Capítulo 11
Processamento de Fluxo

Matheus Rotondano de Camargo

Sumário

Introdução

Introdução

- *Batch Processing*
  - É presumido que o conjunto de entrada é de um tamanho finito e conhecido, para que se saiba quando a entrada terminou.
  - Em uma aplicação real muitos conjuntos de dados não tem um tamanho definido pois eles são recebidos gradativamente. Usuários produziram dados ontem, produzem hoje e produzirão amanhã.
  - Isso significa que um dataset nunca está “completo”, e por isso o *Batch Processing* deve dividir os dados em blocos de tamanho específico (processar dados referentes a um dia ou hora, por exemplo)
  - O problema disso é que as mudanças da entrada podem demorar para serem refletidas na saída.
  - Para reduzir o atraso nós podemos fazer o processo mais frequentemente… como na medida em que os eventos ocorrem.

Introdução

- Processamento de fluxo
  - Fluxo se refere a dados que se tornam disponíveis com o passar do tempo.
  - Se assemelha com *Batch Processing*, porém se abandona o conceito de bloco e se processa os eventos na medida em que eles ocorrem.

Transmissão de fluxo de dados

Transmissão de fluxo de dados

- Em processamento de fluxo as entradas são tratadas como arquivo binário, que é analisado em uma sequência de registros chamados de eventos.
  - Um evento contém um *timestamp*
  - Um evento pode ser uma ação do usuário ou originado de uma máquina
  - Um evento pode ser codificado em uma string, em JSON ou em binário
- A princípio um arquivo ou banco de dados é suficiente para conectar produtores e consumidores.
  - Um produtor escreve todo evento gerado em um *datastore* e cada consumidor checa o *datastore* periodicamente para checar novos eventos
- Quando o processamento é feito com muita frequência, busca por novos eventos se torna caro .

Transmissão de fluxo de dados

- Por causa disso é melhor que os consumidores sejam notificados quando um novo evento ocorre.
- Tradicionalmente banco de dados não suportam um mecanismo de notificação muito bem, por isso foram desenvolvidas ferramentas especializadas em enviar notificações de eventos.
- Por causa disso vamos ver como funciona o sistema de mensagens de notificação.

Sistemas de mensagens

Sistemas de mensagens

- Diferentes sistemas adotam uma gama de abordagens no modelo de publicação/inscrição e não existe uma abordagem correta para todas as situações.
- Para escolher uma abordagem é útil fazer duas perguntas:
  - **O que acontece se o produtor envia mensagens mais rápido do que o consumidor pode processar?**
    - Nessa situação o sistema pode *droppar* mensagens, guardar as mensagens em uma fila ou aplicar *backpressure* (impedir que o produtor envie novas mensagens)
  - **O que acontece se um nodo falha ou fica offline?**
    - Se não há problema de às vezes perder mensagens pode-se optar por uma taxa de transferência maior e uma latência menor.

