High Load & C++

Медведик Давид

Старший инженер программист в “Positive technologies”

Немного обо мне

Медведик Давид

LinkedIn: www.linkedin.com/in/medvedik-david
Telegram: @dmedvedik

Последние 7 лет пишу на C++

Positive Technologies
- Telecom Attack Discovery

EPAM
- **Acronis**
  - Acronis True Image

MFI Soft
- Internet traffic filtering systems

Orion Innovation
- **Ericsson**
  - SS7 protocols
- **Honeywell**
  - Smart meter

High load: коротко о главном

Хайлоад - определённые условия работы системы, при которых возникает дефицит вычислительных ресурсов

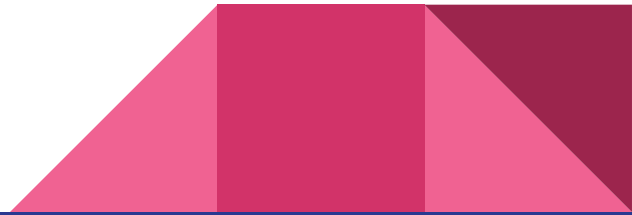
High load: симптомы

Симптомы приближения highload:

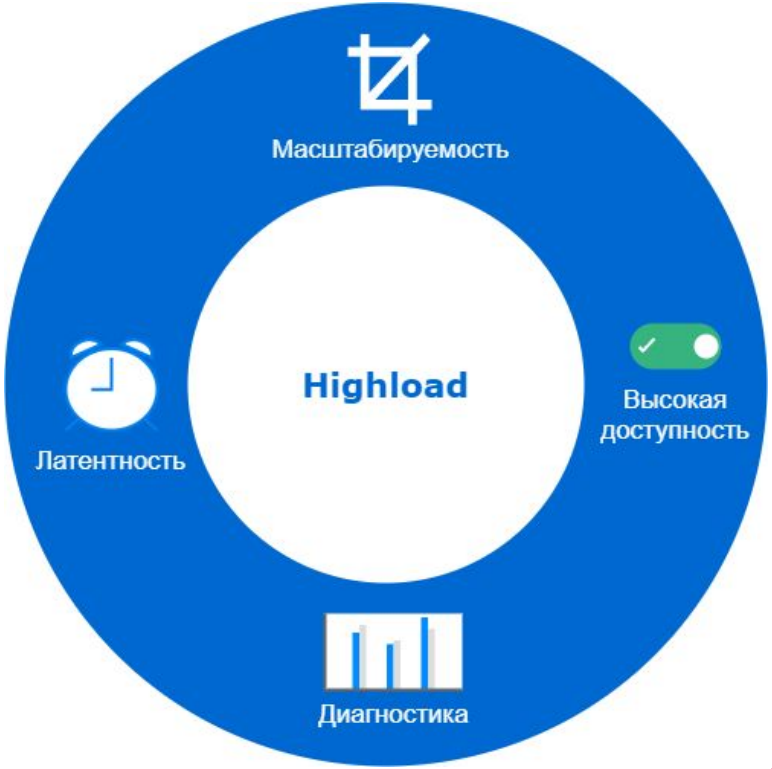
- Долгое выполнение/отсутствие выполнения запроса
- Периодические «плавающие» ошибки
- Обрывы соединения, падение модулей/всей системы
- Стремительное сокращение свободной памяти

Виды нагрузок:

- Увеличение количества запросов
- Увеличение объёма запросов
- Увеличение объёма обрабатываемых данных
- Неправильная настройка/обработка



Формула high load



Как поможет C++

Одна из главных особенностей C++ то что он позволяет писать высокопроизводительный и верхнеуровневый код.

Это достигается за счет:

- Работа с памятью напрямую
- Семантика перемещения(Move semantics)
- Пропуск копии(Copy elision)
- Работа с потоками

Работа с памятью

C++ позволяет запросить память в куче напрямую у операционной системы при помощи оператора new.

```
int main{
    int *ptr = new int();
    cout<<*ptr<<endl;
    delete ptr;
    return 0;
}
```

Работа с памятью, идиома RAII

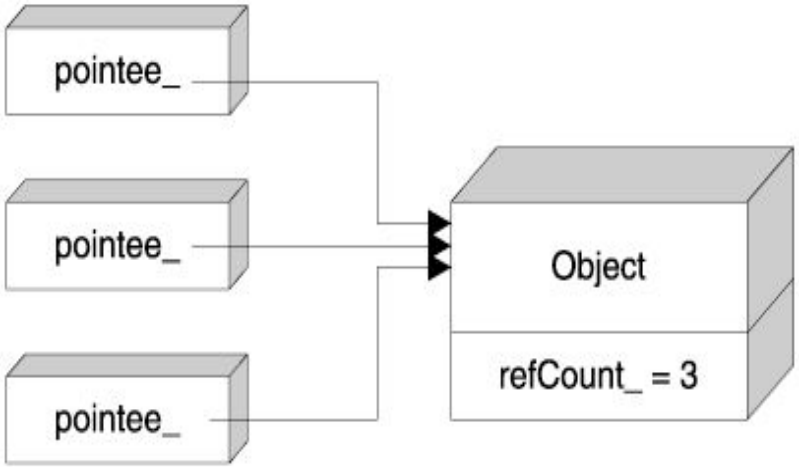
Получение ресурса есть инициализация (англ. **Resource Acquisition Is Initialization (RAII)**) - получение некоторого ресурса неразрывно совмещается с инициализацией, а освобождение - с уничтожением объекта.

