4
Arquitetura

Neste capítulo apresentamos uma visão geral da arquitetura de monitoração de sistemas de componentes distribuídos desenvolvida neste trabalho. Nas seções inicias (4.1, 4.2 e 4.3) analisamos alguns aspectos de sistemas distribuídos. Essa análise nos ajudou a levantar quais são as características relevantes que devemos monitorar nesse tipo de sistema.

Na seção seguinte apresentamos um desenho da arquitetura proposta, ilustrada com um exemplo simples de componentes. Como nossa solução oferece um mecanismo transparente para monitoração, essa seção ajuda a diferenciar como será a visão do desenvolvedor dos componentes de negócio da visão do administrador do ambiente de execução.

4.1
Comunicação

Sistemas distribuídos podem ser formados através da composição de diversos componentes, que por sua vez podem estar espalhados por diversas máquinas. Como os componentes de uma aplicação distribuída, muitas vezes, implementam funcionalidades complementares que requerem troca de informações entre eles, é comum nesse tipo de sistema a existência de um alto número de troca de mensagens entre as diversas máquinas envolvidas na aplicação.

Um sistema distribuído por um número grande de máquinas torna muito difícil a identificação daquelas onde a comunicação está se tornando um gargalo para o desempenho do sistema como um todo. Desta forma um indicador que possa ajudar a encontrar onde estão ocorrendo esses gargalos irá ajudar o administrador da aplicação a manter o bom desempenho da mesma.

Com a medição dos tempos gastos na troca de mensagens entre os diversos componentes, pode-se ter uma idéia de onde está o gargalo de comunicação.

Este tipo de informação aliada a facilidade de reconfiguração apresentada pelo SCS (seção 3.2), faz com que o desenvolvedor do sistema possa reorganizar os componentes da aplicação de forma a otimizar a comunicação entre os diversos nós participantes da aplicação.

4.2 Topologia

Em uma arquitetura distribuída, os componentes pertencentes a uma determinada aplicação estarão executando em máquinas distintas, e poderão estar concorrendo com outros componentes que não pertencem à sua aplicação. Dessa forma é importante para o desenvolvedor ter uma forma centralizada de obter informações sobre como os componentes encontram-se espalhados pelas máquinas que executam a aplicação, e entre quais máquinas existe troca de mensagens.

Com esse tipo de informação o desenvolvedor da aplicação poderá identificar por onde seus componentes estão distribuídos, analisando se existem outras formas de distribuição que aproveitem melhor as máquinas disponíveis.

Como a arquitetura do SCS possui um ambiente de execução bem definido, podemos obter através do *ExecutionNode* as informações dos *containers* que estão executando em cada máquina; já com os *containers* podemos obter a lista dos componentes que estão executando naquele processo. Desta forma, podemos informar ao administrador da aplicação a topologia do seu sistema.

Já para obter informações sobre entre quais nós da aplicação existe comunicação, usaremos técnicas de interceptação de chamadas, que nos permite identificar o destino de uma mensagem interceptando-a em pontos pré-definidos. Na seção 5.3.1 esse mecanismo é explicado com maiores detalhes.

4.3 Recursos

Para uma aplicação que contempla diversas máquinas durante sua execução é importante para o seu administrador acompanhar como estão sendo utilizados os recursos de cada nó participante. Nesse contexto muitas poderiam ser as informações coletadas, tais como: uso de memória, tempo de utilização de CPU, número de acesso a disco e outras. As informações que serão mais relevantes, e que devem ser coletadas, podem ser muito dependentes da aplicação que está sendo monitorada.

Em uma aplicação que leia e grave muitos dados em arquivos, informações sobre o tempo e quantidade de operações de E/S podem ser importantes para entender o comportamento do sistema. Já em casos onde exista uma demanda muito grande por processamento de dados, será importante coletar informações sobre do tempo de uso de CPU dos processos que representam essa aplicação.

