Processamento de Dados em Tempo Real (Streaming)

Centro Universitário Christus- UNICHRISTUS

Especialização em Ciência de Dados e Inteligência de Negócios (Big Data e BI)

Prof. Dr. Manoel Ribeiro

Unichristus
Centro Universitário Christus

Conteúdo da disciplina

- Dia 1 (sexta 18:00 às 22:00h)
  - Apresentação
  - Aula motivacional - Qual a importância do processamento de dados em tempo real?
- Dia 2 (sábado 8:00 às 18:00)
  - Fundamentos de sistema de processamento de tempo real
  - Fundamentos da arquitetura Apache Kafka, Apache ZooKeeper
  - Prática com Apache Kafka (tarde)

Conteúdo da disciplina

- Dia 3 (sexta 18:00 às 22:00h)
  - Fundamentos Apache Flume
  - Fundamentos Spark Streaming
  - Configuração Hadoop e Spark
- Dia 4 (sábado 8:00 às 18:00)
  - Fundamentos Introdução ao Apache Storm
  - Implementação de projeto prático/aplicado envolvendo Real Time Analytics
  - Avaliação

Repositório

https://github.com/antoniomralmeida/streaming


Fundamentação

Escalonamento de processos

O escalonamento de processos ou agendador de tarefas (em inglês scheduling) é uma atividade realizada pelo escalonador (scheduler) de sistema operacional ou aplicação, possibilitando executar os processos mais viáveis e concorrentes, priorizando determinados tipos de processos, utilizando critérios bem definidos.

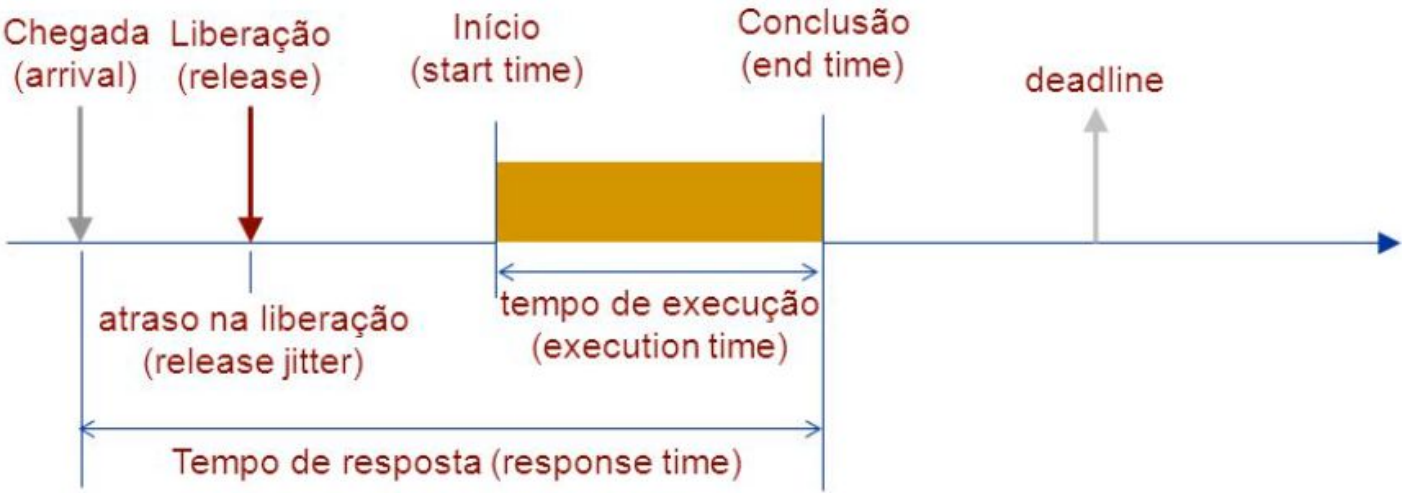
Escalonamento de processos em Tempo Real

- Tempo real
    - Não crítico (soft): processos críticos devem receber prioridade sobre outros e.g.: Linux e Windows
    - Crítico (hard): tarefas críticas devem executar dentro um período de tempo estabelecido (deadline)
        - Estático: as previsões são feitas antes da execução
        - Dinâmico: as decisões são tomados em tempo de execução

Notação

B -> tempo de bloqueio de pior caso (se aplicável)
C -> tempo de computação de pior caso (WCET)
D -> deadline
I -> tempo de interferência
J -> jitter de disparo
N -> número de processos no sistema
P -> prioridade atribuída ao processo (se aplicável)
R -> tempo resposta de pior caso
T -> tempo mínimo entre disparos de um processo (período)
U -> utilização de cada processo (U=C/T)

Modelo de tarefa



- Folga (*slack*) = deadline – liberação – tempo de execução
- Atraso = MÁX(0, conclusão – deadline)

Modelo básico de processo

É difícil prever o pior caso de programas concorrentes de tempo real quando a complexidade aumenta → impõe-se restrições na estrutura de tais programas:

- A aplicação consiste de um conjunto fixo de processos, bem diferente dos SOs de uso geral onde os processos são criados e destruídos dinamicamente
- Todos processos são periódicos com períodos conhecidos
- Os processos são completamente independentes i.e. não se comunicam/sincronizam entre si (são concorrentes mas não são cooperantes)
- Instante crítico: instante em que todos processos são disparados simultaneamente (carga máxima processador)
- **Todos processos tem um deadline igual ao seu período (i.e. o processo deve completar antes ser novamente disparado)**
- **Processos de tempo real tem um deadline interno**

Necessidade de Diferentes Abordagens

- Mercado para tempo real é amplo
- Sistemas de tempo real variam enormemente
  – Sistema de emergência em usina petroquímica
  – Controle de temperatura do freezer
  – Videogame
- Principais variações:
  – Crítico ou não crítico
  – Carga estática ou dinâmica
  – Importância associada com cumprimento dos deadlines
- Diferentes abordagens são necessárias

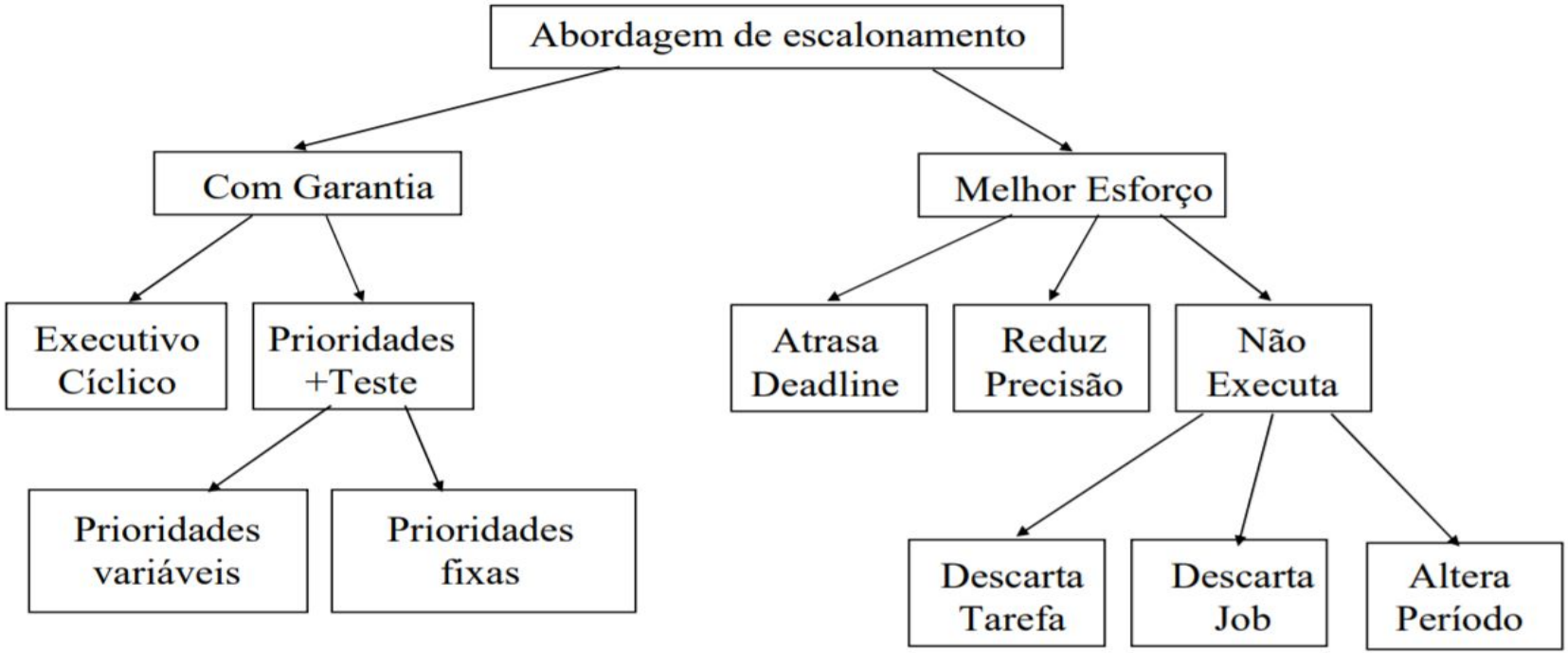
Abordagem Síncrona

Supõe que a velocidade do computador é infinita

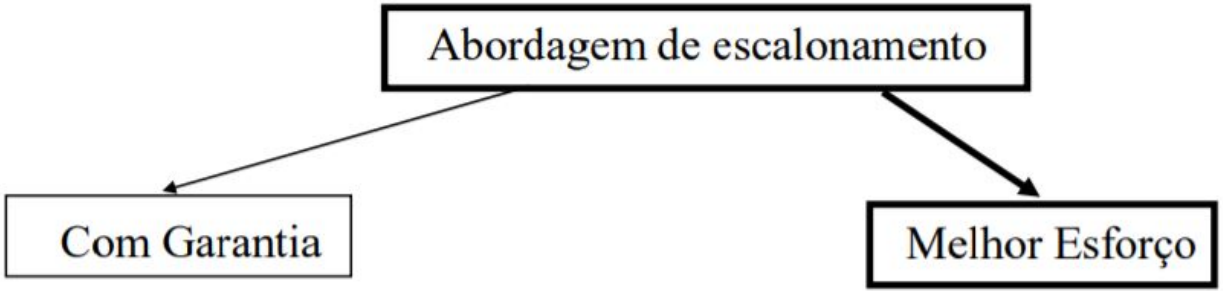
- Logo, tempo de computação é zero
- Tempo de resposta é zero
- Tudo é instantâneo
- Todos os deadlines estão automaticamente atendidos
- Não existe a necessidade de escalonamento
- Velocidade do computador sempre será finita, porém
  – Se o computador for muito mais rápido que o ambiente que estabelece os requisitos temporais
  – É possível assumir que a premissa é verdadeira
  – Exemplo: relógio despertador, processo químico, térmico

Abordagem Assíncrona

- Abordagens básicas para o escalonamento tempo real
  - Visão Assíncrona



Abordagem com Melhor Esforço



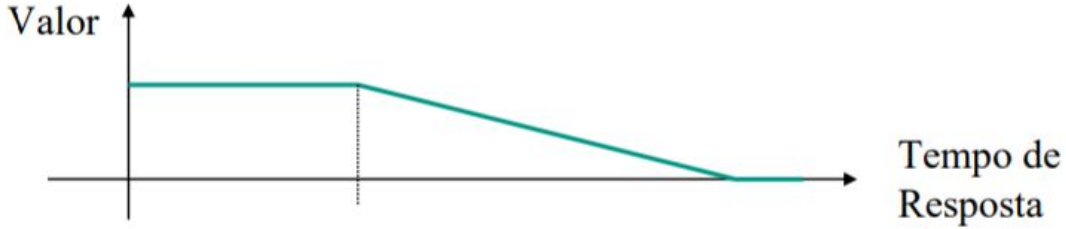
- Não existe garantia que os deadlines serão cumpridos
- O "Melhor Esforço" será feito neste sentido
- Capaz de fornecer análise probabilista
  - Simulação, teoria das filas de tempo real, etc
- Algumas abordagens oferecem Garantia Dinâmica
  - Garante o deadline (ou não) no início da ativação do job

Abordagem com Melhor Esforço

- Existe a possibilidade de Sobrecarga (overload)
- Sobrecarga:
  – Não é possível cumprir todos os deadlines
  – Não é uma falha do projeto
  – É uma situação natural uma vez que não existe garantia antecipada
- Vantagens desta abordagem
  – Não é necessário conhecer o pior caso
  – Sistemas mais baratos, não são projetados para o pior caso
  – Não é necessário conhecer a carga exatamente
- Desvantagens
  – A princípio qualquer deadline poderá ser perdido
- Questão fundamental: Como tratar a sobrecarga ?