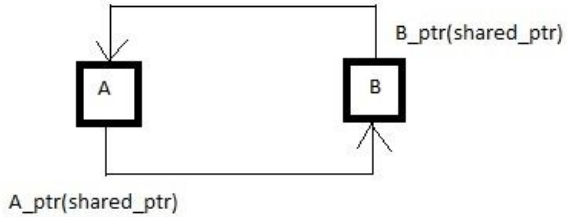
```
class LockNet{
public:
     LockNet(const Url &url){
          m_net = new Network(url);
     }
     ~LockNet (){
          delete m_net;
     }
     operator Network * (){
          return network;
     }
private:
    Network *m_net;
};
```

Работа с памятью, умные указатели

C++ предоставляет следующие умные указатели:

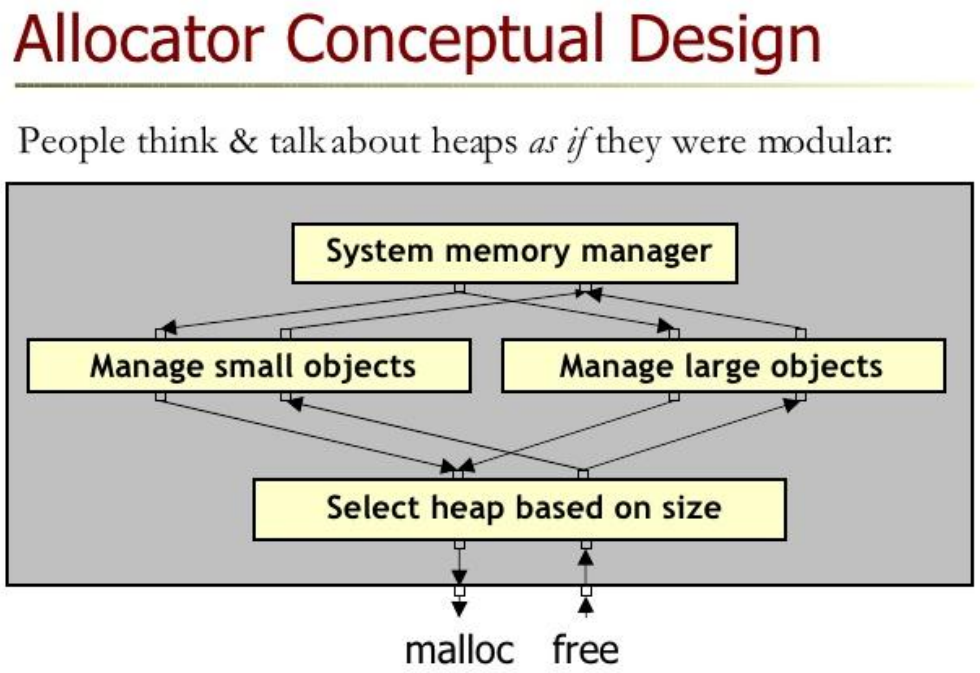
- shared_ptr
- unique_ptr
- weak_ptr





Circular Reference

Работа с памятью, аллокаторы памяти



Работа с памятью, аллокаторы памяти в C++

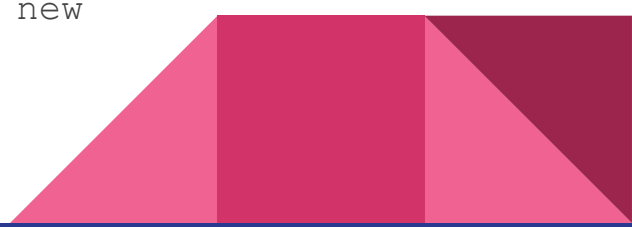
Пример стандартных аллокаторов:

- allocator
- allocate_shared - оболочка для аллокатора чтобы использовать с shared_ptr

```
explicit vector( const Allocator& alloc );
```

Для стандартных контейнеров STL, а так же для умных указателей можно задать свой аллокатор памяти.

```
int* ptr = new (pointer)  int; //использован placement new
```



Семантика перемещения(Move semantics)

Семантика перемещения позволяет избежать ненужных копий при работе с временными объектами, которые вот-вот будут уничтожены, и ресурсы которых могут быть безопасно взяты из этого временного объекта и использованы другим.

