UNIVERSIDADE DE AVEIRO

Departamento de Eletrónica, Telecomunicações e Informática

Mestrado em Engenharia Informática

Arquitetura de Software

Trabalho Prático nº2

Rafael da Fonseca Fernandes | 95319

Gonçalo Junqueira | 95314

Aveiro | 15 de maio de 2022

Índice Geral

1. Introdução

No âmbito da unidade curricular de Arquitetura de Software é proposto a realização de um trabalho prático, a fim de avaliar a componente teórico-prática da disciplina e os conhecimentos obtidos durante as aulas.

Este projeto tem como objetivo a implementação de uma plataforma centralizada capaz de supervisionar o estado de sensores, além disso o design e implementação da plataforma foi criada em Java, em conjunto com o Apache Kafka.

O tema do trabalho é uma simulação de processamento de dados de vários sensores num cluster Kafka. Para cada um dos diversos casos de uso é construída uma solução independente. Cada caso de uso é composto por um Kafka Cluster, um Zookeeper, um Topic com o nome de Sensor, seis Partitions, tres Replicas e dois min.insync.replicas.

Para além do Kafka Cluster (Brokers e Zookeeper), existem pelo menos três processos adicionais. Um processo responsável por ler os dados do sensor (registos) do ficheiro sensor.txt e enviá-los para um ou mais produtores via Java Sockets.

O PProducer é responsável por receber os registos do PSource e enviá-los ao Kafka Cluster, que poderá ser constituído por um ou mais Kafka Producers. A GUI contém o número total de registos por ID de sensor e o número total de registos mais outros detalhes conforme o UC (User Case).

Por fim. o PConsumer é responsável por ler os registos do Kafka Cluster e processá-los, podendo incluir um ou mais Kafka Consumer. Tal como no PProducer, existe um GUI com o número total de registos por ID de sensor e o número total de registos.

2. Propriedades

Para implementar uma solução capaz de cumprir com os requisitos dos casos de uso, foi necessário selecionar um conjunto de propriedades que serão manipulados para obter o resultado pretendido. De seguida será descrita essas propriedades para cada caso em específico.

2.1. Propriedades do Produto

As propriedades selecionadas para o Kafka Producer serão apresentadas de seguida.

2.1.1. buffer.memory

Esta propriedade consiste no total de bytes de memória que o produtor pode utilizar para guardar os registos à espera de serem enviados para o servidor. Este valor deve ser pelo menos tão grande quanto o batch size, e ainda capaz de acomodar compressão e pedidos in-flight.

2.1.2. acks

A propriedade acks consiste no número de confirmações que o produtor requer que o líder tenha recebido antes de considerar um pedido como completo, para controlar a durabilidade dos registos que são enviados.

Os possíveis valores são os seguintes:

- acks=0: significa que o produtor não irá esperar por nenhuma confirmação por parte do servidor. Não é possível garantir que o servidor tenha recebido o registo por isso, poderá existir perda de mensagens;
- acks=1: significa que o líder irá avançar sem a confirmação por parte de todos os seguidores. Existe a possibilidade de perda de mensagens;
- acks=all: significa que o líder vai esperar que todas as réplicas sincronizadas confirmem o registo. Assim é possível garantir que os registos não serão perdidos enquanto houver pelo menos uma réplica sincronizada.

Assim, a necessidade de obter confirmações de todas as réplicas aumentam a latência entre o envio da mensagem por parte do produtor e a receção da confirmação.

2.1.3. batch.size

O produtor tentará agrupar os registos em menos pedidos sempre que vários registos forem enviados para a mesma partição. Caso o batch.size seja demasiado grande para a frequência das mensagens produzidas, então estará a acrescentar um atraso desnecessário às mensagens em espera no buffer. Caso contrário, as mensagens maiores poderão ser atrasadas. O que se ganha no aumento do throughput, concede-se num aumento da latência à entrega das mensagens.

