UM ESTUDO COMPARATIVO ENTRE MODELOS DE CONCORRÊNCIA COM ÊNFASE EM ESCALABILIDADE PARA APLICAÇÕES WEB

Mauricio Colognese Concatto

Orientador: Prof. Pablo Dall'Oglio

Roteiro

- Introdução
  - Motivação
  - Objetivos
  - Metodologia
- Conceitos importantes
  - Processos
  - *Threads*
  - *Context switch*
  - Entrada/Saída (E/S)
- Trabalho Proposto
  - Cenários
  - Métricas
  - Modelos de concorrência
  - Arquitetura
  - Testes
- Resultados
- Considerações finais

Introdução

- Web 2.0
  - Aplicações Web mais acessíveis, dinâmicas e interativas
  - Aplicações mais focadas em problemas específicos
  - APIs para integração
- Ascenção da computação em nuvem: IaaS, PaaS, SaaS...
- *“The Free Lunch Is Over”*. Herb Sutter (2005).
  - *Clock* das CPUs chegando ao limite
  - Necessidade de explorar os novos recursos disponíveis nos hardwares e softwares atuais
  - Escalabilidade através da concorrência

Motivação

- Desafios de aplicações Web operando no modelo SaaS
  - Alta variação de carga e acessos concorrentes
- Diferentes modelos para a implementação de servidores
  - *Thread-per-connection (multithread):* Apache, Tomcat (*Servlets*)
  - *Event-driven:* Nginx, NodeJS
- Comparativos existentes entre os modelos abordam o problema de maneira superficial
- Que benefícios podem ser obtidos ao aplicar estes conceitos em toda a arquitetura de uma aplicação ?
- Todos os tipos de aplicações podem se beneficiar ?

Objetivos

- Realizar um estudo comparativo detalhado sobre o modelo *multithread* e o modelo de atores em aplicações Web
- Preencher as lacunas existentes nos diversos estudos encontrados sobre o assunto
- Objetivos específicos:
  - Revisão bibliográfica sobre tema
  - Definir um conjunto de cenários sobre os quais serão feitas as avaliações
  - Implementar protótipos dos cenários definidos em ambos os modelos selecionados
  - Realizar simulações de carga nos protótipos desenvolvidos
  - Coletar os dados resultantes das simulações
  - Analisar quantitativamente os dados coletados

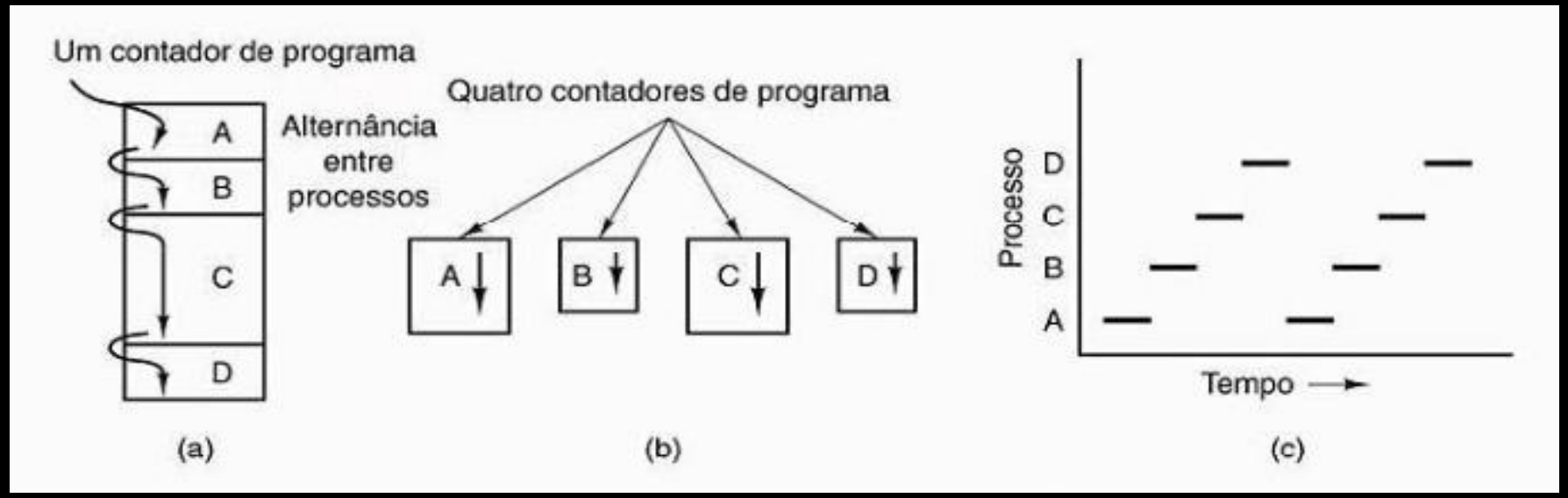
Metodologia

- Pesquisa exploratória
  - Aproximação com o tema
- Pesquisa bibliográfica
  - Revisão da literatura existente sobre os assuntos que compõem o estudo
- Pesquisa quantitativa
  - Comparação de dados obtidos através dos experimentos realizados
- Pesquisa de laboratório
  - Uso de ambientes controlados para execução de testes e obtenção de dados

