UNIVERSIDADE DO MINHO
Mestrado Engenharia Informática

Paradigmas de Sistemas Distribuídos
Sistemas Distribuídos em Grande Escala

Agregação para dispositivos *IoT*

Autores
PG47329 João Pedro da Santa Guedes
PG47556 Paulo Silva Sousa
A89474 Luís Filipe Cruz Sobral

8 de junho de 2022

Conteúdo

1 Introdução

Este relatório descreve o trabalho prático realizado no âmbito de Paradigmas de Sistemas Distribuídos e Sistemas Distribuídos em Larga Escala.

Este trabalho tem como objetivo o desenvolvimento de um protótipo de uma plataforma para suporte a recolha e agregação de dados enviados por dispositivos *IoT* (e.g., drones, veículos), em número potencialmente elevado. Esta plataforma é instanciada em vários nós espalhados geograficamente em diferentes zonas, para permitir escalabilidade e disponibilizar informação agregada.

Os clientes deverão ser capazes de registar (e cancelar) o interesse em ser notificados em tempo real sobre diferentes ocorrências na sua zona. Cada cliente tem também a opção de fazer pedidos relativos ao estado global do sistema.

2 Análise do problema

Em cada zona deverão existir dois componentes de software que comunicam entre si: o Coletor e o Agregador. Para permitir testar o funcionamento da plataforma será também necessário a implementação de Dispositivos e Clientes.

2.1 Coletor

O Coletor deverá ser desenvolvido em *Erlang* e permitir a autenticação de muitos dispositivos em simultâneo que possam enviar eventos. A informação será posteriormente encaminhada para o agregador.

2.2 Agregador

O Agregador deverá resumir a informação vinda dos Coletores de modo a poder notificar clientes e a responder a perguntas sobre o estado global do sistema. Não é viável enviar uma mensagem por cada evento devido ao número elevado de dispositivos e eventos por dispositivo.

2.3 Cliente

O Cliente pode interagir com o sistema de duas formas distintas. A primeira é através da subscrição de acontecimentos na sua zona, sendo que o cliente pode escolher receber 4 tipos de notificações.

Outra opção é receber informações relativas a todo o sistema através de pedidos enviados ao mesmo , onde existem 4 tipos de pedidos.

2.4 Dispositivo

Deverá ter um identificador único, uma senha e um tipo. Cada dispositivo depois de autenticado, pode enviar eventos identificados para um tipo.

3 Desenho da Arquitetura

3.1 Visão geral

Como podemos ver pela figura 1, a nossa arquitetura tem 4 componentes: os dispositivos, os coletores, os agregadores e os clientes. Para comunicar entre os vários componentes, utilizamos diversos *Sockets* de *ZeroMQ* que serão explicados mais detalhadamente de seguida.