Perda de Deadlines na Sobrecarga

- Em sobrecarga ATRASA algumas tarefas
- Baseado em função tempo-valor (time-value function)
- Cada tarefa possui uma função que
  - Indica o valor da tarefa para o sistema em função do seu instante de conclusão
- Objetivo é maximizar o valor total do sistema
  - Valor do sistema é o somatório do valor das tarefas executadas
  - Tarefas podem possuir importância (peso) diferentes



Redução da Precisão na Sobrecarga

- Em sobrecarga DIMINUI a precisão de algumas tarefas
- Objetivo é “fazer o trabalho possível dentro do tempo disponível”
- Exemplos:
  – Ignorar bits menos significativos de cada pixel
  – Trabalhar com amostras de áudio menos precisas
  – Alterar resolução e tamanho de imagem na tela
  – Simplificar animações
  – Usar algoritmos de controle mais simples
  – Interromper pesquisa em algoritmos de inteligência artificial
- Aparece associada com a técnica de
  Computação Imprecisa (Imprecise Computation)

Imprecise Computation

- Cada tarefa dividida em
  – Parte obrigatória (mandatory part)
  – Parte opcional (optional part)
- Parte Obrigatória gera resultado com qualidade mínima
- Parte Opcional refina resultado até qualidade máxima
- Podem existir tarefas com
  – apenas parte obrigatória ou
  – apenas parte opcional
- Situações normais: executa as duas partes de cada tarefa
- Sobrecarga: executa apenas a parte obrigatória
- Objetivo é maximizar a qualidade total da aplicação, cumprindo os deadlines

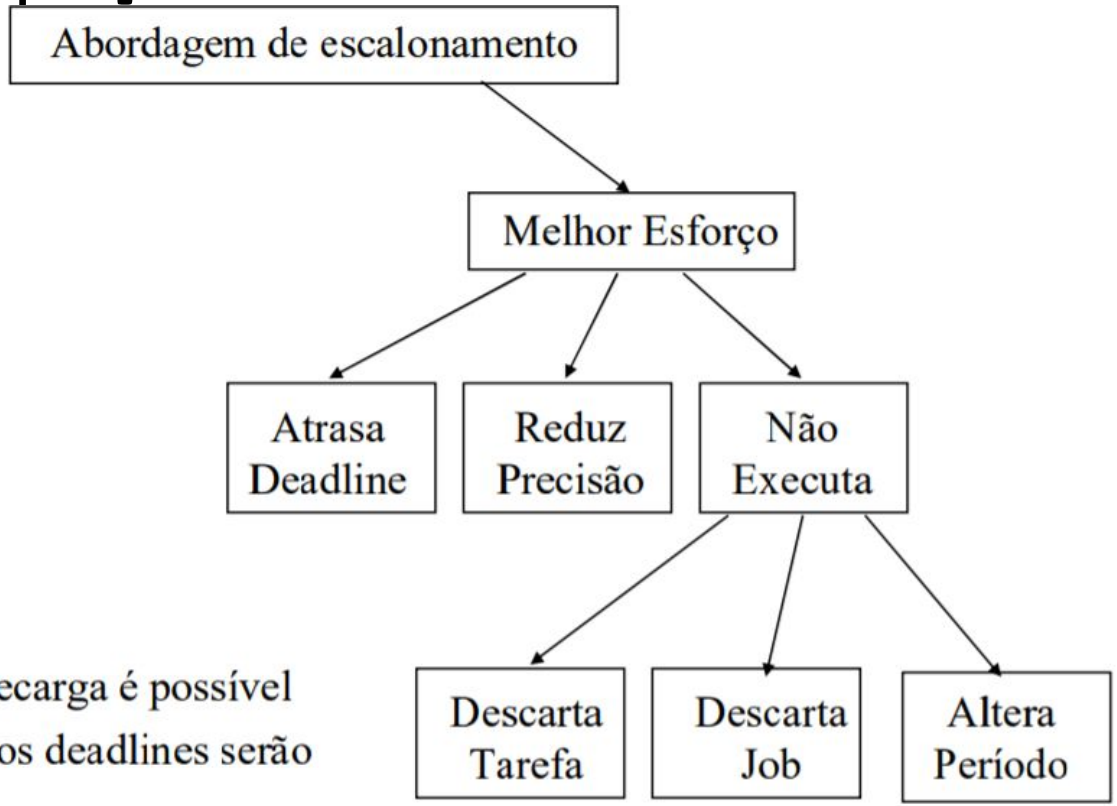
Descarte de Tarefas na Sobrecarga

- Em sobrecarga NÃO EXECUTA algumas tarefas
- As tarefas executadas cumprem o deadline
  – Mais apropriado para tarefas com deadline firm
- Pode cancelar:
  – Ativações individuais (jobs)
  – Tarefas completas
- Objetivo é maximizar o número de tarefas executadas
- Tarefas podem ter importância (peso) diferentes
  – Maximiza o somatório dos pesos das tarefas executadas
- Algumas soluções utilizam um parâmetro de descarte **s**
  – Distância temporal entre ativações descartadas deve ser **s**
- Outras permitem o descarte de até m a cada k ativações

Alteração do Período na Sobrecarga

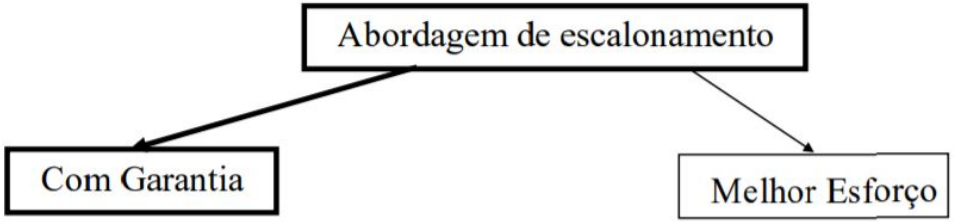
- Em sobrecarga aumenta o período de algumas tarefas
- Objetivo é não perder deadlines através da redução da utilização do processador que cada tarefa representa
- Exemplos:
  – Amostragem de variáveis em laço de controle
  – Taxa de apresentação dos quadros de um vídeo
  – Freqüência da atualização da tela do operador
- Chamado às vezes de
  Rate Modulation

Resumo - Melhor Esforço



- Seja como for:
  - Nesta abordagem a sobrecarga é possível
  - Não existe garantia que os deadlines serão cumpridos

Abordagens com Garantia em Projeto



- Oferece previsibilidade determinista
- Análise feita em projeto
  - Carga precisa ser limitada e conhecida em projeto (Hipótese de Carga)
  - É Suposto um limite para faltas (Hipótese de Faltas)
- Dividida em duas partes:
  - Análise da escalonabilidade (cumprimento dos deadlines)
  - Construção da escala de execução

Abordagens com Garantia em Projeto

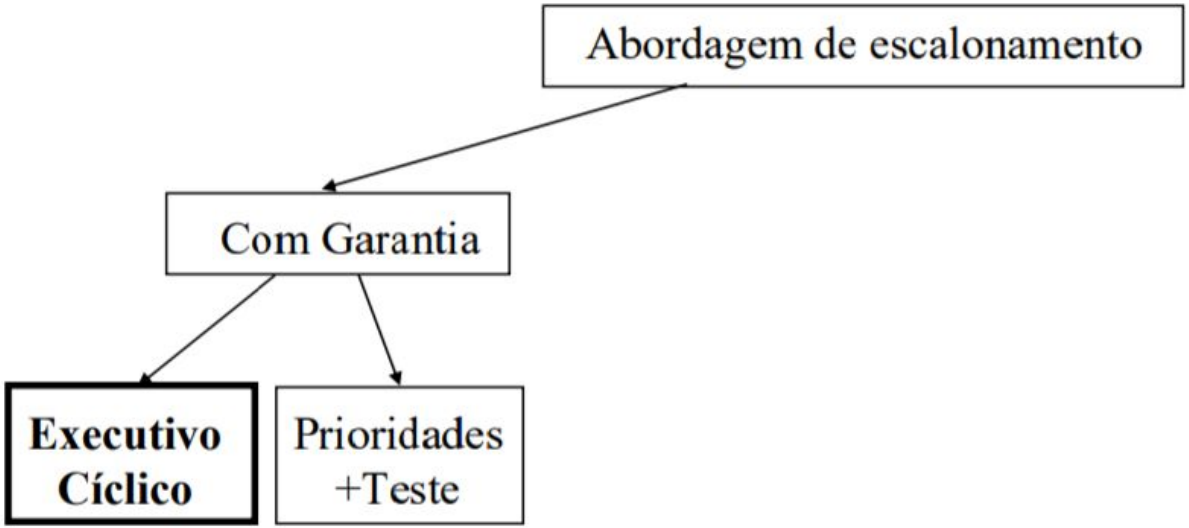
- Necessário conhecer o comportamento do sistema no pior caso, tanto software quanto hardware
- Isto significa
  – Pior fluxo de controle para cada tarefa
  – Pior cenário de sincronização entre tarefas (exclusão mútua, etc)
  – Piores dados de entrada
  – Pior combinação de eventos externos (interrupções, sensores, etc)
  – Pior comportamento das caches no hardware
  – Pior comportamento do processador (pipeline, barramentos, etc)
  – Pior tudo
- Necessário conhecer WCET de cada tarefa, C
  – Worst-case execution time (WCET)
  – Tempo de execução no pior caso

Abordagens com Garantia em Projeto

- Vantagens
  – Determina em projeto que todos os deadlines serão cumpridos
  – Necessário para aplicações críticas
  – Teoria serve de base para abordagens sem garantia
- Desvantagens
  – Necessário conhecer exatamente a carga
  – Necessário reservar recursos para o pior caso
  – Difícil determinar o pior caso em soluções COTS (commercial off-the-shelf)
  – Gera enorme sub-utilização de recursos

Abordagens com Garantia em Projeto

- Existem duas formas de obter garantia em projeto
  - Executivo cíclico
  - Prioridades + Teste de escalonabilidade

