FIAP GRADUAÇÃO

FIAP

DATA SCIENCE

BIG DATA ARCHITECTURING & DATA INTEGRATION

Prof. Dr. Renê de Ávila Mendes

Objetivos da disciplina

DISCIPLINA: Big Data Architecturing & Data Integration

OBJETIVOS: Entenda as principais **arquiteturas** para ingestão, processamento e análise de grandes volumes de dados. Conheça as principais **ferramentas** open-source de Big Data como Hadoop, MapReduce, Spark, Sqoop, NiFi, Flume, Kafka, Zookeeper, HBase, Hive e as **integre** com as ferramentas de **extração, transformação e carga** de dados em modelos dimensionais. Entenda conceitos sobre **computação paralela e distribuída**, aplicação do Hadoop e bases Apache e arquiteturas *serverless* e desacopladas. Veja como **visualizar os dados** estruturados ou não estruturados com ferramentas de Self-Service Business Intelligence como PowerBI, utilizando as melhores práticas de **visualização de dados**.

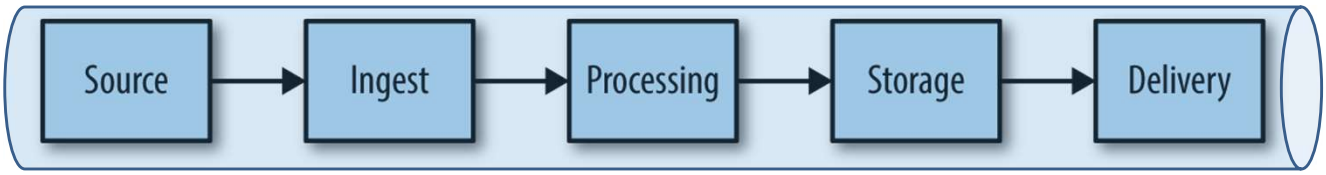
Assuntos – 2º Semestre

- Arquiteturas para Big Data
- Data Pipelines
- Conceito de Data Lake
- Knime, HIVE, Spark, Data Streaming

Objetivo dessa aula

OBJETIVOS: Entender o funcionamento das arquiteturas de Big Data Kappa e Liquid.

O pipeline de dados analíticos

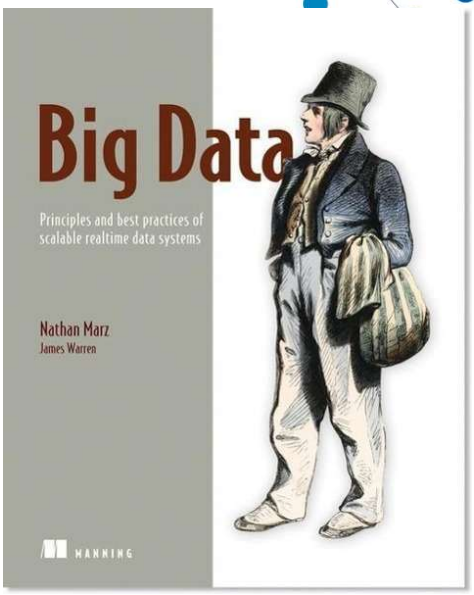


Os dados não se formatam por si mesmos para fins de análise. Eles passam por uma série de passos que envolvem a coleta a partir das origens, o tratamento para as formas necessárias à análise e finalmente sua disponibilização na forma de resultados para serem consumidos. Estes passos podem ser pensados como um “tubo”.
Este modelo ajuda a entender onde aplicar cada tecnologia.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark**." O'Reilly Media, Inc., 2017.

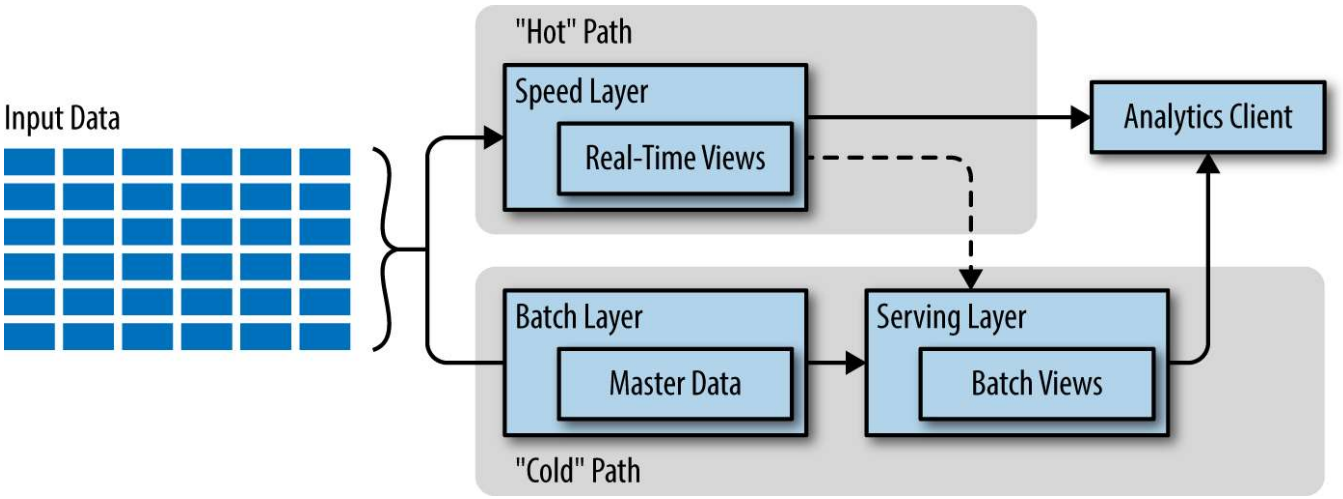
Big Data – Arquitetura Lambda

- Marz e Warren (2013)
- Implementa funções arbitrárias em um conjunto arbitrário de dados
- Resposta em uma baixa latência pela combinação de diferentes ferramentas e técnicas
- Divide o sistema em três camadas dependentes e complementares



MARZ, N.; WARREN, J. Big Data: Principles and best practices of scalable realtime data systems. [S.l.]: Manning Publications Co., 2015.

Lambda – Modelo da Arquitetura



Na arquitetura Lambda há dois caminhos para o dado fluir pelo “tubo” de dados analítico:

- Um caminho quente (hot path) no qual dados sensíveis a latência fluem para serem rapidamente consumidos por clientes analíticos;
- Um caminho frio (cold path) pelo qual todos os dados fluem e são processados em lotes que podem tolerar grandes latências até que o resultado seja produzido.