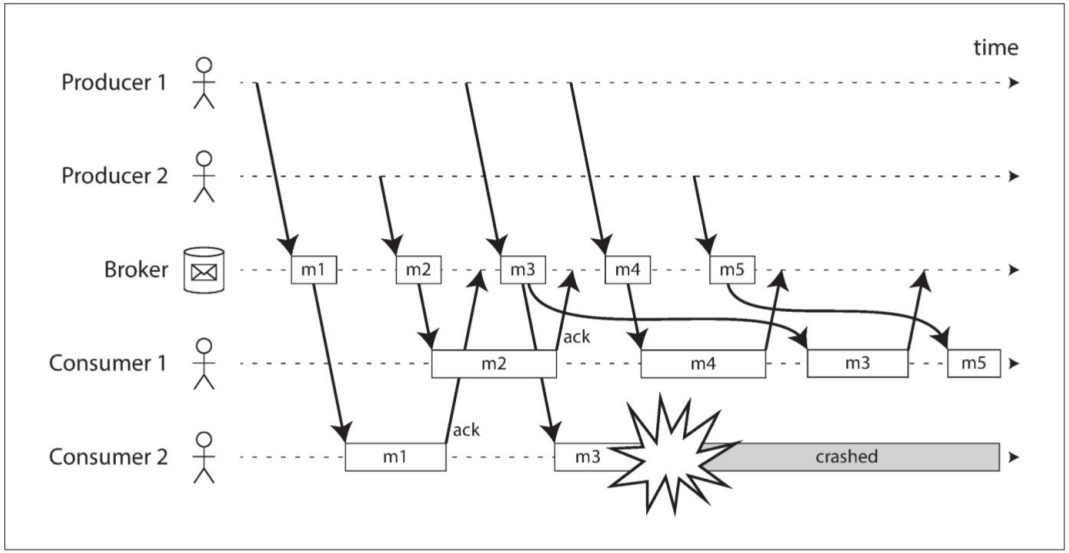
Sistemas de mensagens

- Mensagens diretas:
  - Funcionam bem em sistemas específicos
  - Geralmente é necessário que o código da aplicação esteja ciente da possibilidade de perda de mensagens.
  - Assumem que o produtor e consumidor estão sempre online.
  - Alguns protocolos permitem que o produtor tente reenviar mensagens.

Sistemas de mensagens

- Fila de mensagens:
  - Alternativa muito mais utilizada que mensagens diretas.
  - É essencialmente um banco de dados otimizado para aguentar fluxos de mensagens.
  - Existem dois grandes tipos de filas de mensagem:
    - **AMQP/JMS**
      - Designa mensagens individuais para os consumidores e os consumidores confirmam o recebimento de cada mensagem quando eles terminam de processá-la com sucesso.
      - Mensagens são deletadas da fila quando o recebimento é confirmado.
      - Se a conexão com um cliente é encerrada ou ocorre um *time out* antes do recebimento do *ack*, se assume que a mensagem não foi processada e ela é reenviada para outro consumidor.
      - Isso pode gerar o seguinte problema:

Sistemas de mensagens



- Consumidor 2 *crasha* processando m3 enquanto o consumidor 1 processa m4.
- Como o *ack* de m3 não foi recebido, a mensagem é reenviada para outro consumidor.
- Por causa disso o consumidor 1 recebe as mensagens na ordem m4, m3, m5. Ou seja, as mensagens estão fora de ordem.

