Capítulo 12

O futuro dos sistemas de dados

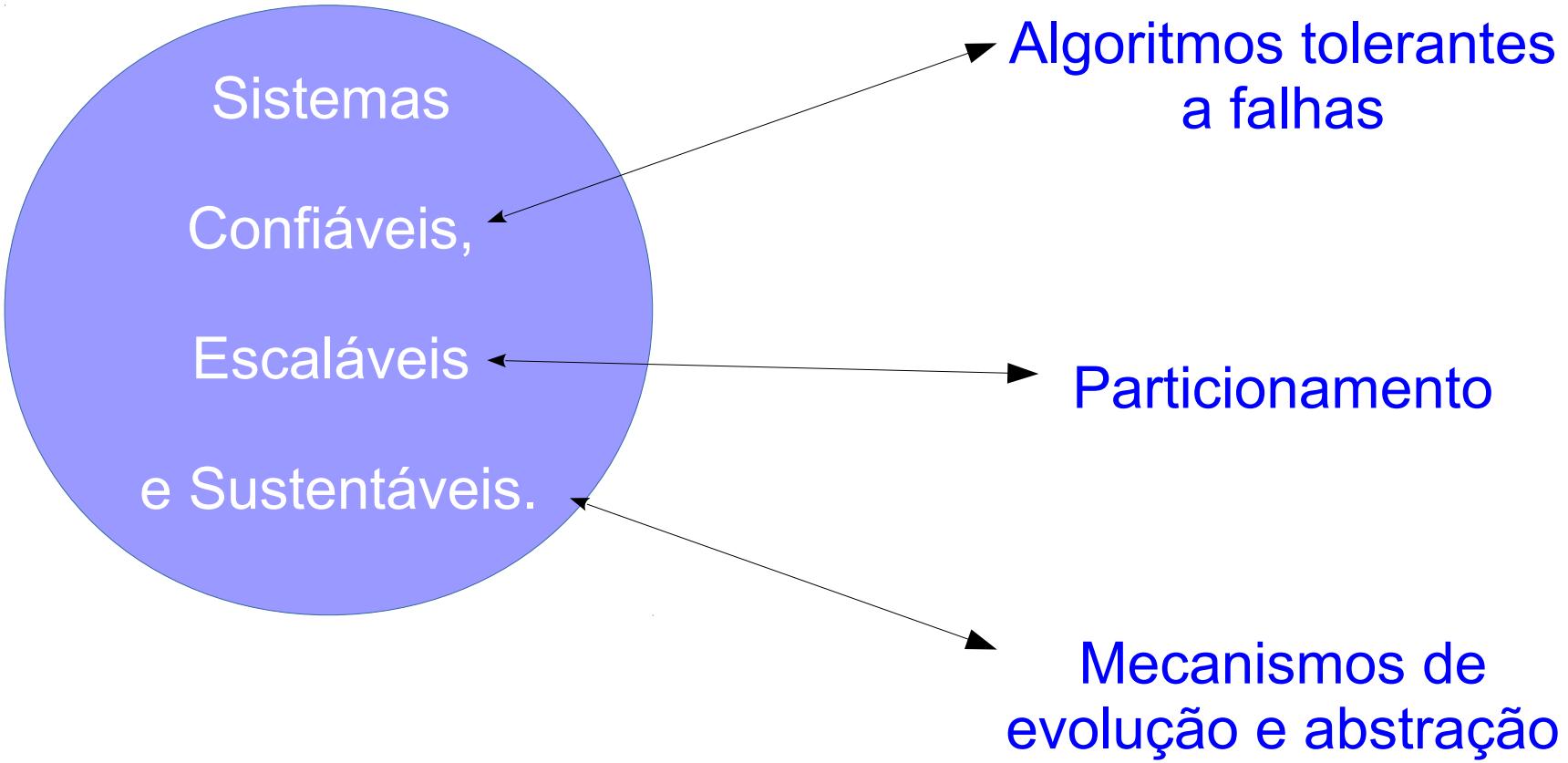
Aluna: Marina A. H. Pimentel
marina@inf.ufpr.br