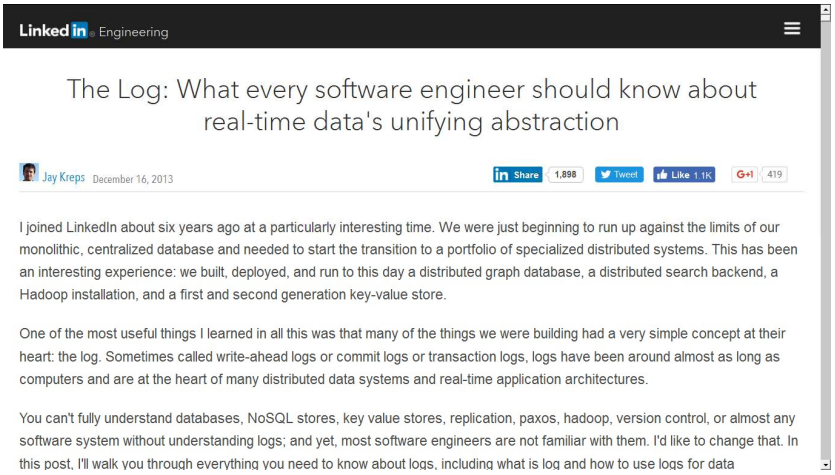
Em certas implementações a camada serving pode hospedar dados das visões real-time e das visões batch.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark**." O'Reilly Media, Inc., 2017.

ARQUITETURA KAPPA

Big Data – Arquitetura Kappa

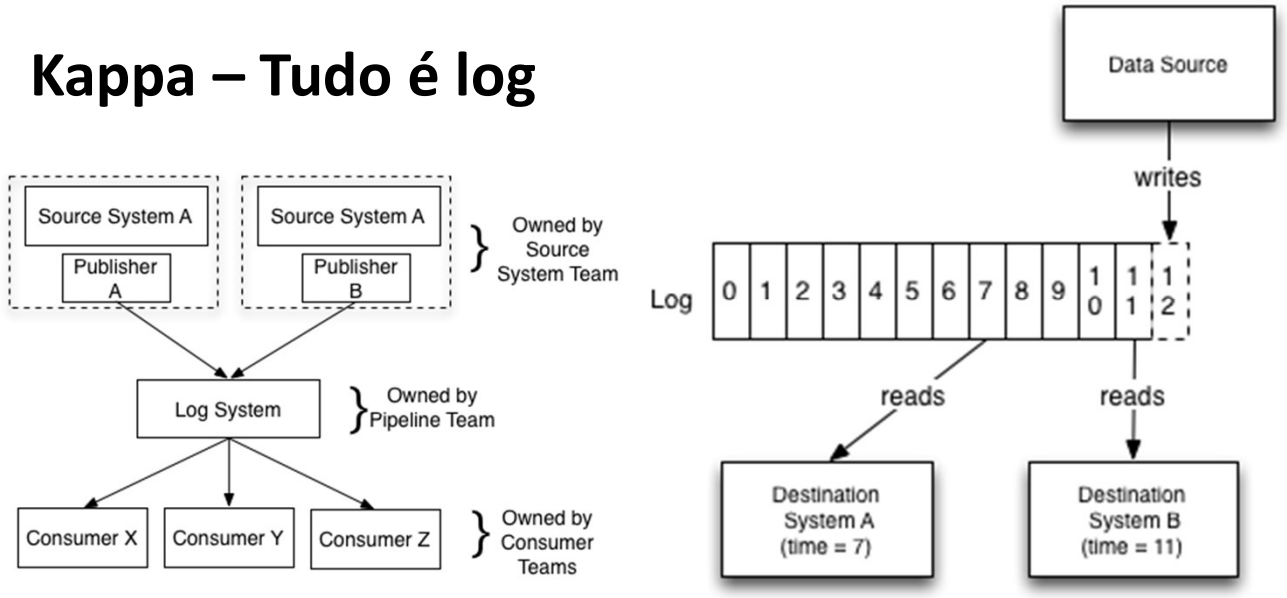
- Jay Kreps (2013)
- Todo o processamento é responsabilidade de uma camada paralelizada de processamento em **fluxo**
- Baseada em **logs**



- Baseada em uma fila central escalável;
- Todos os dados (eventos) são armazenados nesta fila pelos nós produtores (aplicações);
- Cada evento é imutável e armazenado na sequência em que é gerado;
- O evento só pode ser alterado por um novo evento sendo adicionado à fila;
- Os nós consumidores consomem os eventos assincronamente e podem reprocessar;
- Os eventos são retidos por um período de tempo determinado

KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction. https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Kappa – Tudo é log



Kappa – O que é log ?

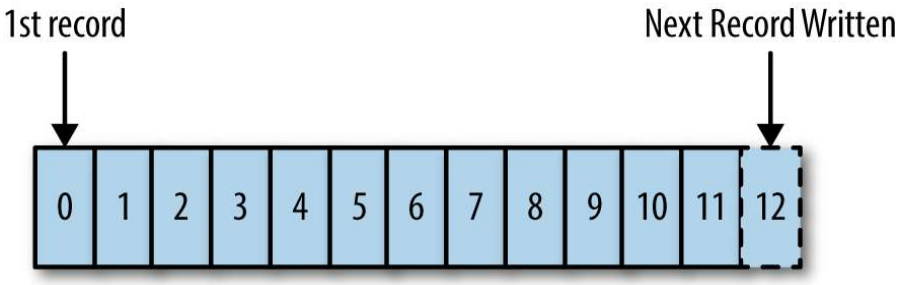
```
jkreps-mn:~ jkreps$ tail -f -n 20 /var/log/apache2/access_log
::1 - - [23/Mar/2014:15:07:00 -0700] "GET /images/apache_feather.gif HTTP/1.1" 200 4128
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/producer_consumer.png HTTP/1.1" 200 8(
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/log_anatomy.png HTTP/1.1" 200 19579
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/consumer-groups.png HTTP/1.1" 200 268:
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/log_compaction.png HTTP/1.1" 200 4141(
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /documentation.html HTTP/1.1" 200 189893
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/log_cleaner_anatomy.png HTTP/1.1" 200
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/kafka_log.png HTTP/1.1" 200 134321
::1 - - [23/Mar/2014:15:07:04 -0700] "GET /images/mirror-maker.png HTTP/1.1" 200 17054
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /documentation.html HTTP/1.1" 200 189937
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /styles.css HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/kafka_logo.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/producer_consumer.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/log_anatomy.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/consumer-groups.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/log_cleaner_anatomy.png HTTP/1.1" 304
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/log_compaction.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/kafka_log.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:08:07 -0700] "GET /images/mirror-maker.png HTTP/1.1" 304 -
::1 - - [23/Mar/2014:15:09:55 -0700] "GET /documentation.html HTTP/1.1" 200 195264
```

Nossa tendência é a associação imediata com logs como os do Apache.

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration**. " O'Reilly Media, Inc.", 2015.

Kappa – O que é log ?

- Uma sequência de registros ordenados por tempo;
- *Append-only*;
- Semelhante a logs de bancos de dados (*commit logs* ou *journals*);
- JSON ou qualquer conteúdo.

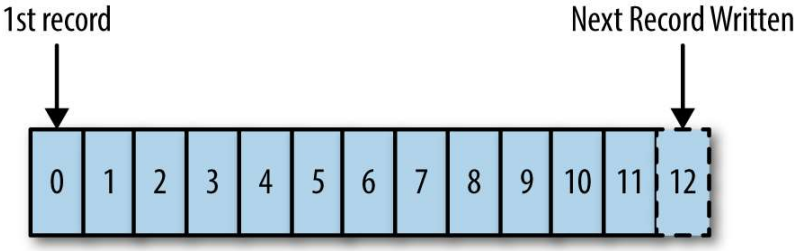


Cada retângulo representa um registro que foi anexado ao log. Os registros são armazenados na ordem em que foram anexados. As leituras procedem da esquerda para a direita. Cada entrada anexada ao log recebe um número de entrada de log sequencial exclusivo que atua como sua chave exclusiva. O conteúdo e o formato dos registros não são importantes para os propósitos desta discussão. Para ser concreto, podemos apenas imaginar cada registro como um blob JSON, mas é claro que qualquer formato de dados serve.

