Сервер приложений на pl/pgsql

взгляд со стороны бизнеса

Артём Макаров
«Проект 111», руководитель отдела IT
art@gifts.ru

Кто мы?

- gifts.ru — B2B eCommerce
- дневная нагрузка в пике:
  ~ 2300 "тяжелых“ (OLTP) пользователей
  ~ 15K визитов на сайте
- 97% заказов — самообслуживание через сайт
- оборот 1.5 млрд. руб. в 2014 году
- ~ 200 сотрудников
- больше информации о компании gifts.ru/about

Свой сервер приложений…
Зачем?

- «Сайтоцентричная» ERP
- Проблемы с «коробками»:
  - производительность
  - функционал
  - лицензирование
- Vendor lock-in

Мифы и стереотипы о хранимых процедурах

«Теряется переносимость между СУБД»

«Сложно поддерживать»

«Сложно вести разработку в команде»

«Недоступен шардинг, распределенные конфигурации и т.д.»

Мифы и стереотипы о хранимых процедурах

Стереотипы

Проблема	Поставщик	Покупатель
Переносимость между СУБД	+	-
Сложно поддерживать коробочный продукт	+	-
Сложно вести разработку в большой команде	+	?
Трудно защитить код от копирования и модификации	+	;-)

Преимущества архитектуры для поставщика ПО (продавца)
выдаются за достоинства ПО для потребителя (покупателя)

Мифы
о хранимых процедурах

Что осталось?
«шардинг, распределенные конфигурации и т.д.»

Что мы имеем?
pl/proxy, dblink, FDW

Уверены, что вам это уже нужно?

Почему Postgresql?

- зрелый pl/pgsql
- строгость, функциональность, академический подход
- открытый код и свободная лицензия
- используем с 1998 года (~ release 6.4)

Хорошие стороны pl/pgsql

+ не нужно думать о выделении памяти, встроенном скриптовом языке, профилировании и т.д.

+ локальность данных и производительность

+ простой язык, низкий порог входа

+ безопасность: приложение не имеет прямого доступа к данным, невозможны sql injections*

+ изоляция от интерфейсов: возможна плавная замена или параллельная работа frontend сервисов

+ долгое время жизни бизнес-логики

* - нужно очень постараться с **`Execute`**

Плохие стороны pl/pgsql

- слишком хорошая изоляция от интерфейсов*
- нет хорошо интегрированных средств разработки
- принудительная транзакционность иногда мешает
- если потребуется высокоуровневое кеширование, появится еще один слой абстракции
- иногда трудно вовремя остановиться**

* с появлением JSON это не вполне справедливо
** создание писем с помощью pl/perl & MIME:Lite — это уж слишком…

Что внутри? (1)

Ядро системы

Postgresql 9.4 (разработку начинали на 7.4, стартовали на 8.x)

Вся бизнес логика ~3000 функций pl/pgsql, код пишем вручную

Строк кода, включая заголовки и комменты:
2007 — 50 000
2009 — 100 000
2015 — 200 000

~700 таблиц, из них 600 — OLTP

~150Гб, рост на 30Гб в год (30% — аналитика и история)

Любим: короткие транзакции, constraints, FK, денормализацию и partial indexes

Ненавидим: ORM и кодо-генераторы SQL

Что внутри? (2)

Сервисы и пользовательские интерфейсы

Веб-сервисы: Tomcat + nginx
- gifts.ru: сильно модифицированный Liferay (мало что осталось)
- CMS, CRM etc
- Reporting сервис: сервлет Jasper Report, Apache POI etc