Sumário

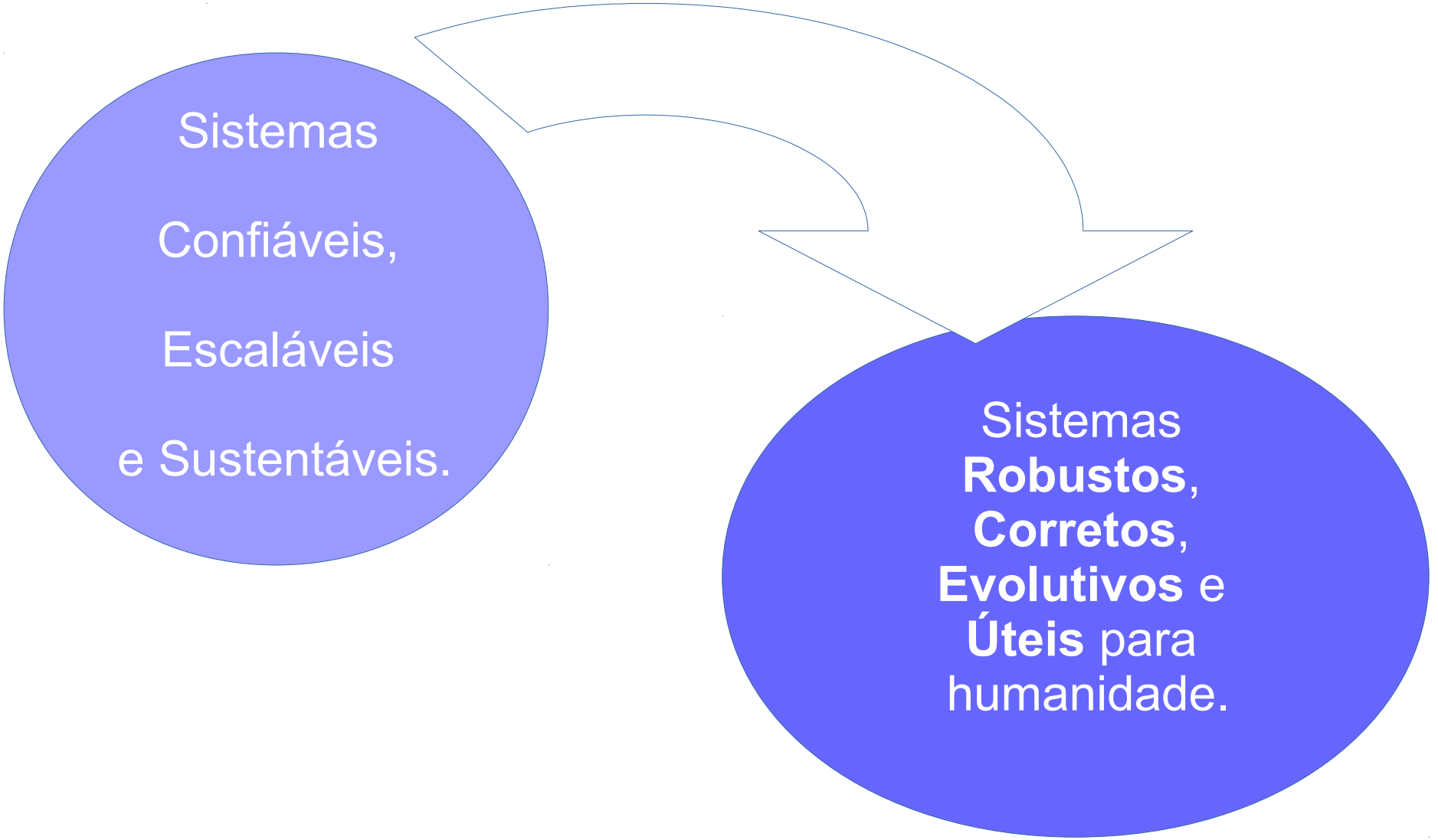
1. Introdução
2. Integração de dados
3. Bases de dados derivadas
4. Visando exatidão
5. Fazendo a coisa certa
6. Perguntas

1. Introdução

1. Projetando sistemas



1. Projetando sistemas



2. Integração de dados

2. Integração de dados

Um problema → diversas soluções

1º desafio:
Mapeamento **Software** x **Uso apropriado**

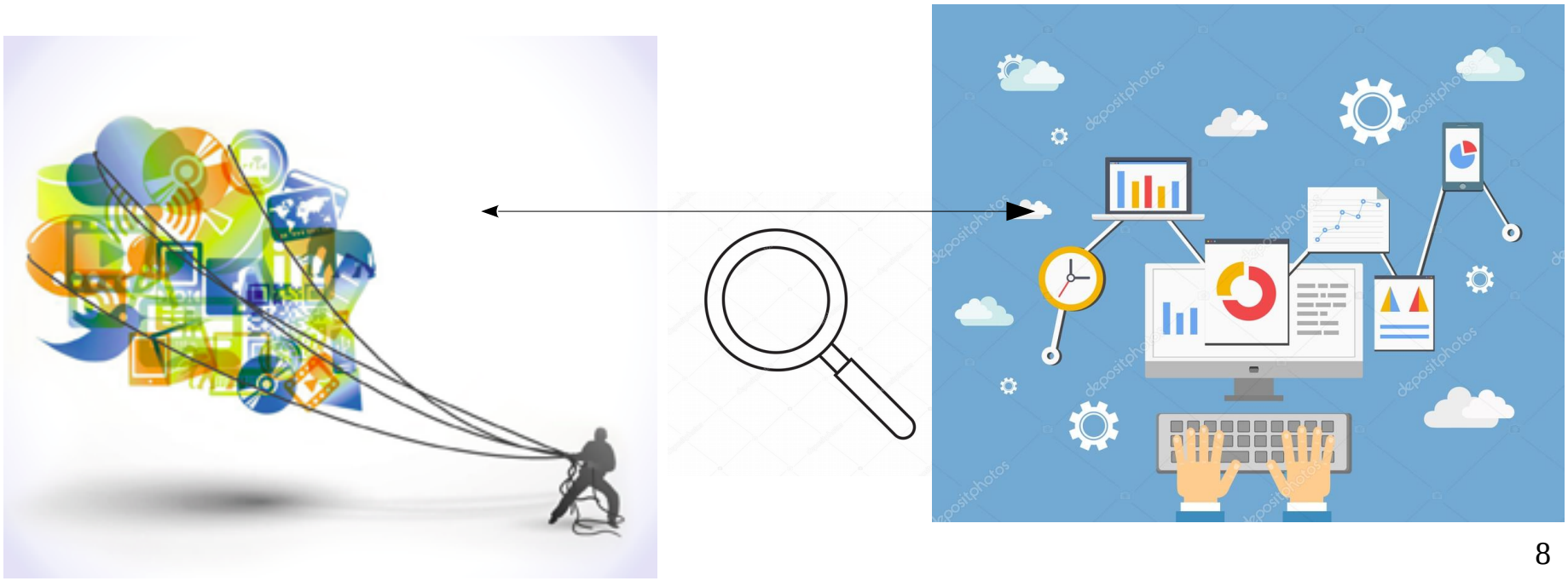
2º desafio:
Dados x **diferentes maneiras de uso**

