# GABDI: Deliverable 01

UFCG

# Sumário

- Objetivos
- Requisitos
- Arquitetura LAMBDA
- Prova de Conceito: WikiTrends
  - Visão Geral
  - Aplicação
  - Infraestrutura
- Wikitrends => GABDI
- Próximos Passos

# Objetivos

# Do Plano de Trabalho:

ME5: relatório e prova de conceito das tecnologias e arquitetura definidas para armazenamento Big Data.

# Objetivos da análise em andamento

- Proposição de uma arquitetura Big Data
  - Capacidade de manipulação de dados em larga escala
    - Redes sociais (posts, tweets, fotos, etc.)
    - Web analytics (acessos, IPs, localizações, etc.)
  - Resolução dos problemas de escalabilidade e complexidade gerados pelos tradicionais modelos de RDBMS
- Implementação incremental da arquitetura proposta em contexto reduzido
  - Desenvolvimento de aplicação real que utilize os conceitos estudados num contexto Big Data
  - Verificação da viabilidade da arquitetura
  - Analise de desempenho real
  - Resultado: WikiTrends!
- Definição de passo-a-passo para implantação genérica
  - Infraestrutura a ser utilizada
  - Tecnologias a serem empregadas

# Requisitos

# Requisitos

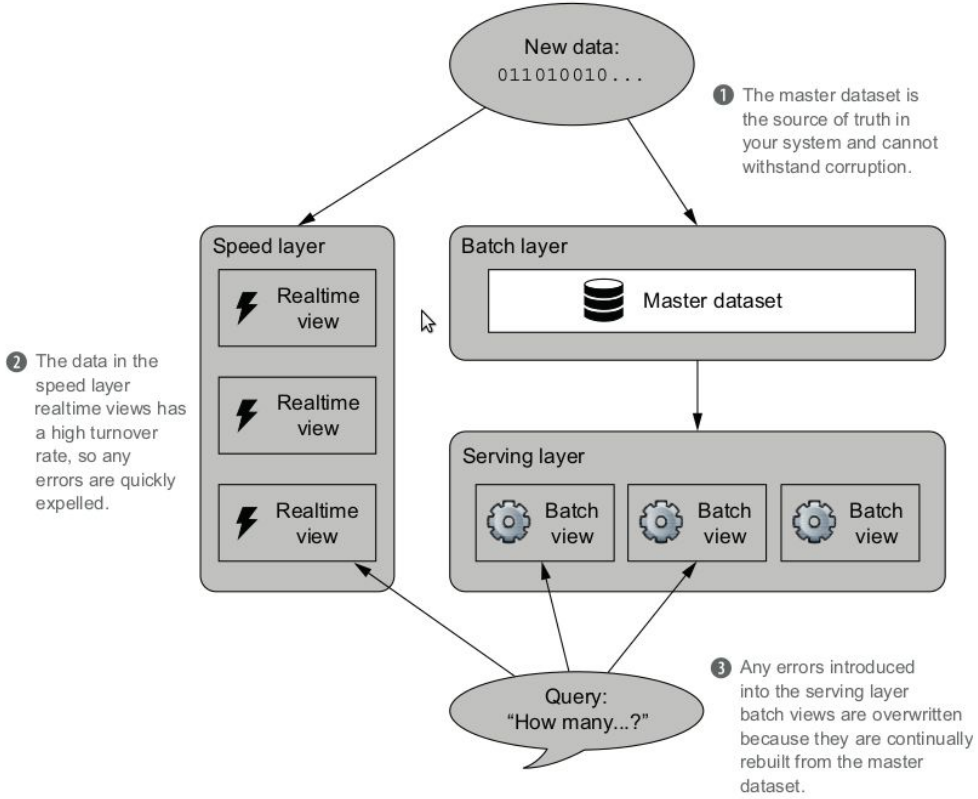
- Características de arquitetura Big Data ideal:
  - Robustez e tolerância a falha
  - Baixa latência
  - Escalabilidade
  - Generalização
  - Extensibilidade
  - Consultas ad-hoc
  - Mínima manutenção
  - Debuggability