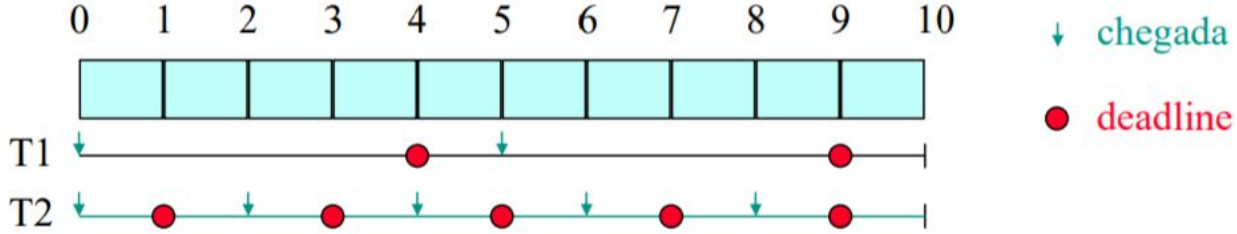


Executivo Cíclico

- Todo o trabalho de escalonamento é feito em projeto
- Resultado é uma grade de execução (time grid)
- Grade determina qual tarefa executa quando
- Garantia obtida através de uma simples inspeção da escala
- Durante a execução:
  – Pequeno programa lê a grade e dispara a tarefa aproprida
  – Quando a grade termina ela é novamente repetida
- Vantagem: Comportamento completamente conhecido
- Desvantagem: Escalonamento muito rígido, tamanho da grade
- Muito usado em aplicações embutidas (Embedded Systems)

Executivo Cíclico

- Restrições devem ser observadas na construção da grade
  - Período, tempo máximo de computação
  - Precedências, exclusões, etc
- Escalonamento é em geral não preemptivo
  - Facilita lidar com recursos compartilhados
  - Mais difícil achar uma escala satisfatória
- Exemplo:
  - T1: P1=5 C1=2 D1=4
  - T2: P2=2 C2=1 D2=1

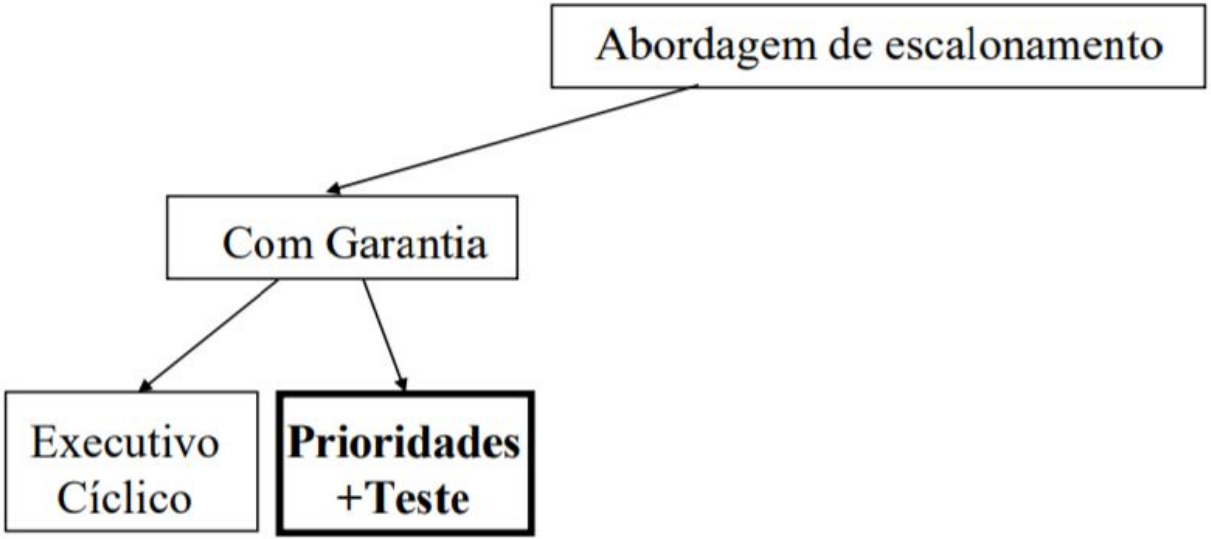


Executivo Cíclico

- Vantagens
  – É a forma tradicional para sistemas críticos
  – Comportamento completamente conhecido
  – Fácil detectar qualquer falha de projeto
  – Apropriado para tarefas periódicas, as quais são maioria nos sistemas críticos
- Desvantagens
  – Não lida bem com tarefas que não são periódicas
  – Tabela pode ficar grande, caso períodos não sejam múltiplos entre si
  – No caso de WCET mal calculado, o que fazer ?
  – Tarefas muito longas precisam ser quebradas em várias sub-tarefas

Abordagens com Garantia em Projeto

- Existem duas formas de obter garantia em projeto
  - Executivo cíclico
  - Prioridades + Teste de escalonabilidade



Prioridades + Teste de Escalonabilidade

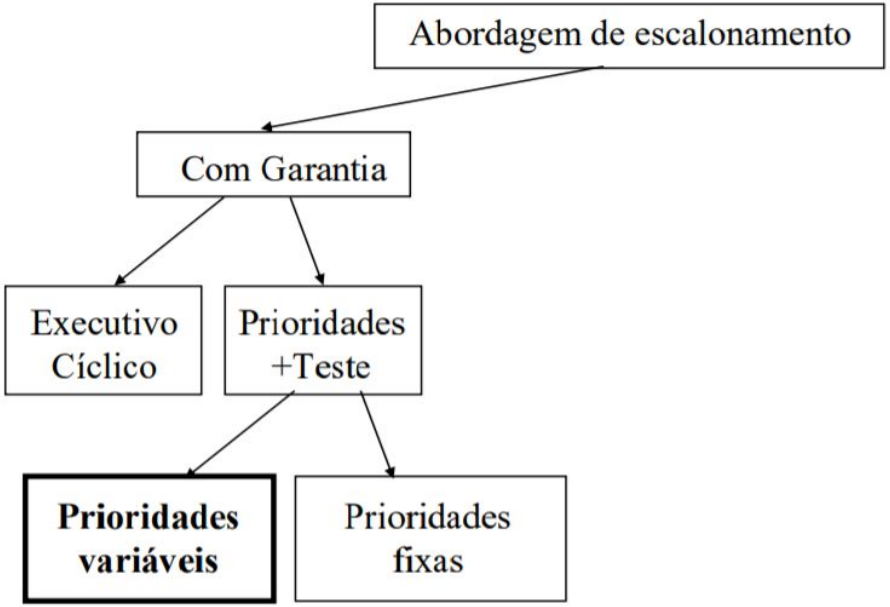
- Cada tarefa recebe uma prioridade
- Escalonamento em geral é preemptivo
- Teste realizado antes da execução determina escalonabilidade
  – Teste considera como são as tarefas (modelo de tarefas)
  - Periódica, esporádica, D<=P, bloqueios, etc
  – Teste considera forma como prioridades são atribuídas
  – Validade do teste é demonstrada como teorema
  – Complexidade do teste depende do modelo de tarefas
- Na execução:
  – Escalonador dispara as tarefas conforme as prioridades

Prioridades + Teste de Escalonabilidade

- Vantagens:
  – Suporta tarefas esporádicas com facilidade
  – Suporta tarefas aperiódicas com facilidade
  – Não é necessário gerar grade de tempo
  – Oferece determinismo para os deadlines
  – Comportamento no caso de falhas pode ser gerenciado
- Desvantagens:
  – Existem testes apenas para alguns modelos de tarefas
  – É difícil criar novos testes (significa provar um teorema)
  – Não oferece determinismo para a escala de execução
  – Mais difícil detectar faltas
- Usado em aplicações que exigem garantia mas também requerem flexibilidade na escala de execução

Prioridades + Teste de Escalonabilidade

- Existem dois tipos básicos de prioridades
  - Prioridades variáveis
  - Prioridades fixas

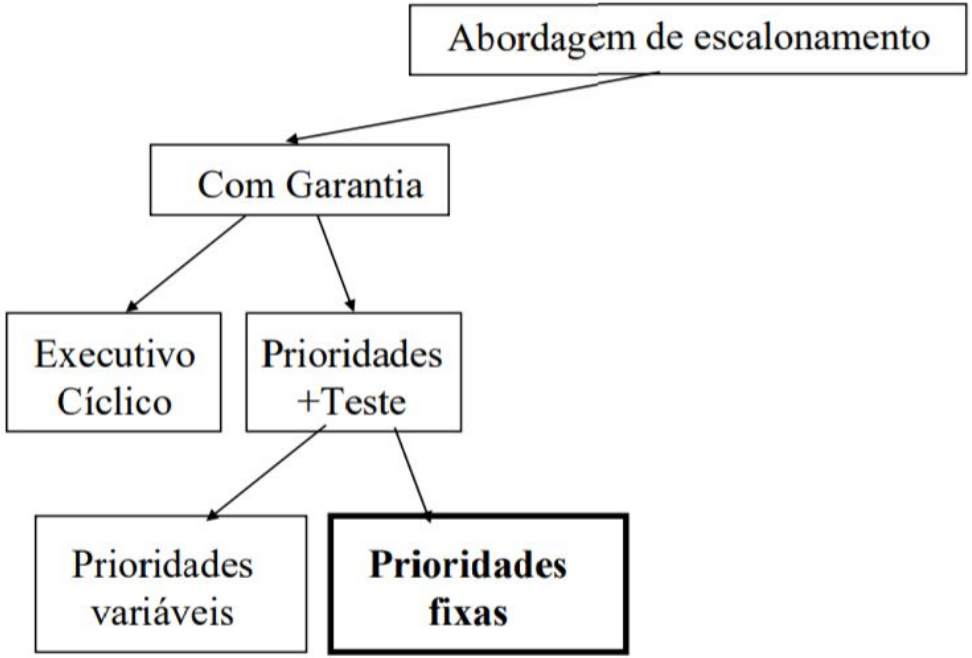


Prioridades Variáveis

- Earliest Deadline First – EDF
  – Prioridade mais alta para a tarefa com deadline absoluto menor
  – Prioridade variável
- Least Laxity First – LLF
  – Prioridade mais alta para a tarefa com menor folga
  – Folga = deadline absoluto – agora – computação que falta
  – Prioridade variável

Prioridades + Teste de Escalonabilidade

- Existem dois tipos básicos de prioridades
  - Prioridades variáveis
  - Prioridades fixas



Prioridades fixas

- Rate Monotonic – RM
  - Prioridade mais alta para a tarefa com período menor
  - Prioridade fixa
- Deadline Monotonic - DM
  - Prioridade mais alta para a tarefa com deadline relativo menor
  - Prioridade fixa
  - Igual ao RM quando D = P

Prioridades + Teste de Escalonabilidade

- Apenas atribuir prioridades não basta
- É necessário um teste de escalonabilidade
- Testes podem ser classificados de diferentes formas
- Quando a cobertura
  – Teste suficiente mas não necessário (muito rigoroso)
  – Teste suficiente e necessário (exato)
  – Teste necessário mas não suficiente (muito frágil)
- Quanto ao tipo
  – Baseado em utilização
  – Baseado em tempo de resposta