Dada essa variação entre quais são as métricas relevantes para cada tipo de aplicação, foi montado um mecanismo de coleta onde o administrador do sistema pode a qualquer momento da execução do sistema adicionar novas métricas para serem coletadas. Dessa forma, a ferramenta de monitoração pode ser adaptada de acordo com o tipo de sistema que está sendo monitorado.

Neste trabalho identificamos e implementamos coletores para algumas métricas que consideramos importantes para sistemas distribuídos de um modo geral. Na seção 5.2, apresentamos e justificamos as escolhas dessas métricas.

4.4 Desenho da Arquitetura

Esta seção apresenta uma visão geral da arquitetura desenvolvida neste trabalho, detalhes sobre a implementação realizada são descritos no capítulo 5.

Para coletar as informações do ambiente de execução do SCS, acrescentamos alguns mecanismos em sua arquitetura básica. Para coleta de dados utilizamos um recurso de CORBA, os interceptadores (Seção 5.1), e criamos coletores de dados que funcionam como *daemons* disparados pelos *containers*. Para realizar a distribuição dos dados coletados, usamos um canal de eventos,

no qual as informações são publicadas.

Desenvolvemos também um cliente para esse canal de eventos, capaz de receber os dados coletados de diversas máquinas através dos canais de eventos, e agregar essas informações. Esse cliente mantém uma estrutura de dados com as informações coletadas, e também possui uma interface com métodos capazes de realizar consultas nessa estrutura de dados.

Para visualizar os dados coletados temos um cliente visualizador de informações que requisita as métricas coletadas e as apresenta na forma de uma árvore de dados, onde o usuário pode navegar pelas máquinas monitoradas e observar os dados coletados. Esse visualizador foi desenvolvido como sendo um exemplo de cliente do serviço de coleta, e também foi usado para validar se os dados estavam sendo coletados corretamente.

Com essa arquitetura conseguimos capturar métricas relevantes sobre a comunicação, a topologia e os recursos utilizados pela aplicação. Para monitorar questões de comunicação, utilizamos os interceptadores. Eles interceptam as chamadas, recebidas e enviadas, realizadas por um determinado componente. Assim, podemos introduzir nas chamadas alguns dados que permitirão, por exemplo, que seja calculado o tempo que uma determina operação usou durante a sua execução no servidor.

Para questões de topologia, também podemos utilizar os interceptadores. Com eles podemos verificar qual o destino de cada requisição que é feita, e assim, inferir entre quais *containers* ocorrem trocas de mensagem. Os recursos implementados pelo próprio SCS, que permitem ao usuário descobrir quais componentes estão executando em cada *container* também ajudaram na verificação da disposição dos componentes. Informações sobre o tempo de resposta de cada método invocado remotamente também podem ser obtidas com a ajuda dos interceptadores.

Para realizar a coleta das informações dos recursos utilizados pelo sistema, cada *container* dispara uma *thread* que, de tempos em tempos, coleta uma série de informações daquele processo.

As métricas coletadas podem ser de interesse de diversos clientes, como o administrador da aplicação e uma outra ferramenta que utilize os dados coletados para diagnosticar falhas na execução do sistema. Além dos múltiplos clientes, também existe a preocupação de não onerar a máquina onde foi feita a coleta, com atividades de armazenamento e distribuição dessas informações, pois é nessa máquina que estão executando os componentes da aplicação.

Para lidar com esses requisitos optamos por utilizar um serviço de eventos para publicação das informações coletadas. Assim, imediatamente após coletar uma informação, seja ela feita nos interceptadores ou nos *daemons* dos *containers*, ela é avaliada e enviada para um canal de eventos que, preferencialmente, estará em uma máquina que a aplicação não utiliza. Nesse canal de eventos poderemos conectar quantos clientes desejarmos para receber as informações coletadas. A figura 4.1 mostra um desenho geral da arquitetura com dois componentes, chamados de PingPong, carregados em *containers* distíntos.