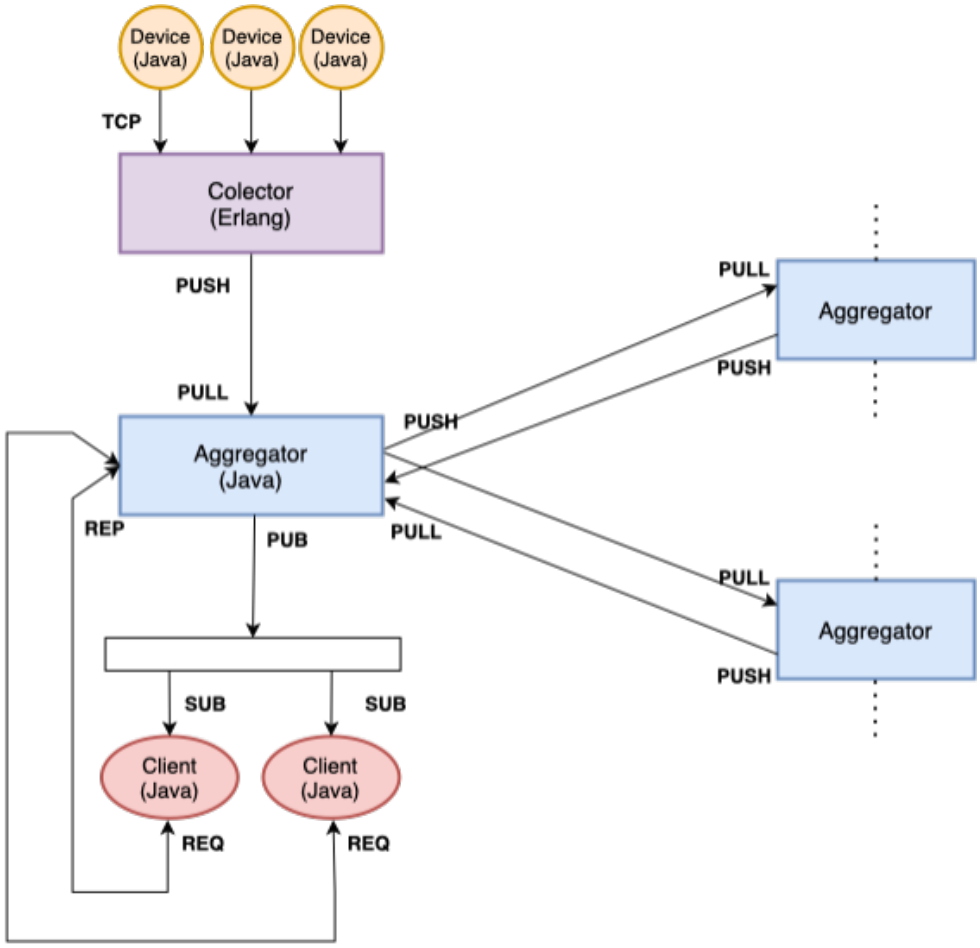

Figura 1: Arquitetura do sistema

3.2 Componentes

3.2.1 Dispositivo

O Dispositivo é um simples protótipo de um dispositivo *IoT* construído em *Java*, que executa o *login*, e de seguida envia eventos constantes para o coletor da sua zona. Os eventos que este dispositivo envia, são aleatórios dentro de uma lista de eventos, de um determinado tipo, enviado como argumento. Este dispositivo não é nada mais, nada menos que um simples programa para que seja possível o teste e o desenvolvimento deste trabalho.

3.2.2 Coletor

O Coletor é o componente que aceita as conexões dos vários dispositivos, e interpreta as suas mensagens. Este componente permite coletar eventos dos dispositivos, mas nunca antes da autenticação dos mesmos.

Devido ao Coletor ser construído em *Erlang*, torna-se muito fácil controlar os estados de sessão entre autenticado, não autenticado, ativo ou inativo, de uma forma fácil e leve, sem nunca abdicar da segurança do estado. Assim, para cada conexão, é criado um processo encarregue de autenticar o dispositivo e tratar os eventos enviados pelo mesmo, sem nunca ter acesso direto ao estado, não colocando-o em perigo.

Aquando da receção de uma mensagem válida, o Coletor irá enviar a informação para o Agregador da sua zona, para que possa ser feita a agregação correta da informação.

3.2.3 Agregador

O agregador tem várias funcionalidades importantes no sistema.

Primeiramente, é o componente que comunica com o Coletor da sua zona, através de um *Socket PULL*, e agrega e guarda (em memória) toda essa informação.

Além disso, comunica com os outros agregadores enviando a informação da sua zona e recebendo a das outras. Para isso, é construída uma topologia *overlay* sem isolamento de zonas, através de um ficheiro *topology*, em que cada agregador só estabelece ligação com os seus vizinhos e o tráfego é espalhado pela rede.

Por último, comunica com o cliente através de dois padrões. Primeiro, num padrão *Request-Reply*, responde a pedidos feitos pelo cliente sobre o sistema todo.

Para além disso, comunica num padrão *Publish-Subscribe*, em que quando o agregador recebe informação do coletor e a armazena, corre umas verificações sobre a informação e, conforme o resultado, publica para os clientes que estão subscritos.

3.2.4 Cliente

O cliente funciona como uma interface de interação com o sistema, usa um padrão *Request-Reply* para enviar pedidos para o agregador e receber resposta.

Além disso, também utiliza um padrão *Publish-Subscribe* para subscrever acontecimentos do seu interesse e receber as respetivas notificações.

O cliente pode a qualquer momento subscrever ou cancelar a subscrição de acontecimentos na sua zona, tal como obter informações relativas ao estado global do sistema através dos pedidos.

3.3 Comunicação entre componentes

3.3.1 Ligação *TCP*

Esta é uma ligação que é feita a partir dos vários dispositivos IoT para se conectarem ao Coletor da sua zona. Estes dispositivos irão ter de fazer *login* antes de poder enviar eventos através desta ligação *TCP*. O envio da informação de *login* e de eventos terão de ser feitos, respetivamente, da seguinte forma:

- Login: $login < user >< password >< type >$
- Evento: $event < event >$

3.3.2 Agregação de Informação

Toda a informação que é recolhida pelo Coletor, irá ser enviada e para o Agregador onde, como o nome indica, irá ser agregada. Esta interação é feita usando um padrão *PUSH-PULL*, onde o Coletor irá fazer uso do seu socket *PUSH* para fazer um envio constante de toda a informação que lhe chega dos devices a quem está conectado. Do outro lado, irá estar um Agregador, com um socket *PULL*, que irá fazer a deserialização das mensagem recebidas e agir da forma como deve agir.