2.1.4. max.in.flight.requests.per.connection

Esta propriedade é o número máximo de pedidos não confirmados que o cliente enviará numa única ligação antes do bloqueio. Se esta propriedade tiver um valor superior a 1, então existe o risco de reordenamento de mensagens devido às novas tentativas, caso estas estejam ativadas.

2.1.5. linger.ms

Caso seja utilizado o batch.size no máximo é enviado um pedido quando as mensagens acumuladas chegam ao tamanho máximo, ou quando as mensagens estiverem em espera à mais tempo que o linger.ms. Nesta propriedade o que se ganha com o aumento do throughput, concede-se no aumento da latência na entrega da mensagem.

2.1.6. delivery.timeout.ms

O ajuste do tempo máximo de espera antes de uma mensagem ser entregue e completar um pedido poderá aumentar o throughput. O valor desta propriedade deve ser maior ou igual à soma do request.timeout.ms e linger.ms.

2.1.7. retries

Permite definir quantas vezes o produtor deve tentar enviar a mensagem para um broker se o envio falhar. É importante referir que um valor elevado ajuda a evitar perdas de dados. A utilização desta propriedade requer atenção em dois aspetos, sendo eles:

- Caso o delivery.timeout.ms for pequeno, mas o número de retries for elevado, a entrega da mensagem falhará na mesma;
- Caso a ordem das mensagens for relevante, então é fundamental definir o max.in.flight.requests.per.connection para 1, isto serve para evitar o reordenamento das mensagens.

2.1.8. compression.type

Esta propriedade é útil para melhorar o throughput e reduzir a carga no armazenamento, contudo, nos casos onde a latência tem de ser baixa a operação de compressão e descompressão não é adequada.

2.2. Propriedades do Consumidor

As propriedades selecionadas para o Kafka Consumer serão apresentadas de seguida.

2.2.1. auto.commit.interval.ms

Esta propriedade consiste na frequência, em milissegundos, com a qual os offsets dos consumidores serão atualizados no Kafka.

2.2.2. fetch.min.bytes

A propriedade descrita acima consiste na quantidade de informação mínima que o servidor deve retornar num pedido de fetch. Caso não existir informação suficiente, o pedido irá aguardar que acumule a quantidade necessária de informação. Um valor maior que 1 irá fazer com que o servidor espere até acumular mais dados, o que pode aumentar o throughput do servidor, há custa de alguma latência adicional.

2.2.3. enable.auto.commit

Caso os commits automáticos estiverem ativados, o offset do consumidor será periodicamente atualizado, assim existe o risco de duplicação e perda de mensagens. Caso o processo tenha de ser realizado manualmente, deve ser definida a estratégia para atualizar o offset dos consumidores.

3. Implementação dos Casos de Uso

Para cada caso de uso, foi desenvolvida uma solução configurando as propriedades apresentadas, de forma que fossem compridos os requisitos.

3.1. Caso de Uso 1

Os valores definidos para as propriedades do produtor foram:

- acks = 0: visto que podem ser perdidas mensagens e o throughput deve ser maximizado;
- buffer.memory = 33554432 (default): valor capaz de armazenar toda a informação necessária, porém talvez pudesse ser reduzido de forma a não ocupar espaço desnecessariamente;
- batch.size = 100000: um valor elevado aumenta o throughput, um dos requisitos do caso de uso;
- linger.ms = 50: estando a utilizar um valor elevado de batch.size, deve ser definido um valor de linger.ms maior que zero. Melhora o throughput;
- max.in.flight.requests.per.connection = 1: diminui o throughput, porém é necessário, pois não pode haver reordenamento de mensagens, isto é, tem que se manter a ordem original;
- compression.type = gzip: a utilização de compressão aumenta o throughput;
- delivery.timeout.ms = 35000: um valor mais reduzido de timeout, mas que cumpra com o valor mínimo necessário, ajuda a aumentar o throughput;
- retries = 0: aumenta o throughput. Não é necessário utilizar visto que podem ser perdidas mensagens.

