РАЗРАБОТКА ВЫСОКО-НАГРУЖЕННЫХ СИСТЕМ

ПАВЕЛ МАТЛАШОВ

UDEV TECH EVENTS

О КОМПАНИИ

200 000 000
ИГРОКОВ

150
СТРАН

12
УСПЕШНЫХ ПРОЕКТОВ

115
РАСПРЕДЕЛЕННЫХ СЕРВЕРОВ

СТАТИСТИКА

200 МЛН
ПОЛЬЗОВАТЕЛЕЙ

2.2 МЛН
DAU

22 МЛН
СЕССИЙ В СУТКИ

30 МЛН
СРАЖЕНИЙ В СУТКИ

64
СЕРВЕРА ПРИЛОЖЕНИЙ

21
СЕРВЕР БД

8500
ЗАПРОСОВ В СЕКУНДУ

ГДЕ ДЕРЖАТЬ МОЩНОСТИ?

2 реальных варианта — выделенные серверы в аренду или облако. Свои серверы в нашей стране — не вариант :)

ОБЛАЧНЫЕ СЕРВИСЫ

ВЫДЕЛЕННЫЕ СЕРВЕРЫ

ГДЕ ДЕРЖАТЬ МОЩНОСТИ?

ОБЛАЧНЫЕ СЕРВИСЫ

\+

- очень много быстродоступных ресурсов
- большой набор вариантов хранения данных и сервисов (очереди, балансировка, защита от атак, кэширование и т. д.)
- возможность сэкономить при непостоянных нагрузках
- меньше затрат на администрирование

–

- высокая стоимость при постоянных больших нагрузках сервисов

ГДЕ ДЕРЖАТЬ МОЩНОСТИ?

ВЫДЕЛЕННЫЕ СЕРВЕРЫ

\+

- невысокая стоимость при условии постоянной большой нагрузки
- больше контроля над ПО и железом

–

- меньше интересных сервисов
- низкая скорость получения новых ресурсов
- меньше безопасности

ЧТО ВЫБРАЛИ МЫ?

- Выделенные серверы в аренду для приложений
- CDN для раздачи графики
- Ограниченное использование облачных сервисов для смягчения атак

КАК МАСШТАБИРОВАТЬСЯ?

ВЕРТИКАЛЬНОЕ МАСШТАБИРОВАНИЕ /SCALE-UP/

- достаточно просто по концепции
- быстро достигаем предела стоимости и возможностей

ГОРИЗОНТАЛЬНОЕ МАСШТАБИРОВАНИЕ /SCALE-OUT/

- сложнее достичь предела
- линейный рост стоимости
- более сложная архитектура, возникают задачи по согласованности, распределенным вычислениям, кэшированию и т. д.

ЧТО ВЫБРАЛИ МЫ?

ВЕРТИКАЛЬНОЕ МАСШТАБИРОВАНИЕ

- sharding для основных игровых сервисов
- разбивка игровых серверов на кластеры (мастер + сегменты)
- вспомогательные сервисы отдельно (платежи, нотификации)

СТЕК ТЕХНОЛОГИЙ

- выбирайте более удобный и знакомый вам стек
- разные технологии не всегда хорошо стыкуются между собой
- не все технологии выдерживают большие нагрузки
- выбор стека напрямую влияет на стоимость решения

КАКОЙ СТЕК ВЫБРАЛИ МЫ?

.NET FRAMEWORK 4.6.1,
ASP.NET,
MS SQL SERVER

~~большинство стандартных модулей, сессии, безопасность, кэш~~

~~WCF, Web Api, SOAP~~

~~ORM~~

~~MVC, Spring.NET, Castle и т. д.~~

КАКОЙ СТЕК ВЫБРАЛИ МЫ?

- свой механизм Eventual Consistency для распределенных вычислений
- свое кэширование
- своя сериализация в JSON
- просто ADO.NET плюс легковесная обертка
- JSON over HTTP
- log4Net + Kafka
- свой инструментарий для мониторинга производительности и развертывания

КАКОЕ ХРАНИЛИЩЕ ВЫБРАТЬ?

NOSQL | RDBMS

КАКОЕ ХРАНИЛИЩЕ ВЫБРАТЬ?

NOSQL

\+

- может держать очень большие нагрузки
- масса вариантов и форматов хранения
- стоимость

—

- NoACID :)

КАКОЕ ХРАНИЛИЩЕ ВЫБРАТЬ?

RDBMS

\+

- многолетний опыт использования
- ACID
- высокая стабильность

—

- высокая стоимость
- тяжело масштабируется
- низкая производительность

ЧТО ВЫБРАЛИ МЫ?

MS SQL SERVER с применением
NOSQL-подхода

- производительность при вычитке/ сохранении данных
- полноценный ACID на уровне сегмента при необходимости
- миграция схем пользователей
- согласованность между сегментами и масштабирование

ACID ИЛИ BASE?