Windows App: C++ Builder, Devart Pg DAC, Dev Express

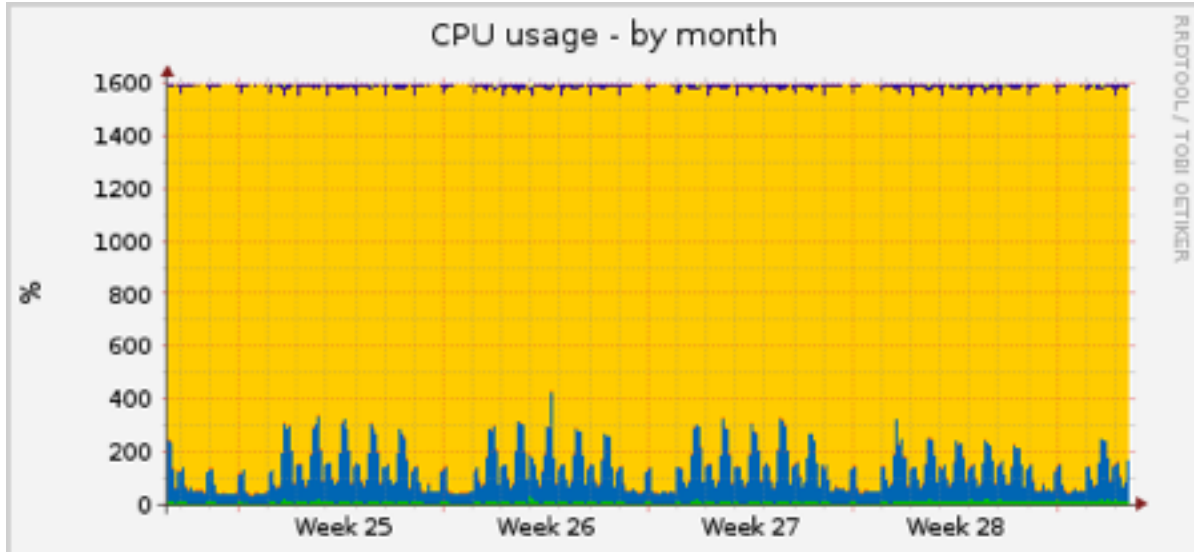
(всё это работает через pgBouncer)

Асинхронные задачи — crond
- mailer: java app планировщик для отправки email, SMS
- аналитика и OLAP: bash скрипты, psql, python

Разное: Asterisk PBX, SEO аналитика

Что внутри? (3)

Серверы (x2)
- 2x 8 core CPU
- 48 Gb ECC RAM (16 Gb shared buffers)
- 8x 15K rpm SFF SAS



Нагрузка:
- LA 2.8 — 3.0
- 150-200 tps, пики до 500 tps
- hitrate 99.8%
- 150 IOPS avg, пики до 800 IOPS

Факты:
- цена сервера ~ [gross margin за 40 минут в сезон]
- не используем реплику для offload
- не используем SSD

Что внутри? (схемы и роли)

```
CREATE ROLE "WebGroup" -- сайт и веб-сервисы
  NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE NOREPLICATION;
CREATE ROLE "EmplGroup" -- внутренние приложения
  NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE NOREPLICATION;

CREATE ROLE web LOGIN ENCRYPTED PASSWORD 'md5**********'
  NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE NOREPLICATION;
ALTER ROLE web SET statement_timeout = '10000';
GRANT "WebGroup" TO web;

-- Схема для внешних пользователей
create schema custm; grant usage on schema custm to "WebGroup", "EmplGroup";

-- Только для сотрудников (финансы, логистика и пр.)
create schema empl; grant usage on schema empl to "EmplGroup";

-- Запретим приложениям видеть код хранимых процедур и схему данных
revoke select on pg_catalog.pg_proc, information_schema.routines
   from public;
revoke all on schema public from public;
```

Структура функций

```
-- Авторизация
custm.SessionCreate(in_CompanyCode       text,
                    in_Login             text,
                    in_Password          text)
  RETURNS SessionID uuid as
$$
begin
  /* any staff */
return SessionID;
end;
$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER;
GRANT EXECUTE ON FUNCTION custm.sessionopen(..) TO “EmplGroup";
GRANT EXECUTE ON FUNCTION custm.sessionopen(..) TO "WebGroup";
```

Структура функций

```
-- Функции предметной области — первый параметр in_SessionID
custm.OrderGet(in_SessionID uuid, in_OrderID int)....;

-- Функция для внешних пользователей урезаны по возможностям
custm.OrderCreate(in_SessionID uuid, ...) RETURNS OrderID ....
 declare
   m_CompanyID INT :=     internal.SessionCompanyID(in_SessionID);
   m_OrderID   INT;
 begin
   m_OrderID :=
  empl.OrderCreate(in_SessionID,m_CompanyID,...);
           /*  any staff   */
   return m_OrderID;
 end;
```