2. Fluxo de dados

Combinação de ferramentas especializadas com dados derivados.

Ex. de aplicativo: OLTP DB + Dados em memória + Índice de pesquisa de texto completo + OLAP DB

Fluxo de dados → Integração de dados



2. Fluxo de dados

Diferentes padrões de acesso aos dados → cópia dos dados em diversos sistemas de armazenamento.

Mas...

1. Onde gravar os dados primeiro?
2. Quais representações derivam de quais fontes?
3. Como manter os dados nos lugares corretos?
4. Como manter formato correto?

2. Fluxo de dados

Exemplos

Caso 1
1. Aplicação grava no sistema de Banco de Dados
2. Captura de Mudança nos Dados (CDC) aplica a mudança no índice de busca na mesma ordem.

Caso 2
Todas as escritas passam por um sistema único que decide pela ordem de todas as escritas (broadcast atômico).

Dados consistentes.

2. Fluxo de dados

Caso 3
1. Aplicação grava no Banco de Dados
2. e também no índice de busca.

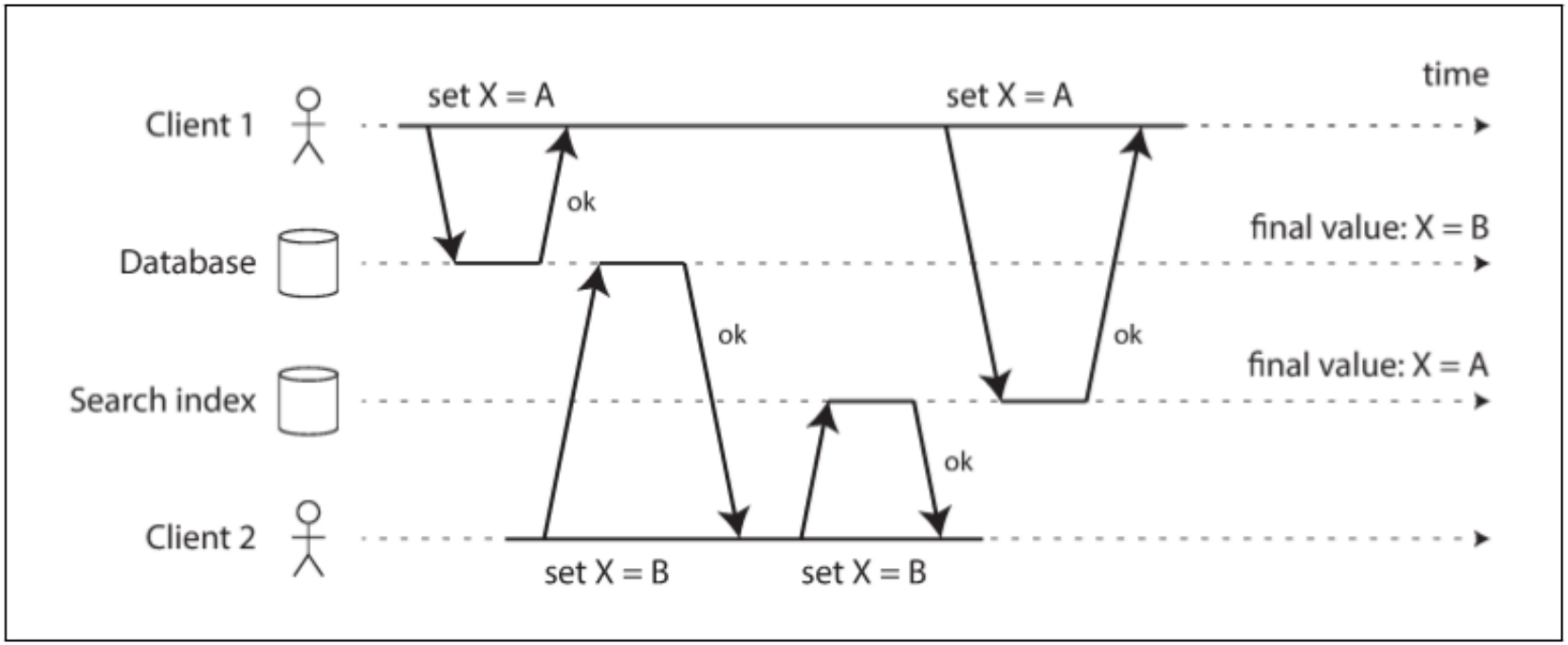


Figure 11-4. In the database, X is first set to A and then to B, while at the search index the writes arrive in the opposite order.