Teste de Escalonabilidade: Utilização

- Utilização de uma tarefa:
  – Tempo máximo de computação dividido pelo período
  – Exemplo: T1 tem C1=12 e P1=50, então U1 = 12 / 50 = 0.24
- Utilização do sistema
  – Somatório da utilização de todas as tarefas
- Dado
  – Um modelo de tarefas
  – Uma política de atribuição de prioridades
- Existe um limiar de utilização para o processador, de tal sorte que:
  – Se a utilização do processador for menor que o limiar
  – Então jamais um deadline será perdido
- Limiar demonstrado como teorema

Teste de Escalonabilidade: Tempo de Resposta

- Dado
  – Um modelo de tarefas
  – Uma política de atribuição de prioridades
- Para cada tarefa Ti
  – Identifica o pior padrão de chegadas possível para Ti
  – Constrói o diagrama de tempo considerando o pior caso para tudo
  – Calcula o tempo de resposta da tarefa neste caso
  – Este é o tempo de resposta no pior caso Ri da tarefa Ti
- Se Ri <= Di
  – Então jamais Ti perderá um deadline
- Dificuldade: Calcular Ri quando o sistema é complexo e a escala de execução não determinista

Resumo sobre Escalonamento em Tempo real

- Existe a necessidade de diferentes abordagens para o escalonamento tempo real
- Principal classificação é com respeito a garantia
- Abordagens com Melhor Esforço
  – Perda de deadlines
  – Redução da Precisão
  – Descarte de tarefas, de jobs, ajusta período
- Abordagens com Garantia em Projeto
  – Executivo Cíclico
  – Prioridades + Teste de Escalonabilidade

Escalonamento: Tarefas Aperiódicas

- Muitas aplicações de tempo real requerem tanto processos periódicos quanto aperiódicos e esporádicos.
- As tarefas periódicas são dirigidas por tempo e executam atividades de controle que são críticas, ou seja, com restrições temporais rígidas e que devem ser ativadas em taxas regulares.

- Tarefas aperiódicas são, por outro lado, dirigidas por eventos e podem ter requisitos temporais rígidos, brandos ou então sem restrições temporais.

Escalonamento: Tarefas Aperiódicas

- A garantia em tempo de projeto de tarefas aperiódicas pode ser conseguida assumindo uma taxa máxima de chegada para os eventos críticos.
- Isto quer dizer que tarefas aperiódicas associadas com eventos críticos são caracterizadas por um intervalo mínimo entre suas ativações, limitando a carga aperiódica.
- Estas tarefas são identificadas como tarefas esporádicas, possuem um comportamento temporal determinista, facilitando assim a obtenção de garantias em tempo de projeto.

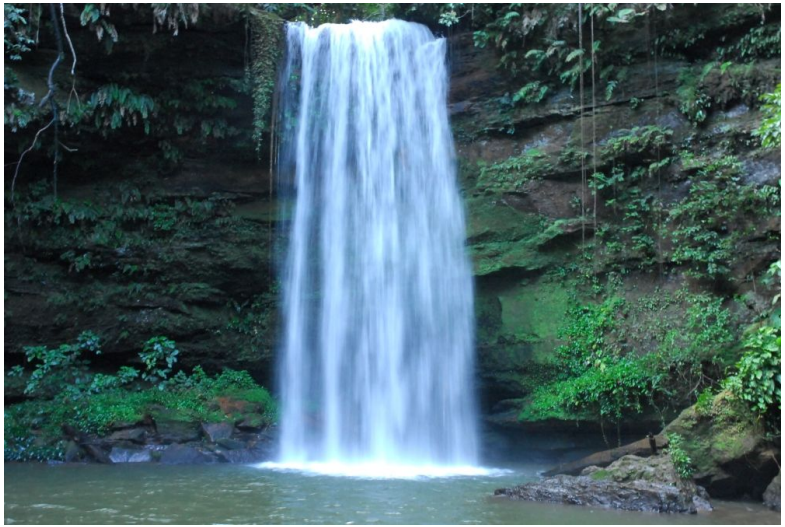
Escalonamento: Tarefas Aperiódicas, Periódicas e Esporádicas

- Assumimos que o sistema mantém filas de prioridade a seguir:
  - tarefas periódicas
    - Resolvida com estratégia de prioridades
  - tarefas aperiódicas
    - Resolvidas intervalo mínimo entre ocorrências
  - tarefas esporádicas
    - Resolvida com critério de aceitação ou não da tarefa dependendo do estado do sistema

Streaming

Streaming

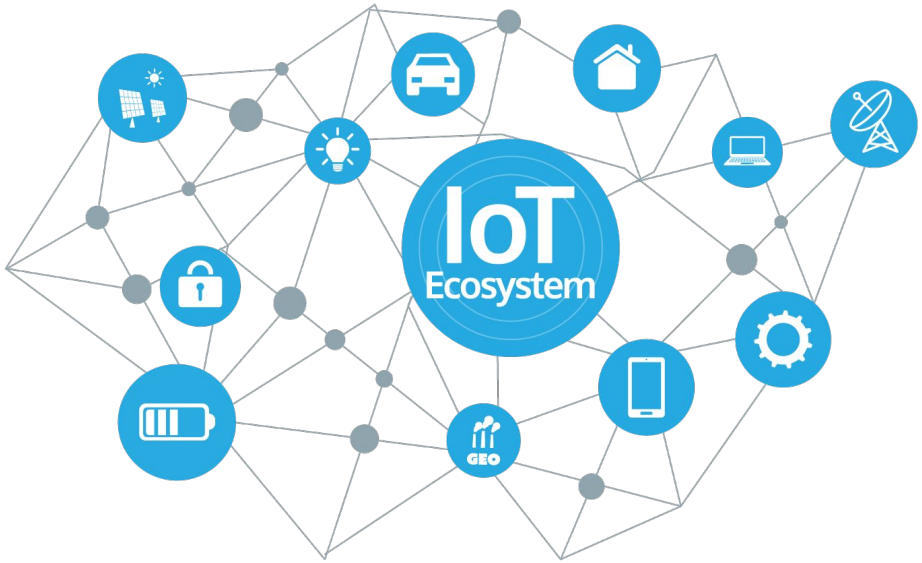
- Fluxo contínuo (contínuo ≠ constante).



Streaming de dados

- Fluxo contínuo de dados.

Spotify®



Streaming de dados: Exemplos

- Sensores (IoT)
- Tráfego de rede
- Registros de call center
- Tendências em redes sociais
- Serviços de áudio e vídeo
- Análise de log
- Estatísticas de sites web

Tipos de streaming de dados

Dados de texto: web, log

Dados relacionais: tabelas, transações

Dados semi-estruturados: XML, json

Dados em grafo: redes sociais

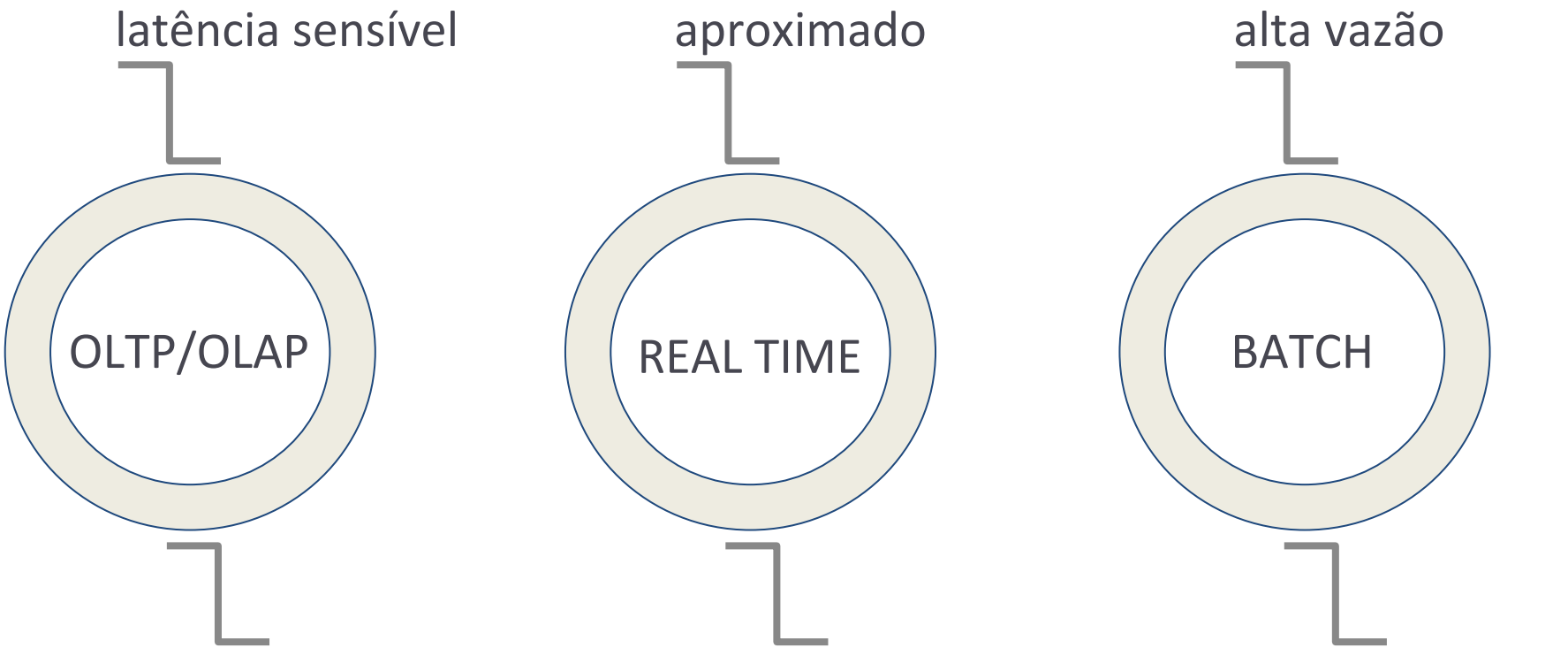
Dados de mobilidade: coordenadas geográficas x tempo

Etc.

O que é tempo real?

Milissegundos, segundos, minutos?

O que é Tempo Real?



O que é Tempo Real?

REAL TIME TRENDS

Emerging break out trends in Twitter (in the form #hashtags)

REAL TIME CONVERSATIONS

Real time sports conversations related with a topic (recent goal or touchdown)

REAL TIME RECOMMENDATIONS

Real time product recommendations based on your behavior & profile

REAL TIME SEARCH

Real time search of tweets with a budget < 200 ms

Fonte: Real-Time Analytics with Apache Storm - https://www.udacity.com/course/ud381

Problemas em streaming

1. Como **obter** os dados a partir de várias fontes em um cluster em tempo real?
2. Como **processar** esses dados?

Apache Kafka

Apache Kafka - o que é

- *O que é o Apache Kafka?*
- *“Apache Kafka é uma plataforma distribuída de mensagens e streaming”.*

Apache Kafka

- Você **produz** uma mensagem.
- Essa mensagem é **anexada** em um tópico.
- Você então **consome** essa mensagem.

Por que usar?