Conceitos Importantes

Processos

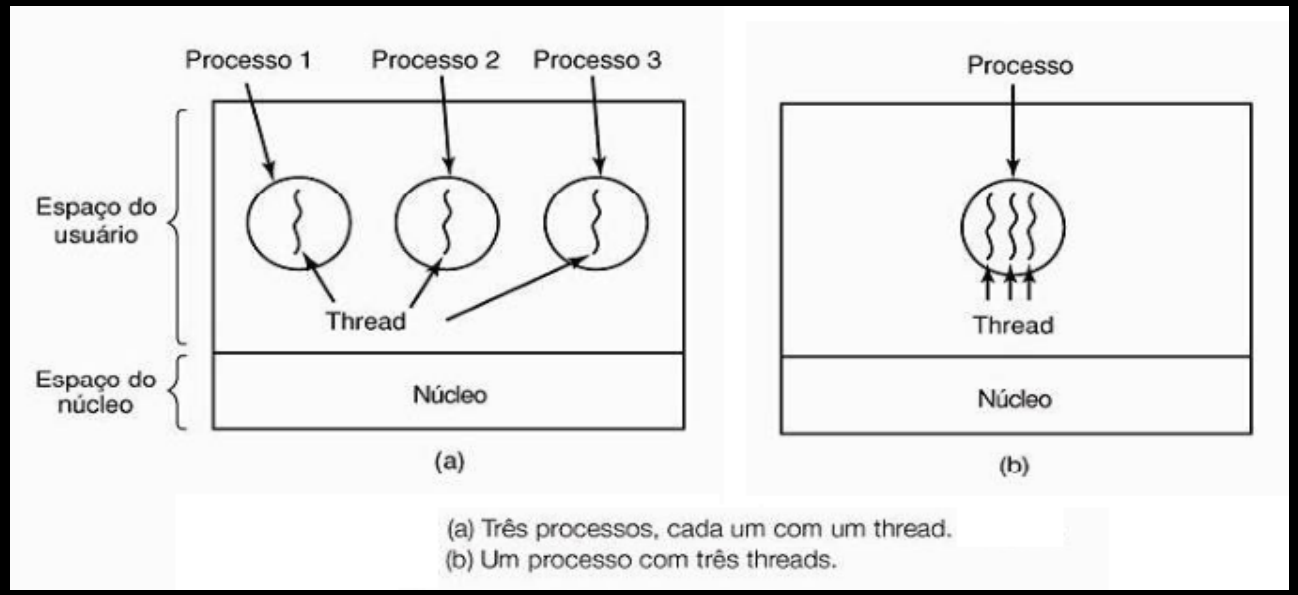
- Abstração de um programa em execução
  - Isolamento
  - Área de memória dedicada
- Comunicação: Uso de mecanismos de *Inter-process Communication* (IPC)
- Sistema Operacional: Uso de algoritmos de escalonamento para definir quando parar um processo e iniciar outro



Fonte: (TANENBAUM, 2010)

Threads

- Fluxos de execução que residem e compartilham recursos já alocados por um processo
- Menor custo de alocação
- Memória compartilhada
- Uso de técnicas de sincronização



Fonte: (TANENBAUM, 2010)

Context switch

- Alternar entre a execução de processos ou *threads* obriga o sistema operacional a salvar e restaurar o contexto de execução
- Custo: ~1.000-3.000ns
  - Arquiteturas *Sandy Bridge*
    - Família Intel 2xxx
      - I7 2600K, I5 2500K, etc.
  - Kernel Linux 2.6+
- Afeta a execução dos programas
  - Processador
    - Invalidação de *cache* (TLB)
    - Invalidação do *pipeline* de execução

Fonte: <http://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make-context.html>. Acesso em novembro 2016.