Было

```
T& operator=(const T& rhs)
{
  T tmp(rhs);
  swap(tmp);
  return *this;
}
```

Стало

```
T& operator=(T&& rhs)
{
  T tmp(std::move(rhs));
  swap(tmp);
  return *this;
}
```

Пропуск копии(Copy elision)

```
struct C {
  C() {}
  C(const C&) { std::cout << "A copy was made.\n"; }
};

C f() {
  return C();
}

int main() {
  std::cout << "Hello World!\n";
  C obj = f();
}
```

```
Hello World!
A copy was made.
A copy was made.
```

```
Hello World!
A copy was made.
```

```
Hello World!
```

Работа с потоками

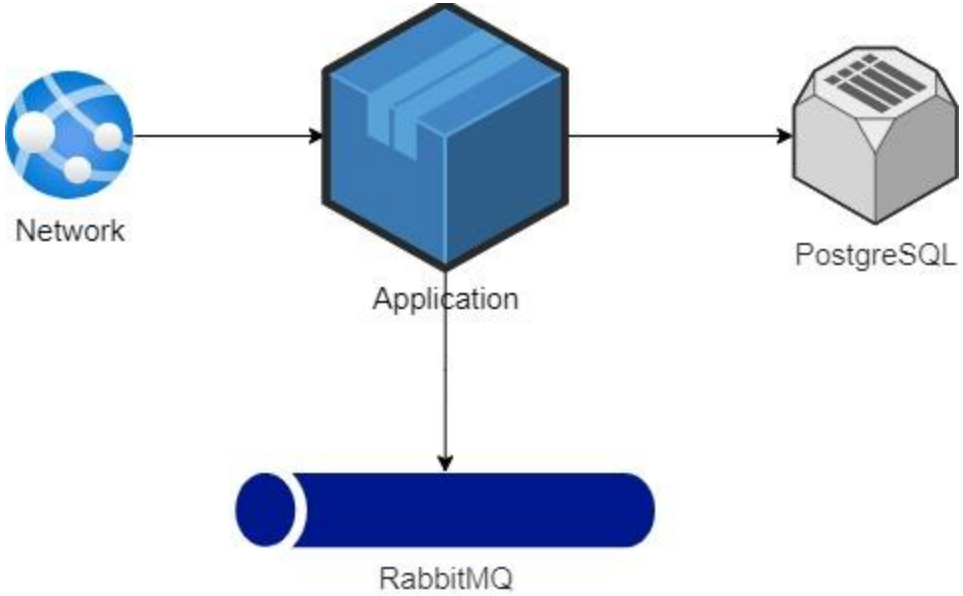
Примитивы синхронизации:

- Mutual exclusion
  - mutex
  - timed_mutex
  - recursive_mutex
- Generic mutex management
  - shared_mutex
  - shared_timed_mutex

Барьеры:

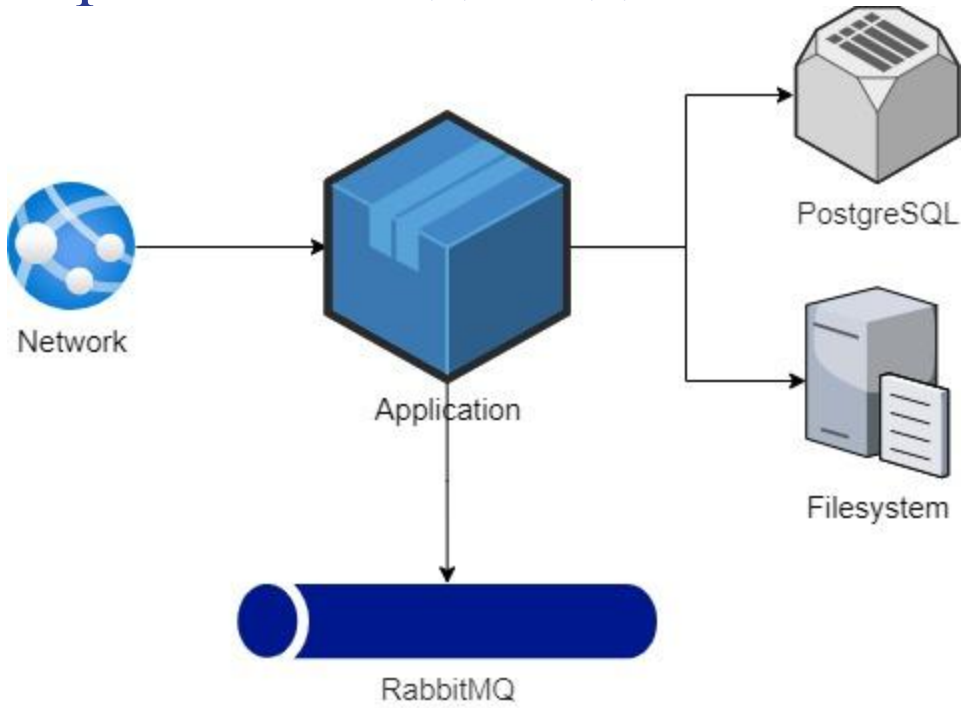
- memory_order_relaxed
- memory_order_consume
- memory_order_acquire
- memory_order_release
- memory_order_acq_rel
- memory_order_seq_cst