Объекты, свойства …

Записи в таблицах предметной области повторно представлены в таблице объектов:

```
CREATE TABLE objects
(
    ObjectID                int NOT NULL,
    ObjectTypeID            int NOT NULL,
    ParentObjectID          int,
    ParentObjecttypeID      int,
    ObjectStatusID          int NOT NULL default 0,
    CreateUserID            int NOT NULL,

  /*......cut........*/
  CONSTRAINT objects_pkey PRIMARY KEY(objectid,OjectTypeid),
   /*......cut........*/);
```

Проверка прав доступа

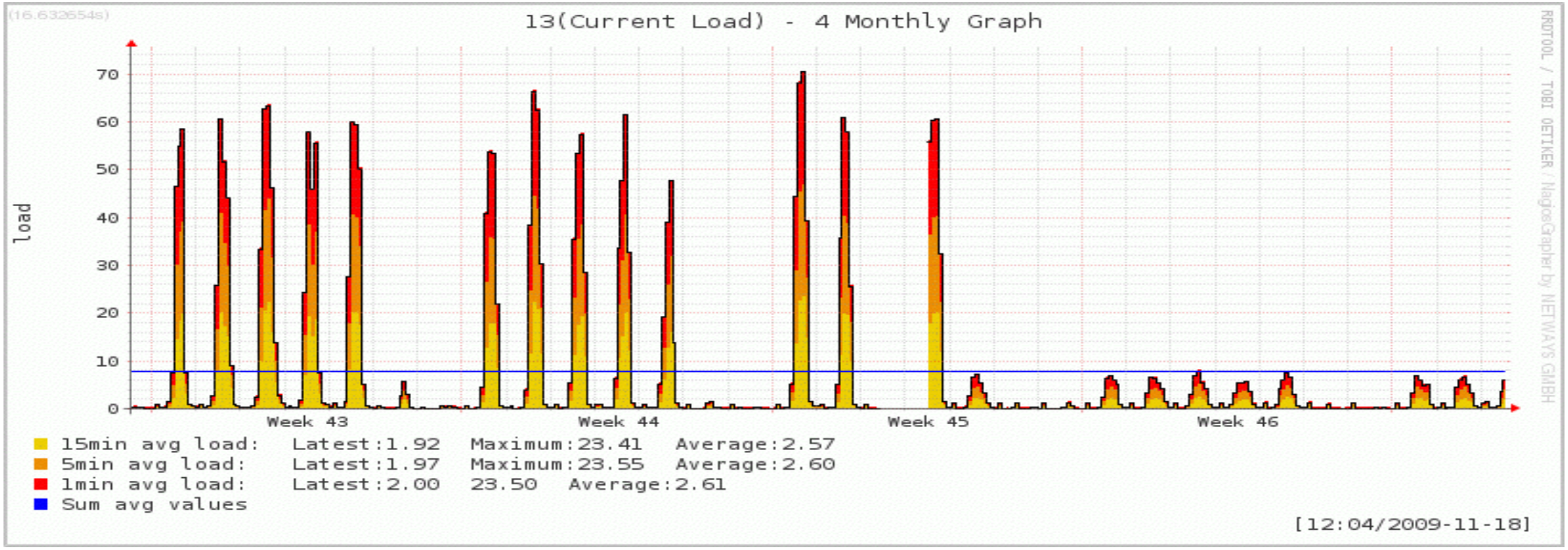
Вызовы хранимых процедур предметной области обрамлены проверкой прав:

```
create function fw.ObjectBeforeAction(
                    in_SessionID          uuid,
                    in_ObjectID           int,
                    in_ObjectTypeID       int,
                    in_ActionID           int,
                    in_RaiseException     bool
                    );
in_RaiseException = true | false
```

Бросаем exception и показываем на клиенте,
либо используем для фильтрации списков

Проблемы

“on commit delete rows” — ужас первых недель



Выводы:

1. **У вас нет причин <u>не использовать</u> pgBouncer**
2. **Community помогает**
3. **Open Source полезен даже новичкам**

Проблемки

1. **Синтаксис pl/pgsql не такой уж строгий — missed comma**

```
select i1,i2,i3,i4 into var_i1,var_i2 var_i3,var_i4
  from temp_table;

  raise notice '%,%,%,%',var_i1,var_i2,var_i3,var_i4;
```