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration**. " O'Reilly Media, Inc.", 2015.

Kappa – O que é log ?

- A posição do log confere a noção de tempo;
- É um tipo de tabela ou de arquivo onde os registros são ordenados por tempo;
- Uma estrutura de dados abstrata, não um arquivo de texto;
- Registram **o que** acontece e **quando** acontece;



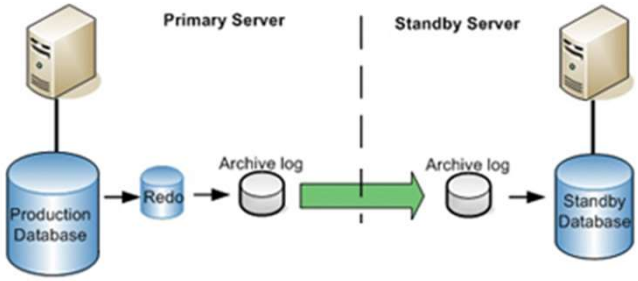
A ordenação dos registros define uma noção de “tempo”, já que as entradas à esquerda são definidas como mais antigas que as entradas à direita. O número da entrada de log pode ser considerado como o “carimbo de data/hora” da entrada. Descrever essa ordenação como uma noção de tempo parece um pouco estranho a princípio, mas tem a propriedade conveniente de ser desacoplado de qualquer relógio físico específico. Essa propriedade se tornará essencial quando chegarmos aos sistemas distribuídos.

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration**. " O'Reilly Media, Inc.", 2015.

Log – Onde é usado

Em bancos de dados:

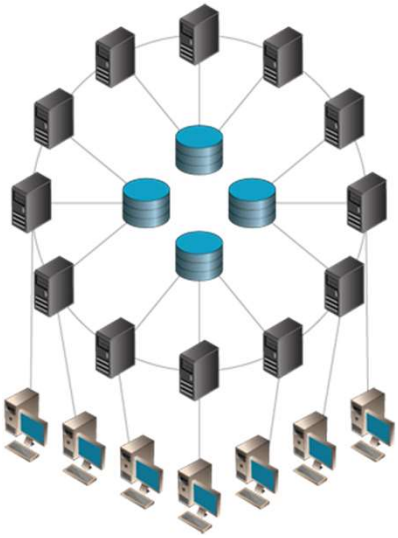
- O log é usado como um mecanismo de publicação/assinatura para transmitir dados para outras réplicas;
- O log é usado como um mecanismo de consistência para executar as atualizações que são aplicadas em múltiplas réplicas.



Exemplo: replicação entre um banco Oracle primário e seu banco *standby*. Note que os dados trafegam por meio de logs (archived redo logs).

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration**. " O'Reilly Media, Inc.", 2015.

Log – Onde é usado



Em sistemas distribuídos:

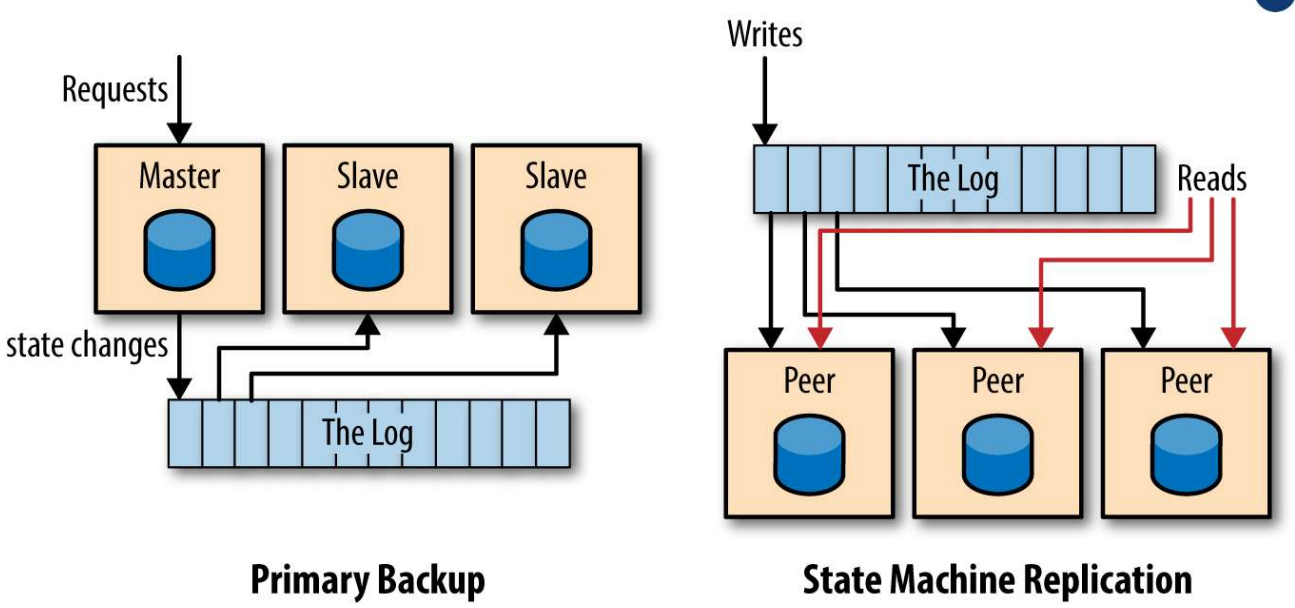
- A natureza temporal de uma sequência de logs é a garantia de que os resultados serão obtidos em ordem em um sistema distribuído.
- Princípio de replicação da máquina de estados:

"Se dois processos idênticos e determinísticos começarem no mesmo estado e receberem as mesmas entradas na mesma ordem, eles produzirão a mesma saída e terminarão no mesmo estado."

Os mesmos problemas que os bancos de dados resolvem com logs (como distribuir dados para réplicas e concordar com a ordem de atualização) estão entre os problemas mais fundamentais para todos os sistemas distribuídos.
…
Uma das coisas bonitas sobre isso é que os números de entrada de log discretos agora agem como um relógio para o estado das réplicas - você pode descrever o estado de cada réplica por um único número: o registro de data e hora para a entrada de log máxima que ela processou . Duas réplicas ao mesmo tempo estarão no mesmo estado. Assim, esse carimbo de data/hora combinado com o log captura exclusivamente todo o estado da réplica. Isso fornece uma noção de tempo discreta e orientada a eventos que, ao contrário dos relógios locais da máquina, é facilmente comparável entre diferentes máquinas.

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration**. " O'Reilly Media, Inc.", 2015.

