2
Coordenação

Um programa distribuído ou paralelo pode ser visto como a combinação de duas partes distintas: a parte computacional propriamente dita, formada por processos envolvidos com a manipulação de dados e a parte de *coordenação*, responsável pela comunicação e cooperação entre os processos. O conceito de coordenação pode ser usado para distinguir os interesses computacionais de uma aplicação distribuída ou paralela dos interesses de comunicação, permitindo tanto o desenvolvimento independente quanto a integração final dessas duas facetas de uma aplicação distribuída ou paralela [10].

O propósito de um modelo de coordenação e da linguagem a ele associada é prover meios de se integrar componentes heterogêneos, de modo que a interface com esses componentes forme uma única aplicação que pode ser executada e que tire proveito de sistemas paralelos e distribuídos. Para este trabalho, um componente pode ser uma função, uma classe, uma aplicação, um servidor, etc, que oferece e requer serviços.

O conceito de coordenação também está intimamente relacionado com os conceitos de heterogeneidade e de programação multilinguagem. Uma vez que o componente de coordenação está separado do computacional, o primeiro vê os processos que formam o segundo como caixas pretas, portanto, as linguagens de programação propriamente ditas, usadas para escrever código computacional, não têm um papel importante na montagem da estrutura de coordenação. Além disso, uma vez que o modelo de coordenação oferece uma forma homogênea de comunicação entre processos e abstrai os detalhes dependentes de plataforma, a coordenação encoraja o uso de composições heterogêneas de sistemas.

A maioria dos modelos de coordenação tem a intenção de oferecer um *framework* que melhora a modularidade, o reuso de componentes existentes (seqüenciais ou paralelos), a portabilidade e a interoperabilidade entre linguagens. Entretanto, cada modelo difere dos demais na maneira como define a noção de coordenação, o

PUC-Rio - Certificação Digital Nº 9824820/CA

que exatamente está sendo coordenado e como a coordenação é atingida.

Um dos trabalhos pioneiros em modelos de coordenação foi a linguagem Linda [6]. Linda introduziu uma forma simples de se separar interesses de computação e comunicação, que desacopla processos tanto no espaço quanto no tempo: nenhum processo precisa saber a identidade de outros processos, nem é necessário que todos os processos envolvidos na mesma computação estejam ativos ao mesmo tempo. Em Linda, processos se comunicam através de uma memória associativa globalmente compartilhada, chamada de espaço de tuplas. Um processo que deseja comunicar uma informação gera um novo objeto de dados, chamado de tupla, e o adiciona ao espaço de tuplas. Um processo que necessita uma informação a procura no espaço de tuplas, fazendo uma busca associativa na qual a tupla apropriada é localizada pela estrutura de seus campos e por alguns de seus valores.

Um modelo de computação e um modelo de coordenação podem ser integrados em uma única linguagem, mas também podem estar separados em duas linguagens distintas. Muitos modelos de coordenação enfatizam uma separação entre os aspectos de coordenação e de computação [20, 21, 22], que normalmente é atingida utilizando-se uma linguagem de coordenação onde as partes computacionais são tratadas como caixas pretas com interfaces de entrada e saída claramente definidas. Por exemplo, Linda é uma linguagem de coordenação [23] que possui um conjunto de primitivas de coordenação independentes da linguagem hospedeira e por isso é possível derivar variantes naturais de Linda de quase qualquer linguagem de programação [24, 25, 26].

Características comuns em linguagens de *script*, como dinamismo, flexibilidade, facilidade de programação e integração com programas hospedeiros, são bastante úteis para linguagens de coordenação [27, 28]. Mas se por um lado linguagens de script oferecem flexibilidade, por outro elas não dão prioridade ao alto desempenho, uma vez que são interpretadas. Por isso, muitas vezes essas linguagens não são consideradas boas candidatas para implementação de componentes computacionais.

Em programação paralela, por causa da ênfase que convencionalmente é dada ao desempenho de seus programas, linguagens de script não têm recebido muita atenção como candidatas para linguagens de coordenação. Além disso, linguagens compiladas também podem oferecer verificação estática de tipos nas ligações entre componentes [21]. Isto é adequado ao contexto de máquinas paralelas com memória distribuída, como clusters de PCs ou redes locais, no qual a informação sobre o ambiente de execução (como quantidade de usuários) geralmente está disponível antes da execução do programa.

Entretanto, programadores de aplicações paralelas estão enfrentando atualmente novos problemas relacionados ao aproveitamento do poder de computação

PUC-Rio - Certificação Digital Nº 9824820/CA

disponível através de redes geograficamente distribuídas. Por exemplo, iniciativas e projetos de computação em grade [11, 12] procuram criar uma infra-estrutura de computação para aplicações científicas e de engenharia com alta demanda de recursos computacionais. Grades apresentam tipicamente uma configuração altamente heterogênea e dinâmica, em contraste com máquinas paralelas convencionais. A disponibilidade de recursos em uma grade pode sofrer alta variação ao longo do tempo e do espaço. Como discutido em [29, 30], uma técnica importante para lidar com estas variações é utilizar estratégias de adaptação, permitindo que o programa reaja dinamicamente a mudanças em seu ambiente de execução. Além disso, em geral, programas que podem se beneficiar de serem executados em uma grade computacional são aplicações com longo tempo de execução. Freqüentemente, não faz sentido ter que parar a aplicação e começar todo o processo novamente apenas porque o programador disparou a aplicação com parâmetros ou procedimentos inadequados. Deveria ser possível coletar dados sobre a execução e agir sobre o programa enquanto ele está em execução [30].