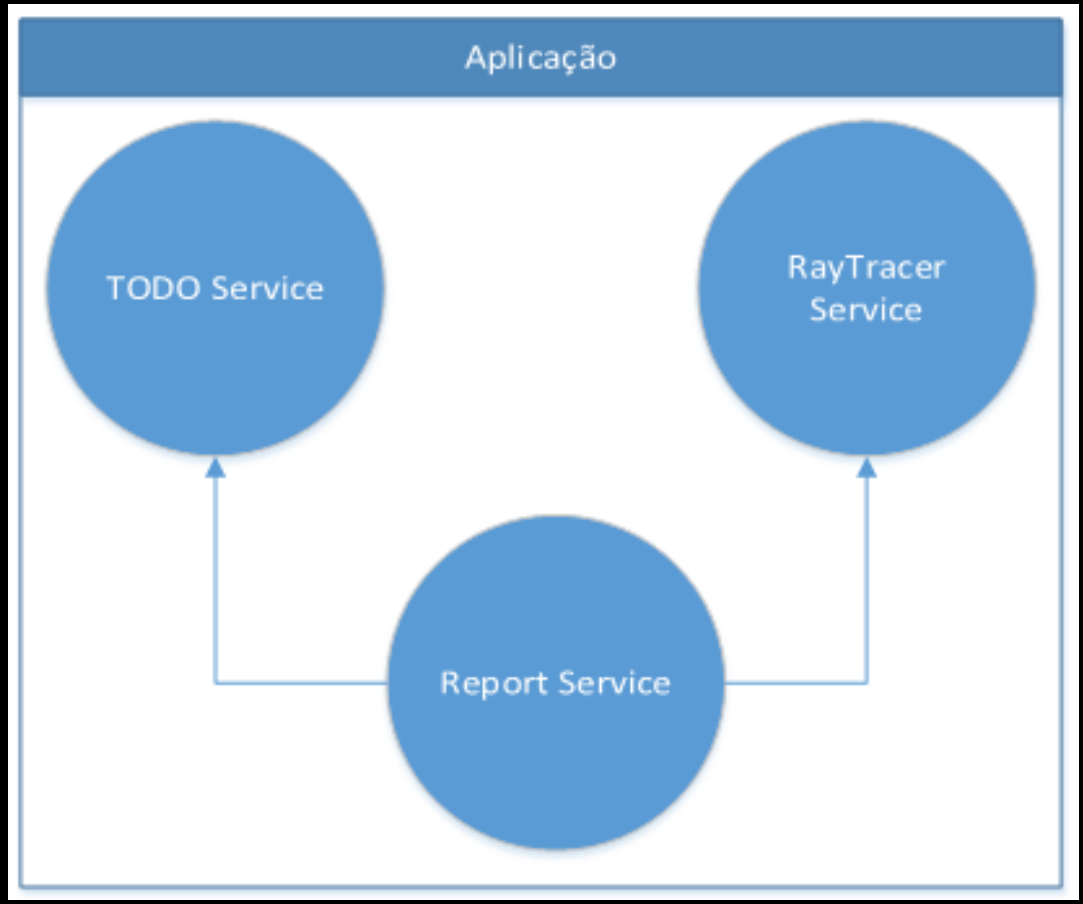
Entrada/Saída (E/S)

- Operações envolvendo leitura e escrita em descritores
- Extremamente lentas (em comparação ao processador)
- E/S Síncrona
  - *Thread*/Processo fica bloqueado até toda operação de E/S ser finalizada
- E/S Assíncrona
  - Dispara a execução e retorna o controle imediatamente
  - Coordenado por operações de *callback* invocadas pelo SO quando o descritor esta pronto para leitura/escrita