ACID

/ATOMICITY,
CONSISTENCY,
ISOLATION,
DURABILITY/

BASE

/BASICALLY AVAILABLE,
SOFT STATE,
EVENTUAL
CONSISTENCY/

ACID ИЛИ BASE?

ACID /ATOMICITY, CONSISTENCY, ISOLATION, DURABILITY/

\+

- классический подход, есть готовые решения
- проще разрабатывать (пока не наступят взаимоблокировки)

—

- низкая масштабируемость (взаимоблокировки, задержки)

ACID ИЛИ BASE?

BASE /BASICALLY AVAILABLE, SOFT STATE, EVENTUAL CONSISTENCY/

\+

- прекрасная масштабируемость и доступность
- масса приемов и паттернов
- возможность подобрать нужную степень согласованности в каждом случае

—

- усложненная разработка

ЧТО ВЫБРАЛИ МЫ?

BASE

- собственная очередь
- отличная масштабируемость
- более сложная модель программирования

Прекрасные статьи для начала:

CAP

Base: An Acid Alternative

РАБОТА С ПАМЯТЬЮ

Необходимо четко знать,
что вы храните в памяти

- работа с большими объектами
- сложность графа
- ротация памяти
- Cache is the King!

РАБОТА С ПАМЯТЬЮ. БОЛЬШИЕ ОБЪЕКТЫ

- выделение больших массивов
- разбивка данных на части, где это возможно
- снижение размера кучи больших объектов
- уменьшение затрат на сборку мусора
- снижение ротации памяти

РАБОТА С ПАМЯТЬЮ. КЭШИРОВАНИЕ

Кэширование вне процесса:

позволяет пережить перезагрузки системы и сделать кэш обще-доступным, но снижает скорость доступа, требует дополнительной обработки

Кэширование в памяти процесса:

- выбор коллекции и способа синхронизации
- предварительный разогрев некоторых кэшей
- политики устаревания и доверия данным

\+ кэширование в базе

РАБОТА С ПАМЯТЬЮ. БОЛЬШЕ ДЕТАЛЕЙ

- пулы
- снова сложность графа
- использование структур

JSON-СЕРИАЛИЗАЦИЯ

Тратим до 50 % времени процессора на сериализацию и десериализацию

- написали сериализацию самостоятельно
- оказалось в 10 раз быстрее JSON.Net, в 3 раза быстрее fastJSON
- совместимость между данными была критически важной

ПРОФИЛИРОВЩИК

- сложность нагрузочного тестирования
- необходимость мониторинга
- традиционные планировщики

ПРОФИЛИРОВЩИК

Используем свою систему счетчиков производительности

```
using (new PerfWatch("Read") )
   ints = File.ReadAllText("inst.txt")
      .Split(new[] {"  "}, StringSplitOptions.RemoveEmplyEntries)
      .Select(int.Parse).ToList();

using (new PerfWatch("Sort") )
   ints.Sort();

using (new PerfWatch("Write") )
   File.WriteAllText("sorted inst.txt", string.Join(" ", ints));
```

ПРОФИЛИРОВЩИК

Статистика накапливается в памяти и собирается с серверов вместе с другой полезной информацией.

Поддерживаются иерархии счетчиков для анализа цепочек вызовов.

Name	Calls per second / total	Total Time time / cpu time (no child)	Time total / nochild	CPU Time total / nochild	Wait Time total / nochild	Time Details <100(3) <500(5) <3K(6) <5K(3)
Read	33,3% 0,08 1	68% 56,4% (68%) 75,86% (56,4%)	8575 / 8575 ms	6505 / 6505 ms	2070 / 2070 ms	8575 (6505) ... 8575 (6505) ms
Write	33,3% 0,08 1	23,7% 24% (23,7%) 92,53% (24%)	2984 / 2984 ms	2761 / 2761 ms	223 / 223 ms	2984 (2761) ... 2984 (2761) ms
Sort	33,3% 0,08 1	8,4% 9,2% (8,4%) 100% (9,2%)	1057 / 1057 ms	1060 / 1060 ms	0 / 0 ms	1057 (1060) ... 1057 (1060) ms

ПРОФИЛИРОВЩИК

Counter Name	Value
Monitoring period	UTC 5/12/2016 1:47:59 AM ... 5/18/2016 8:52:43 AM
Duration	151:04:44.516
CPU: Available time / Process time	2417:15:52.268 (16 CPUs) / 934:47:24.836 (38.67%)
Profiler overhead	0.868% of CPU time, 11.29ms per 1000 calls

- простой и гибкий подход к указанию профилируемых участков
- автоматическая генерация счетчиков для точек входа (сетевых запросов, методов)
- возможность просмотра и агрегации по принципу parent-child

ЛОГИРОВАНИЕ

- часто единственный способ диагностики ошибок
- используем два формата: human readable и JSON
- пишем всё, что можно писать
- используем log4net и Kafka для сбора логов в реальном времени
- генерируем порядка 130 Гб логов для аналитики в сутки (примерно 220 млн полезных событий в сутки)