Escritas conflitantes e concorrentes podem gerar inconsistências permanentes entre bases.

2. Transações distribuídas x Dados derivados

Sist. transações distribuídas	**Sitemas de dados derivados**
Ordem de escrita usando lock exclusão mútua (2PL)	CDC e event sourcing - Ordem usando log
Escrita com commit atômico – Garantia de escrita única	Escrita baseadas em log – escritas determinísticas e idempotentes
Linearizabilidade (leitura da própria escrita)	Atualização assíncrona - (possibilidade de leitura de dados sujos)
Protocolo com baixa tolerância a falhas e desempenho	Abordagem mais promissora - melhor se possibilitar leitura da própria escrita

2. Ordenamento x causalidade

Ordenamento total = broadcast atômico ≡ consenso.
Algoritmos de consenso → dificuldade no ordenamento compartilhado.

- João e Maria são amigos, mas depois terminam namoro.
- Maria exclui amizade com João.
- Maria envia msg reclamando de João aos demais amigos.
- (Status amizade e mensagens armazenados em locais distintos → Dependência causal pode não ser capturada).
- João também recebe a mensagem de Maria.

Necessidade de padrões que permitam:

- capturar dependências causais
- manter estado derivado corretamente
- não depender de broadcast atômico (gargalo)

2. Processamento em lote e fluxo

Integração de dados - objetivo é garantir que os dados terminem:

- no formato certo
- em todos os lugares corretos.

Requisitos:

- Consumo de entradas
- Transformação
- Junção
- Filtragem
- Agregação
- Treinamento de modelos
- Avaliação
- Escrita de saídas apropriadas

{ **Processadores em lote e fluxo**

Saídas: conjuntos de dados derivados.

Principal diferença:

- Processamento em lote - entradas de tamanho finito e conhecido.
- Processamento de fluxo - conjunto de dados sem limites.

2. Manutenção do estado derivado

Como propagar a mudança de estado em um sistema com dados derivados?

1) De forma síncrona.
2) De forma assíncrona - baseado em log de eventos – mais robusto (tolerante a falhas).

2. Reprocessando dados para evolução

Caso de migração de esquema em ferrovias

1. Ferrovias na Inglaterra usavam diferentes bitolas.
2. Restrição nas possíveis interconexões na rede de trens.
3. Em 1846 houve padronização da bitola.
4. Como migrar bitolas fora do padrão sem interromper as linhas durante meses ou anos?
5. Primeiro, converter a ferrovia para bitola mista: adicionando um terceiro trilho gradativamente.
6. Quando todos os trens fossem convertidos para a bitola padrão, o trilho fora do padrão poderia ser removido.

Permitir a coexistência da versão velha com a nova possibilita uma mudança gradativa.

2. Reprocessando dados para evolução

Sem reprocessamento: permite evolução simples (adição de campo opcional).

Reprocessar dados: bom mecanismo para evoluções mais complexas.

Processamento de fluxo e em lote possibilitam manter dados derivados:

- fluxo: reflete mudanças recentes com pouco atraso
- lote: processa grande quantidade de dados históricos derivando novas visões.

Visões derivadas permitem uma evolução gradual:

- Sem necessidade de migrações bruscas
- Manutenção do esquema velho e novo lado a lado
- Em qualquer estágio pode-se reverter a migração facilmente
- Incremento gradual acessando novo esquema

2. Unificando processamento em lote e de fluxo

Trabalhos recentes permitem reprocessamento de dados históricos e computação de fluxo num único sistema:

- Reprocessamento de eventos históricos usando mesmo motor que processa o fluxo de eventos recentes.
- Semântica de execução única, mesmo em caso de falhas.
- “Janelamento” por tempo de evento.

3. Bases de dados derivadas

3. Meta-Banco de dados de tudo

Não há modelo de dados ou formato de armazenamento único que seja apropriado para todos os padrões de acesso.

Modelo proposto para a composição de um sistema coeso:

- **Banco de dados federado** (polystore): unificando leituras
  - Segue tradição do banco relacional
  - Interface de consulta única e integrada
  - Linguagem de consulta de alto nível
  - Semântica elegante
  - Implementação complicada
- **Banco de dados desagregados**: unificando escritas
  - Segue tradição do Unix
  - Ferramentas simples
  - Fazem bem uma coisa
  - Comunicação através de API de baixo nível (pipes)
  - Composição usando linguagem de alto nível (shell)

3. Trabalho de derivação ou desagregação

Abordagem atual para escrita sincronizada: transações distribuídas com armazenamento heterogêneo.

Abordagem mais robusta e prática usando log de eventos assíncrono com escritas idempotentes:

- interface poderosa o suficiente para garantir forte consistência;
- genérica suficiente para quase todo tipo de dado;
- baixo acoplamento:
  - eventos assíncronos - mais robustez em interrupções e degradação de performance de componentes individuais.
  - desagregação do sistema de dados permite administração dos componentens de maneira independente e por diferentes times.