Trabalho proposto

- Cenários
- Métricas
- Modelos de concorrência
- Arquitetura
- Testes

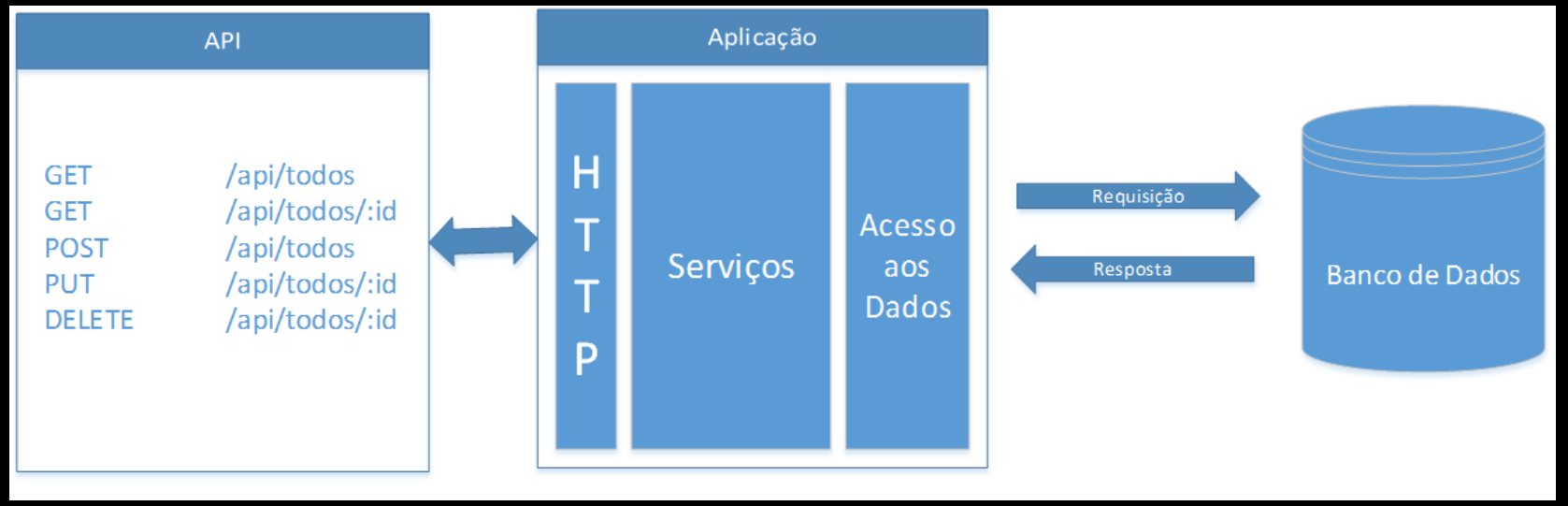
Cenários



Fonte: Autor

TODO service

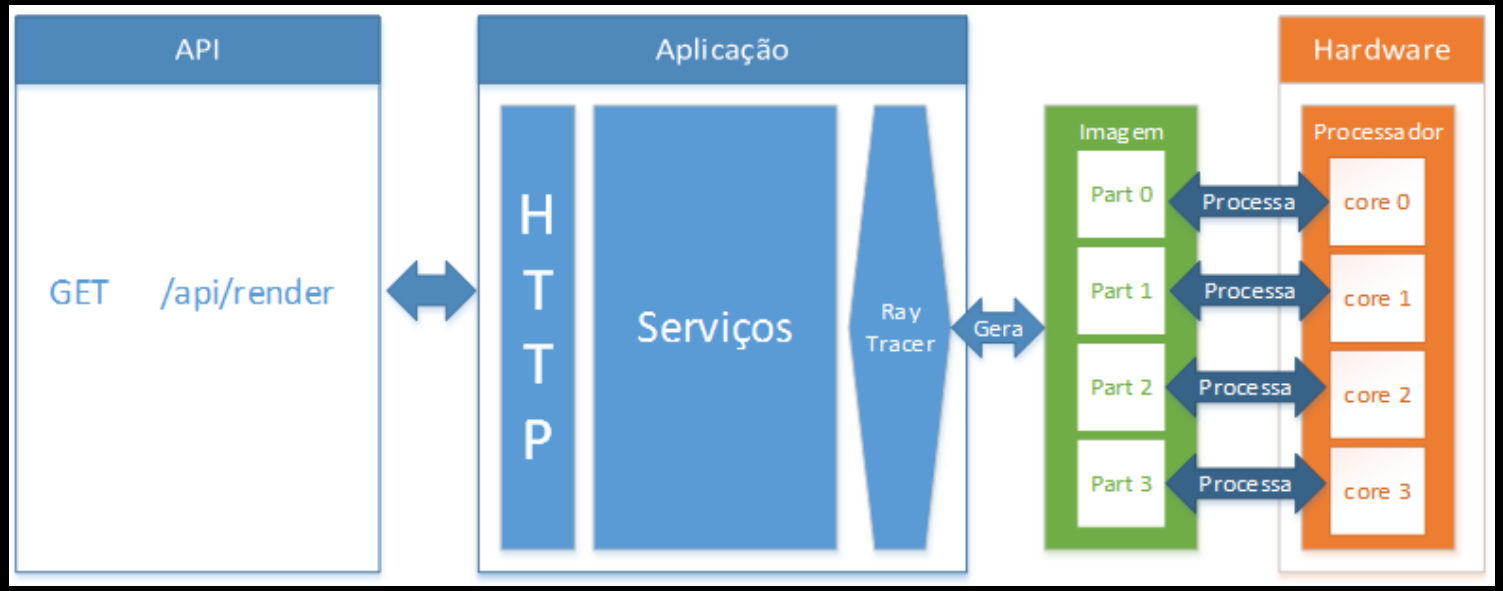
- Objetivo: Gerenciamento de uma lista de tarefas
- Características
  - Utilização de um banco de dados (PostgreSQL)
  - Expõem API via HTTP
  - Carga de trabalho mista
    - Processamento
    - E/S



