправим типовую ошибку новичков:

```
shared_buffers = 1GB          # 25% от общего ОЗУ
work_mem = 100MB              # хотим разом прочитать все данные xml
effective_cache_size = 3GB    # 75% от общего ОЗУ
wal_buffers = 16MB
```

Снова запускаем чтение xml, и… время не поменялось:

Nested Loop Left Join (cost=0.15..10.17 rows=100 width=196) (actual time=1817.516..1841.140 rows=254898 loops=1)

-> Index Scan using file_cache_pkey on file_cache f (cost=0.15..8.17 rows=1 width=36) (actual time=0.018..0.023 rows=1 loops=1)

Index Cond: (id = 5)

-> Table Function Scan on "xmltable" f2 (cost=0.01..1.00 rows=100 width=192) (**actual time=1817.493..1824.454** rows=254898 loops=1)

Planning Time: 0.086 ms

Execution Time: 1848.785 ms

Выводы

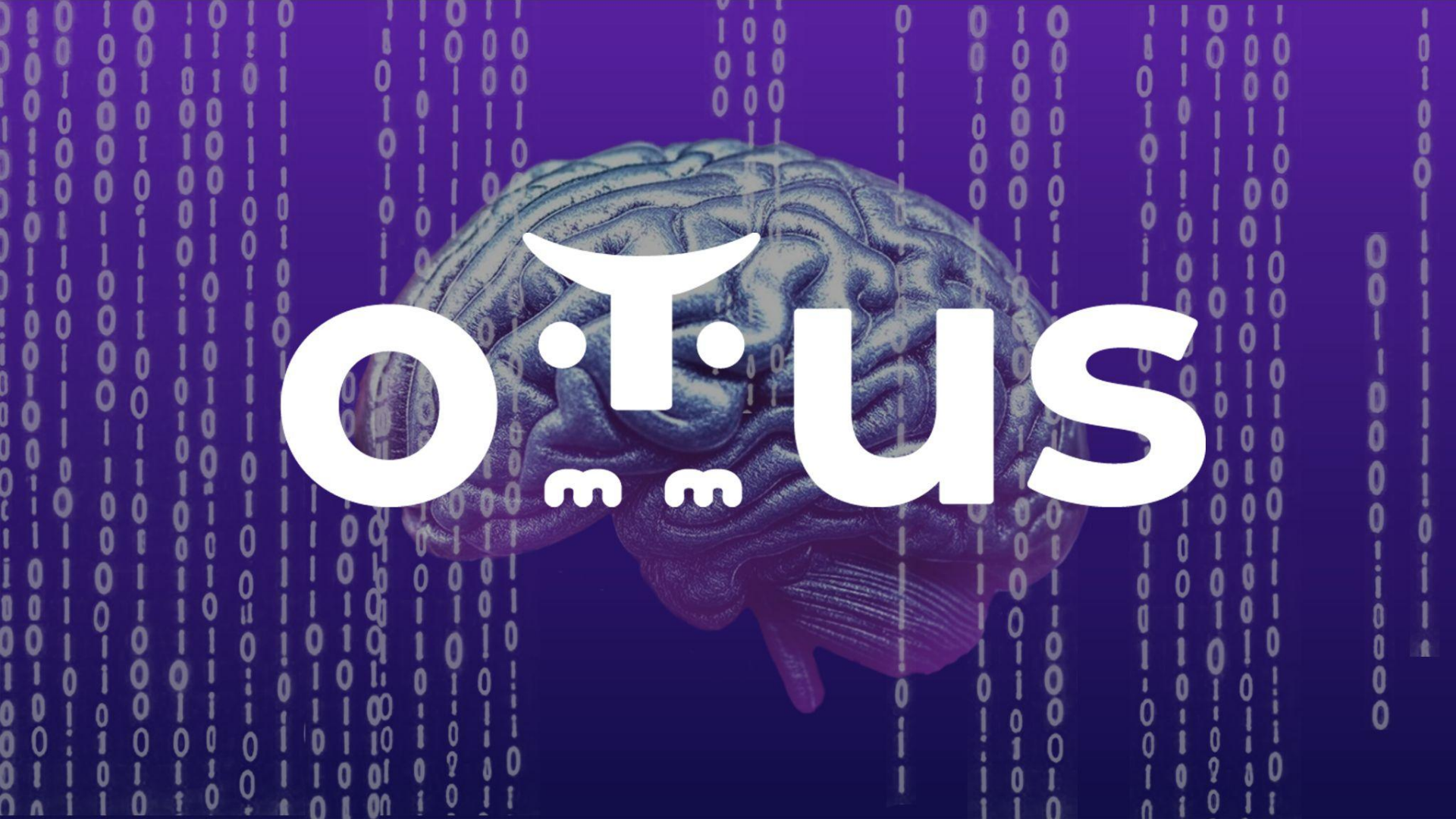
1. Работа с xml-данными в PostgreSQL вполне дружественная и неприхотливая
2. Скорости, которые показывает PostgreSQL, вполне сопоставимы со скоростью ***последовательного*** чтения дерева xml в MSSQL. При этом можно упрощать исходный код, не перекладывая сконвертированные бинарные данные, а читать их на лету без “катастрофических” тормозов, как это у MSSQL
3. Даже на минимальных настройках выделенной оперативной памяти, PostgreSQL читает 250_000 строк “весом” в 60+ мегабайт без каких-либо потерь в скорости
4. В итоге, переезд с MSSQL на PostgreSQL в области работы с xml получается совершенно безболезненным, без каких-либо сюрпризов по скорости и ресурсам, важно не забыть только один

Вопросы и рекомендации

\+ если есть вопросы

— если вопросов нет

Спасибо за внимание!


OTUS