3. Composição de tecnologias - O que falta?

Banco de dados derivados equivalentes a shell Unix, com linguagem de alto nível simples e declarativa.

Mysql | elasticsearch

Seria equivalente a:

- pegar todos os documentos MySQL
- indexá-los no Elasticsearch
- capturar e propagar mudanças de um para o outro de forma contínua e automática.

3. Projetando aplicações com fluxo de dados

Planilha eletrônica: capacidade de programação baseada em fluxo de dados.

Função em uma célula: mudança em qualquer dado usado na fórmula, o resultado é recalculado automaticamente.

É exatamente o que gostaríamos num sistema de dados:

- quando um registro em um banco de dados mudar;
- queremos atualização automática de todos os índices relacionados;
- atualização automática de agregações e visões.

3. Separando estado e o código da aplicação

Tendência: não colocar lógica da aplicação no banco de dados e não colocar persistência de estado na aplicação.

Banco de dados:

- Variável compartilhada mutável e durável;
- Acessada através da rede com controles de concorrência e tolerância a falhas;
- Só se descobre mudança nos dados executando consultas periodicamente.

Aplicação:

- Lê e atualiza a variável;
- pode-se programar o padrão observador caso não possa se increver para receber notificação de mudanças em variáveis mutáveis.

3. Fluxo de dados: mudança de estado x código aplicação

Trigger, ou atualização automática de índice secundário: é o que deveria ocorrer com banco de dados derivados.

Usar processamento de fluxo ou sistemas de mensagens, onde:

- A ordem da mudança de estados é importante;
- Tolerância a falhas é fator chave – perder uma única mensagem pode tornar o dado derivado fora de sincronismo para sempre.

Similar a pipes Unix, operadores de fluxo podem ser utilizados para compor grandes sistemas baseado no fluxo de dados.

3. Serviços x Processadores de fluxo

Sistema de compras

Compra é feita em uma moeda e o pagamento é feito em outra moeda.

Conversão de moedas: fornecer taxa de conversão atualizada entre as moedas.

Microserviços (REST/API): o código que processa a compra consulta o banco de dados ou um serviço de taxa de câmbio, de forma síncrona.

Fluxo de dados:

- fluxo de atualizações das taxas de câmbio é enviado de forma assíncrona para os incritos.
- Sistema inscrito recebe e armazena a taxa atualizada num banco local.
- Ao processar uma compra, basta consultar localmente.

Questões em abertos: junções dependentes do tempo.

3. Observando estados derivados

Caminho da escrita: a mudança na informação deve-se propagar até os dados derivados, através de processamento em lote ou de fluxo.

Caminho da leitura: leitura nas bases de dados derivados para atender requisições do usuário.