Foram assim definidas as seguintes mensagens:

- login: $login, < user >, < type >$
- logout: $login, < user >$
- event: $event, < user >, < data >$
- inactive: $inactive, < user >$

3.3.3 Convergência de Informação

De modo a todos os agregadores terem a informação atualizada sobre todo o sistema, cada Agregador tem uma classe $CRDT$ que contém um *HashMap* com a porta de cada agregador e com a informação respetiva. Além disso, também é guardado num *HashMap* um vetor de versões com a última versão recebida de cada agregador.

Para receber a informação de outros Agregadores, cada um tem uma *thread* que está à escuta, a receber pacotes do *Socket PULL*. Aquando da receção da mensagem, verifica se a versão que recebeu é maior que a que está no vetor de versões. Caso afirmativo, deserializa a informação, guarda no respetivo *HashMap*, atualiza o vetor de versões e envia a informação a todos os seus vizinhos da rede *overlay*. Caso contrário, não atualiza a informação nem partilha com os vizinhos.

Para enviar a sua informação aos outros Agregadores, é inicializado um *Scheduler* que envia de 5 em 5 segundos o seu estado serializado a todos os seus vizinhos, caso este tenha sofrido alterações. Além disso, se for recebida

do Coletor uma atualização sobre o estado ou atividade de um dispositivo, essa informação é enviada imediatamente aos vizinhos.

As mensagens entre agregadores, seguem o seguinte padrão:

- Para mudanças de estado de um dispositivo:

$$< version >, < source >, < msg >, < payload >$$

- Para envio de estados:

$$< version >, < source >, < state >, < payload >$$

3.3.4 Notificações

De modo a possibilitar aos clientes subscrever os 4 tipos de notificações definidos foi desenvolvido entre os agregadores e clientes um padrão *Publish-Subscribe*. Foram assim definidos os seguintes tipos de mensagem:

- Notificação 'Sem dispositivos online de um dado tipo na zona':

$$offline- < type >$$

- Notificação 'Recorde de dispositivos online de um dado tipo na zona':

$$record- < type >$$

$$record$$

- Notificação 'Subida de dispositivos online na zona para mais de x%':

$$percentUp- < x > x \in \{10, 20, .., 90\}$$

- Notificação 'Descida de dispositivos online na zona para menos de x%':

$$percentDown- < x > x \in \{10, 20, .., 90\}$$

3.3.5 Pedidos

De modo a processar os pedidos, cada cliente tem um *Socket REQ* onde envia o pedido desejado. O agregador tem num *Socket REP* a correr em ciclo numa *thread* que recebe a mensagem, executa o pedido e responde ao cliente nos mesmo *sockets* apenas com o resultado do pedido.

Foram assim definidos os tipos de mensagem que o cliente envia ao servidor:

- Número de dispositivos de um dado tipo online no sistema: $1, < type >$
- Estado de um dado dispositivo no sistema: $2, < id >$
- Número de dispositivos ativos no sistema: 3
- Número de eventos de um dado tipo ocorridos no sistema: $4, < event >$

4 Conclusão

Com este projeto foi possível aperfeiçoar alguns dos conhecimentos aprendidos em Paradigmas de Sistemas Distribuídos (programação por atores e *ZeroMQ*) e Sistemas Distribuídos em Grande Escala (utilização de mecanismos apropriados para um elevado número de eventos e dispositivos). Depois de desenhada a arquitetura do projeto e identificadas as diferentes tecnologias a ser utilizadas, a implementação revelou-se uma tarefa bastante mais simples do que o inicialmente esperado. Foi possível também tirar proveito de diferentes padrões de mensagens oferecidos pelo *ZeroMQ*, tendo sido usado padrões *Publish-Subscribe*, *Request-Reply* e *Push-Pull* para a comunicação entre os componentes do sistema.

No entanto, poderíamos ter aperfeiçoado a criação da rede *overlay*, através de um processo dinâmico em vez de utilizar um ficheiro com a topologia da rede. Além disso também podia ser utilizado um protocolo epidémico mais eficiente para espalhar a informação por todo o sistema.

No geral, o grupo encontra-se satisfeito com o trabalho realizado e os resultados obtidos e por ter conseguido desenvolver um sistema distribuído usando as tecnologias lecionadas nas aulas de ambas as áreas.