Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
Mestrado em Engenharia Informática
Ano Letivo 2020/2021

Arquiteturas de Software

Trabalho Prático 2

Aveiro, 24 de maio de 2021

Luís Laranjeira, 81526
Rafael Sá, 104552

Índice

1. Introdução

No âmbito da unidade curricular de Arquiteturas 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.

O foco deste trabalho é a implementação de uma plataforma centralizada capaz de supervisionar o estado de sensores. 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*, para além do *Zookeeper*, que contém 6 *Brokers*, 1 *Topic* com o nome de Sensor, 6 *Partitions*, 3 *Replicas* e 2 *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 registos de 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.

O PConsumer 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 manipuladas para obter o resultado pretendido.

2.1 Propriedades do Produtor

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

2.1.1. acks

A propriedade consiste no número de confirmações que o produtor requer que o líder tenha recebido antes de considerar um pedido como completo. Isto controla a durabilidade dos registos que são enviados [1].

A propriedade pode ter os seguintes valores [1]:

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

A necessidade de obter confirmações de todas as réplicas aumenta a latência entre o envio da mensagem por parte do produtor e a receção da confirmação [2].

2.1.2. buffer.memory

Consiste no total de bytes de memória que o produtor pode utilizar 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* [1].

2.1.3. max.in.flight.requests.per.connection

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, existe o risco de reordenamento de mensagens devido às novas tentativas, caso estas estejam ativadas [1].

2.1.4. batch.size

O produtor tentará agrupar os registos em menos pedidos sempre que vários registos forem enviados para a mesma partição [1].

Se o batch.size for demasiado grande para a frequência das mensagens produzidas, estará a acrescentar um atraso desnecessário às mensagens em espera no *buffer*. Por outro lado,

se o batch.size for demasiado pequeno, mas mensagens maiores poderão ser atrasadas. O que se ganha no aumento do *throughput*, concede-se num aumento de latência à entrega das mensagens [2].

2.1.5. linger.ms

Se também for utilizado o batch.size máximo, é enviado um pedido quando as mensagens acumuladas chegam ao tamanho máximo, ou quando as mensagens estiverem esperem em espera há mais tempo que linger.ms [2].

O que se ganha com o aumento do *throughput*, concede-se no aumento da latência na entrega da mensagem [2].

2.1.6. compression.type

A compressão é útil para melhorar o throughput e reduzir a carga no armazenamento. Porém, nos casos onde a latência tem de ser baixa, a operação de compressão e descompressão não é adequada [2].

2.1.7. 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 mais ou igual à soma do request.timeout.ms e linger.ms [2].

2.1.8. retries

Esta propriedade permite definir quantas vezes o produtor deve tentar enviar a mensagem para um *broker* se o envio falhar. Um valor elevado ajuda a evitar a perda de dados. Ao utilizar esta propriedade é necessário ter em atenção dois aspetos [2]:

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

2.2 Propriedades do Consumidor

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

2.2.1. fetch.min.bytes

Consiste na quantidade de informação mínima que o servidor deve retornar num pedido de *fetch*. Se não existir informação suficiente, o pedido irá aguardar que acumule a quantidade necessária de informação [1].

Um valor maior que 1 irá fazer com que o servidor espere por acumulem mais dados, o que pode aumentar o throughput do servidor, a custo de alguma latência adicional [1].

2.2.2. enable.auto.commit

Se os *commits* automáticos estiverem ativados, o *offset* do consumidor será periodicamente atualizado [1]. Com o *commit* automático existe o risco de duplicação e perda de mensagens [3]. Caso o processo tenha de ser realizado manualmente, deve ser definida a estratégia para atualizar o *offset* dos consumidores.

2.2.3. auto.commit.interval.ms

Consiste na frequência, em milissegundos, com a qual os *offsets* dos consumidores serão atualizados no Kafka [1].

3. Implementação dos Casos de Uso

Para cada caso de uso, foi desenvolvida uma solução configurando as propriedades apresentadas, de forma a que fossem cumpridos 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 feita 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** = 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:** Existe um total de seis produtores, cada um com um respetivo ServerSocket. A cada produtor está associado um sensor ID, ficando encarregue de tratar de todos os registos a ele associado. O processamento das mensagens é feito 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 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 feita linha a linha. A cada linha lida é enviada para um PProducer. A escolha do Producer que irá tratar o registo é determinada pelo sensor ID.

3.3 Caso de Uso 3

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** = 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** = 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 um total de seis produtores, cada um com um respetivo ServerSocket. A cada produtor está associado um sensor ID, ficando encarregue de tratar de todos os registos a ele associado. O processamento das mensagens é feito 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 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 feita linha a linha. A cada linha lida é enviada para um PProducer. A escolha do Producer que irá tratar o registo é determinada pelo sensor ID.

3.4 Caso de Uso 4

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** = 5 (default): aumenta o throughput, já que a ordem dos registos não é relevante.
- **compression.type** = none: não pode ser utilizada compressão, caso contrário aumenta a latência.
- **delivery.timeout.ms =** 60000: balanceamento entre a minimização da possibilidade de se perder um registo, e a tentativa de se 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** = 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 o registo para o tópico. A utilização de *threads* deve-se ao facto de os registos poderem ser reordenados.
- **Kafka Consumer:** Existem 3 consumidores idênticos, que servem de réplicas para a *Voting Replication Tactic*. Cada Kafka Consumer possui o seu próprio group.id, de forma a todos os consumidores poderem ler a mesma informação. Visto que o processamento dos registos pode ser feito em qualquer ordem, o seu processamento também é feito de forma assíncrona, sendo que é lançada uma *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.

3.5 Caso de Uso 5

Os valores definidos para as propriedades do produtor foram:

- **acks** = all: apesar de diminuir o *throughput*, não existe perda de mensagens, que é mais relevante.
- **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: 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 =** 60000: balanceamento entre a minimização da possibilidade de se perder um registo, e a tentativa de se 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:** 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:** Existem 3 consumidores idênticos, que servem de réplicas para a *Voting Replication Tactic*. Cada Kafka Consumer possui o seu próprio group.id, de forma a todos os consumidores poderem ler a mesma informação. Visto que o processamento dos registos pode ser feito em qualquer ordem, o seu processamento também é feito de forma assíncrona, sendo que é lançada uma *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.

4. Conclusões

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 propriedades do Kafka Producers e Kafka Consumers foi possível cumprir com os mais diversos requisitos, cobrindo atributos como a durabilidade, desempenho, entre outros.

Referências

[1] Apache Kafka. (2021). *Kafka Documentation.*
https://kafka.apache.org/documentation/#configuration

[2] Strimzi. (Outubro 2020). *Optimizing Kafka Producers*.
https://strimzi.io/blog/2020/10/15/producer-tuning/

[3] Strimzi. (Janeiro 2021). *Optimizing Kafka Consumers*.
https://strimzi.io/blog/2021/01/07/consumer-tuning/