# Arquitetura LAMBDA

# Arquitetura Lambda

- Proposta por Nathan Marz
  - BackType (adquirida pelo Twitter)
  - Twitter
- Livro: *“Big Data: Principles and best practices of scalable real-time data systems”*
- Arquitetura em camadas (*layers*) atende a todos os requisitos mencionados
  - *Batch Layer*
  - *Speed Layer*
  - *Serving Layer*
- Por que Arquitetura “Lambda”?
  - “*Data comes in one side, goes out in two other directions*”

# Arquitetura Lambda



# Arquitetura Lambda

- Problema da arquitetura Lambda: Complexidade
  - Lógica de processamento de *views* duplicada em duas camadas
  - Difícil manter e fazer debugging
- Solução: Apache Spark
  - *Batch processing*
  - *Real-time/streaming processing* com **Spark Streaming**
  - Mesma lógica de processamento aplicada a camadas diferentes, evitando retrabalho

```
def countErrors(rdd: RDD[String]): RDD[(String, Int)] = {
  rdd
    .filter(_.contains("ERROR")) // Keep "ERROR" lines
    .map( s =&gt; (s.split(" ")(0), 1) ) // Return tuple with date &amp;amp; count
    .reduceByKey(_+_) // Sum counts for each date
}
```

```
val sc = new SparkContext(conf)

val lines = sc.textFile(...)
val errCount = countErrors(lines)
errCount.saveAsTextFile(...)
```

```
val ssc = new StreamingContext(sparkConf, 60)

// Create the DStream from data sent over the network
val dStream = ssc.socketTextStream(args(1), args(2).toInt, StorageLevel.MEMORY_AND_DISK_

// Counting the errors in each RDD in the stream
val errCountStream = dStream.transform(rdd =&gt; ErrorCount.countErrors(rdd))

// printing out the current error count
errCountStream.foreachRDD(rdd =&gt; {
      System.out.println("Errors this minute:%d".format(rdd.first()._2))
})

// creating a stream with running error count
val stateStream = errCountStream.updateStateByKey[Int](updateFunc)

// printing the running error count
stateStream.foreachRDD(rdd =&gt; {
      System.out.println("Errors today:%d".format(rdd.first()._2))
})
```

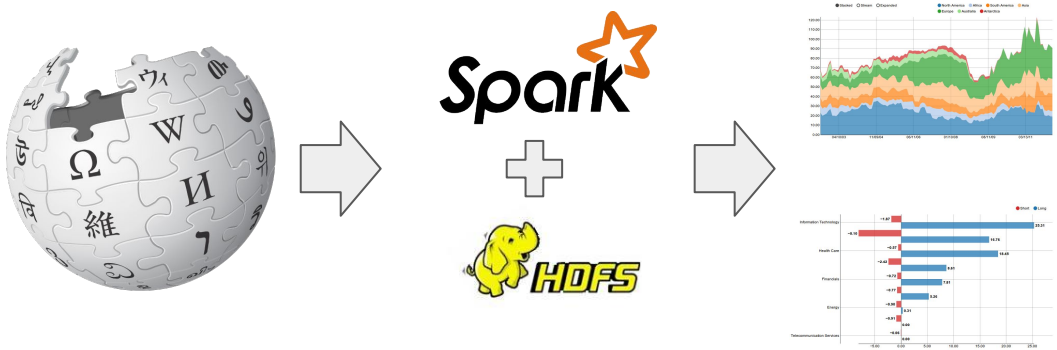
# Prova de Conceito: WikiTrends

# Objetivo da Prova de Conceito

- Instanciar arquitetura LAMBDA numa infraestrutura realista usando tecnologias *off-the-shelf* para executar processamento Big Data
- **Produto Final da prova de conceito:** *trending topics* da Wikipedia
  - processamento agendado de *trending topics*
  - processamento em tempo real de estatísticas do *stream*