- *“Se você quer mover e transformar um grande volume de dados em tempo real entre diferentes sistemas, então Apache Kafka pode ser exatamente o que você precisa”.*
- *Todo mundo usa, inclusive Microsoft*
- *Possui integração com Python e outras linguagens*

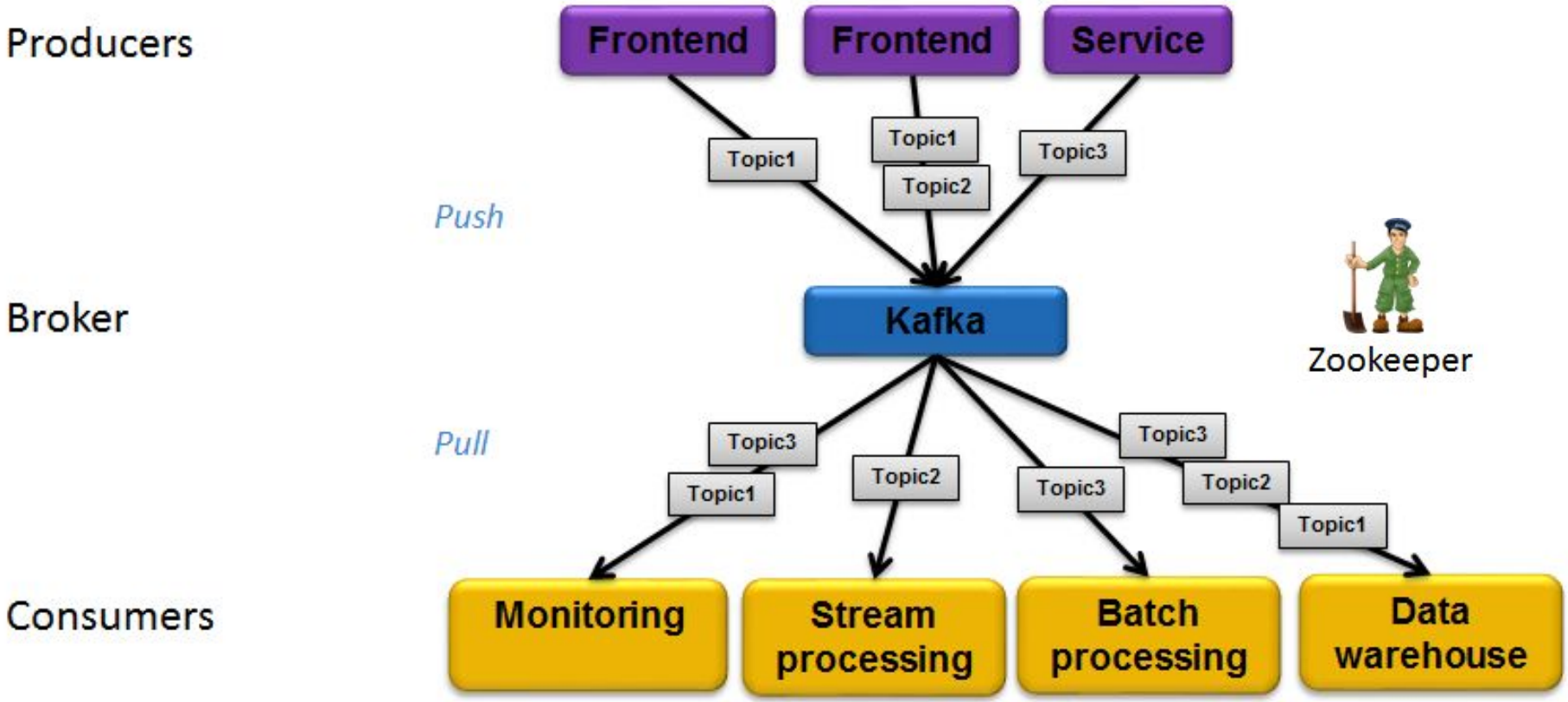
Apache Kafka

- Sistema de mensagens
  - Distribuído
  - Com alta vazão (*throughput*)
    - De geração (publicação) e leitura (sub-inscrição)
- Principais casos de uso:
  - Agregação de log
  - Processamento em tempo real
  - Monitoramento

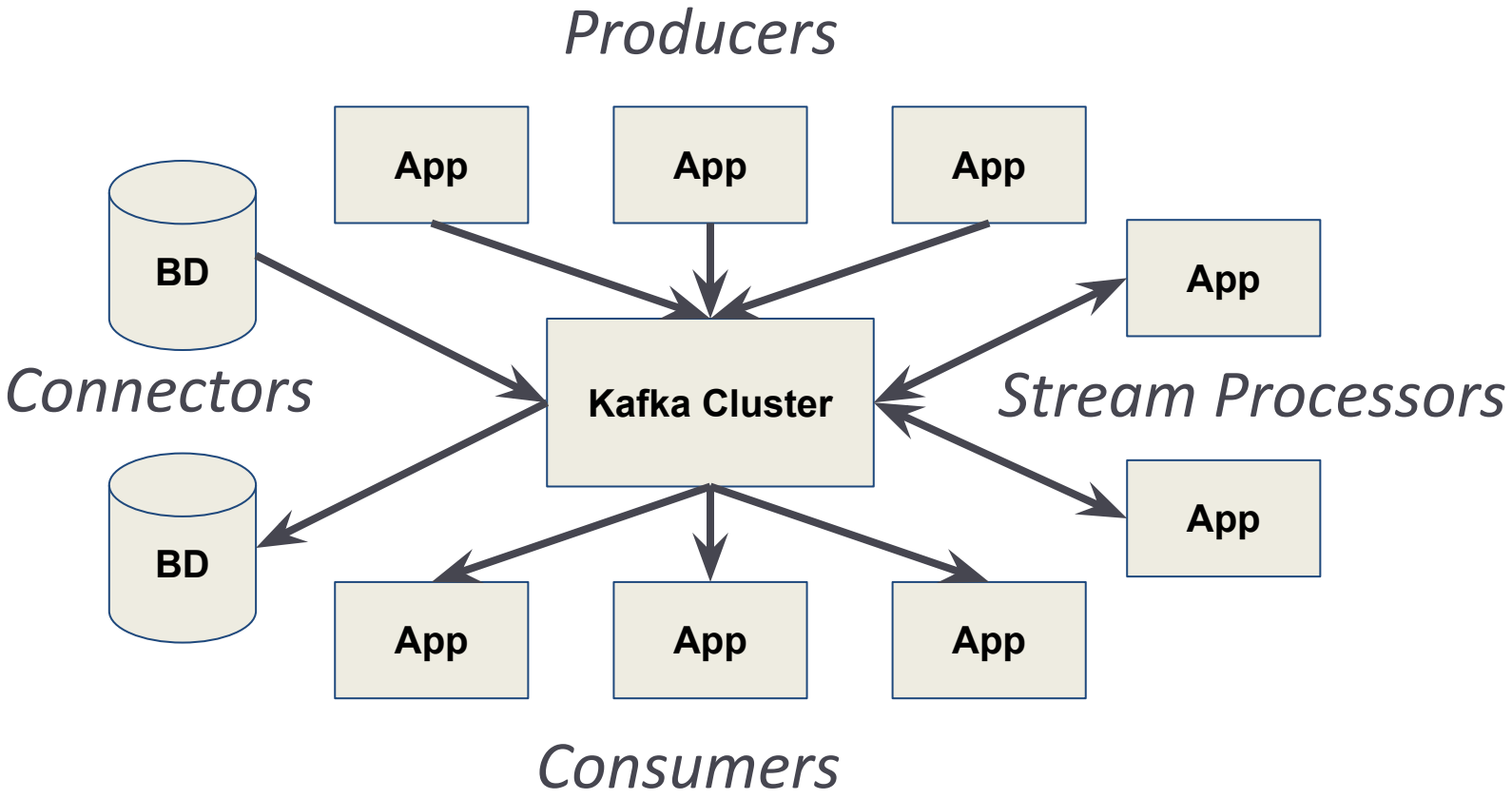
Apache Kafka

- Originalmente desenvolvido pelo LinkedIn.
- Implementado em scala/Java.
- *Producers* & *Consumers.*
- Mensagens são associadas a tópicos, os quais representam um stream específico.
  - Logs web
  - Dados de sensores
- *Consumers* se inscrevem em um ou mais tópicos.

Kafka: conceitos

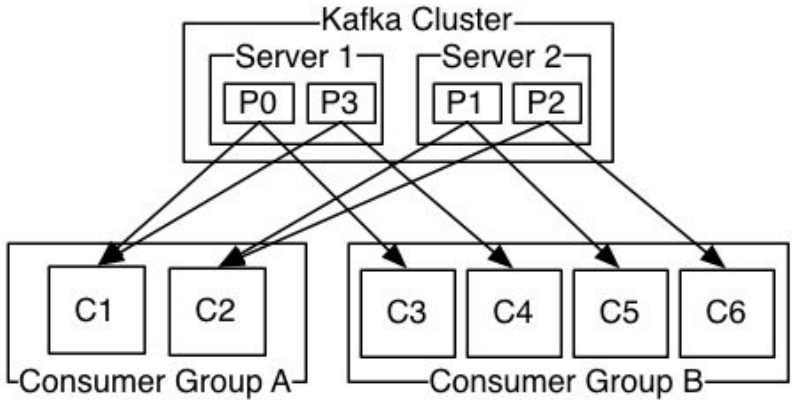


Kafka: arquitetura



Kafka: escalabilidade

- Kafka pode ser distribuído entre muitos processos em vários servidores.
- *Consumers* também podem ser distribuídos.
- Tolerante a falhas.



Fonte: https://kafka.apache.org/intro.html

Kafka: pontos a considerar

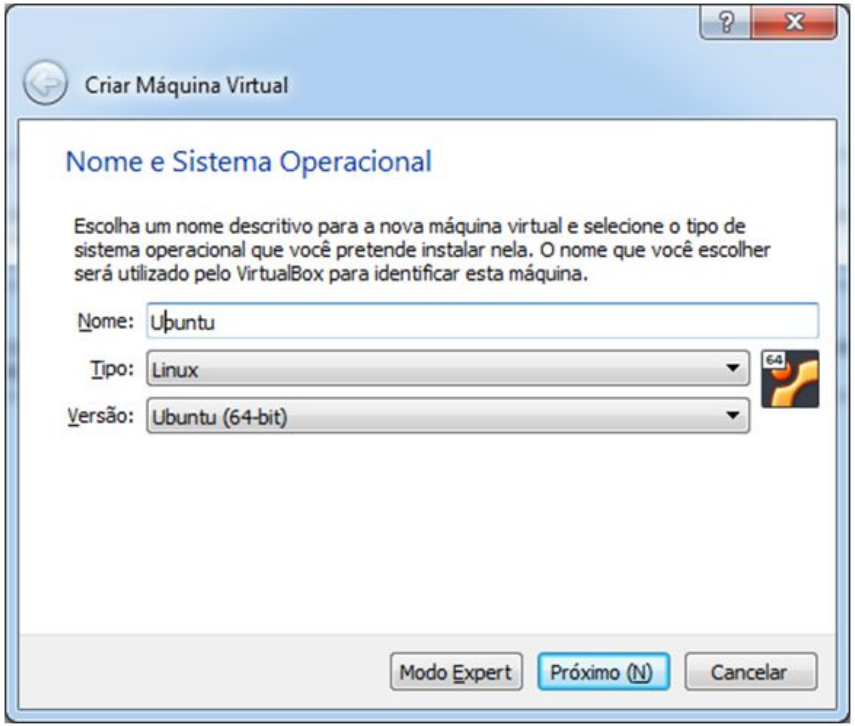
- Simples sistema de mensagens, não de processamento.
- Não vive sem o **Zookeeper**, o qual pode se tornar um gargalo quando o número de tópicos/partições é muito grande (>>10000).
- Não otimizado para latências de milissegundos.