Как играть в highload и не проиграть



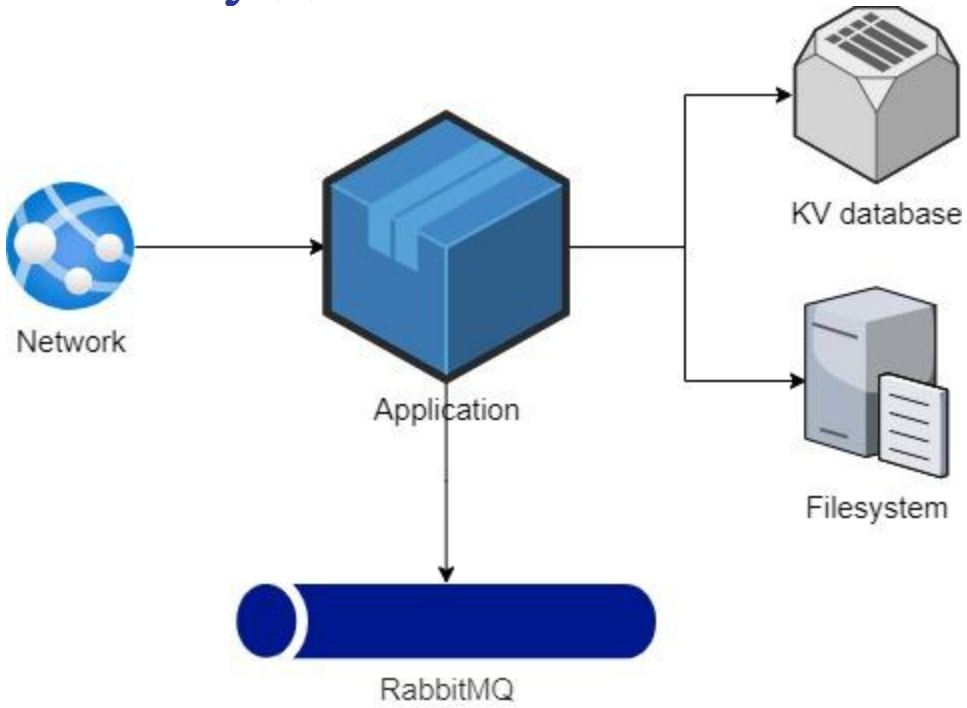
Производительность: 10 попугаев в секунду

Добавляем сохранение на диск данных



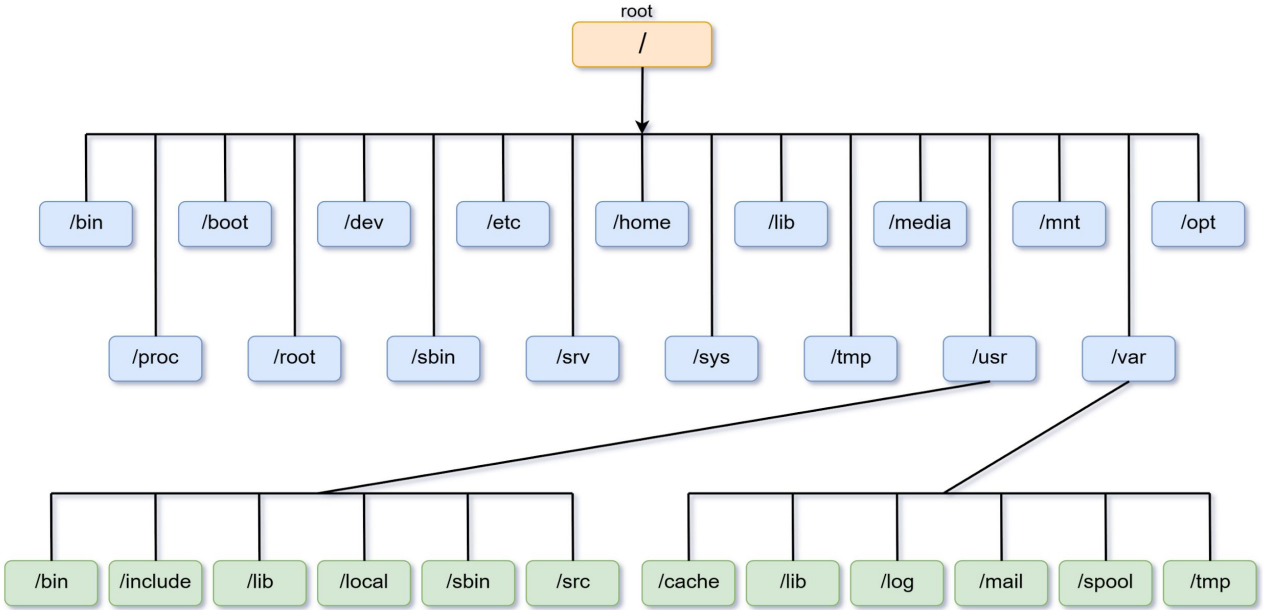
Производительность: 15 попугаев в секунду