# Visão Geral

- Dados abertos originados do *stream* de edições da Wikimedia com detalhes, em tempo real, de cada edição feita na Wikipedia[1]
- Armazenamento em sistema de arquivos distribuídos
- Processamento usando Apache Spark




1 - https://wikitech.wikimedia.org/wiki/Stream.wikimedia.org

# Como é a estrutura de uma edição?


```
{
  "comment": "/* Lista de histórias */",
  "wiki": "ptwiki",
  "server_name": "pt.wikipedia.org",
  "title": "As Mil e Uma Noites",
  "timestamp": 1445278063,
  "server_script_path": "/w",
  "namespace": 0,
  "server_url": "https://pt.wikipedia.org",
  "length": {
    "new": 17210,
    "old": 17182
  },
  "user": "177.43.166.162",
  "bot": false,
  "patrolled": false,
  "type": "edit",
  "id": 61005650,
  "minor": false,
  "revision": {
    "new": 43691245,
    "old": 43470145
  }
}
```

# Como é a estrutura de uma edição? - QUEM?


```
{
  "comment": "/* Lista de histórias */",
  "wiki": "ptwiki",
  "server_name": "pt.wikipedia.org",
  "title": "As Mil e Uma Noites",
  "timestamp": 1445278063,
  "server_script_path": "/w",
  "namespace": 0,
  "server_url": "https://pt.wikipedia.org",
  "length": {
    "new": 17210,
    "old": 17182
  },
  "user": "177.43.166.162",
  "bot": false,
  "patrolled": false,
  "type": "edit",
  "id": 61005650,
  "minor": false,
  "revision": {
    "new": 43691245,
    "old": 43470145
  }
}
```

# Como é a estrutura de uma edição? - QUANDO?


```
{
  "comment": "/* Lista de histórias */",
  "wiki": "ptwiki",
  "server_name": "pt.wikipedia.org",
  "title": "As Mil e Uma Noites",
  "timestamp": 1445278063,
  "server_script_path": "/w",
  "namespace": 0,
  "server_url": "https://pt.wikipedia.org",
  "length": {
    "new": 17210,
    "old": 17182
  },
  "user": "177.43.166.162",
  "bot": false,
  "patrolled": false,
  "type": "edit",
  "id": 61005650,
  "minor": false,
  "revision": {
    "new": 43691245,
    "old": 43470145
  }
}
```

# Como é a estrutura de uma edição? - ONDE?


```
{
  "comment": "/* Lista de histórias */",
  "wiki": "ptwiki",
  "server_name": "pt.wikipedia.org",
  "title": "As Mil e Uma Noites",
  "timestamp": 1445278063,
  "server_script_path": "/w",
  "namespace": 0,
  "server_url": "https://pt.wikipedia.org",
  "length": {
    "new": 17210,
    "old": 17182
  },
  "user": "177.43.166.162",
  "bot": false,
  "patrolled": false,
  "type": "edit",
  "id": 61005650,
  "minor": false,
  "revision": {
    "new": 43691245,
    "old": 43470145
  }
}
```

# Como é a estrutura de uma edição? - O QUE?


```
{
  "comment": "/* Lista de histórias */",
  "wiki": "ptwiki",
  "server_name": "pt.wikipedia.org",
  "title": "As Mil e Uma Noites",
  "timestamp": 1445278063,
  "server_script_path": "/w",
  "namespace": 0,
  "server_url": "https://pt.wikipedia.org",
  "length": {
    "new": 17210,
    "old": 17182
  },
  "user": "177.43.166.162",
  "bot": false,
  "patrolled": false,
  "type": "edit",
  "id": 61005650,
  "minor": false,
  "revision": {
    "new": 43691245,
    "old": 43470145
  }
}
```