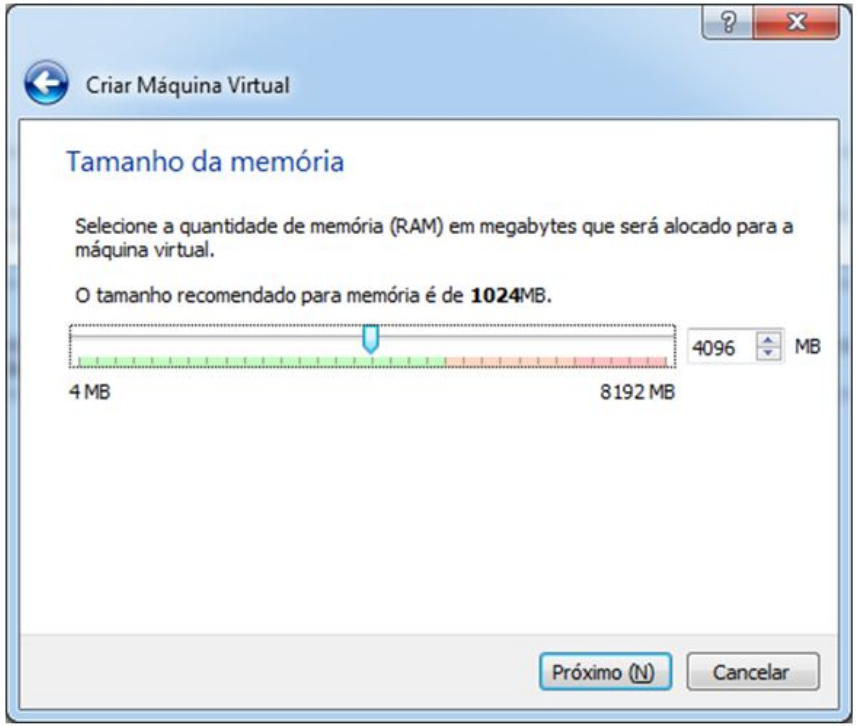

Prática

Prática dia 2

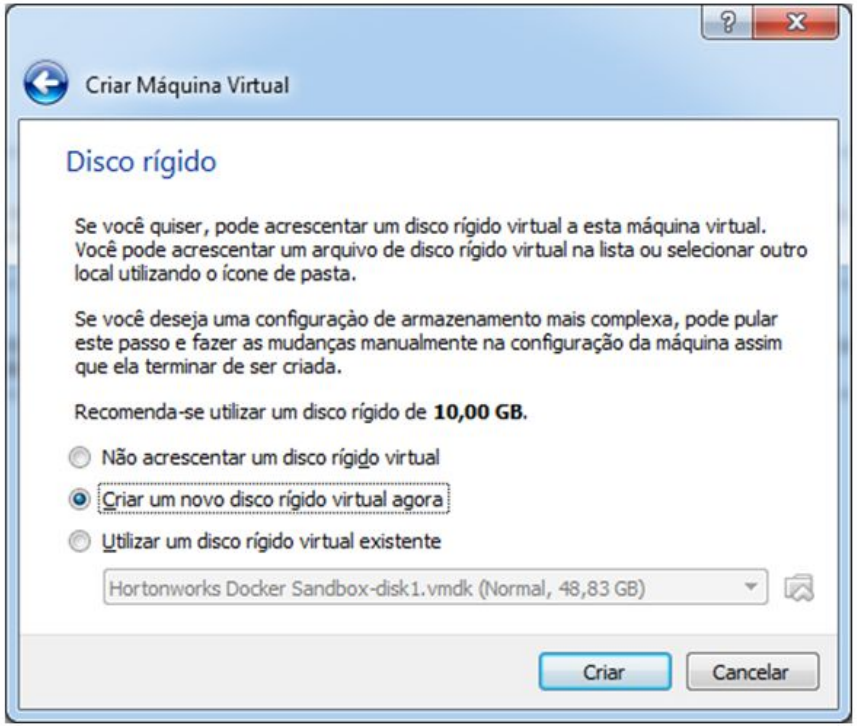
- Instalar Oracle VirtualBox
- Criar instância para Linux Ubuntu
- Instalar Ubuntu
- Instalar Apache Kafka
  - Pré-requisito Zookeeper
- Tarefa 1 - Hello World Kafka
  - Criar tópico
  - Gerar dados no tópico
  - Consumir dados
- Tarefa 2 - produtor / consumidor em Python
  - instalar o python
  - instalar liblioteca kafka-python
  - executar código producer_consumer.py