Os valores definidos para as propriedades do consumidor foram:

- fetch.min.bytes = 100000: um valor elevado de forma a aumentar o throughput;
- enable.auto.commit = true: o auto commit pode estar ativado uma vez que os dados podem ser reprocessados;
- auto.commit.interval.ms = 10000: um valor elevado de forma a aumentar o throughput.

A interação entre os diferentes componentes processa-se da seguinte forma:

- **Kafka Producer:** O envio das mensagens para o tópico é feito consoante as mensagens que vão sendo recebidas do PSource. Não é utilizada concorrência visto que é necessário manter a ordem original dos registos;
- **Kafka Consumer:** A leitura dos registos do tópico é feita de forma síncrona, um registo de cada vez. Não é utilizada concorrência visto que é necessário manter a ordem original dos registos.
- **PSource:** A leitura e processamento do ficheiro de texto é realizada linha a linha. A cada linha lida é enviada para o PProducer. Não é utilizada concorrência visto que é necessário manter a ordem original dos registos.

3.2. Caso de Uso 2

Os valores definidos para as propriedades do produtor foram:

- acks = 1: diminui a possibilidade de perda de mensagens, e diminui menos o throughput que o acks = all. Foi feito um balanceamento entre a possibilidade de perda de mensagens e throughput / latência;
- buffer.memory = 33554432 (default): valor capaz de armazenar toda a informação necessária, porém talvez pudesse ser reduzido de forma a não ocupar espaço desnecessariamente;
- batch.size = 16384 (default): um valor mais reduzido, caso contrário a latência irá aumentar;
- linger.ms = 0 (default): valor nulo, caso contrário a latência irá aumentar;
- max.in.flight.requests.per.connection = 1: é necessário, pois não pode haver reordenamento de mensagens, isto é, tem que se manter a ordem original;
- compression.type = none: não pode ser utilizada compressão, caso contrário aumenta a latência;
- delivery.timeout.ms = 35000: um valor mais reduzido de timeout, mas que cumpra com o valor mínimo necessário, ajuda a aumentar o throughput;
- retries = 0: aumenta o throughput. Não é necessário utilizar visto que podem ser perdidas mensagens.

Os valores definidos para as propriedades do consumidor foram:

- fetch.min.bytes = 100000: um valor elevado de forma a aumentar o throughput;
- enable.auto.commit = true: o auto commit pode estar ativado uma vez que os dados podem ser reprocessados;
- auto.commit.interval.ms = 10000: um valor elevado de forma a aumentar o throughput.

A interação entre os diferentes componentes processa-se da seguinte forma:

- **Kafka Producer:** No total existem seis produtores, cada um com um respetivo ServerSocket. A cada produtor está associado um sensor ID, que está encarregue de tratar de todos os registos a ele associado. O processamento das mensagens é realizado de forma síncrona, consoante as mensagens que são recebidas do PSource, uma vez que deve ser mantida a ordem por sensor ID;
- **Kafka Consumer:** A leitura dos registos do tópico é feita de forma síncrona, um registo de cada vez. Não é utilizada a concorrência visto que é necessário manter a ordem pela qual os registos são escritos no tópico;
- **PSource:** A leitura e processamento do ficheiro de texto é realizada linha a linha. A cada linha lida é enviada para um PProducer. A escolha do Producer que irá tretar o registo é determinado pelo sensor ID.