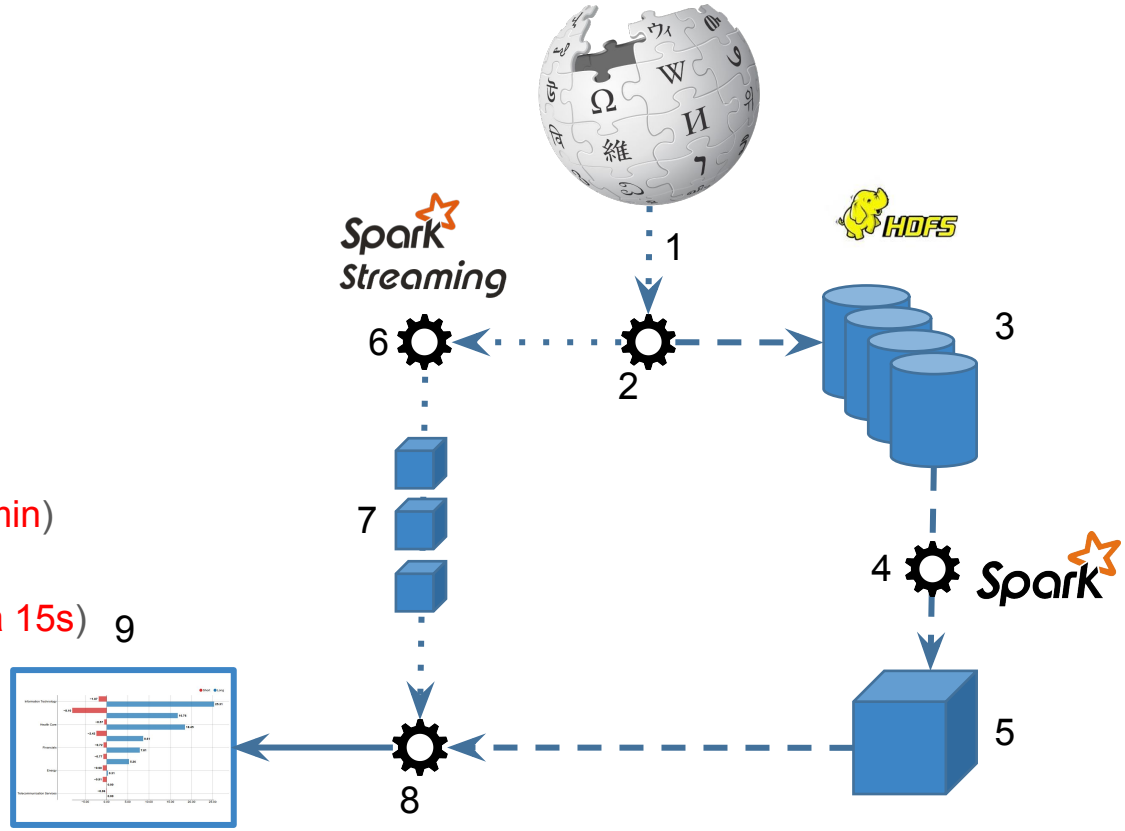
# Aplicação

- Quais idiomas são mais editados?
- Quais páginas são mais editadas?
- Quem são os editores mais frequentes?

# Aplicação

Arquitetura LAMBDA

1. Stream da Wikimedia
2. Cliente do Stream da Wikimedia
3. Master Dataset
4. Processamento Batch (a cada 30min)
5. Batch View
6. Processamento Real Time (a cada 15s)
7. Real Time Views
8. Serving Layer
9. WikiTrends




Como organizar a infraestrutura?

# Infraestrutura

- OpenStack@LSD (Laboratório de Sistemas Distribuídos)
  - Servidor de Máquinas Virtuais nos moldes da Amazon AWS
  - Quota alocada
    - 14 CPU's virtuais
    - 28 GB de RAM

# Infraestrutura

Arquiteturalmente falando, temos três papéis distintos:

1. Servidor da aplicação WikiTrends
2. Gerenciador do backend BigData
3. Backend BigData em si

# Infraestrutura

- Automatizamos a instalação da infraestrutura com scripts
- Realiza pouca transferência
- Configuração flexível
- Fácil de migrar para outra infraestrutura com pouca ou nenhuma adaptação
- Processo pode ficar demorado se instalação de dependências for demorada (não é o nosso caso atualmente)