2. **Опасные возможности — truncate cascade, сравните с drop cascade**

3. **Pl/proxy & refcursor**

Оптимизация (1)

pg_stat_user_functions

Измеряем параметры работы функций

Храним данные за каждую рабочую смену

Выявляем отклонения: изменения в коде, перекосы статистики, новые версии Postgresql

funcid oid	schemaname name	funcname name	calls bigint	total_time double precision	self_time double precision
855247	internal	article_reservesarray	1398237	1237684.098	1237684.098
852909	buh	invoicecalcservicecostspread	3554364	1036313.33	1036313.33
852910	buh	invoicecalculate	188424	1949681.146	1949681.146
853509	custm	stat_manufactureloadwidgetget	121825	246774.638	200351
852912	buh	invoicecreate	21812	1457736.185	1983.717
852913	buh	invoicecreate	57265	5270800.73	3409.211
852914	buh	invoicecreate	184304	7655407.089	5941.609
852918	buh	invoiceitemcreate	444098	1186883.801	6798.933

pg_stat_user_functions

Срез статистики дважды в день:

```
insert into stat.pg_functionstatistics
  (statid,funcid,schema,function,calls,total_time,self_time)
select currval('stat.pg_statistics_seq'),
  funcid,schemaname,funcname,calls,
  total_time,self_time
from pg_stat_user_functions
where to_char(now(), 'HH') = '08';
```

Показатели дня

```
update  stat.pg_functionstatistics as s2
set avg_time          = (s2.total_time - s1.total_time)::float
                                /(s2.calls - s1.calls),
  avg_self_time       = (s2.self_time - s1.self_time)::float
                           /(s2.calls - s1.calls),
  diff_total_time     = s2.total_time - s1.total_time,
  diff_self_time      = s2.self_time - s1.self_time,
  diff_count          = s2.calls - s1.calls
from stat.pg_functionstatistics as s1
where to_char(s2.stamp, 'HH24') = '20'
  and to_char(s1.stamp, 'HH24') = '08'
  and s2.statid = currval('stat.pg_statistics_seq')
  and s2.calls > s1.calls
  and s1.funcid = s2.funcid
  and s1.date = s2.date
  and to_char(now(), 'HH') = '08';
```

Оптимизация (2)

pg_stat_user_functions – не для всех проблем:

- курсоры
- ветвления функций
- точечные нагрузки (среднее время — не показатель)

Идем глубже — pg_stat_statements

- храним данные за каждую рабочую смену
- выявляем отклонения: изменения в коде, перекосы статистики, новые версии Postgresql

Полезно: почасовые слепки, списки исключений, разные роли у разных приложений

pg_stat_statements

Вызовы функций попадают в статистику несколько раз:
- сами функции (для каждой роли)
- элементарные запросы из тела функций *
- fetch refcursor

```
select u.usename,
     regexp_replace(query,'\r?\n','') as query,
     calls,
     (total_time/calls)::numeric(14,2) as avgtime,
     total_time::int,
     shared_blks_read,shared_blks_dirtied
from pg_stat_statements as s
inner join pg_user as u on (u.usesysid = s.userid)
order by total_time desc
```

* - включая форматирование и комментарии!

Refcursor и его запрос.
Вызов функции, вернувшей курсор, где-то глубоко внизу:

usename name	query text	calls bigint	avgtime numeric(14,2)	total_time integer	shared_blks_read bigint	shared_blks_dirtied bigint
postgres	SELECT internal.BL OrderReserveBalanceUpdate(in O	3922	38.60	151390	14244	1229
web	FETCH ALL IN "refcursor/*custm.liferay latestpu	5	190.11	149236	32	359
web	select CATEGORIES.categoryid,	5	190.10	149230	32	359
postgres	SELECT internal.BL ObjectBalanceUpdate(in OrderID	3	37.89	148849	14046	1226
postgres	select out widget	19	765.50	148507	0	6
postgres	select '{name: "По регионам",	194	759.56	147355	0	6
postgres	select json agg(FINAL.json row)	54338	0.58	146640	602	0

usename name	query text	calls bigint	avgtime numeric(14,2)
postgres	SELECT internal.BL OrderReserveBalanceUpdate(in O	3922	38.60
web	FETCH ALL IN "refcursor/*custm.liferay latestpubl	785	190.11
web	select CATEGORIES.categoryid,	785	190.10