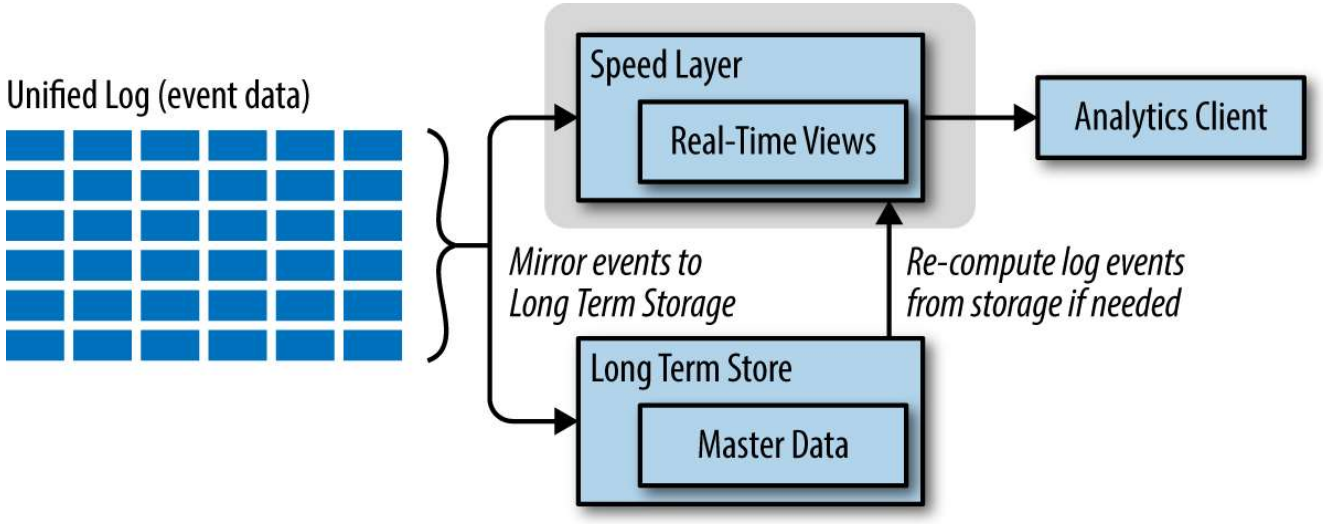
Log – Onde é usado



No modelo de backup primário, um nó mestre é escolhido para lidar com todas as leituras e gravações. Cada gravação é postada no log. Os escravos se inscrevem no log e aplicam as alterações que o mestre executou em seu estado local. Se o mestre falhar, um novo mestre é escolhido entre os escravos. No modelo de replicação da máquina de estado, todos os nós são pares. As gravações vão primeiro para o log e todos os nós aplicam a gravação na ordem determinada pelo log.

KREPS, Jay. **I Heart Logs: Event Data, Stream Processing, and Data Integration**. " O'Reilly Media, Inc.", 2015.

Kappa – Descrição



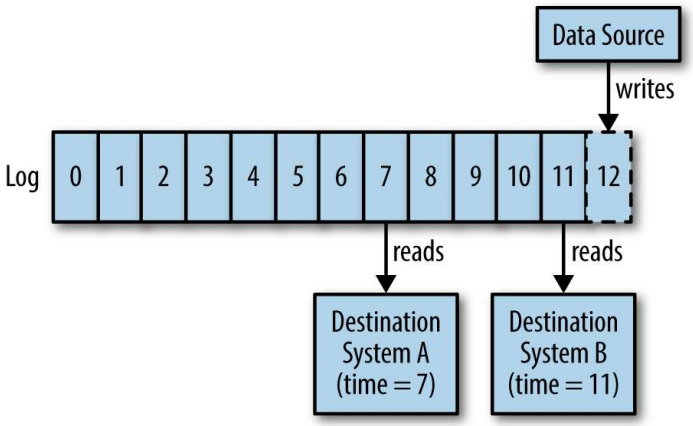
TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark**." O'Reilly Media, Inc., 2017.

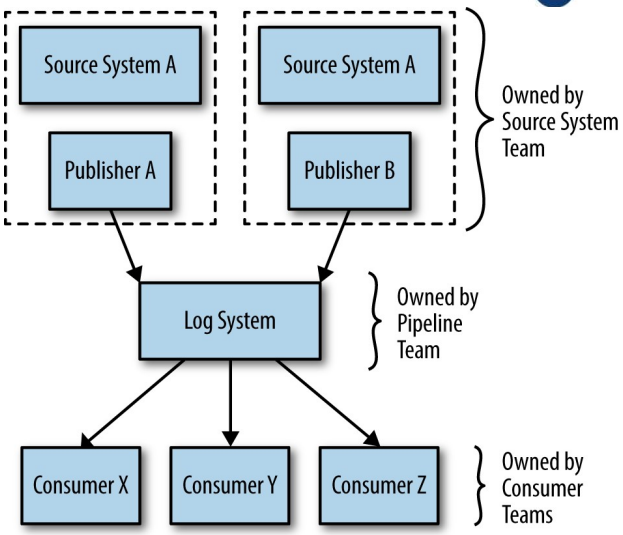
Kappa – Descrição

- Baseada em uma fila central escalável;
- Todos os dados (eventos) são armazenados nesta fila pelos nós produtores (aplicações);
- Cada evento é imutável e armazenado na sequência em que é gerado;
- O evento só pode ser alterado por um novo evento sendo adicionado à fila;
- Os nós consumidores consomem os eventos assincronamente e podem reprocessar;
- Os eventos são retidos por um período de tempo determinado.

TEJADA, Zoiner. **Mastering Azure Analytics: Architecting in the Cloud with Azure Data Lake, HDInsight, and Spark**." O'Reilly Media, Inc., 2017.