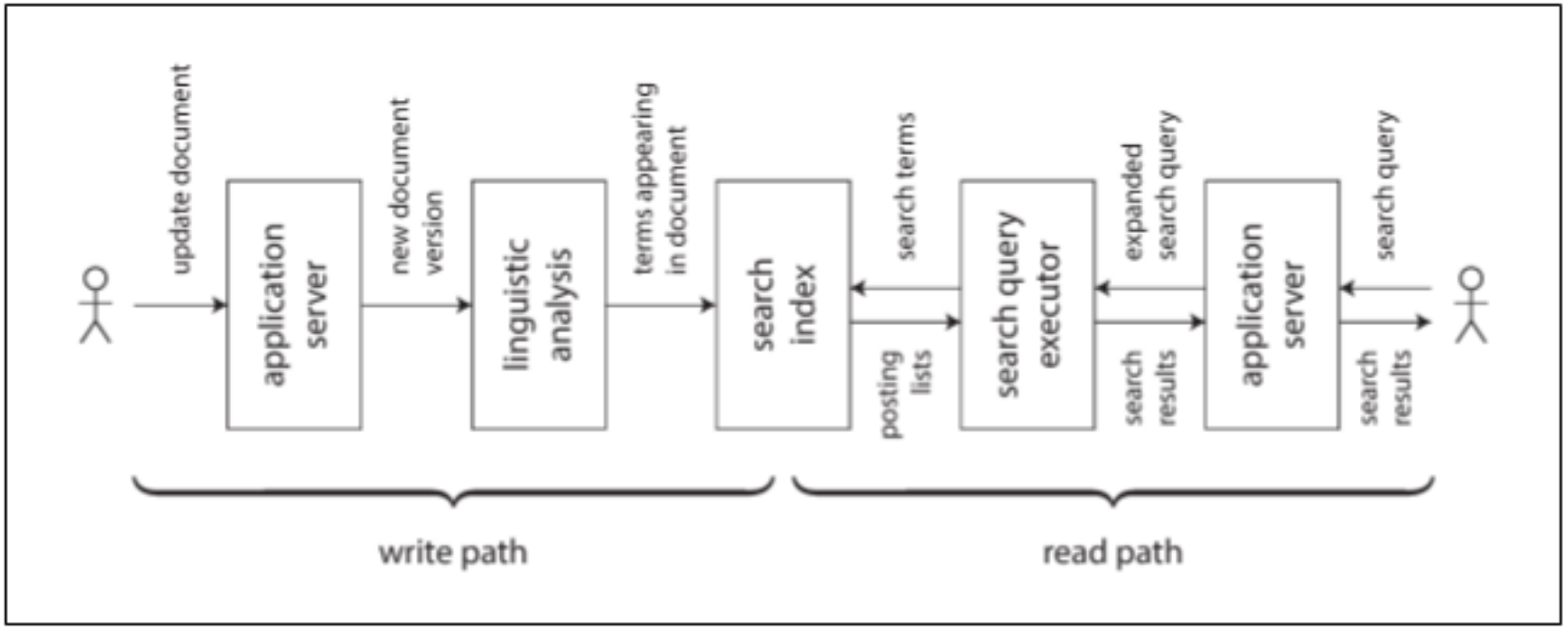


Figure 12-1. In a search index, writes (document updates) meet reads (queries).

3. Visões materializadas e caching

Exemplo: índice de pesquisa de texto completo.

- Caminho de escrita atualiza o índice.
- Caminho de leitura busca palavras no índice.
- Ambos precisam realizar algum trabalho.
- Menos trabalho em um dos lados significa mais trabalho do outro lado.

Outra opção:

- pré-computar os resultados das pesquisas mais comuns ;
- colocá-los em cache (visões materializadas), sem buscá-los no índice.

Papel de índices, caches, visões materializadas é:

- mudar o limite entre o caminho de escrita e o caminho de leitura;
- mais trabalho no caminho de escrita menos esforço no caminho da leitura.

3. Clientes com estado e capacidade offline

Modelo cliente-servidor: clientes sem estado e servidores com autoridade sobre os dados.

Existem outras abordagens?

Aplicativos web e mobile com:

- capacidade de guardar estados
- interface de interação no lado cliente
- armazenamento persistente local no web browser.

Aumento do interesse em aplicativos offline que:

- realizam o máximo possível usando a base local sem conexão com internet;
- sincronização com servidores remotos em background quando houver a conexão disponível.

3. Empurrando as mudanças de estado para o cliente

Típica aplicação web:

- estado em cache estático;
- para atualizar, cliente busca por mudanças no servidor.

Protocolos mais recentes:

- vão além do tradicional HTTP: API EventSource e WebSockets;
- browser mantém conexão TCP com servidor;
- servidor ativamente envia mensagens para o browser, informando mudanças de estado ocorridas nos dados;
- cliente armazena localmente;
- extende o caminho de escrita até o cliente.

3. Eventos de fluxo ponta-a-ponta

Abordagem onde a mudança de estado flui de ponta-a-ponta:

- Um dispositivo dispara uma mudança de estado;
- A mudança é adicionada ao log de eventos;
- A mudança reflete-se nos sistemas de dados derivados;
- Processadores de fluxo enviam mensagem da mudança até outro usuário que observa o estado em outro dispositivo.

Exemplos reais: mensagens instantâneas, jogos online.

Nova filosofia: mudar da interação requisição/resposta para fluxo de dados publicação/inscrição.

Desafio: além da cultura arraigada, poucos bancos de dados fornecem a funcionalidade de inscrição.

3. Leituras também são eventos

Abordagem atual:

- escritas passam por um log de eventos;
- leituras vão diretamente para os nodos que contém o dado requerido.

Outra forma:

- considerar leituras como fluxo de eventos;
- tratá-las por meio do processador de fluxo emitindo o resultado num fluxo de saída.

Vantagens de registrar um log de eventos de leitura:

- rastreamento de dependências causais;
- proveniência dos dados;
- permite reconstrução do que o usuário viu antes de tomar uma determinada decisão.

Dificuldades: registrar as leituras geram mais operações de I/O e custo de armazenamento adicional.

3. Processamento de dados multi-particionados

Para aplicações que usam uma única partição, usar processamento de fluxos talvez seja um exagero.

Entretanto a infraestrutura de roteamento de mensagens, particionamento e junção fornecido por processadores de fluxo viabilizam consultas complexas que compõe dados de várias partições.

Exemplo: calcular nr de pessoas que viram uma URL no Twitter.

4. Visando exatidão

4. Visando exatidão

Sistemas com estado, tais como banco de dados, devem lembrar das coisas “para sempre”.

Se algo der errado durante uma transação, os efeitos podem potencialmente durar para sempre.

Se a aplicação necessita de fortes garantias de integridade, então serializabilidade e commit atômico são as abordagens estabelecidas,

MAS isso tem seus custos: normalmente operam em um único datacenter e limitam a escalabilidade e a tolerância a falhas.

4. Argumentos ponta-a-ponta para Banco de Dados

O uso de sistemas de dados com propriedades fortes de segurança, como transações serializáveis, não significa que a aplicação está garantida contra perda ou corrupção dos dados.

Bugs no código e erros humanos ocorrem:

- por isso dados imutáveis e append-only facilitam a recuperação e correção;
- melhor então tirar a capacidade de códigos defeituosos destruírem dados corretos.

4. Execução única de uma operação

Se ocorre um erro durante o processamento de uma mensagem, pode-se:

- desistir e remover a mensagem, incorrendo em perda de dados ou;
- reprocessar, com o risco de duplicidade.