Ищем потенциально «плохие» места:
- update, delete
- горячие таблицы
- собственные комментарии (--todo, --WTF?)

```
select u.usename, regexp_replace(query,'\r?\n','') as query,calls,
      (total_time/calls)::numeric(14,2) as avgtime, total_time::int
            7*…cut…*/
from pg_stat_statements as s
inner join pg_user as u on (u.usesysid = s.userid)
where query ilike 'update%'
order by total_time desc
```

usename name	query text	calls bigint	avgtime numeric(14,2)	total_time integer	shared_blks_read bigint	shared_blks_dirtied bigint
liferay	update User set uuid =$1, companyId=$2, c	1301	185.31	241084	32	44
postgres	update cache.SalePriceArticlesCache as CA	7026	2.11	14826	495	4734
postgres	update public.Sessions set UserAccessTi	3374	4.33	14619	14	758
postgres	update cache.SalePriceArticlesCache as CA	462	28.65	13236	352	1249
postgres	update Cache.SalePriceArticlesCache as CA	70	167.70	11739	40	869
postgres	update Orders set CurrencyRateDate	816	13.25	10815	1269	3022
postgres	update StorePlaceItems set Amount	6387	1.25	7992	1435	4699
postgres	update cache.searchcaches set IsValid = fa	1574	4.78	7529	2991	8570
postgres	update Objects set StateID = in Ne	3804	1.62	6155	989	2119
postgres	update Balance set Balance	6054	0.74	4461	633	1423
postgres	update ObjectTasks set IsDone = m NewIs	7049	0.62	4397	742	440

Оптимизация (3)

Еще глубже: pg_stat_user_tables, pg_statio_user_tables

Сравним теорию и практику:
- сколько раз обновляется таблица на одну бизнес-транзакцию?
- сколько раз она обновилась на самом деле?
- какой тип обращений ожидается?

Источники проблем:
- циклы
- забытые условия в where
- иногда нужен distinct в подзапросе update … from
- забытые индексы

Оптимизация (4)

«Вы не поверите,
каких бездн достигает
терпение пользователя,
если производительность
ухудшается медленно…»

Дела на каждый день

- смотрим postgresql.log
- заменяем legasy код на новый: window functions, with recursive, filter (where ….)
- во всех сомнительных случаях: raise notice ‘timing=%’, clock_timestamp() – clck_before;
- избавляемся от временных таблиц (готово)

Оптимизация (5)

Чистим систему от устаревшего кода

```
for rec in
 select PC.oid, PC.proname
 from pg_Proc as PC
 inner join pg_namespace as SC on(SC.oid=PC.pronamespace)
 left join pg_stat_user_functions as SF on(SF.funcid = PC.oid)
 where SC.nspname in ('custm', 'empl')
  and SF.funcid is null
loop
  /* staff 2 move functions */
end loop;
```

За 7 лет разработки накопилось много мусора… После очередного редизайна за неделю избавились от 900 «мертвых» функций.

Надежды и ожидания

1. JSON и JSONB
- **фасетные фильтры на сайте, еще удобнее**
- **компактные и быстрые ACL**
- **Android**

2. IDE для pl/pgsql: подстановка кода, интеграция с VCS

2. BRIN, PGStorm для ETL & OLAP

3. Отказ от курсоров и pl/proxy offload

4. BDR — еще сами не поняли, зачем -)

Вопросы?

Как это сделать?

1. **Нанять профессиональную компанию, которая любит и знает PG*.**

2. **Вместе понять, что вы хотите от ядра системы на старте.**

3. **На ранней стадии включить своего программиста в разработку**

4. **Самим писать код для миграции. Тестировать миграцию несколько раз в неделю. Лучше – каждый день.**

5. **Запускать систему своими силами. Это повышает ответственность.**

6. **Иметь верифицируемые цели и контрольные точки.**

* теперь это намного проще

Мысли и факты..

1. **Средний стаж программиста в компании — 9 лет.**
   Нельзя так долго программировать на 1С. На pl/pgsql — можно.
   Скажите это своему HR менеджеру.

2. **Поддержкой legasy системы и параллельной разработкой перед стартом занимались 2.5 программиста.**

3. **15-20% кода вносится в боевую систему «на ходу».**