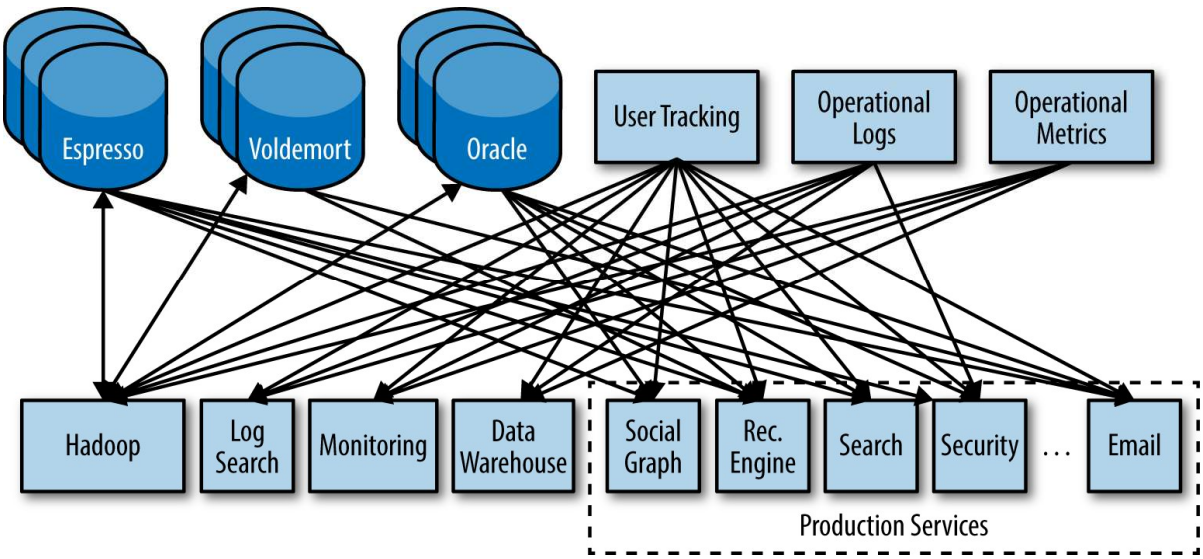
Kappa – Funcionamento





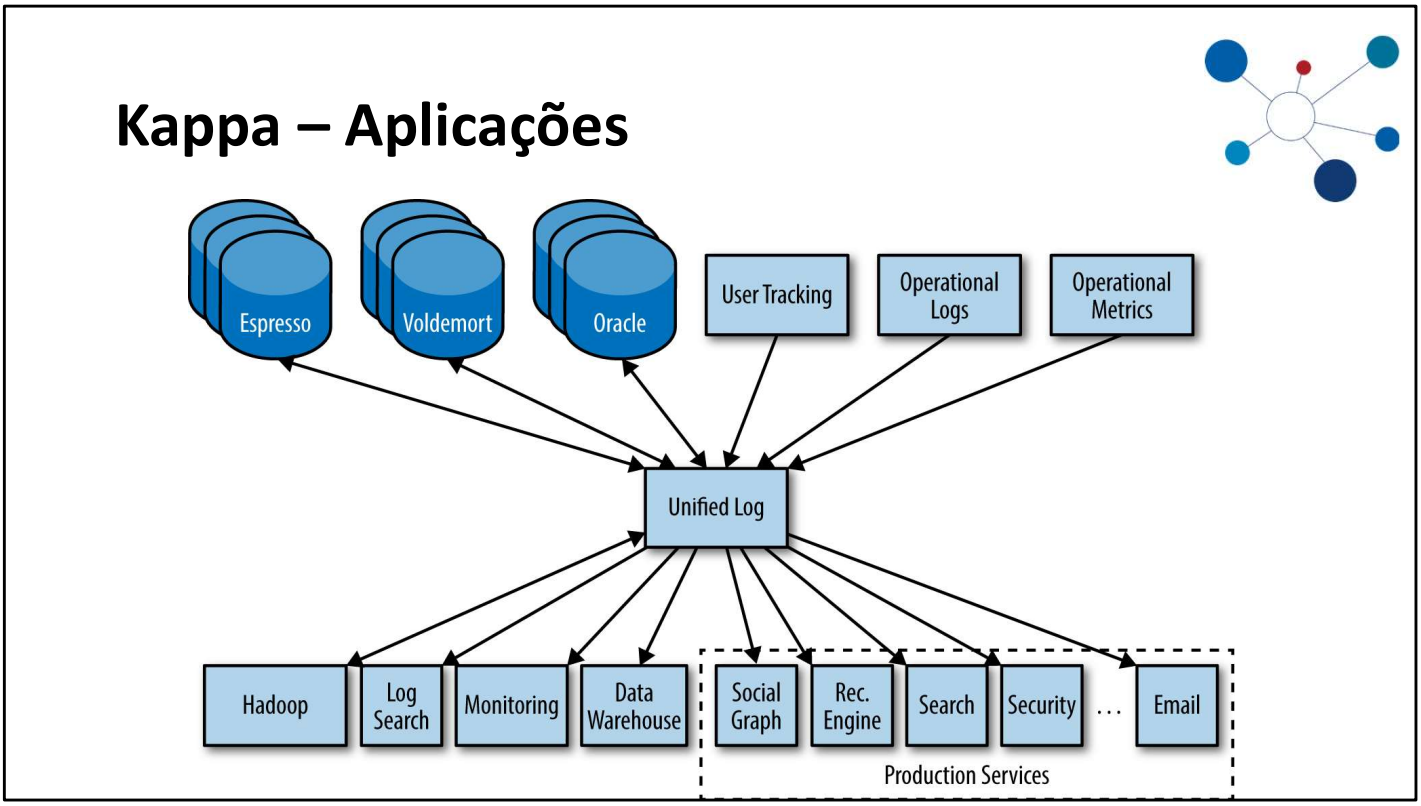
KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction. https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Kappa – Aplicações



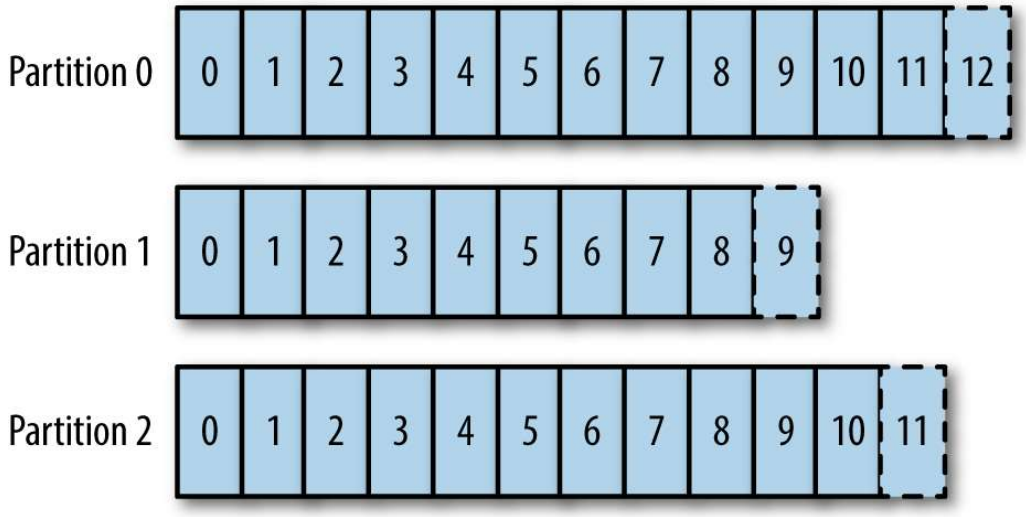
KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction. https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Kappa – Aplicações



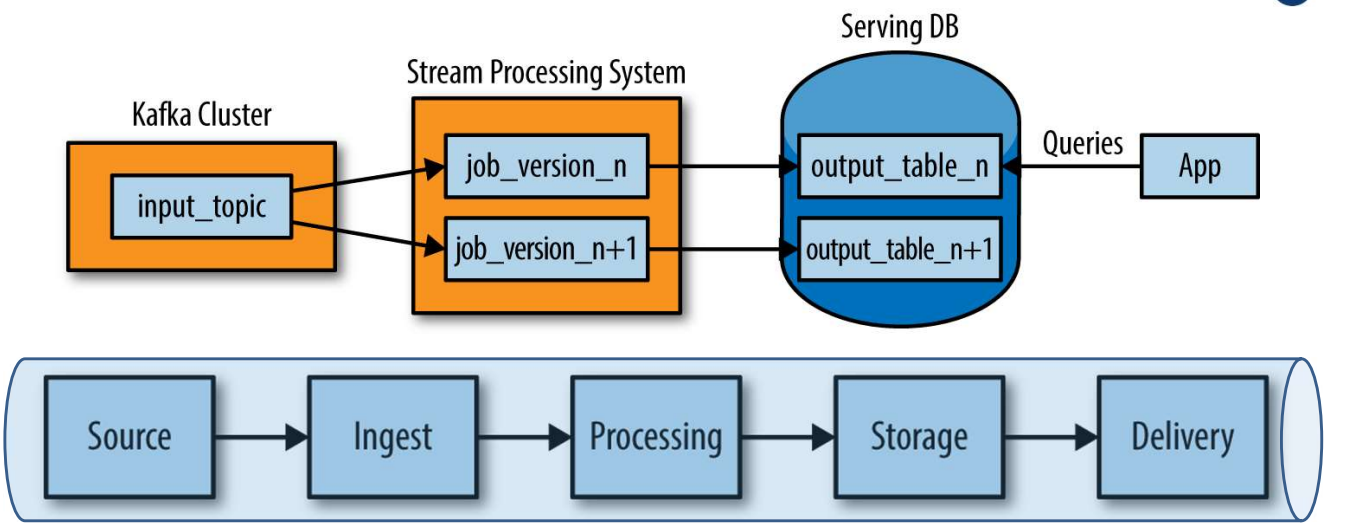
KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction. https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Kappa – Escalamento do log



KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction. https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Kappa – Implementação

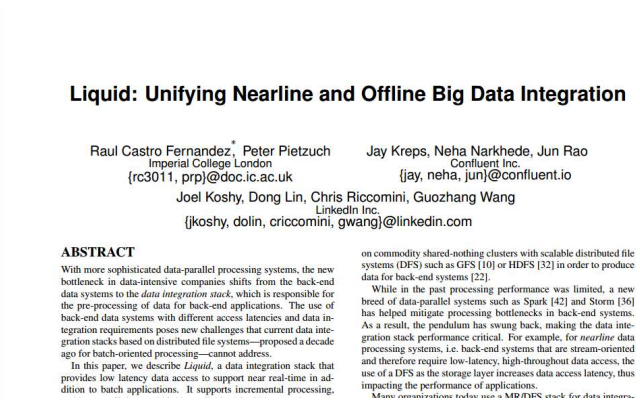


KREPS, J. The Log: What every software engineer should know about real-time data's unifying abstraction. https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

ARQUITETURA LIQUID

Big Data – Arquitetura Liquid

- Fernandez e outros (2015)
- *Stack* de integração de dados
- Oferece acesso de baixa latência
- Suporta processamentos em lote e em tempo quase real
- Suporta processamento incremental, eficiência de custo e alta disponibilidade.



Liquid: Unifying Nearline and Offline Big Data Integration

Raul Castro Fernandez,* Peter Pietzuch
Imperial College London
{rc3011, prp}@doc.ic.ac.uk

Jay Kreps, Neha Narkhede, Jun Rao
Confluent Inc.
{jay, neha, jun}@confluent.io

Joel Koshy, Dong Lin, Chris Riccomini, Guozhang Wang
LinkedIn Inc.
{jkoshy, dolin, criccomini, gwang}@linkedin.com

ABSTRACT

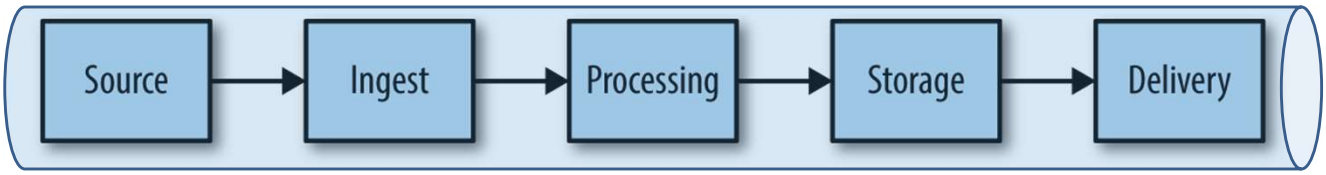
With more sophisticated data-parallel processing systems, the new bottleneck in data-intensive companies shifts from the back-end data systems to the *data integration stack*, which is responsible for the pre-processing of data for back-end applications. The use of back-end data systems with different access latencies and data integration requirements poses new challenges that current data integration stacks based on distributed file systems—proposed a decade ago for batch-oriented processing—cannot address.

In this paper, we describe *Liquid*, a data integration stack that provides low latency data access to support near real-time in addition to batch applications. It supports incremental processing, on commodity shared-nothing clusters with scalable distributed file systems (DFS) such as GFS [10] or HDFS [32] in order to produce data for back-end systems [22].

While in the past processing performance was limited, a new breed of data-parallel systems such as Spark [42] and Storm [36] has helped mitigate processing bottlenecks in back-end systems. As a result, the pendulum has swung back, making the data integration stack performance critical. For example, for *nearline* data processing systems, i.e. back-end systems that are stream-oriented and therefore require low-latency, high-throughout data access, the use of a DFS as the storage layer increases data access latency, thus impacting the performance of applications.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

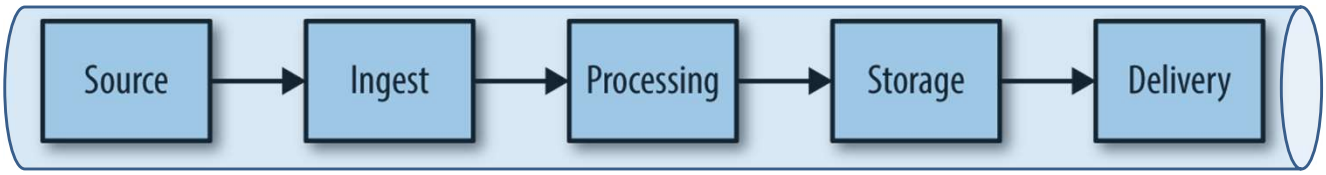
Tipos de processamento



Lote (batch)

Fluxo (stream)

Tipos de processamento



Lote (batch)

Fluxo (stream)

online **nearline**

Tipos de processamento

Online

Executado por aplicações que oferecem sua resposta com uma latência de milissegundos.

Nearline (near online)

Executado por aplicações que oferecem sua resposta com uma latência de segundos.

Quanto antes oferecerem o resultado maior é o valor para os usuários.

Near online - Aplicativos que consultam um gráfico social, pesquisam dados, normalizam dados ou monitoram alterações são classificados como nearline: quanto mais cedo eles fornecerem resultados, maior será o valor para os usuários finais. Como os pipelines de processamento têm mais estágios, a latência de ponta a ponta também aumenta, o que torna o processamento nearline mais desafiador. Fundamentalmente, as pilhas baseadas em DFS não suportam processamento de baixa latência porque têm uma alta sobrecarga por estágio: elas são projetadas para leituras e gravações de dados granulares.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Tipos de processamento

Offline

Processamento onde a latência aceitável varia entre minutos a horas.