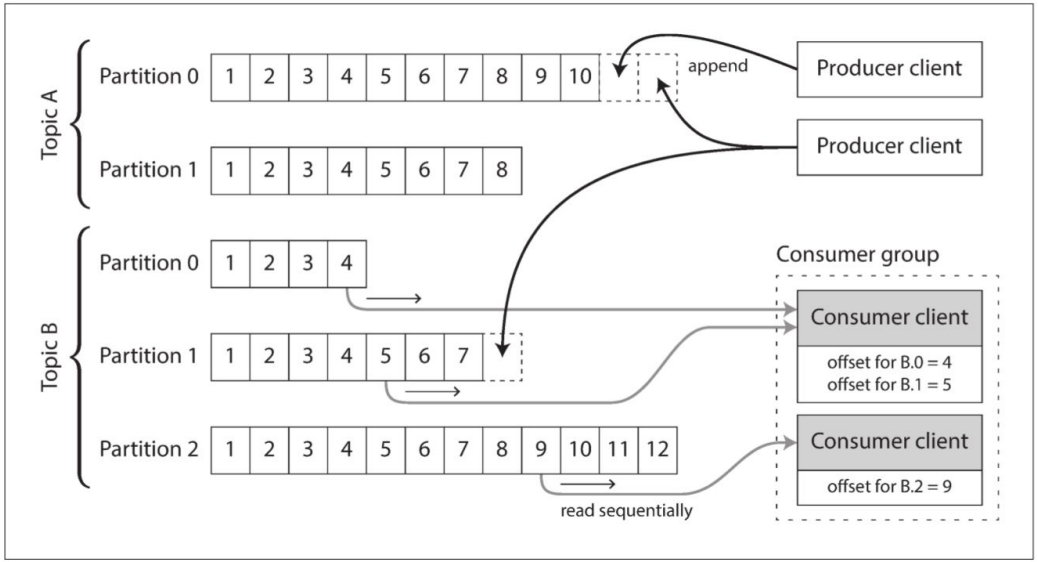
Sistemas de mensagens

- Por causa desse problema a abordagem AMQP/JSM é recomendada para situações em que a ordem as mensagens não é importante.
- Além disso, pelo fato dessa abordagem deletar as mensagens após o *ack*, ela também é recomendada para situações em que não há a necessidade de ler novamente mensagens que já foram processadas.

Sistemas de mensagens

- Fila de mensagens:
  - Existem dois grandes tipos de filas de mensagem:
    - **Fila baseada em log**
      - O log é particionado e diferentes partições podem ser *hosteadas* em diferentes máquinas.
      - Um tópico pode ser definido por um grupo de partições que possuem o mesmo tipo de mensagem.
      - Todas as mensagens de uma partição são designadas para o mesmo nó consumidor.
      - Mensagens são sempre entregues na mesma ordem.

Sistemas de mensagens



- Produtores enviam mensagens incluindo elas no final do log de uma partição e os consumidores lêem esse log.
- Esse particionamento possibilita paralelismo.
- O consumidor monitora o seu progresso criando um *checkpoint* no *offset* da última mensagem processada.

Sistemas de mensagens

- Uso de disco
  - Se o log sempre for incrementado, eventualmente você não terá mais espaço em disco.
  - Por isso o log é segmentado e, de tempos em tempos, segmentos antigos são deletados ou movidos para outro local.
  - Isso significa que se o consumidor for muito lento, o seu offset pode estar em um segmento que já foi deletado.
- Quando o consumidor é muito lento
  - Como já foi discutido anteriormente, as opções são *droppar* mensagens, guardar as mensagens em uma fila ou aplicar *backpressure*.

Bancos de dados e fluxos

Bancos de dados e fluxos

- Já discutimos que um evento pode ser uma ação do usuário ou originado de uma máquina, mas ele também pode ser uma escrita em um banco de dados.
- O log de replicação é um fluxo de eventos de write de um banco de dados.
    - Com isso podemos capturar o *changelog* (o histórico de todas as mudanças feitas no banco de dados) através de CDC ou event sourcing (veremos o que é isso a seguir).
    - O log pode ser compactado da seguinte maneira: registros do log com a mesma chave são identificados e duplicações são removidas, deixando apenas a atualização mais recente de cada chave.
    - Compactação do log permite que o fluxo guarde uma cópia completa do conteúdo do banco de dados.

Bancos de dados e fluxos

- Change Data Capture (CDC)
  - A aplicação utiliza o banco de dados de uma maneira mutável. Ou seja, a aplicação pode atualizar e remover registros.
  - O *changelog* é extraído do banco de dados em baixo nível; isso garante que a ordem dos *writes* extraídos do banco de dados correspondem com a ordem em que eles realmente ocorreram.
  - A aplicação que está escrevendo no banco de dados não precisa estar ciente que o CDC está sendo executado.

Bancos de dados e fluxos

- Event Sourcing
  - Ao contrário de CDC, o banco de dados é utilizado de maneira imutável.
  - Neste caso o *changelog* é apenas de inserção; atualização e remoção de registros são desencorajadas ou proibidas.
  - Eventos refletem ocorrências do mundo real no nível de aplicação, ao invés de mudanças em baixo nível.
    - Exemplo: Guardar o evento “o estudante cancelou a sua matrícula em uma matéria” expressa uma única ação, enquanto os efeitos colaterais “um aluno foi removido da tabela de matrículas” e “um motivo de cancelamento foi adicionado à tabela de feedback do aluno” mostram como os dados serão utilizados.
    - Se uma aplicação como “a vaga foi oferecida para a próxima pessoa na lista de espera” for implementada, a abordagem de Event Sourcing permite que os efeitos colaterais sejam facilmente inferidos a partir desse evento.