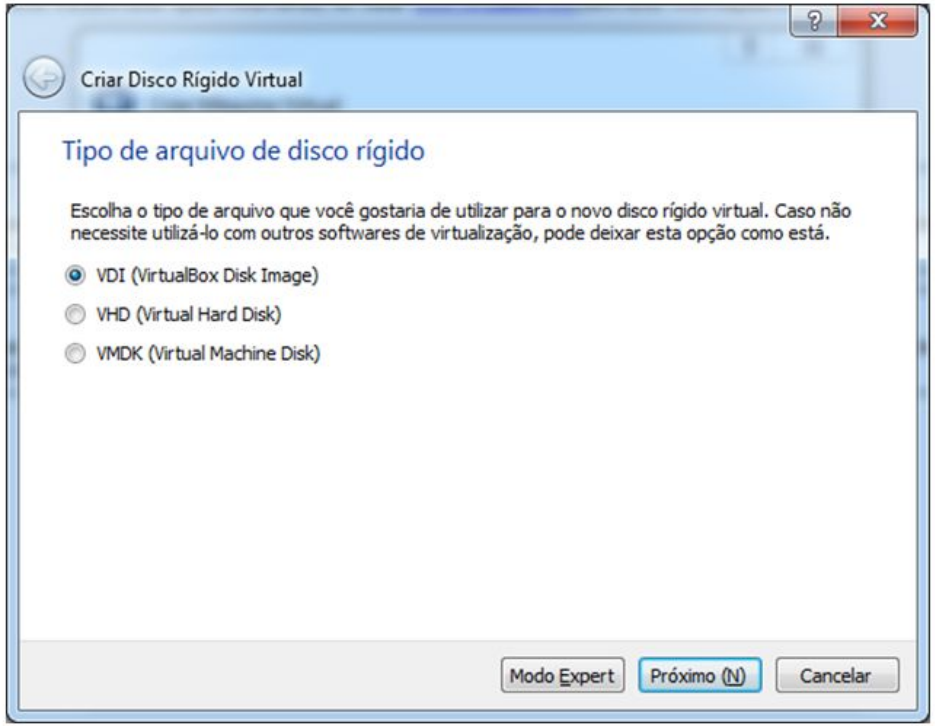
Criar nova máquina virtual



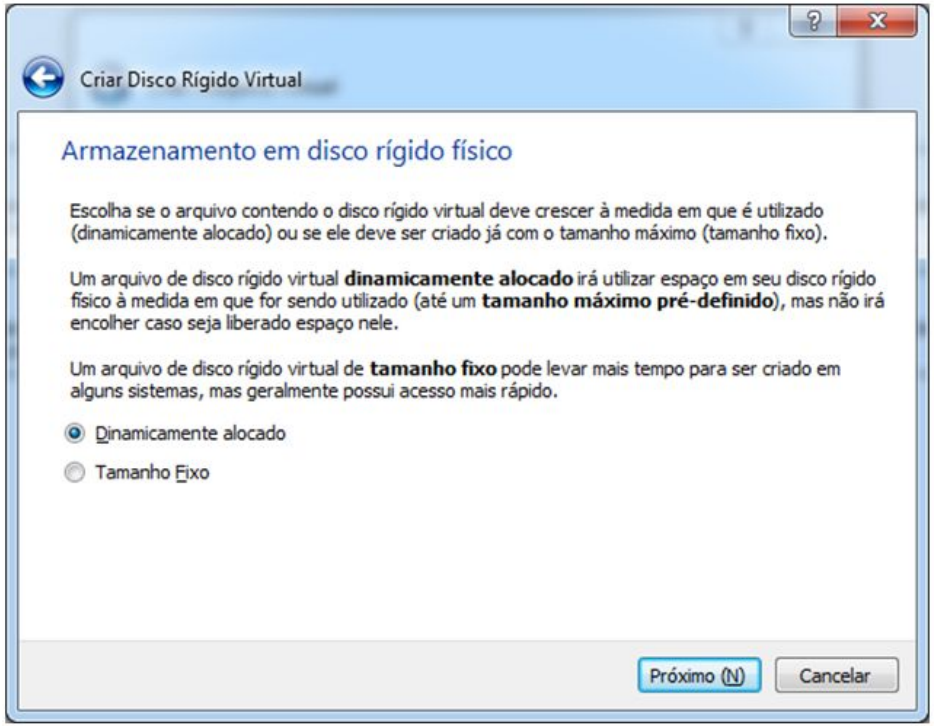


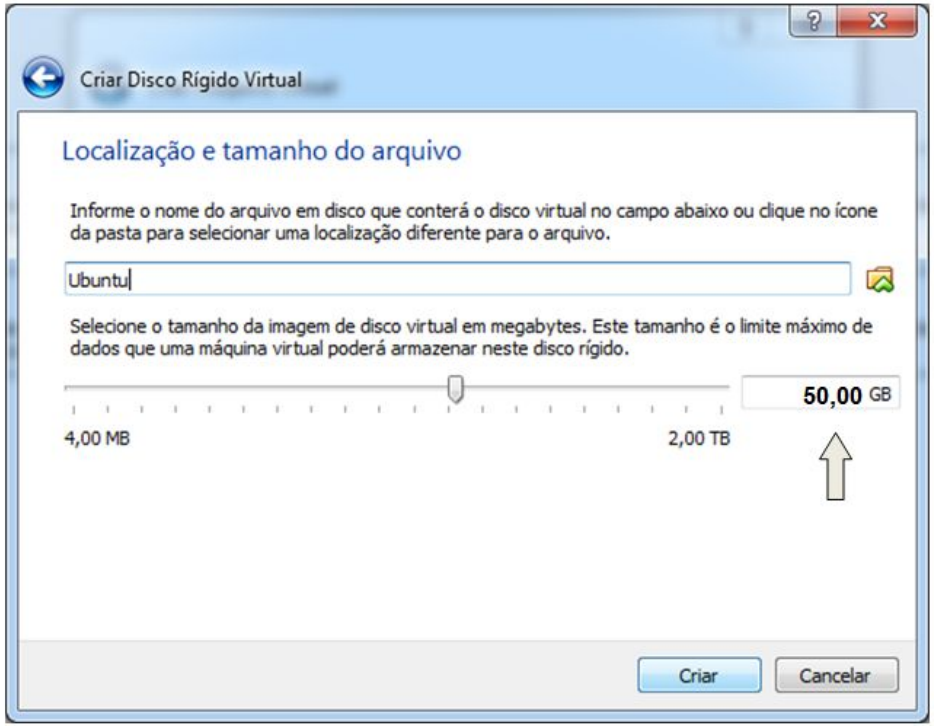
Criar nova máquina virtual



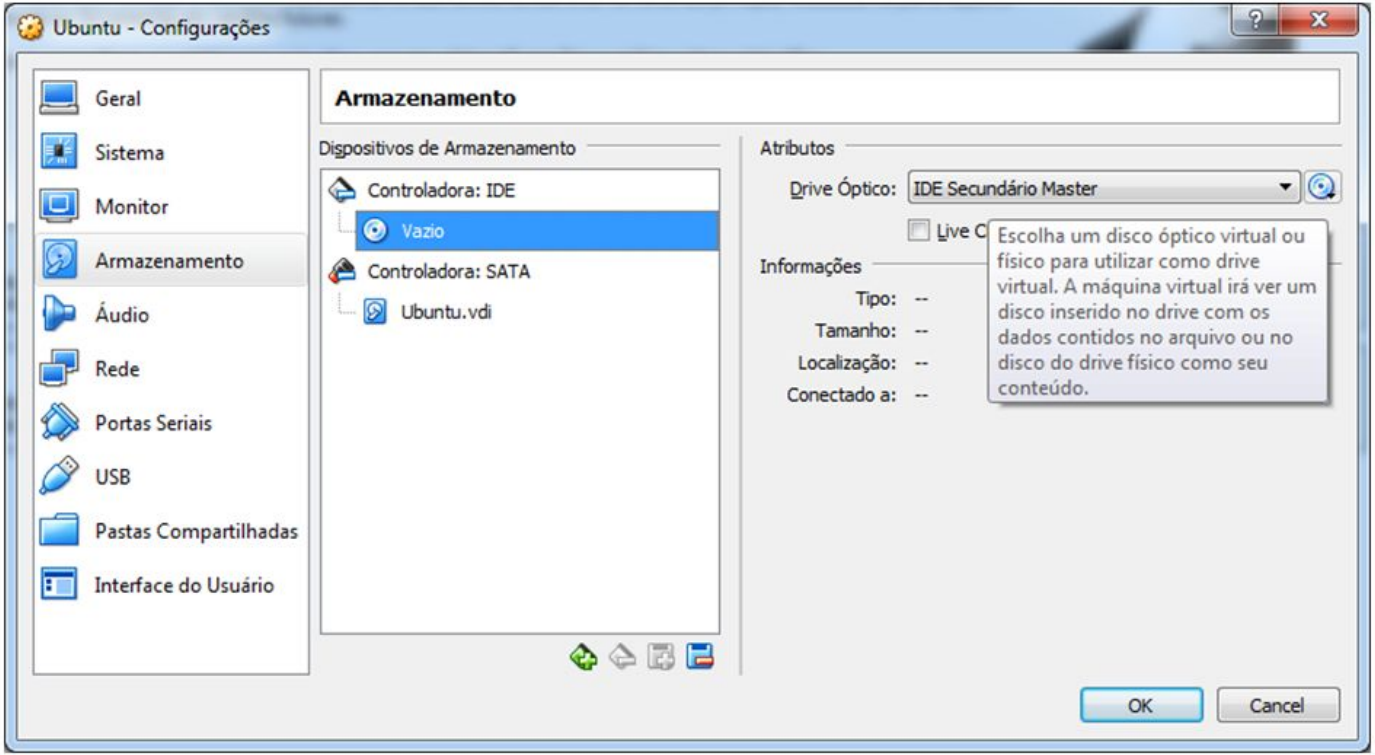


Criar nova máquina virtual

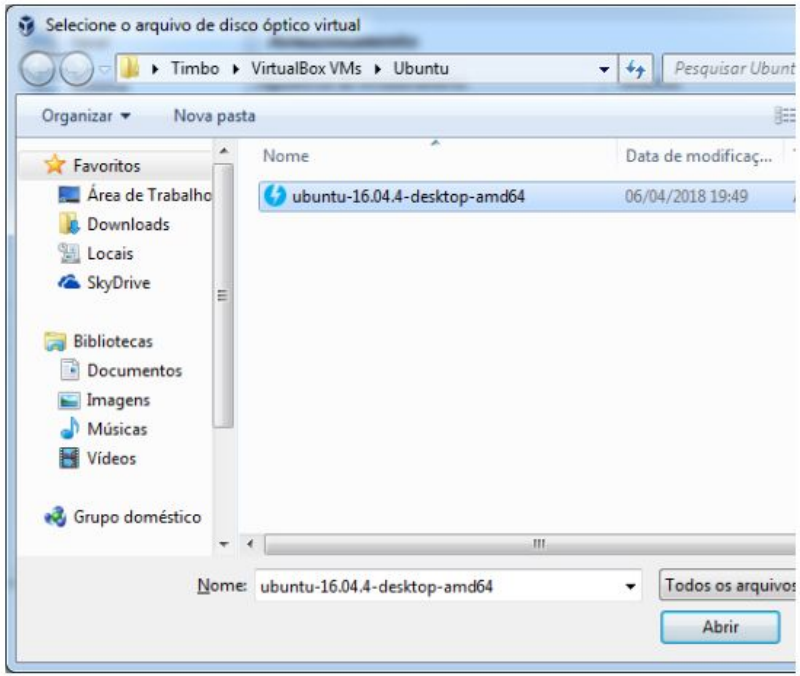


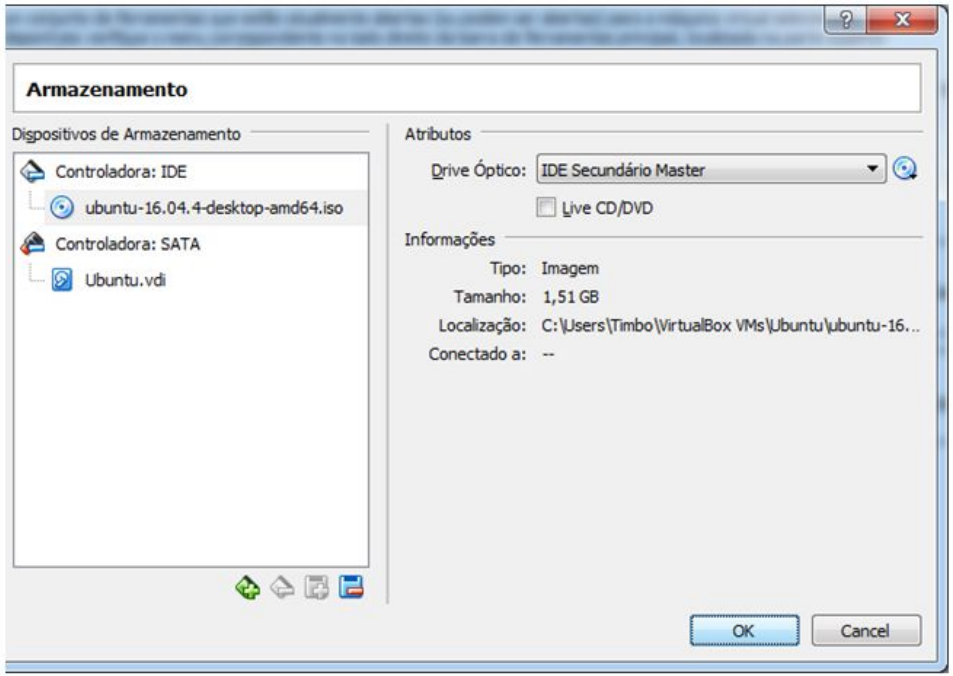


Configurações -> Armazenamento

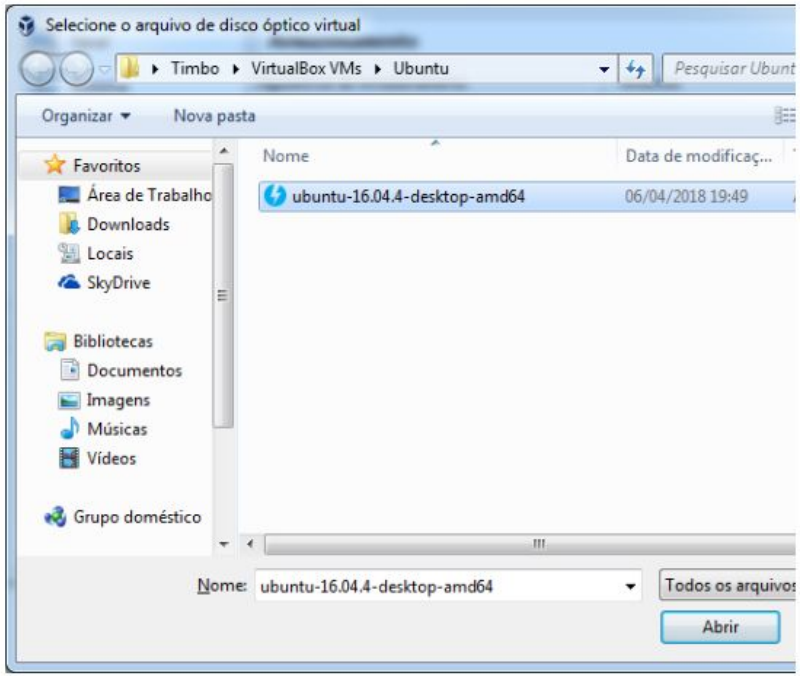


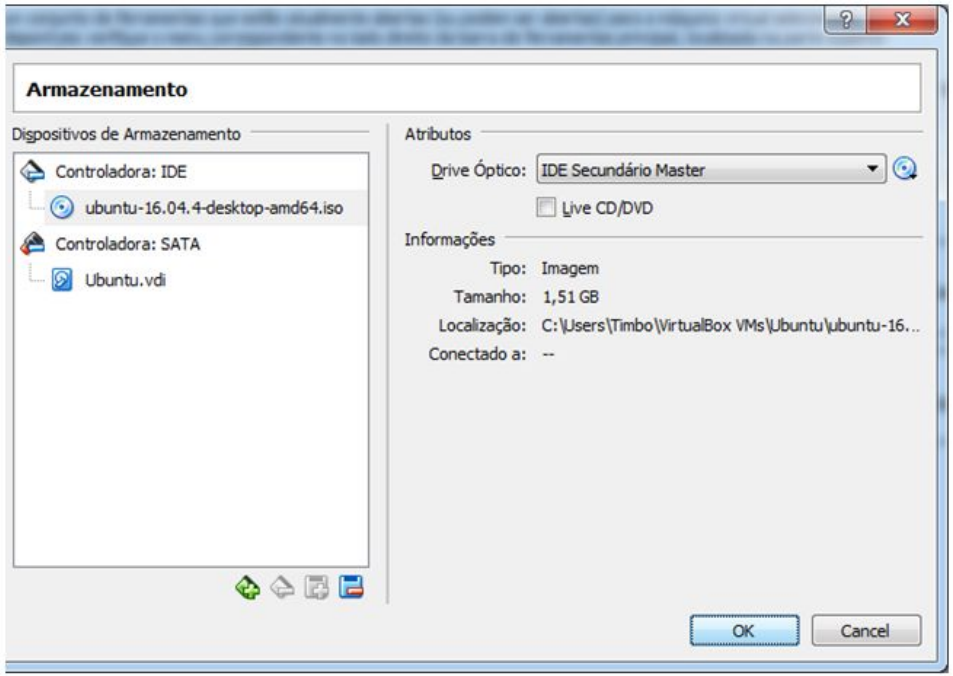
Selecionar arquivo de disco óptico virtual