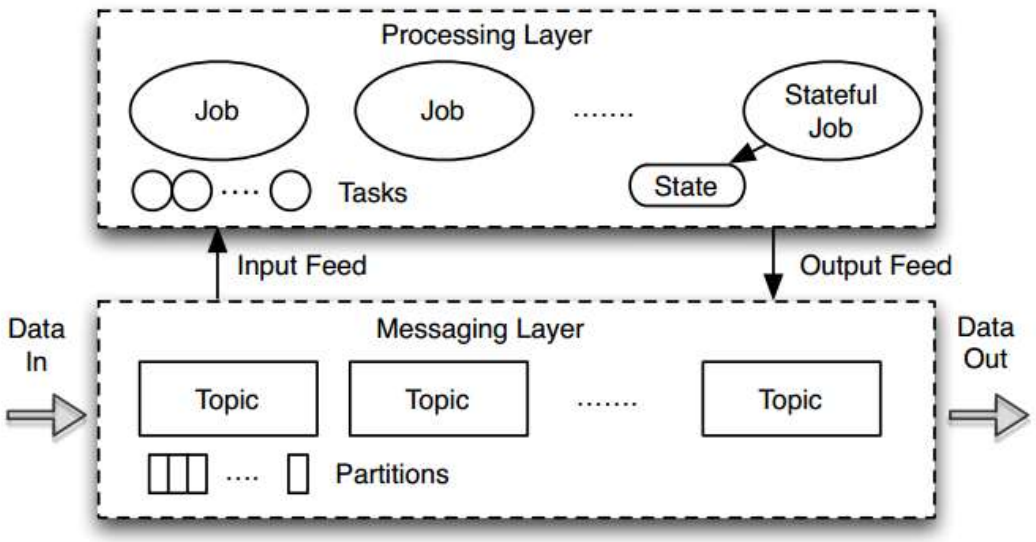
Neste tipo de processamento o armazenamento em DFS é aceitável e recomendado porque sua latência não compromete o processamento.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Stateful e Stateless

- Adjetivos que descrevem se um programa, processo ou computador é capaz de anotar e lembrar um ou mais eventos precedentes em uma determinada sequência de interações.
- ***Stateful*** – o processo mantém a trilha histórica do estado das interações, normalmente atribuindo valores em um campo de armazenamento destinado a este fim;
- ***Stateless*** – o processo não mantém registro de interações anteriores e cada nova interação é processada unicamente com base nos dados fornecidos ao processo.

Liquid – Descrição da arquitetura



Camada de mensagem – provê acesso aos dados baseado em metadados, os quais permitem às aplicações consumidoras lerem dados a partir de um ponto específico no tempo. Esta camada é distribuída para escalar para grandes volumes de dados mantendo alta disponibilidade;

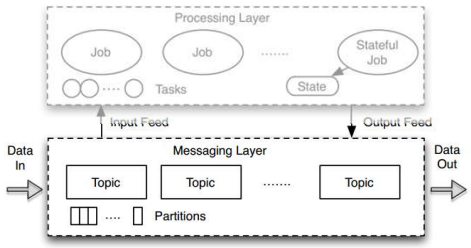
Camada de processamento – executa tarefas do tipo ETL para sistemas consumidores, garantindo um acesso aos dados com baixa latência. Esta camada pode executar processamentos arbitrários antes de entregar os dados aos sistemas consumidores finais, indo de limpeza dos dados e normalização para computação de estatísticas agregadas ou detecção de anomalias nos dados.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Descrição da arquitetura

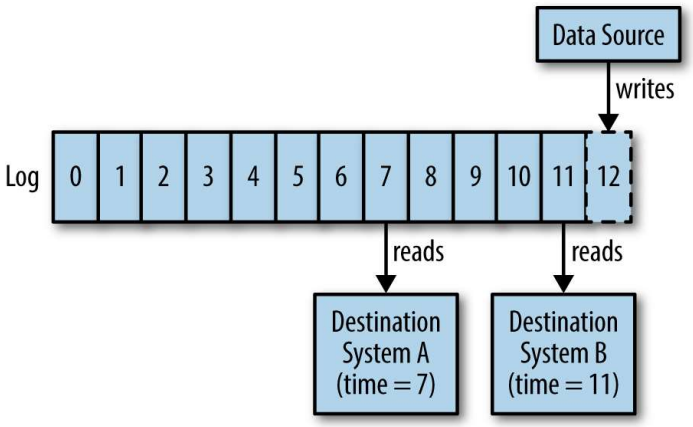
Camada de mensagem

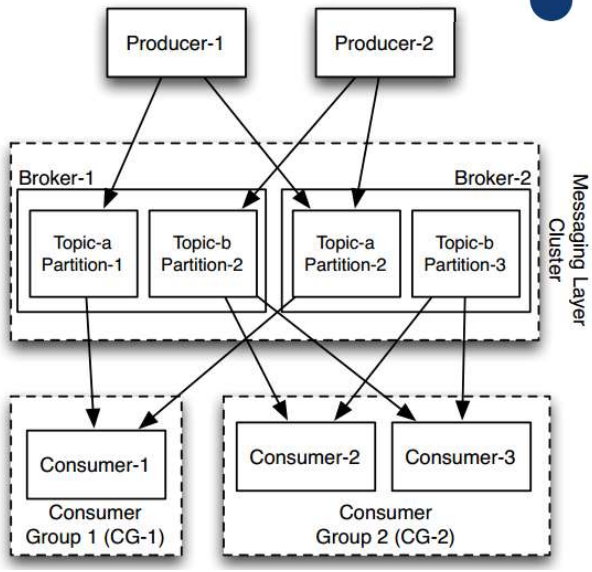
- Suporta a camada de processamento;
- Armazena um grande volume de dados com alta disponibilidade;
- Oferece a possibilidade de acessar os dados com base em anotações por metadados (*rewindability*).



FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Camada de mensagem





Baseia-se no modelo publicador/assinante, armazenando os dados em tópicos. Pode utilizar o Apache Kafka.

Liquid – Camada de mensagem

Tópicos

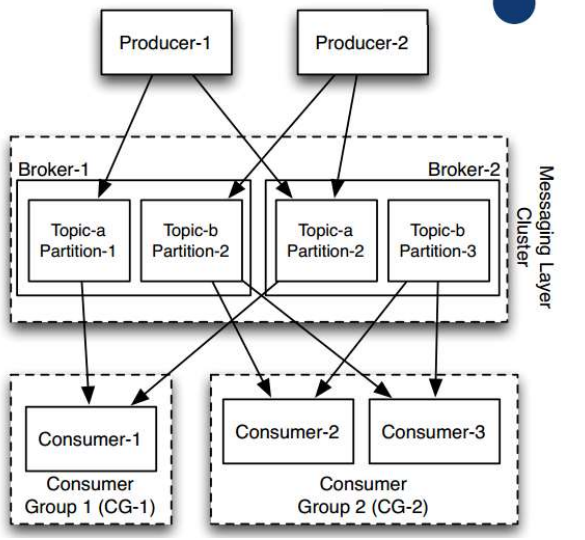
Separam os dados em categorias que façam sentido para as aplicações que produzem os dados.

Produtores

Aplicações que produzem os eventos, os quais são armazenados em diferentes tópicos.

Consumidores

São aplicações assinantes dos tópicos, lendo dados da camada de mensagem.