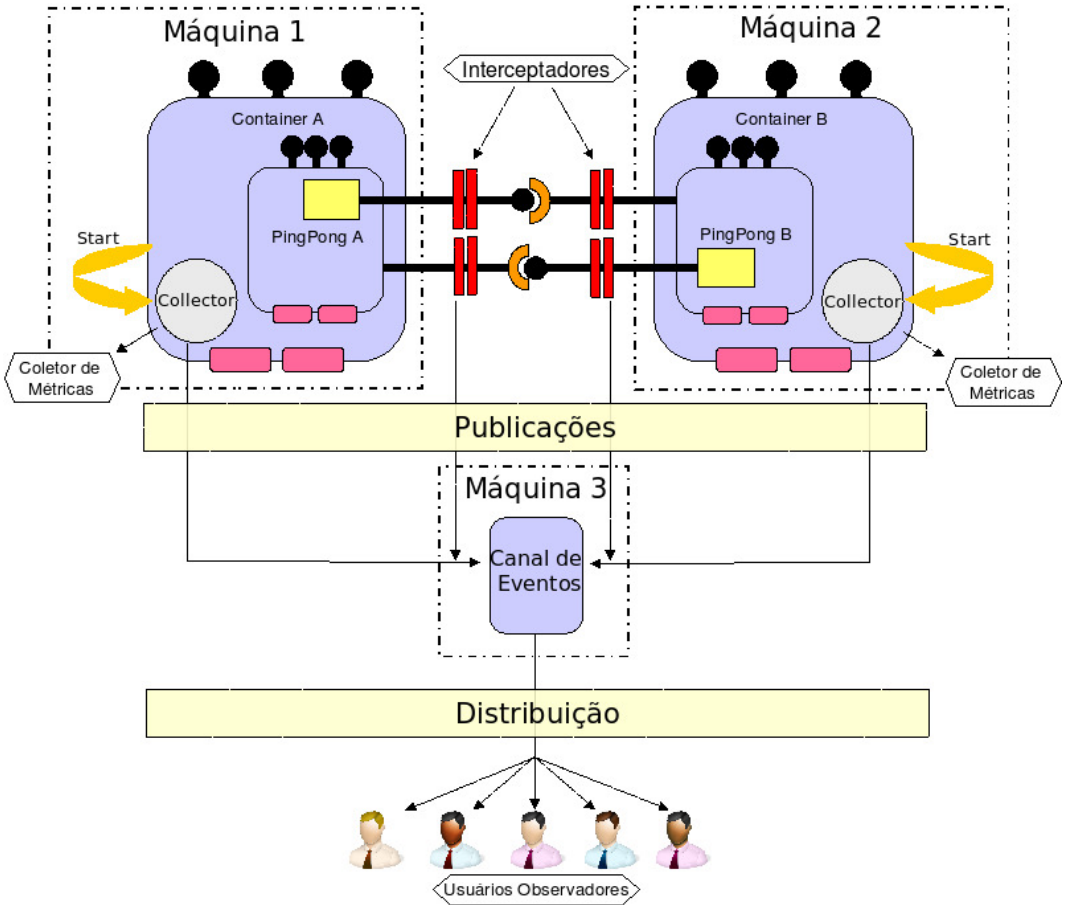

Figura 4.1: Configuração do Exemplo *PingPong*

A avaliação dos dados coletados é realizada para determinarmos se aquele valor necessita ou não ser informado. Para isso implementamos diversas políticas de publicação que podem ser associadas a cada métrica. Com isso pretendemos aumentar a relevância dos valores informados bem como evitar a sobrecarga da rede com o tráfego de informações pouco relevantes.

Os dados que serão coletados são colocados em uma estrutura capaz de transportar várias métricas por vez, de forma a reduzir o número de trocas de mensagens. Para agrupar essas informações implementamos um cliente

desse canal de eventos chamado *StatsCollection*. Um usuário da arquitetura de monitoração poderá escolher utilizar as informações vindas diretamente do canal de eventos, bastando para isso conectar-se diretamente a ele, ou utilizar o *StatsCollection* que é uma alternativa onde as informações estão agrupadas, e existem métodos específicos para buscar as informações. Nesse caso o usuário do *StatsCollection* é quem será o responsável por buscar as informações.