Используем KV базу данных

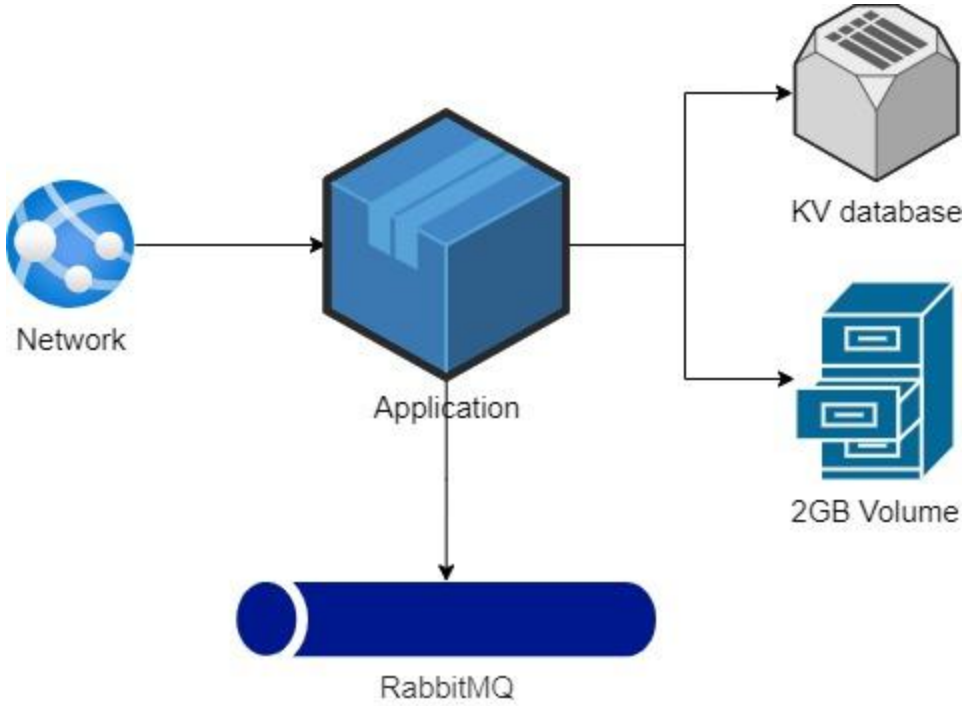


Производительность: 20 попугаев в секунду

Немного теории: работа с файлами



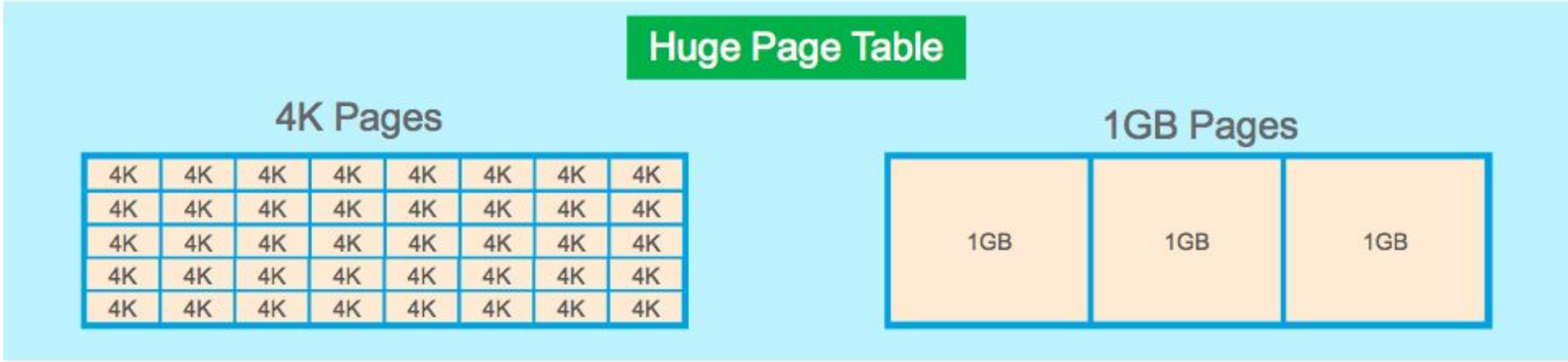
Сохраняем томами по 2GB



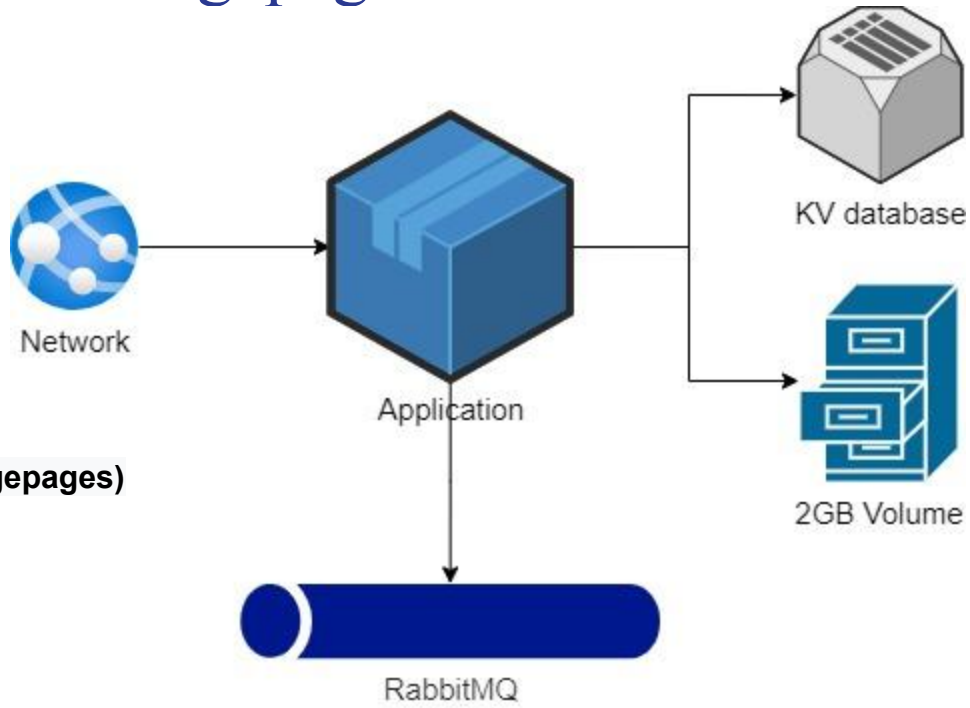