Fonte: Autor

Raytracer service

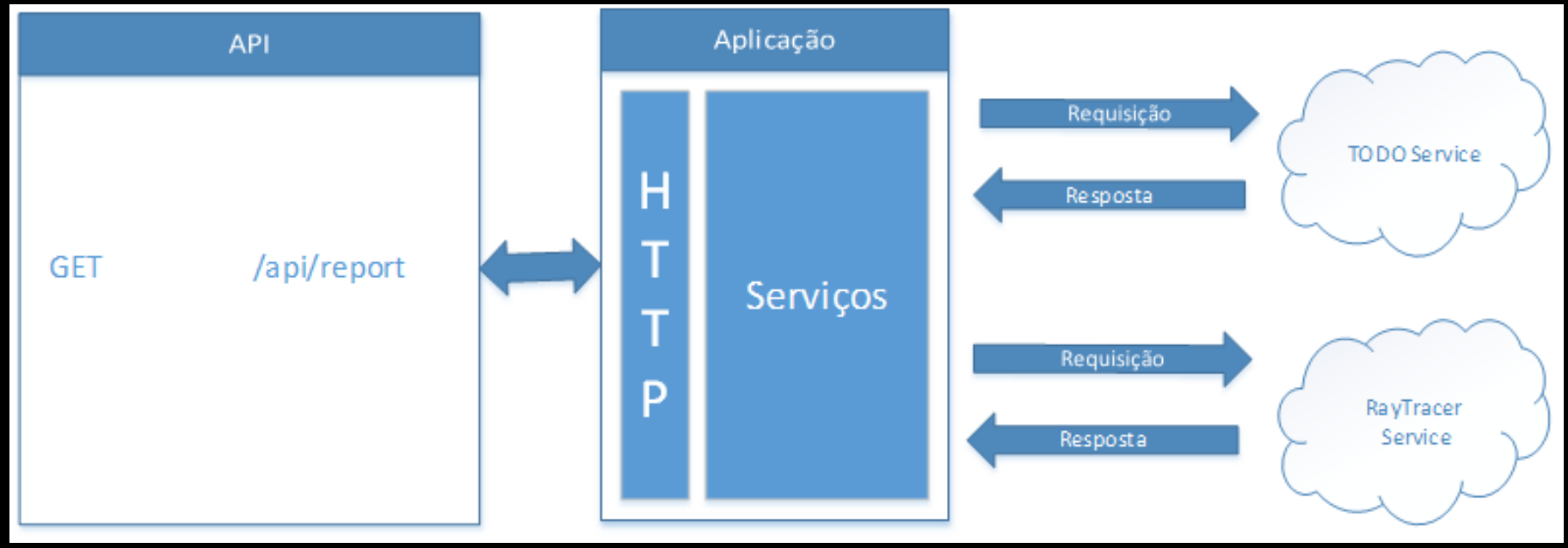
- Objetivo: Geração imagens utilizando o método de renderização *Ray tracing*
- Características
  - Expõem API via HTTP
  - Busca simular aplicações que fazem uso intenso do processador e devido a isso são altamente bloqueantes



Fonte: Autor

Report service

- Objetivo: Geração de relatório HTML baseado no resultado coletado das outras duas aplicações
- Características
  - Expõem API via HTTP
  - Busca simular aplicações que possuam alta dependência de operações de E/S.