Na arquitetura de monitoração desenvolvida implementamos dois pontos de extensão, um para as métricas e outro para as políticas. Métricas podem ser acrescidas ou removidas dinamicamente, permitindo ao administrador adaptar a estrutura de monitoração de acordo com as especificidades do sistema que está executando. As políticas de publicação, que definem quando um valor coletado deve ou não ser enviado para o cliente, podem ser associadas a cada uma das métricas que estão sendo coletadas, logo, uma métrica recém definida pode necessitar de uma nova política. Nesta dissertação implementamos métricas e políticas que na nossa visão poderão ser usadas na monitoração de sistemas distribuídos de um modo geral. Com esses pontos de extensão tornamos possível o atendimento de requisitos específicos de determinados sistemas.

4.4.1 Exemplo de Uso

Para exemplificar como será a configuração de uma aplicação implementada utilizando a arquitetura proposta, usaremos um exemplo simples de aplicação distribuída, o *PingPong*.

Nessa aplicação existem dois componentes *PingPong* conectados entre sí, e cada componente simplesmente retorna uma mensagem recebida para o outro. Como podemos verificar na figura 4.2, um componente *PingPong* possui uma faceta e um receptáculo nos quais o outro *PingPong* irá se conectar. Basicamente a figura 4.2 ilustra como deve ser a visão do desenvolvedor dos componentes *PingPong*.

A implementação desse componente oferece dois métodos principais o *ping* e o *pong*, a listagem 4.1 mostra a implementação do método *ping* em Java.

```
1  public void ping() {
2    // Obtém a lista facetas conectadas ao seu receptáculo
3    ArrayList<ConnectionDescription> conns =
4      this.myComponent.getReceptacles().get(
5        PingPongServerFactory.FACET_PP).getConnections();
```

```
6       // Para cada faceta encontrada chama o método pong() dessa faceta
7       for (ConnectionDescription desc : conns) {
8         PingPongServer ppFacet = PingPongServerHelper.narrow(desc.objref);
9         try {
10        // Aguarda 3 segundos para chamar o método pong()
11          Thread.sleep(3000);
12        }
13        catch (InterruptedException e) {
14          e.printStackTrace();
15        }
16        // Chamada ao método pong() do componente conectado a esse
17        ppFacet.pong();
18      }
19    }
```

Listagem 4.1: Implementação do Método Ping em Java

Com os componentes implementados, os próximos passos serão distribuí-los e inicializá-los. Como estamos utilizando o SCS para implementação dos componentes não teremos dificuldade para inicializar cada um deles em uma máquina distinta e em seguida conectá-los. Dessa forma, cada mensagem trocada entre os componentes será feita remotamente. A figura 4.1 mostra como esses dois componentes estarão juntos com o SCS e a estrutura de monitoração. Nela os componentes *PingPong* estão executando dentro dos *containers* A e B. As barras paralelas (interceptadores) indicam os pontos onde ocorreram interceptações na troca de mensagens. O objeto chamado de *Collector*, é um *daemon* disparado pelo *container* responsável por coletar informações do processo. Tanto os interceptadores como os coletores irão publicar os dados em um canal de eventos que irá distribuir as informações para os diversos clientes que estiverem conectados a ele.

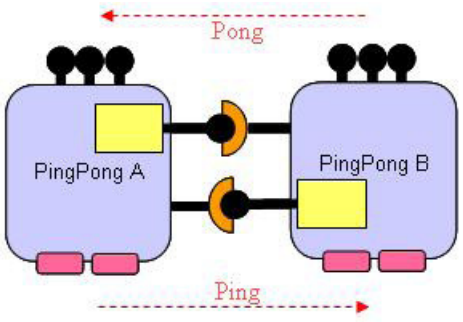

Figura 4.2: Componentes PingPong Conectados