Processamento duplicado também é dado corrompido.

Abordagem mais efetiva: tornar a **operação idempotente**.

Requer esforço e cuidado: manter metadados adicionais (conjunto de IDs das operações que atualizaram o valor) e assegurar não progagação de falhas.

4. Identificadores de operação

Operação idempotente:

- necessário controle ponta-a-ponta do fluxo da requisição,
- uso de identificador único (ou hash dos campos do formulário) como identificador da operação.

```
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance + 11.00 WHERE account_id = 1234;
UPDATE accounts SET balance = balance - 11.00 WHERE account_id = 4321;
COMMIT;
```

```
ALTER TABLE requests ADD UNIQUE (request_id);
BEGIN TRANSACTION;
INSERT INTO requests
(request_id, from_account, to_account, amount)
VALUES('0286FDB8-D7E1-423F-B40B-792B3608036C', 4321, 1234, 11.00);
UPDATE accounts SET balance = balance + 11.00 WHERE account_id = 1234;
UPDATE accounts SET balance = balance - 11.00 WHERE account_id = 4321;
COMMIT;
```

4. Restrição de unicidade requer consenso

O identificador de operação resolve remoção da duplicação ponta-a-ponta.

Mas e os outros tipos de restrições?

Configurações distribuídas: restrição de unicidade por consenso (líder único), implicando em gargalo.

Unicidade escalável: particionamento baseado no valor único.

Se ID da operação deve ser único, todas as operações com mesmo ID devem ser roteadas para a mesma partição.

Replicação assíncrona multi-líder: não garante unicidade.

Diferentes líderes concorrentemente podem aceitar escritas conflitantes.

4. Unicidade em mensagens baseadas em log

Logs garantem que todos os consumidores vejam as mensagens na mesma ordem – broadcast total – equivalente a consenso.

Vários usuários requerem o mesmo username;

1. toda requisição é codificada como mensagem e adicionada na partição correspondente ao hash do username;
2. o processador lê sequencialmente as requisições na log, usando uma base local para registrar os usernames utilizados. Se username está disponível envia mensagem de sucesso para um fluxo de saída.
3. o cliente observa o fluxo de saída e aguarda pela mensagem de sucesso ou rejeição.

Esta abordagem funciona para outros tipos de restrição.

Princípio fundamental: escritas conflitantes são roteadas para a mesma partição e processadas sequencialmente.

4. Processando requisições multi-particionadas

```
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance + 11.00 WHERE account_id = 1234;
UPDATE accounts SET balance = balance - 11.00 WHERE account_id = 4321;
COMMIT;
```

Banco de dados tradicionais: requer commit atômico envolvendo todas as partições, com coordenação de partição cruzada, e com prejuízo na taxa de transferência.

Solução com logs particionadas e sem commit atômico:

1) Logs particionados com base no ID (único) das requisições;
2) transferência ganha ID único e é adicionada ao log da partição correspondente;
3) processador de fluxo lê as mensagens no log;
4) então emite 2 mensagens no fluxo de saída: uma instrução de débito para o pagador A (partição A), e outra instrução de crédito para o beneficiário B (partição B). O ID da requisição original é incluído nessas mensagens.
5) então os processadores consomem os fluxos de instruções de crédito e débito, deduplicado pelo ID da requisição, aplicando as mudanças no saldo das contas.

4. Pontualidade e integridade

Pontualidade: assegurar que o usuário observe o sistema em um estado atualizado.

Integridade: ausência de corrupção (sem perda de dados e sem dados falsos ou contraditórios).

Violação de pontualidade pode ser chato e confuso.
Mas a violação de integridade pode ser catastrófico.
Integridade é mais importante que pontualidade.

Exemplo:
É ruim não aparecer os lançamentos mais recentes no extrato de cartão de crédito,
mas é muito pior se o valor de uma compra for debitado de seu cartão e não for pago para o credor.

4. Integridade dos sistemas de fluxo de dados

Mecanismos para manutençaõ de integridade:

- conteúdo da operação de escrita em **mensagem única**, escrita **atomicamente**;
- a partir dessa mensagem única, enviar todas as outras atualizações de estado através de **funções de derivação determinísticas**
- **propagar o ID** da requisição inicial **de ponta-a-ponta**, permitindo **abolição de duplicação e idempotência**
- tornar **mensagens imutáveis**, permitindo que dados derivados possam ser **reprocessados** de tempos em tempos.

4. Restrições não tão rígidas

Forçar restrição de unicidade requer consenso → gargalo nodo único.

Em muitos casos, é aceitável violar restrições temporariamente e utilizar os procedimentos de pedido de desculpas e recompensas.

O custo disto normalmente é baixo.

Noções mais fracas de unicidade:

- transação de compensação (pedir desculpas e oferecer outra alternativa);
- se vendeu mais do que tem em estoque (compra no concorrente, pede desculpas pelo atraso e oferece um desconto);
- muitas companhias aéreas e hotéis vendem mais que a capacidade real, na esperança de que alguns clientes cancelem ou não apareçam.

5. Fazendo a coisa certa

5. Fazendo a coisa certa

Em que tipo de mundo queremos viver?