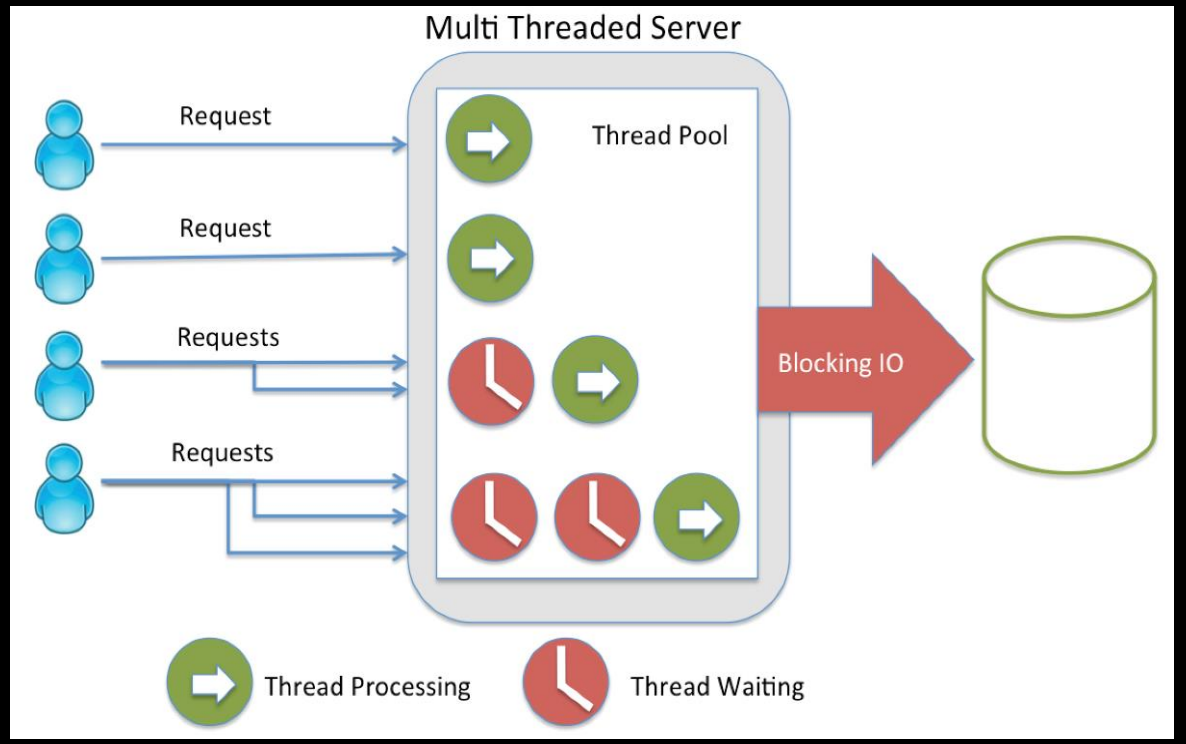
Fonte: Autor

Métricas

- Métricas coletadas durante a execução das simulações de carga:
  - Tempo de resposta mínimo, médio e máximo
  - Uso do processador pela aplicação
  - Memória utilizada pela aplicação
  - Número de *threads* alocadas durante a simulação

Modelo Multithread

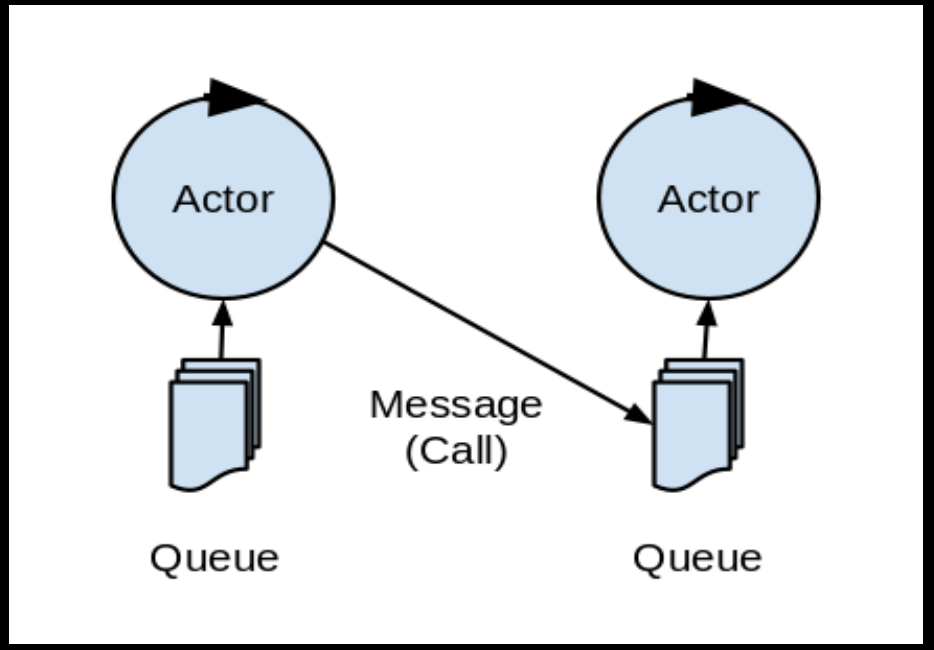
- *Threads*
- Memória compartilhada
- Estruturas de sincronização
  - *Locks*



Fonte: Retirado da página *Strongloop*

Modelo de atores

- Foco na troca de mensagens
- Canais de comunicação assíncronos
- Atores
  - Mensagens imutáveis
  - Encapsulamento do estado
  - Uso de filas (caixa de mensagens)
  - Evitar processamento bloqueante