DEPLOYMENT

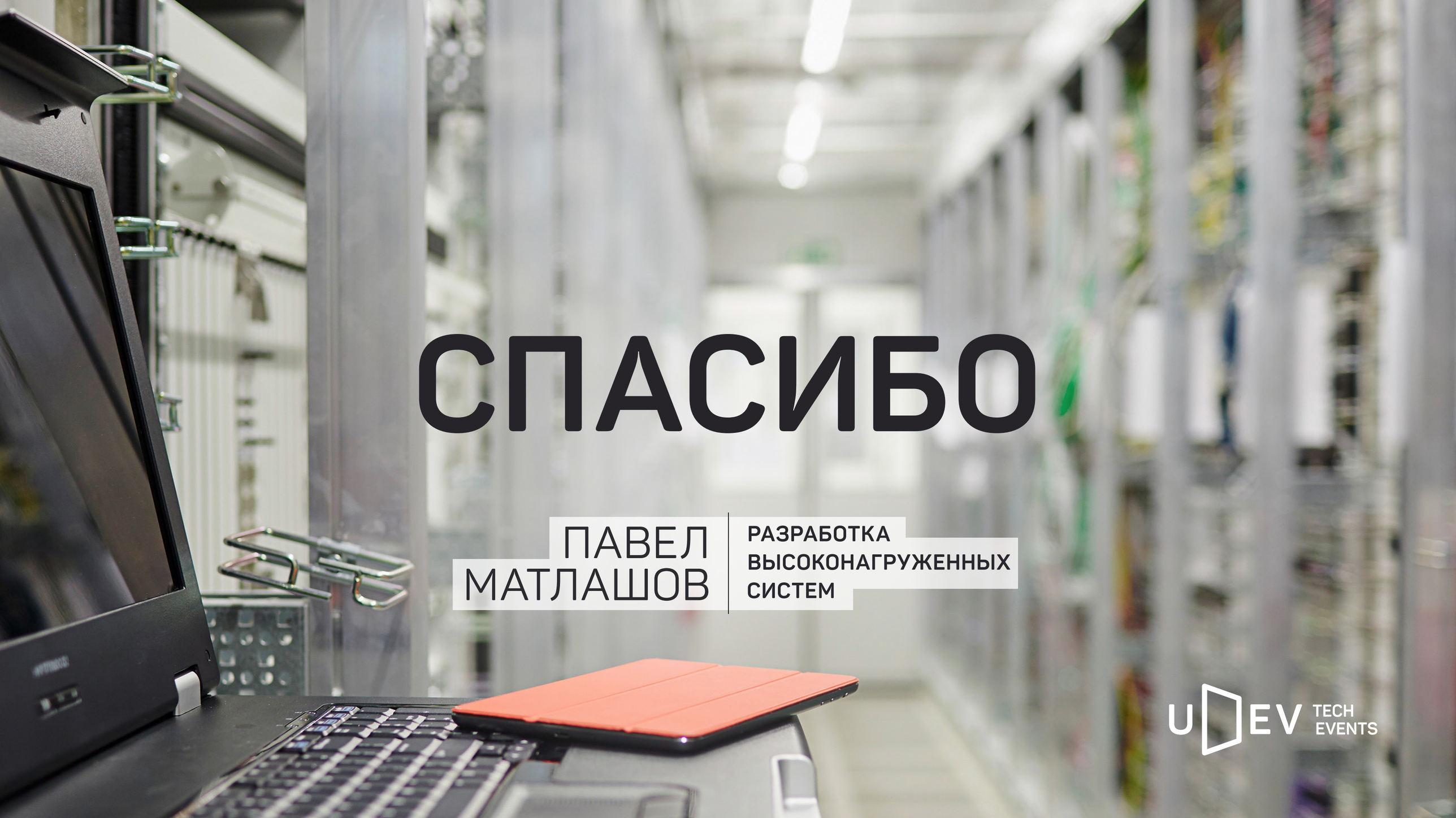
- делаем примерно **52 релиза** в год
- **64 сервера** приложений и **21 сервер** баз данных
- разработали простую, но эффективную инфраструктуру для развертывания и конфигурирования серверов
- есть инструменты для мониторинга состояния серверов (свои и сторонние)

ОБО ВСЁМ ПОНЕМНОГУ

- проектируйте максимально простую архитектуру с запасом прочности
- исходите из требований к нагрузке, реального использования и здравого смысла
- оценивайте новые веяния прагматично
- используйте только хорошо знакомый стек

ОБО ВСЁМ ПОНЕМНОГУ

- понимайте архитектуру железа
- проанализируйте работу основных структур данных и временные затраты
- обязательно используйте инструментарий для мониторинга жизнедеятельности системы
- микрооптимизации могут дать приятный бонус


СПАСИБО
ПАВЕЛ МАТЛАШОВ
РАЗРАБОТКА ВЫСОКОНАГРУЖЕННЫХ СИСТЕМ
UDEV TECH EVENTS