Производительность: 40 попугаев в секунду

Немного теории: работа с памятью

Немного теории, огромные страницы(Hugepages)



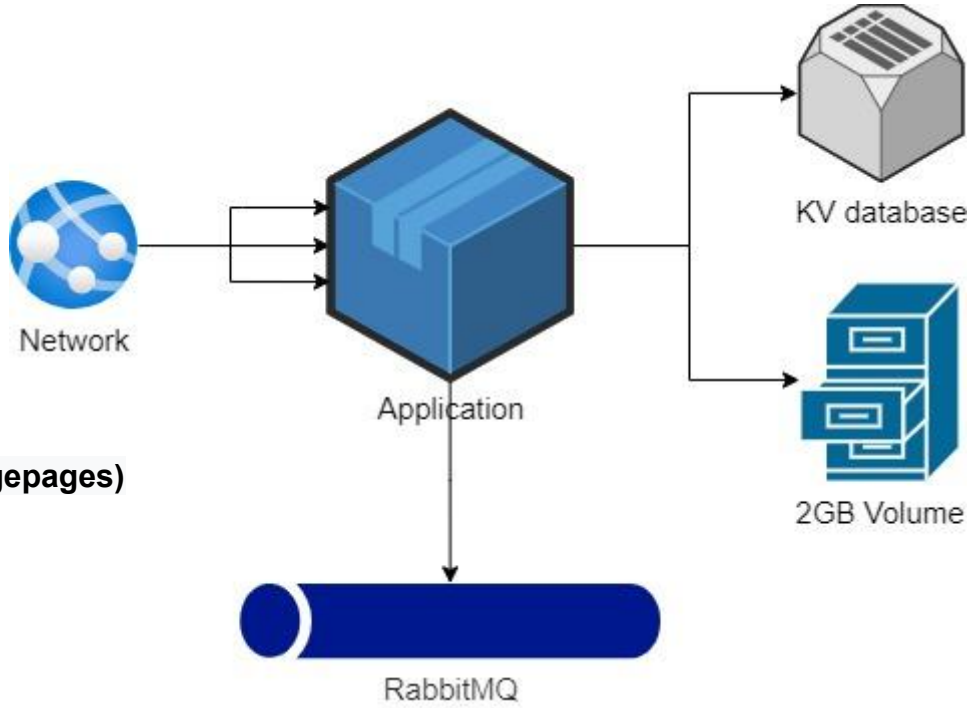
Использование Hugepages



+Огромные страницы(Hugepages)