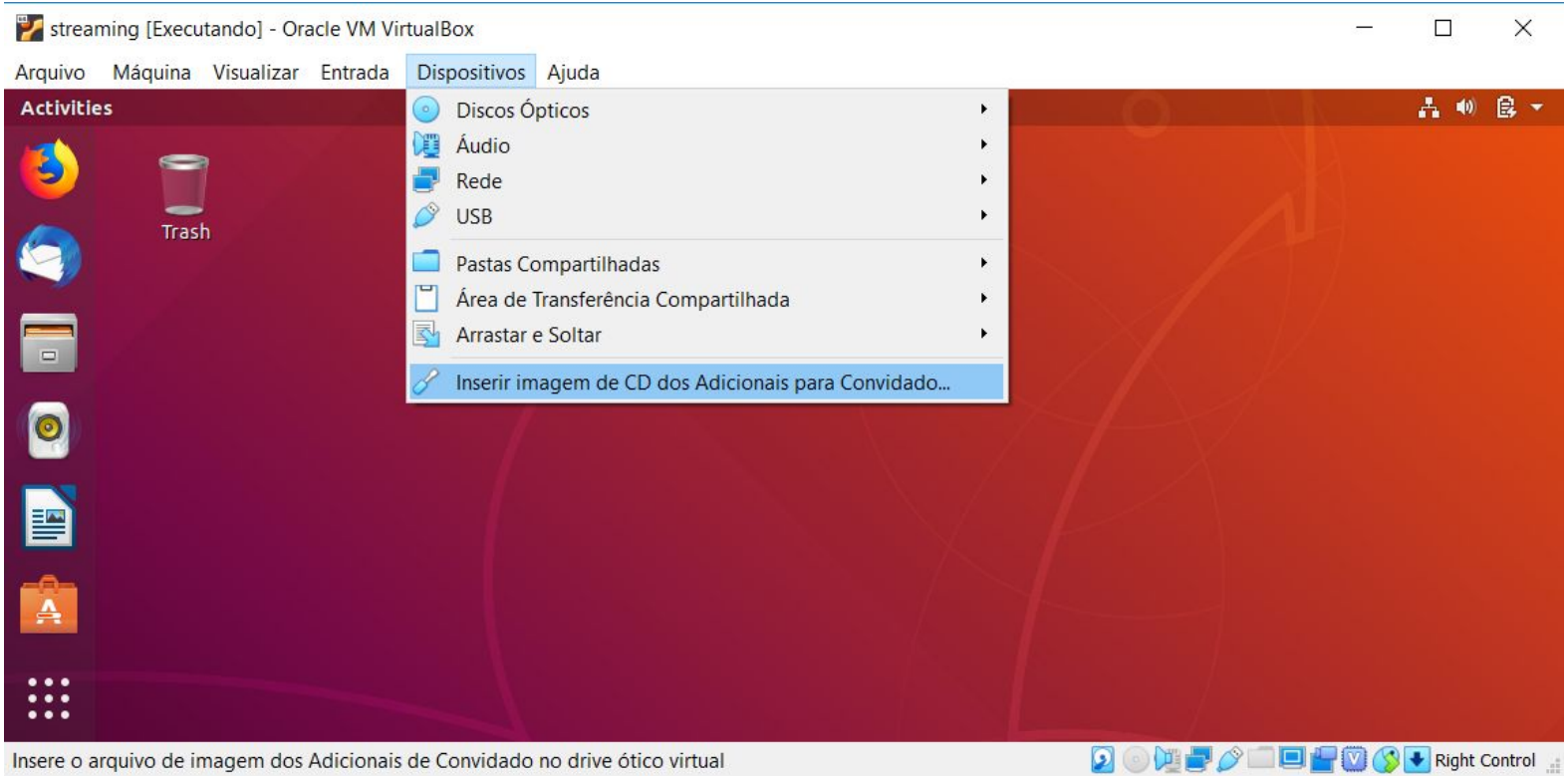


Selecionar arquivo de disco óptico virtual





Configurar tela cheia



Instalar KAFKA

Primeiros passos

1. Acessar máquina virtual
2. Acessar terminal
3. Atualizar Ubuntu 16.04
   - `sudo apt-get update -y`
   - `sudo apt-get upgrade -y`
4. Esperar alguns minutos

Instalar o Java

```
sudo add-apt-repository -y ppa:webupd8team/java
```

```
gpg: keyring `/tmp/tmpkjrm4mnm/secring.gpg' created
gpg: keyring `/tmp/tmpkjrm4mnm/pubring.gpg' created
gpg: requesting key EEA14886 from hkp server keyserver.ubuntu.com
gpg: /tmp/tmpkjrm4mnm/trustdb.gpg: trustdb created
gpg: key EEA14886: public key "Launchpad VLC" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
OK
```

```
sudo apt-get update
sudo apt-get install oracle-java8-installer -y
sudo java -version
```

```
java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b17, mixed mode)
```

Instalar o ZooKeeper

- `sudo apt-get install zookeeperd`

Para testar a instalação:

- `netstat -ant | grep :2181`

```
tcp6       0      0 :::2181                 :::*                    LISTEN
```

Instalar o Kafka

Baixar o kafka:

- `wget https://archive.apache.org/dist/kafka/0.10.0.1/kafka_2.10-0.10.0.1.tgz`

Criar um diretório para a instalação do kafka:

- `sudo mkdir kafka`

Extrair os dados para o diretório:

- `sudo tar -xvf kafka_2.10-0.10.0.1.tgz -C kafka/`

Inicializar o Servidor Kafka

- cd kafka/kafka_2.10-0.10.0.1/bin
- sudo ./kafka-server-start.sh ~/kafka/kafka_2.10-0.10.0.1/config/server.properties

```
[2016-08-22 21:43:48,279] WARN No meta.properties file under dir /tmp/kafka-logs/meta.properties
(kafka.server.BrokerMetadataCheckpoint)
[2016-08-22 21:43:48,516] INFO Kafka version : 0.10.0.1 (org.apache.kafka.common.utils.AppInfoParser)
[2016-08-22 21:43:48,525] INFO Kafka commitId : a7a17cdec9eaa6c5 (org.apache.kafka.common.utils.AppInfoParser)
[2016-08-22 21:43:48,527] INFO [Kafka Server 0], started (kafka.server.KafkaServer)
[2016-08-22 21:43:48,555] INFO New leader is 0 (kafka.server.ZookeeperLeaderElector$LeaderChangeListener)
```

Criar novo tópico

7. Inicializar outra janela de terminal
8. Acessar diretório KAFKA
   - `cd kafka/kafka_2.10-0.10.0.1/bin`
9. Criar e listar um novo tópico
   - `./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic any`
   - `./kafka-topics.sh --list --zookeeper localhost:2181`

Gerar dados de um tópico

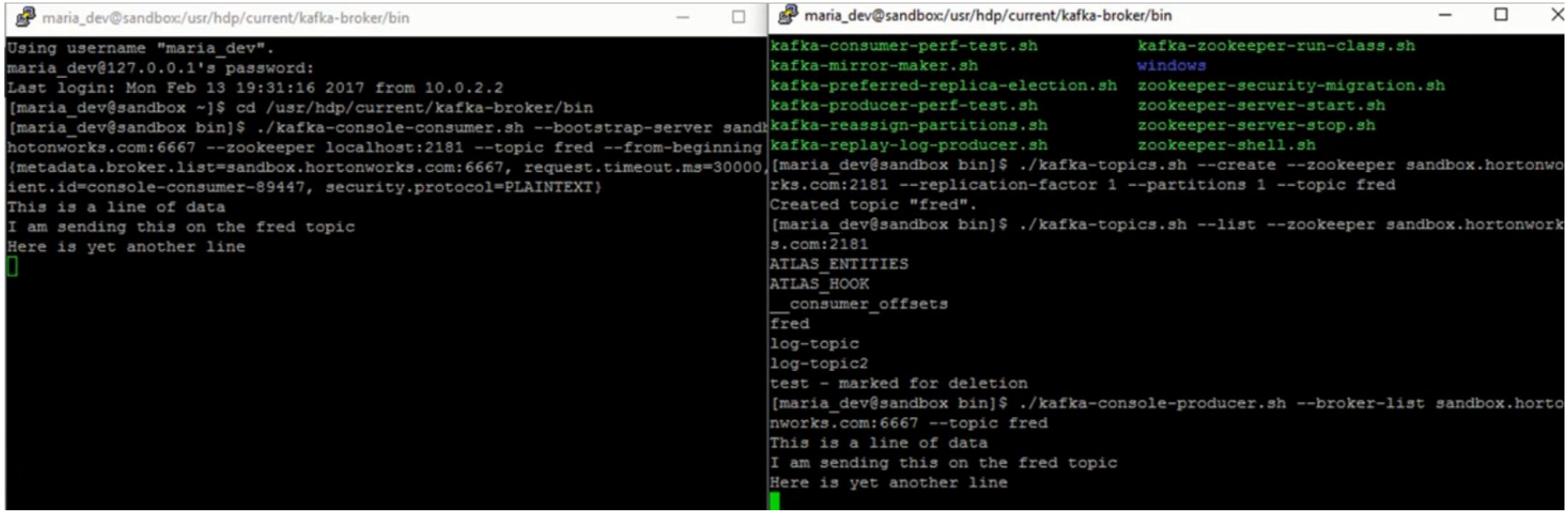
8. Executar o *producer*
   - `./kafka-console-producer.sh --broker-list localhost:9092 --topic any`

Consumir dados de um tópico

9. Inicializar outra janela de terminal
10. Acessar diretório KAFKA
    - `cd kafka/kafka_2.10-0.10.0.1/bin`
11. Obter os dados de um determinado tópico

```
./kafka-console-consumer.sh --zookeeper
localhost:2181 --bootstrap-server localhost:9092
--topic any --from-beginning
```

Produtor x Consumidor de dados



Para sair: `ctrl+C`

KAFKA - Atividade 1

Atividade 1

1. Crie um tópico com o nome de vocês;
2. Liste os tópicos e verifique se o seu foi criado;
3. Gere dados para o tópico criado.

KAFKA-PYTHON - Atividade 2

Produtor/consumidor

1. Baixar código python produtor/consumidor

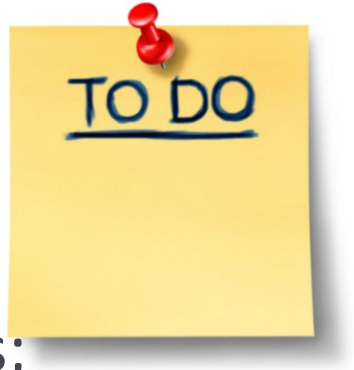
Terminal 2

```
wget https://raw.githubusercontent.com/antoniomralmeida/streaming/master/producer_consumer.py
```

2. Instalar dependência do kafka no python
   - `sudo apt-get install python-pip`
   - `pip install kafka-python`

2. Rodar script
   - `python producer_consumer.py`

Atividade 2 (vale pt)



1. Imprima apenas o conteúdo da tupla;
2. Gere dados de quatro producers simultâneos;
3. Aumente a frequência de geração das tuplas (geração mais rápida);
4. Filtre e imprima apenas por tuplas que possuem valores de peso maiores que 80;
5. Filtre e imprima apenas por tuplas que possuem valores de IMC acima de 35 (*IMC = peso/altura²*).

Fim