3.3. Caso de Uso 3

Os valores definidos para as propriedades do produtor foram:

- acks = 1: diminui a possibilidade de perda de mensagens, e diminui menos o throughput que o acks = all. Foi feito um balanceamento entre a possibilidade de perda de mensagens e throughput / latência;
- buffer.memory = 33554432 (default): valor capaz de armazenar toda a informação necessária, porém talvez pudesse ser reduzido de forma a não ocupar espaço desnecessariamente;
- max.in.flight.requests.per.connection = 5 (default): aumenta o throughput, já que a ordem dos registos não é relevante;

- batch.size = 100000: um valor elevado aumenta o throughput, um dos requisitos do caso de uso;
- linger.ms = 50: estando a utilizar um valor elevado de batch.size, deve ser definido um valor de linger.ms maior que zero. Melhora o throughput;
- compression.type = gzip: a utilização de compressão aumenta o throughput.
- delivery.timeout.ms = 35000: um valor mais reduzido de timeout, mas que cumpra com o valor mínimo necessário, ajuda a aumentar o throughput;
- retries = 0: aumenta o throughput. Não é necessário utilizar visto que podem ser perdidas mensagens.

Os valores definidos para as propriedades do consumidor foram:

- fetch.min.bytes = 100000: um valor elevado de forma a aumentar o throughput;
- enable.auto.commit = true: o auto commit pode estar ativado uma vez que os dados podem ser reprocessados;
- auto.commit.interval.ms = 10000: um valor elevado de forma a aumentar o throughput.

A interação entre os diferentes componentes processa-se da seguinte forma:

- **Kafka Producer:** Existe no total três produtores, cada um com um respetivo ServerSocket. O processamento das mensagens é feito de forma assíncrona, visto que a ordem dos registos não importa. Além disso, a cada mensagem que é recebida do PSource é lançada uma thread para o processar e enviar o registo para o tópico. A utilização de threads deve-se ao facto de os registos poderem ser reordenados;
- **Kafka Consumer:** A leitura dos registos do tópico é feita de forma assíncrona, ou seja, existe três Kafka Consumer que pertencem ao mesmo grupo a consumir registos;
- **PSource:** A leitura e processamento do ficheiro é realizada linha a linha enviando os registos para um fifo, de seguida um dos três threads vai enviar para o seu PProducer.

3.4. Caso de Uso 4

Este caso de uso não foi implementado devido ao grupo não saber como garantir a ordem com o uso de vários produtores.

3.5. Caso de Uso 5

Os valores definidos para as propriedades do produtor foram:

- acks = 1: diminui a possibilidade de perda de mensagens, e diminui menos o throughput que o acks = all. Foi feito um balanceamento entre a possibilidade de perda de mensagens e throughput / latência;
- buffer.memory = 33554432 (default): valor capaz de armazenar toda a informação necessária, porém talvez pudesse ser reduzido de forma a não ocupar espaço desnecessariamente;
- max.in.flight.requests.per.connection = 5 (default): aumenta o throughput, já que a ordem dos registos não é relevante;
- retries = 0: aumenta o throughput. Não é necessário utilizar visto que podem ser perdidas mensagens;

- delivery.timeout.ms = 35000: um valor mais reduzido de timeout, mas que cumpra com o valor mínimo necessário, ajuda a aumentar o throughput;
- batch.size = 100000: um valor elevado aumenta o throughput, um dos requisitos do caso de uso;
- linger.ms = 50: estando a utilizar um valor elevado de batch.size, deve ser definido um valor de linger.ms maior que zero. Melhora o throughput;
- compression.type = gzip: a utilização de compressão aumenta o throughput.

Os valores definidos para as propriedades do consumidor foram:

- fetch.min.bytes = 100000: um valor elevado de forma a aumentar o throughput;
- enable.auto.commit = true: o auto commit pode estar ativado uma vez que os dados podem ser reprocessados;
- auto.commit.interval.ms = 10000: um valor elevado de forma a aumentar o throughput.