Производительность: 50 попугаев в секунду

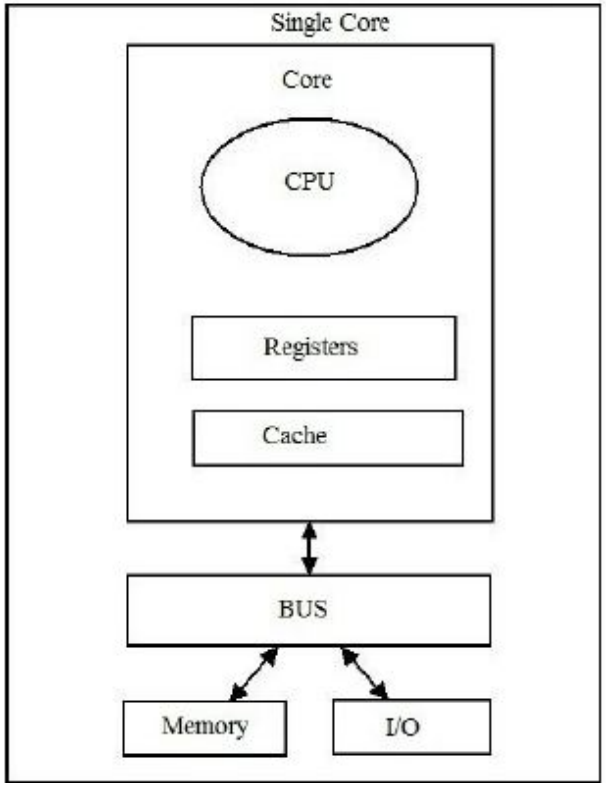
Балансировка входного потока

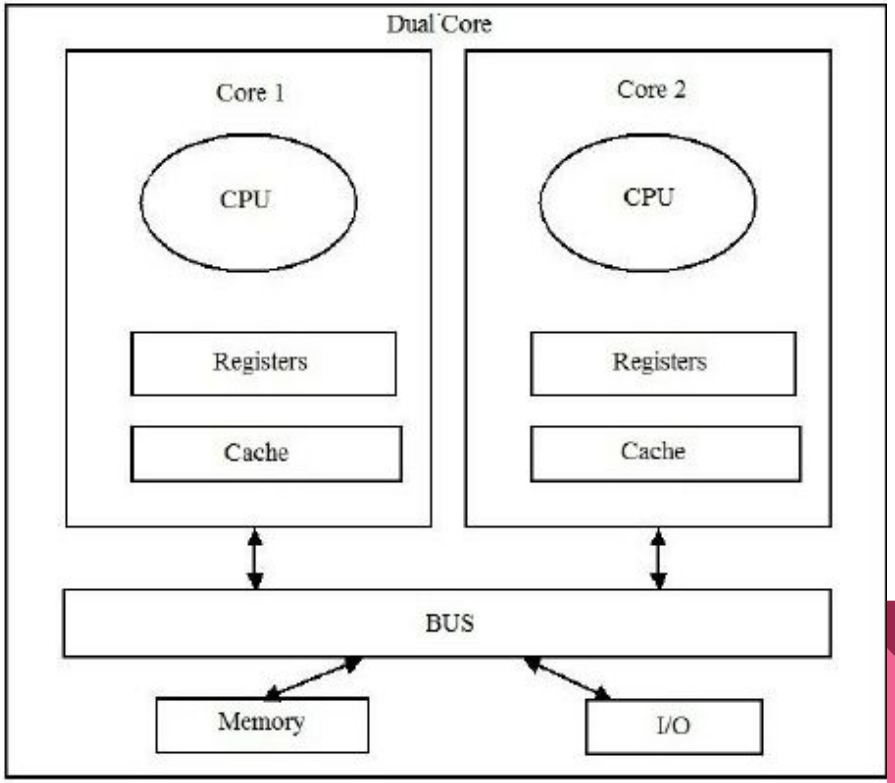


+Огромные страницы(Hugepages)