# Infraestrutura

Produzimos dois scripts*:

- **boot.sh****: realiza configurações durante o primeiro boot das VMs
- **setup****: gerencia a criação, configuração e atualização dos componentes do backend

* os script produzidos estão isponíveis no repositório privado https://github.com/analyticsUfcg/gabdi/tree/master/infrastructure/openstack
** FAQ dos scrips pode ser encontrado no readme no link acima

# Infraestrutura - Armazenamento: HDFS

Precisamos (principalmente) de dois tipos de servidores:

1. **namenode**: mantem a árvore de diretórios
    a. papel do gerenciador do backend Big Data
2. **datanode**: mantem os dados em si
    a. cada um dos nós de processamento do backend Big Data

Costuma-se usar servidores de maior porte para deploying do **namenode**

# Infraestrutura - Processamento

- Problema: precisamos de um gerenciador de aplicações distribuídas
  - YARN (Hadoop 2.0)
    - permite que se execute aplicações MapReduce na mesma infraestrutura
    - usado por quem já tem uma infraestrutura YARN no ar e quer expandir para Spark
  - Spark
    - o pacote do Apache Spark vêm com um gerenciador de aplicações próprio não muito diferente do YARN
  - Mesos
    - permite que se execute aplicações distribuídas (menos MapReduce)
    - usado por quem já tem uma infraestrutura Mesos no ar e quer expandir para Spark
- **Gerenciador do Spark é suficiente para os objetivos do protótipo**

# Infraestrutura - Processamento: Spark

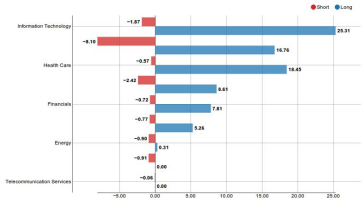
Precisamos de dois tipos de servidores:

1. **master**: gerencia a execução das tarefas
    a. papel do gerenciador do backend Big Data
2. **worker**: executam o processamento dos dados
    a. cada um dos nós de processamento do backend Big Data

OBS: O Spark conta cada CPU disponível como um worker, logo, uma máquina com 4 núcleos será contabilizada como 4 workers pelo Spark.

# Infraestrutura - Servidor WEB

- Usamos uma terceira VM como servidor Web
  - Acesso direto ao HDFS e ao Spark pelo nó de gerência
  - Execução de tarefas usando o serviço **spark-submit**
- Esse servidor também:
  - alimenta sistema de arquivos com novos dados
  - dispara o processamento no Spark
  - combina resultados das camadas *batch* e *speed*



stream

batch

injestão de dados

batch layer

resultados

speed layer

Dados

Servidor Web
(cliente)

Gerenciador
(namenode + spark master)

Nós de Processamento
(datanode + spark worker)

# Infraestrutura

Trabalhos futuros:

- Como fazer ingestão de dados mais eficientemente?
- Como agendar a execução dos Jobs no Spark considerando o tamanho dos dados?
- Como otimizar a produção e o consumo dos resultados calculados?

# WikiTrends => GABDI

# WikiTrends => Gabdi

- Mapeamento das funcionalidades do WikiTrends em um possível subconjunto de funcionalidades do GABDI
  - Prova de viabilidade da arquitetura sugerida no contexto mais amplo do GABDI
- **Top Campaigns & Users do GABDI**
- Utilização dos dados relativos a campanhas para elaboração de rankings
- Exemplo de formato de dado a ser processado (pode ser incrementado):

```
{
        "campaign_name": "Qual a sua rede de fast-food preferida?"
        "campaign_id": 8593409232, # optional (?)
        "timestamp": 1445278063,
        "user_id": 1827328743
        "user_name": "Ricardo Santos"
        "type": "response" # create / response / view
        "response": "Choice: Burger King" # opcional
        “choices”: {“Choice: Burger King”, ”Choice: McDonald’s”} # opcional
}
```