A interação entre os diferentes componentes processa-se da seguinte forma:

- **Kafka Producer:** Existe apenas 1 produtor, porém as mensagens são tratadas de forma assíncrona. A cada mensagem que é recebida do PSource, é lançada uma thread para a processar e enviar os registos para o tópico;
- **Kafka Consumer:** Existem três grupos, cada um deles com três consumidores idênticos, que servem de réplicas para a Voting Replication Tactic. Cada Kafka Group possui o seu próprio group.id, de forma a todos os grupos poderem ler a mesma informação. Visto que o processamento dos registos pode ser realizado em qualquer ordem, o seu processamento também é feito de forma assíncrona, sendo que é lançada um thread para cada um deles;
- **PSource:** A leitura e processamento do ficheiro de texto é feita linha a linha. A cada linha lida é enviada para o PProducer.

3.6. Caso de Uso 6

Os valores definidos para as propriedades do produtor foram:

- acks = 1: diminui a possibilidade de perda de mensagens, e diminui menos o throughput que o acks = all. Foi feito um balanceamento entre a possibilidade de perda de mensagens e throughput / latência;
- buffer.memory = 33554432 (default): valor capaz de armazenar toda a informação necessária, porém talvez pudesse ser reduzido de forma a não ocupar espaço desnecessariamente;
- batch.size = 100000: um valor elevado aumenta o throughput, um dos requisitos do caso de uso;
- linger.ms = 50: estando a utilizar um valor elevado de batch.size, deve ser definido um valor de linger.ms maior que zero. Melhora o throughput;
- max.in.flight.requests.per.connection = 5 (default): aumenta o throughput, já que a ordem dos registos não é relevante;
- compression.type = gzip: a utilização de compressão aumenta o throughput;
- delivery.timeout.ms = 35000: um valor mais reduzido de timeout, mas que cumpra com o valor mínimo necessário, ajuda a aumentar o throughput;
- retries = 1000000: minimiza a possibilidade de se perder dados. Considerou-se que o valor default seria demasiado elevado.

Os valores definidos para as propriedades do consumidor foram:

- fetch.min.bytes = 100000: um valor elevado de forma a aumentar o throughput;
- enable.auto.commit = false: o commit é feito manualmente de forma síncrona, de forma a evitar o reprocessamento de dados. A estratégica seguida consiste em efetuar commit sempre que um bloco de registos é processado.

A interação entre os diferentes componentes processa-se da seguinte forma:

- **Kafka Producer:** Existe apenas 1 produtor, porém as mensagens são tratadas de forma síncrona. A cada mensagem recebida do PSource, processa para enviar o registo para o tópico;
- **Kafka Consumer:** Existem três grupos, cada um deles com três consumidores idênticos, que servem de réplicas para a Voting Replication Tactic. Cada Kafka Group possui o seu próprio group.id, de forma a todos os grupos poderem ler a mesma informação. Visto que o processamento dos registos pode ser realizado em qualquer ordem, o seu processamento também é feito de forma assíncrona, sendo que é lançada um thread para cada um deles;
- **PSource:** A leitura e processamento do ficheiro de texto é feita linha a linha. A casa linha lida é enviada para o PProducer.

4. Conclusão

É possível verificar com a realização deste trabalho, que o Apache Kafka é uma ferramenta versátil e parametrizável, capaz de cumprir com os atributos de qualidade desejados.

Através das propriedas do Kafka Producers e Kafka Consumers foi possível cumprir com os mais diversos requisitos, cobrindo atribuso com a durabilidade e desempenho.

Em suma, depois de aprender a utilizar Kafka foi possível verificar que é bastante útil em diversas empresas pelo mundo demonstra o quanto útil e consistente o Kafka é.

5. Referências

https://kafka.apache.org/documentation/#configuration

https://strimzi.io/blog/2020/10/15/producer-tuning/

https://strimzi.io/blog/2021/01/07/consumer-tuning/

https://www.youtube.com/watch?v=_q1IjK5jjyU

https://www.youtube.com/watch?v=HtrKp6V9KuE&t=1423s

https://www.youtube.com/watch?v=pq_V7YupXvs&t=207s