Processando fluxos

Processando fluxos

- Até agora discutimos de onde os fluxos vêm (ação do usuário, atividade e uma máquina e *writes* em um banco de dados) e como transmitirmos eles (mensagem direta, fila de mensagens e logs)
- Agora vamos discutir o que se pode fazer com um fluxo:
  - Você pode escrever os dados dos eventos em um banco de dados, cache ou sistemas de armazenamento similares, onde o usuário pode busca-los.
  - Você pode enviar eventos para usuários, como enviando e-mails, notificações ou fazendo *stream* dos eventos em tempo real para algum *dashboard*.
  - Você pode processar um ou mais fluxos de entrada e produzir um ou mais fluxos de saída.
- Pelo resto da apresentação vamos focar na última opção.

Processando fluxos

- Busca por padrões de evento (Cemplex Event Processing)
  - É utilizado para monitoração, onde uma organização deseja ser notificada caso certas coisas aconteçam.
    - Detecção de fraude caso o padrão de uso de um cartão de crédito tenha mudado drasticamente
    - Sistemas de *trading* devem examinar mudanças de preços no mercado financeiro e executar os *trades* conforme as regras específicas
    - Sistemas de fabricação devem monitorar o estado das máquinas em uma fábrica e rapidamente identificar o problema caso haja algum defeito
    - Sistemas militares devem traçar as atividades de um agressor em potencial e gerar um alerta caso haja sinais de um ataque
  - Assim como uma expressão regular permite que se encontrem padrões em uma string, CEP permite que se encontrem padrões nos eventos de um fluxo

Processando fluxos

- Busca por padrões de evento (Cemplex Event Processing)
  - Nesses sistemas a relação entre busca e dados é invertido, se comparado aos banco de dados tradicionais.
    - Geralmente um banco de dados armazena dados e tratam as buscas como transições: quando uma busca ocorre o banco de dados procura por dados que satisfaçam a busca, e após isso esquece o *query*.
    - Ferramentas CEP armazenam os *queries* e utiliza eventos do fluxo de entrada para buscar uma *query* que corresponda a um padrão de entrada.

Processando fluxos

- *Windowed Aggregation (Stream Analytics)*
  - A linha entre CEP e *Analytics* é tênue, mas como regra geral:
    - *Analytics* é menos interessado em encontrar padrões específicos nos eventos e é mais orientado à agregações e medições estatísticas de um grande número de eventos:
      - Medir a frequência com a um certo tipo de evento ocorre
      - Calcular a média de um certo valor em um certo período de tempo
      - Comparar estatísticas atuais com estatísticas anteriores
  - O intervalo de tempo referente à agregação é chamado de janela (*window*)

Processando fluxos

- Mantendo os sistemas atualizados (*Materialized Views*)
  - Um fluxo pode ser utilizado para manter sistemas derivados atualizados
    - Cache, *warehouses*, etc.
  - Atualizações são mantidas por meio de um log que é aplicado nos sistemas derivados

Processando fluxos

- Entrando em consenso sobre tempo
  - Imagine a afirmação “processe a média dos últimos 5 minutos”
    - Os últimos 5 minutos de eventos?
    - Os últimos 5 minutos de processamento?
  - Na maioria dos casos o tempo de interesse são os 5 minutos de eventos.
  - Essa abordagem utiliza o relógio do sistema da máquina de processamento e tem a vantagem de ser simples e precisa (se o atraso entre a criação do evento e seu processamento for negligenciável)
- Esses atrasos podem causar desordenamento das mensagens