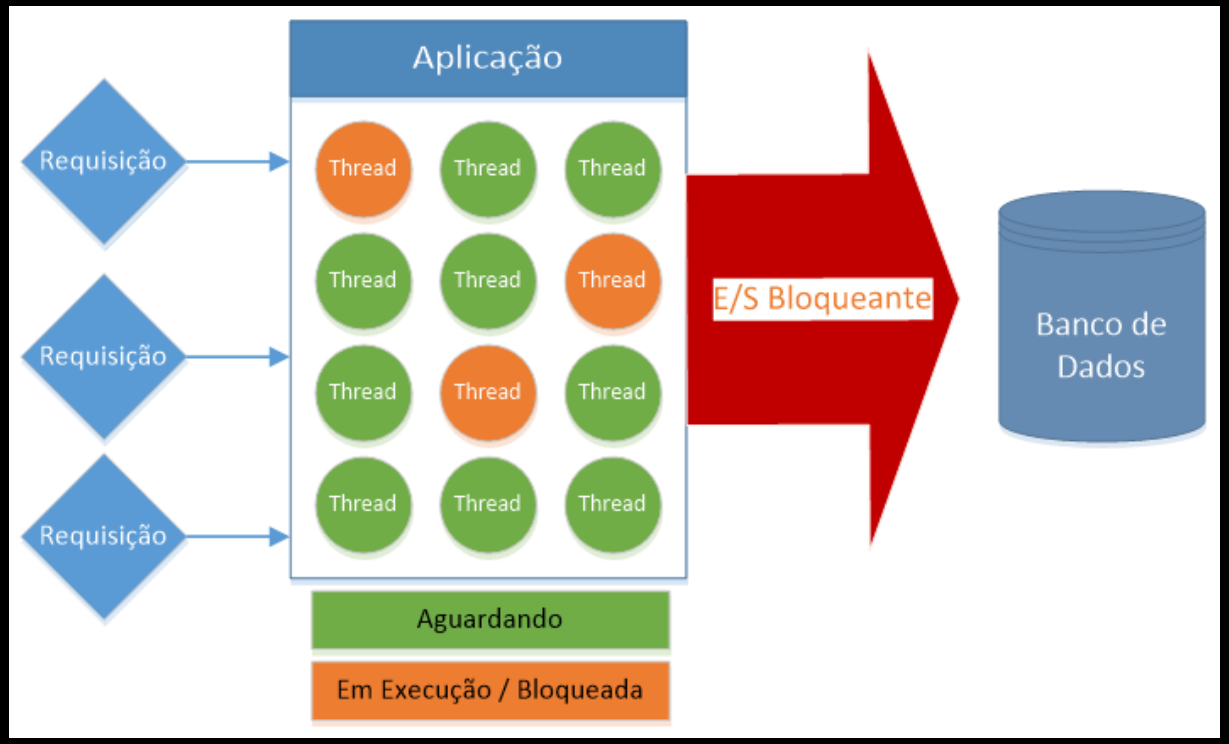


Fonte: Retirado da página *Java is the new C*

Arquitetura

Modelo Multithread

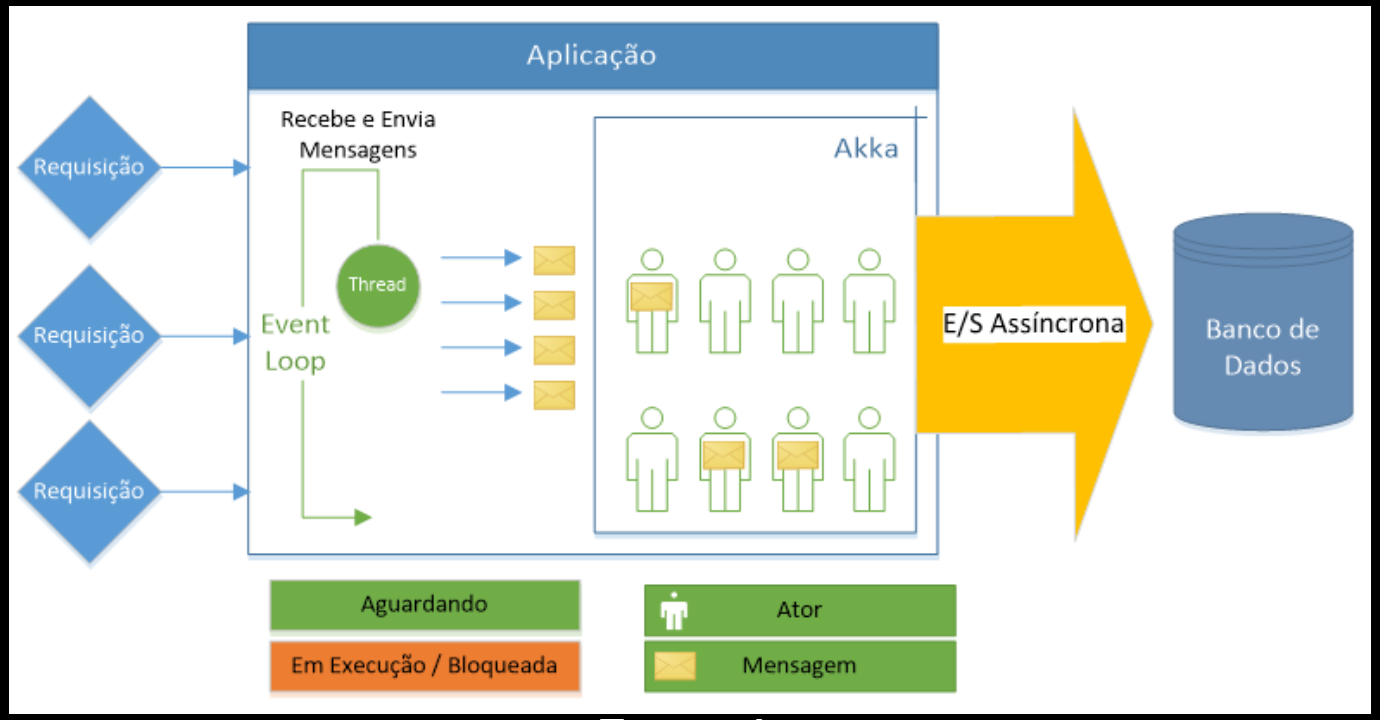
- *Thread-per-connection*
- HTTP: Spring MVC (Tomcat - *Servlet*)
- Acesso a banco de dados: Spring JDBC (JDBC)
- Uso de E/S síncrona