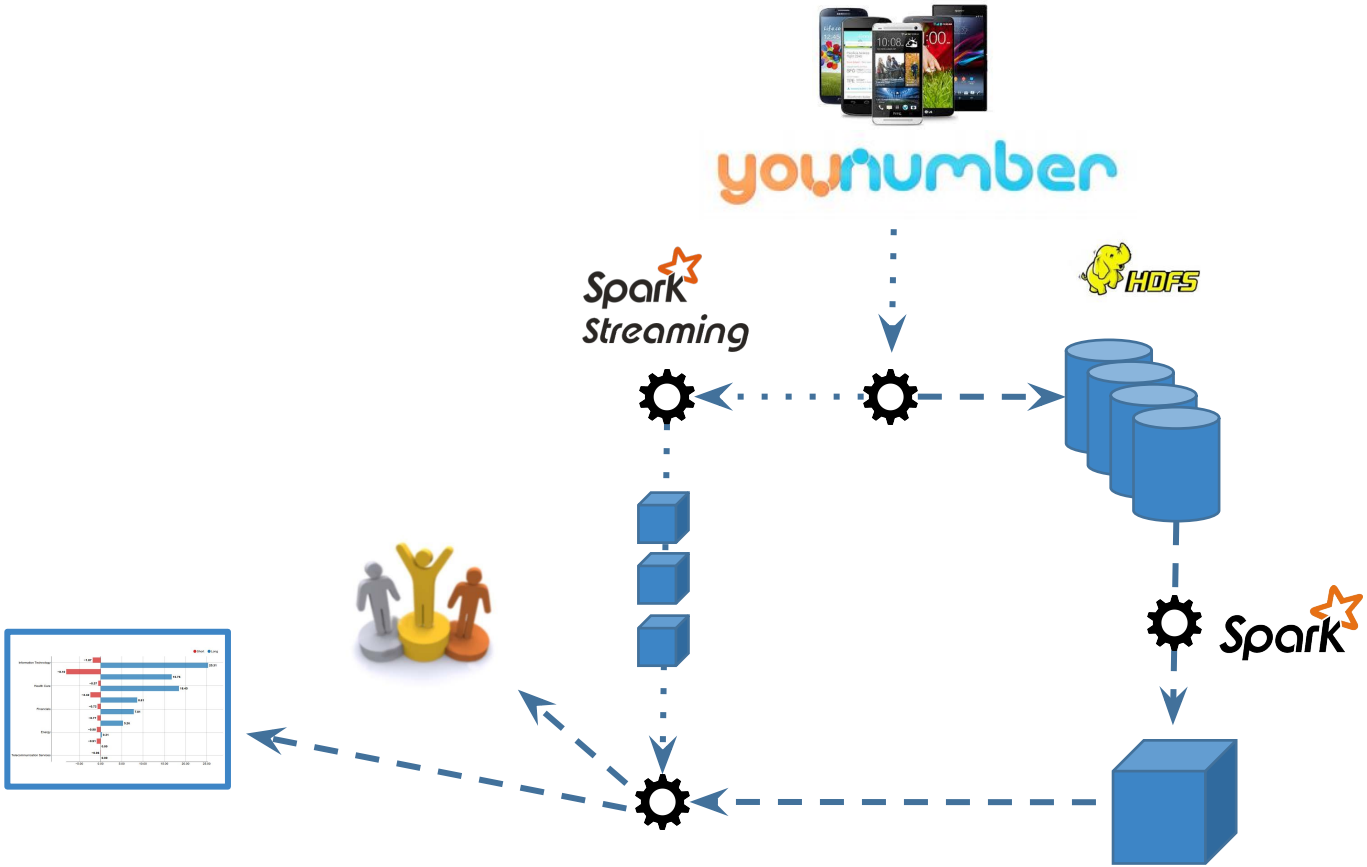
O quê?

Quando?

Quem?

Como?

# WikiTrends => Gabdi



# Próximos Passos

# Do Plano de Trabalho

ME7: relatório e prova de conceito das tecnologias e arquitetura definidas para retorno de consulta e push de campanha.