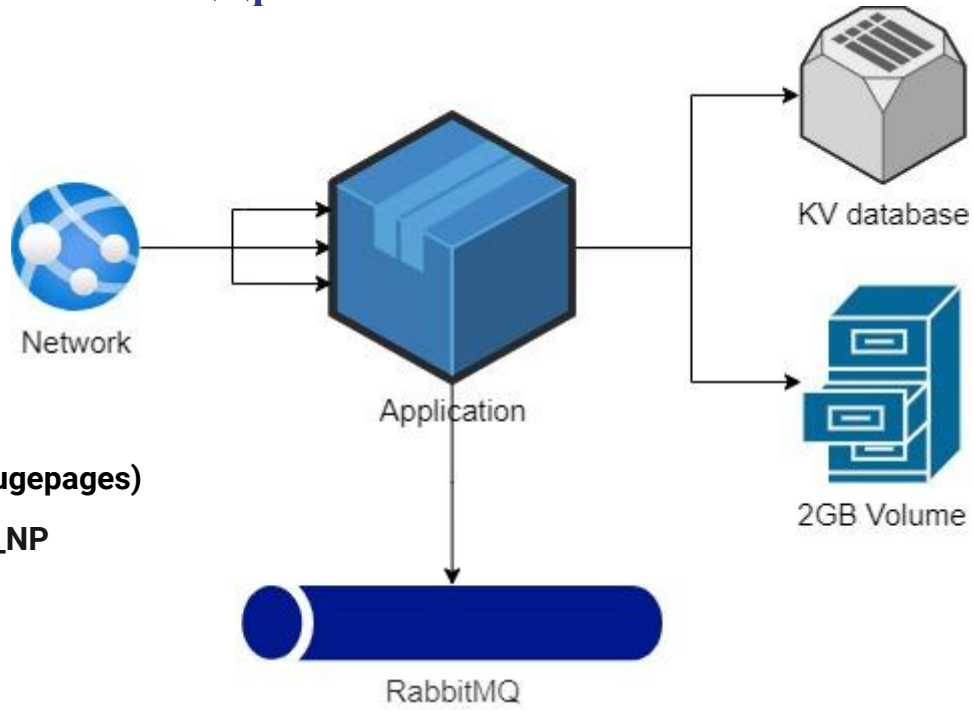
Производительность: 70 попугаев в секунду

Ещё немного теории: работа ядер





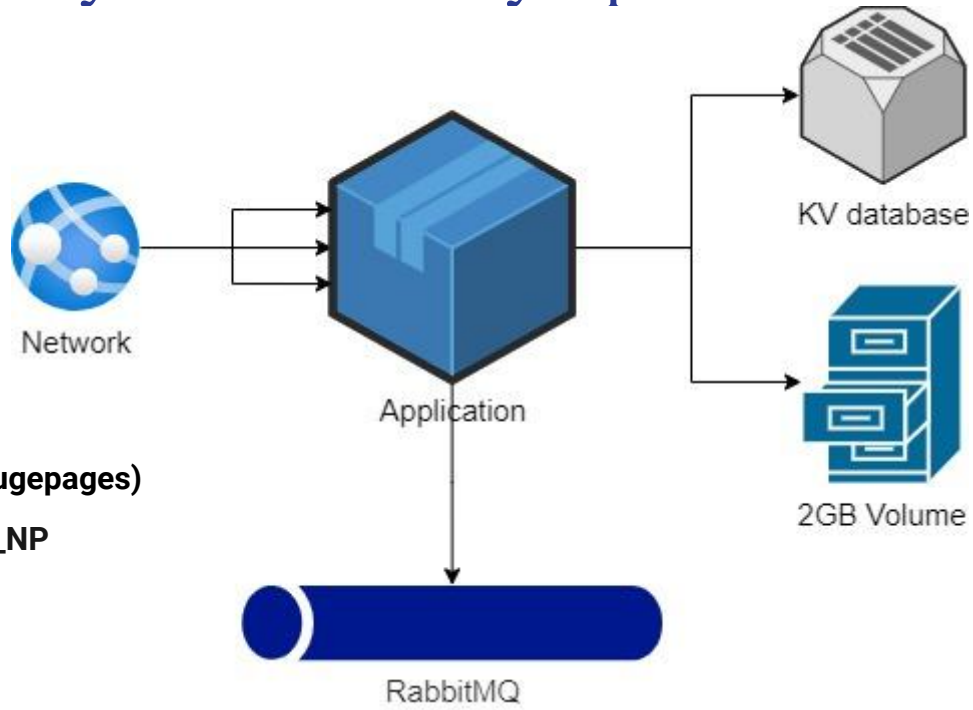
Привязка поток к ядрам



+ **Огромные страницы(Hugepages)**
+ **PTHREAD_SETAFFINITY_NP**

Производительность: 100 попугаев в секунду

Запись напрямую в блочное устройство



+ **Огромные страницы(Hugepages)**
+ **PTHREAD_SETAFFINITY_NP**
+ **/dev/sda**

Производительность: 120 попугаев в секунду

Рефлексия


C# это лучший язык
для highload
Нет, Java подходит лучше
Программисты на C++

Спасибо за внимание!

- Highload
- C++
  - работа с памятью
  - семантика перемещения
  - пропуск копии
  - работа с потоками
- Работа с файлами
- Использование TOMов с данными
- Работа с памятью
- Использование огромных страниц
- Балансировка потока данных
- Привязка потоков
- Запись на блочное устройство

High Load & C++

LinkedIn: www.linkedin.com/in/medvedik-david
Telegram: @dmedvedik