- Todo sistema é construído para um propósito; toda decisão tem sua consequências, intencionais ou não.
- Dados (abstratos) são sobre pessoas: seu comportamento, seus interesses, sua identidade.
- Tratar dados e usuários com humanidade e respeito.
- Ética no desenvolvimento de software: Código de ética e prática profissional na engenharia de software da ACM.
- Ética: raramente discutida, aplicada ou obrigatório na prática.

A tecnologia por si só não é boa nem má – importante é como ela é usada e como afeta as pessoas.

5. Análise preditiva

Análise preditiva: maior promessa em Big Data.

Algoritmos de tomada de decisão: cada vez mais difundidos.

Prisão algorítmica: um indivíduo é classificado como de risco.

Então passa a sofrer sérias consequências: descartado para entrevistas, companhias aéreas, cobertura de seguro, previdência, serviços financeiros e em outros aspectos chave na sociedade.

Sistema de justiça criminal: presume inocência até que se comprove a culpa.

Sistemas automatizados: podem “culpar” pessoas sem nenhuma prova da culpa.

5. Preconceito e discriminação

Sistemas analíticos preditivos:

- regras inferidas a partir dos dados;
- caso haja preconceito nas entradas, o sistema aprenderá e amplificará o preconceito nas saídas;
- padrões aprendidos são obscuros;
- automatiza decisão humana.

Dados e modelos são nossas ferramentas, não nossos guias.

5. Responsabilidade e prestação de contas

Quando um algoritmo comete erros, quem será responsabilizado?

Ex: se um carro autônomo causa um acidente, de quem é a culpa?

Se a decisão tomada pelo seu sistema de aprendizado de máquina cai sob revisão judicial, você conseguirá explicar ao juiz como o algoritmo tomou a decisão?

Como construir algoritmos confiáveis e transparentes, evitando reforçar preconceitos existentes?

Como prevenir que os dados sejam usados para prejudicar pessoas?

5. Loops de feedback

O sistema reforça e amplifica as diferenças existentes entre as pessoas ou tenta combater as injustiças?

Tornar o rico mais rico e o pobre mais pobre.

Sistema de contratação de pessoas usando score de cartão de crédito.

Indivíduo bom trabalhador e bom pagador.

De repente passa por dificuldades financeiras fora de controle.

Score é afetado.

Chance de contratação diminui.

Score torna-se pior.

Menos chance de ser contratado.

E assim por diante.

5. Privacidade, rastreamento e uso dos dados

Privacidade: liberdade de escolher o que deve ser revelado para quem; o que manter secreto e o que é público.

Problemas éticos na coleta de dados dos usuários.

Rastreamento de dados comportamentais.

Termos de serviço e política de privacidade: não há como negociar o quanto de dados deve ser fornecido. Os termos são configurados por serviço e não por usuário.

Serviços gratuitos x publicidade

Os publicitários podem ser os reais clientes, e os interesses dos usuários estão em segundo plano.

5. Vigilância x Valor dos dados

Análise de dados pode revelar coisas surpreendentemente intrusivas.

Ex: O sensor de movimento em um smartwatch ou rastreador fitness pode ser usado para descobrir o que você está digitando (por exemplo, senhas) com boa precisão.

Infraestrutura de vigilância: se a publicidade financia um serviço, então dados comportamentais são o ativo (**ativos tóxicos**) mais valioso do serviço.

Corretores de dados: indústria obscura trabalhando secretamente, comprando, agregando, analisando, inferindo, e revendendo dados sobre as pessoas.

Conhecimento é poder. Por isso todos os governos querem a vigilância: isso dá-lhes **poder de controlar** a população.

5. Relembrando a revolução industrial

Era da informação

Internet, armazenamento de dados, processamento, e automação controlada por software: impacto na economia global e na sociedade humana.

Revolução industrial: avanços tecnológicos e agrícolas, trouxeram crescimento econômino e melhorias por muito tempo.

Mas surgiram os problemas: poluição do ar e da água. Desigualdade social.

Bruce Scheneier:

“Dados são o problema de poluição da era da informação, e proteger a privacidade é o desafio ambiental...”

5. Legislação e auto-regulação

Diretiva européia de proteção dos dados de 1995: dados pessoais devem ser coletados para **propósitos específicos**, **explícitos** e **legítimos** e não podem ser posteriormente processados de maneira incompatível com estes propósitos. E além disso os dados devem ser **adequados**, **relevantes** e **não excessivos** em relação ao **propósito para o qual foram coletados**.

Big data: **maximizar** coleta de dados, combinar com outros conjuntos de dados, realizar experimentos e **explorar** para gerar novas visões.

Explorar significa usar os dados para **propósitos não previstos**, o oposto de “específicos e explícitos”.

Leis de proteção dos dados devem ser capazes de preservar os direitos individuais.

6. Perguntas

6. Perguntas

Questão 1 – Considerando a abordagem de sistemas baseados em dados derivados: cite dificuldades e benefícios obtidos e os reflexos dessa abordagem no caminho de leitura e escrita dos dados.

Questão 2- Explique o que é análise preditiva e como os algoritmos de tomada de decisão podem afetar a humanidade.

Aluna: Marina A. H. Pimentel
E-mail: marina@inf.ufpr.br