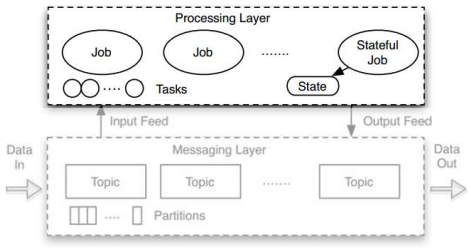
Metadata-based access. A camada de mensagens usa um gerenciador de deslocamento altamente disponível e logicamente centralizado para manter anotações nos dados, que podem ser consultados pelos clientes. Por exemplo, os consumidores podem verificar suas últimas compensações consumidas para salvar seu progresso; após a falha, eles podem solicitar os últimos dados que processaram. Para reprocessar os dados, os clientes podem incluir metadados, como timestamps, com os deslocamentos e recuperar dados de acordo com esses timestamps armazenados anteriormente.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Camada de processamento

Camada de processamento

- Executa *jobs* do tipo ETL para diferentes sistemas consumidores de acordo com modelo de processamento em fluxo que mantém o estado (*stateful*);
- Garante os níveis de serviço através do isolamento de recursos;
- Oferece resultados com baixa latência;
- Permite o processamento incremental.

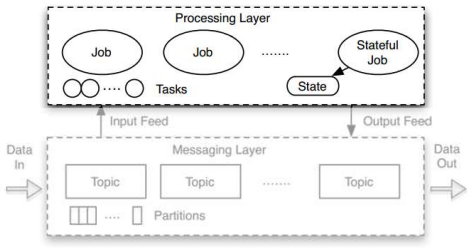


FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Camada de processamento

Jobs

- São divididos em tarefas para permitir o processamento paralelo, processando diferentes partições dos tópicos;
- Podem se comunicar com outros *jobs*, formando um grafo de processamento de dados em fluxo.

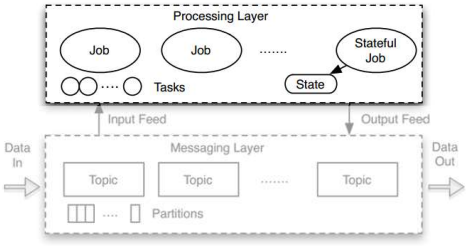


FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Camada de processamento

Processamento *stateful*

- Os estados são carregados localmente por cada *job*;
- Estes estados podem ser representados de formas diversas, tais como uma janela dos dados em fluxo mais recentes, um dicionário de estatísticas ou mesmo um índice invertido utilizado em *queries*;
- Para fins de recuperação de um *job*, a camada de processamento salva o estado atual do *job* como dados derivados, armazenados na cada de mensagem.

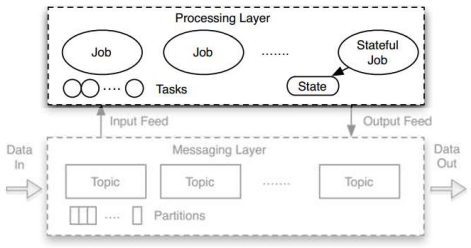


FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Camada de processamento

Processamento incremental

- Ocorre pela combinação do estado de um *job* e dos metadados;
- O *job* pode periodicamente verificar os *offsets* que já consumiu e manter um sumário dos dados de entrada como sendo seu estado atual;
- Quando novos dados são disponibilizados, o *job* pode então ignorar os dados já processados.

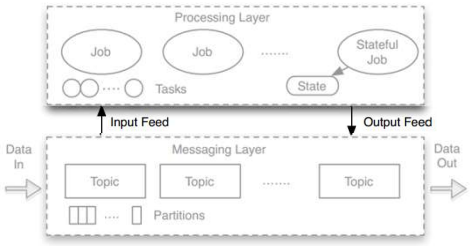


FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Descrição da arquitetura

Comunicação entre as camadas

- Ocorre pela leitura e pela escrita em tópicos da camada de mensagem;
- Os dados "golden record" são primários, isto é, não gerados no sistema;
- Os dados derivados são resultados de processamentos dos *golden records* ou de outros dados derivados;

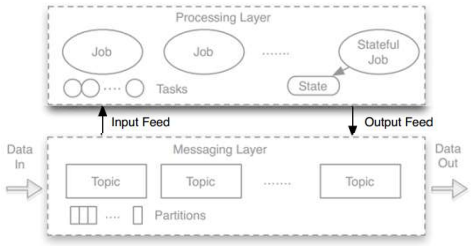


FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR '15).

Liquid – Descrição da arquitetura

Comunicação entre as camadas

- Dados derivados contêm a linhagem dos dados, isto é, metadados sobre como os dados foram processados, os quais são armazenados na camada de mensagem;
- A camada de processamento precisa poder acessar dados com base nestes metadados, e esta camada também produz tais metadados e os escreve na camada de mensagem.



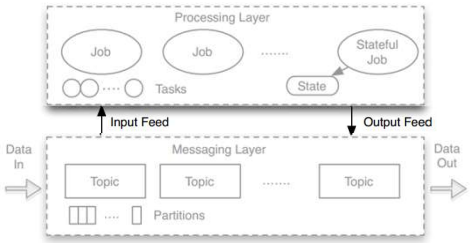
Os sistemas de back-end leem os dados dos feeds de entrada, depois que o Liquid os pré-processou para atender aos requisitos específicos do aplicativo. Esses trabalhos são executados pela camada de processamento, que lê os dados dos feeds de entrada e envia os dados processados para novos feeds de saída.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Descrição da arquitetura

A divisão em camadas

- É fundamental para arquitetura;
- Mantém produtores e consumidores totalmente desacoplados;
- Um *job* da camada de processamento pode consumir dados mais lentamente que a taxa com que um produtor produz este dado sem que os desempenhos de um e de outro sejam afetados;
- Permite que a administração e o desenvolvimento sejam separados.



FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Propriedades

Baixa latência

O acesso rápido aos novos dados deve ser o comportamento padrão. A arquitetura Liquid oferece acesso online para atender aos requisitos de sistemas em tempo quase-real e também oferece dados para sistemas de processamento *batch* processarem *offline*.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Propriedades

Processamento incremental

A arquitetura oferece processamento incremental. Ela adiciona notas aos dados com metadados como *timestamps* ou versões de software, as quais sistemas *back-end* podem usar para ler os dados a partir de um dado ponto. Esta propriedade de reprocessamento é um bloco crucial para o processamento incremental e para a recuperação de falhas.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Propriedades

Alta disponibilidade

A arquitetura Liquid oferece a versão de ouro dos dados que diferentes sistemas back-end consumem. Por esta razão os dados precisam ter alta disponibilidade para as leituras e escritas em clusters tolerantes a falhas.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Propriedades

Isolamento de recursos

A arquitetura Liquid permite que processos de extração de dados do tipo ETL sejam executados centralmente ao invés de serem gerenciados por times independentes dentro da organização. Isto facilita o controle de recursos para as tarefas e a garantia de funcionamento destas tarefas.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).

Liquid – Propriedades

Efetividade de custos

A arquitetura precisa armazenar um alto volume de dados que entram, garantindo um alto *throughput* e mantendo baixos custos operacionais. Isto significa que a arquitetura não pode manter todos os dados em memória RAM para reduzir a latência de acesso.

Throughput significa a quantidade de materiais ou itens que passam através de um sistema ou processo, e no contexto computacional pode ser interpretado como taxa de transferência.

FERNANDEZ, R. et al., **Liquid: Unifying Nearline and Offline Big Data Integration**. 7th Biennial Conference on Innovative Data Systems Research (CIDR ’15).