Processando fluxos

- O número do episódio seria seu *timestamp* e o ano de lançamento o processamento
- Se você assistir os filmes na ordem em que eles foram lançados a narrativa é inconsistente
- Humanos conseguem lidar com isso, porém algoritmos de processamento necessitam que isso seja especificado

Processando fluxos

- Afinal, qual relógio devemos utilizar?
  - Exemplo: aplicativo de celular que envia dados para um servidor; o app pode ser utilizado offline, o que causa com que os dados sejam armazenados em um buffer para depois serem enviados quando houver conexão com a internet.
  - Nesse contexto temos interesse no *timestamp* do evento, porém o relógio do dispositivo pode estar errado. Para resolver isso podemos utilizar 3 *timestamps*:
    - O momento em que o evento ocorreu, no relógio do dispositivo
    - O momento em que o evento foi enviado para o servidor, no relógio do dispositivo
    - O momento em que o evento foi recebido pelo servidor, no relógio do servidor
  - Subtraindo o segundo *timestamp* do terceiro é possível estimar o *offset* entre o relógio do dispositivo e do servidor
  - Após isso se aplica o *offset* no *timestamp* do evento e se estima o momento em que o evento realmente ocorreu

Agrupamento de fluxos

Agrupamento de fluxos

- Assim como em *batch processing* foi discutido a importância do agrupamento, aqui isso também é importante.
- Porém, como agora os eventos podem aparecer a qualquer momento, agrupamentos são mais desafiadores.
- Existem três tipos de agrupamento:
  - *Stream-stream*
  - *Stream-table*
  - *Table-table*

Agrupamento de fluxos

- *Stream-stream*
  - Os dois fluxos de entrada consistem de eventos de atividade
  - O operador de agrupamento procura por eventos relacionados que ocorreram dentro de uma mesma janela de tempo
    - Exemplo: podem-se agrupar duas ações tomadas pelo mesmo usuário em um intervalo de 30 minutos entre cada.
- *Stream-table*
  - Uma entrada é um fluxo de eventos de atividade e a outra é um *changelog*
  - O *changelog* mantém atualizado uma cópia local do banco de dados
  - Para cada evento de atividade é realizado uma busca no banco de dados para encontrar informações sobre o autor da atividade
  - A atividade é agrupada com essas informações
  - Esse processo também é conhecido como enriquecimento, pois enriquece os eventos de atividade com informações do banco de dados

Agrupamento de fluxos

- *Table-table*
  - Os dois fluxos de entrada consistem de *changelogs*
  - Neste caso, cada mudança em um lado é agrupada com o estado mais recente do outro lado
  - O resultado é um fluxo atualizado do agrupamento das duas tabelas
  - Exemplo: Twitter
    - É utilizado uma *timeline cache*, uma espécie de *inbox* em que os tweets são escritos na medida em que são enviados.
    - Para realizar a manutenção dessa cache requer coisas como: adicionar novos tweets à timeline, remover tweets da timeline caso ele seja deletado pelo dono, mostrar tweets recentes quando se começa a seguir um usuário e remover tweets caso deixe de seguir um usuário.
    - Esse tipo de união serve para garantir que todos os seguidores de uma determinada pessoa recebem novos tweets e atualizar suas timelines nos casos de seguir ou deixar de seguir.

Resumo

- *Batch Processing x Stream Processing*
- Transmissão
  - Eventos, produtores e consumidores
- Mensagens
  - Filas AMQP/JMS e baseadas em log
- Changelog
  - CDC e Event Sourcing
- Utilidades de um fluxo
  - CEP, Analytics e Materialized Views
  - Tempo
- Agrupamentos
  - *Stream-stream, stream-table, table-table*

Perguntas

1. Comparando *Batch Processing* com *Stream Processing*, em qual(is) aspecto(s) essas duas abordagens se assemelham e em qual(is) aspecto(s) elas se diferenciam?

2. Cite os tipos de agrupamento discutidos e, para cada um desses tipos, cite uma situação em que ele pode ser utilizado.

Obrigado