Fonte: Autor

Modelo de Atores

- Akka
- HTTP: Akka HTTP
- Acesso a banco de dados: Postgres Async Driver
- Uso de E/S assíncrona



Fonte: Autor

Testes

Ambiente

- 4 VMs tipo **n1-highcpu-2** – Google Cloud
  - Processador: Intel Xeon E5-2697v3 2,30GHz (dois núcleos)
  - Memória: 1,8GB 1600MHz
  - Disco: SSD 10GB sem especificação
  - Sistema Operacional: Ubuntu Server 16.04
- Uma VM por serviço:
  - PostgreSQL
  - TODO Service
  - RayTracer Service
  - Report Service
- Estabelecido limite de 1GB de memória para as aplicações na JVM

Condução dos testes

- Inicialização da aplicação e de suas dependências através de *scripts* (realizam limpeza do ambiente como por exemplo banco de dados, arquivos temporários, etc.)
- Inicialização do VisualVM e conexão com a aplicação para realização do monitoramento
- Realização das três execuções da simulação utilizando o Gatling
- Coleta dos resultados

Carga de teste

Aplicação	Usuários Simulados	Requisições Por Usuário	Total de Requisições
TODO service	1.000	5	5.000
Raytracer service	50	1	50
Report service	50	1	50

Resultados

TODO Service

Critério	*Multithread*	Atores
Tempo total (s)	166	6
Tempo médio resposta (s)	19,63	0,5
***Throughput* (Req./s)**	30,12	833,33
Req. com erros (%)	11,38	0
Uso CPU (%)	~20-50	~90
Uso Memória (MB)	~350-500	~80-125
Threads	~50-223	~20-27

TODO Service

- *Multithread*
  - Alto uso de memória
  - Esgotamento do *thread pool* do Tomcat (max. 200)
  - Alto numero de *threads* alocadas
  - Processador ocioso (aguardando E/S)
    - Context Switch
    - Maior atividade do Garbage Collector
- Atores
  - Baixo consumo de memória:
    - Uso de poucas *threads*
    - E/S assíncrona
  - Uso mais eficiente do processador
    - Menos *threads* = menos operações de *context switch* = execução continua

Raytracer Service

Critério	*Multithread*	Atores
Tempo total (s)	61	21
Tempo médio resposta (s)	60	18,1
***Throughput* (Req./s)**	0,82	2,38
Req. com erros (%)	100	76
Uso CPU (%)	~100	~100
Uso Memória (MB)	~100-300	~100-300
Threads	~70-120	~16-25

Raytracer Service

- Consumo de recursos similar
- *Multithread*
  - 50 requisições competindo pelo processador (2 núcleos)
  - *Context switch*
- Atores
  - 2 execuções simultâneas (2 núcleos)
  - Uso de filas
  - Enfileiramento da execução permitiu um uso mais eficiente do processador
  - Menos operações de *Context switch*

Report Service

Critério	*Multithread*	Atores
Tempo total (s)	61	21
Tempo médio resposta (s)	60	17,4
***Throughput* (Req./s)**	0,82	2,38
Req. com erros (%)	100	76
Uso CPU (%)	~0-10	~0-10
Uso Memória (MB)	~100-250	~20-80
Threads	~30-72	~16-24

Report Service

- Replicação do resultado o *microservice* mais lento (RayTracer Service)
- Modo como o teste foi arquitetado impede uma comparação de performance entre as implementações
- Uso de recursos
  - Modelo de atores consegue trabalhar de maneira mais eficiente e com um menor consumo de memória devido ao baixo número de *threads* alocadas
  - Uso de E/S assíncrona

Conclusão

- O estudo conseguiu complementar a lacuna de informação encontrada em diversos comparativos existentes na Web
- O estudo gerou resultados relevantes:
  - A pesquisa permitiu identificar o potencial do modelo de atores em conjunto com operações de E/S assíncronas para a criação de aplicações de alta performance
  - As implementações e testes realizados permitiram comprovar na pratica este potencial
- Foi possível atingir os objetivos estabelecidos no inicio do trabalho.

Duvidas?