Em vista dessas exigências, o uso de mecanismos mais flexíveis, como os oferecidos por linguagens de script, podem ganhar uma nova importância para coordenar uma aplicação paralela. Uma das características de uma linguagem de script é a possibilidade de interatividade: Com uma linguagem interpretada, o programador pode usar um "console de coordenação" para controlar a aplicação com maior flexibilidade, podendo definir um comportamento que não havia sido previsto.

Além disso, o uso de uma linguagem de programação completa (*fully-fledged*) para coordenar uma aplicação, em contraste com linguagens declarativas como Darwin (ver Seção 6.2), nos permite criar fronteiras menos rígidas no desenvolvimento de componentes, que podem ser escritos na própria linguagem de coordenação, e na interconexão e coordenação entre eles. Essas fronteiras menos rígidas facilitam a prototipação de aplicações, onde podemos inicialmente construir e testar componentes usando a própria linguagem de coordenação para mais tarde implementá-lo na linguagem utilizada na implementação dos demais componentes.

O uso de uma linguagem de script para coordenação também oferece mais poder de adaptação no nível da linguagem de coordenação, permitindo ao programador acrescentar funcionalidade a aplicação em tempo de execução.

PUC-Rio - Certificação Digital Nº 9824820/CA

2.1
Linguagens de Script e de Extensão

Linguagens de *script* são geralmente linguagens interpretadas, que servem como uma cola entre outros programas. Alguns exemplos destas linguagens são AWK, Perl, csh, Python e Tcl.

Uma das facilidades introduzidas por linguagens de scripts é a habilidade de executar um programa externo facilmente e poder usar o valor e as saídas retornadas por este programa. Uma característica interessante de linguagens de script é a habilidade de interagir em vários níveis, não só chamando funções, mas interagindo com o ambiente global, definindo novas funções, etc. Se pensarmos em um programa externo ou em uma função como um componente, podemos dizer que essas linguagens podem ser usadas para desenvolver e configurar aplicações mais complexas, através da conexão de componentes, assim como linguagens de coordenação.

Algumas linguagens de script podem ser usadas como linguagens de configuração e extensão. Uma linguagem de extensão é um interpretador de linguagem de programação que pode ser integrado a um programa aplicativo, de modo que usuários possam escrever macros ou até mesmo programas completos para estender a aplicação original. Linguagens de extensão têm uma interface C (geralmente C, mas poderia ser qualquer outra linguagem compilada), e podem ter acesso a estruturas de dados da aplicação original em C. Do mesmo modo, existem rotinas em C para dar acesso as estruturas de dados da linguagem de extensão.

Geralmente, uma linguagem de extensão deve ter uma sintaxe clara e simples, uma vez que não é a linguagem principal para a maioria de seus usuários. Deve ser pequena, de modo que o custo de adicioná-la ao programa aplicativo não seja alto. Além disso, deve possuir facilidades para descrição de dados, para que seja útil como uma linguagem de configuração. E, finalmente, deve ser adequadamente extensível, para permitir seu uso em níveis de abstração mais altos, de modo a poder ter interfaces com usuários de diversos domínios [17]. Linguagens de extensão não foram feitas com o objetivo principal de se escrever pedaços grandes de software, com centenas de milhares de linhas de código. Por isso, mecanismos como verificação estática de tipos, tratamento de exceções, etc, não são essenciais a estas linguagens.

Linguagens de extensão são, de modo geral, linguagens de programação completas (*fully-fledged*) que um usuário pode aprender e usar para ajustar o programa aplicativo. Usando linguagens de extensão, programadores podem ligar bibliotecas como Tcl [31] ou Lua [16, 17] a seus programas aplicativos e permitir que seus

PUC-Rio - Certificação Digital Nº 9824820/CA

usuários ajustem seus programas utilizando uma linguagem que eles já conhecem.

Nos últimos anos, linguagens de *script* estão se tornando cada vez mais populares para construir rapidamente aplicações pequenas e flexíveis a partir de um conjunto de componentes existentes [32]. Além disso, linguagens de *script* parecem ser ferramentas adequadas para a construção de sistemas distribuídos em geral e para implementar e coordenar agentes de software em particular. A flexibilidade oferecida por essas linguagens associada ao poder de se ter uma linguagem completa para escrever os mecanismos de coordenação, e até mesmo desenvolver componentes da aplicação, fazem com que linguagens de script sejam apropriadas para implementar aplicações com fronteiras menos rígidas entre o modelo de coordenação utilizado e os componentes desenvolvidos.

2.2
Coordenação Orientada a Eventos

PUC-Rio - Certificação Digital Nº 9824820/CA

O padrão de chamadas remotas de procedimentos RPC foi uma solução amplamente adotada em sistemas distribuídos no domínio de redes locais. Mas, segundo [23], para o desenvolvedor de aplicações paralelas, o mecanismo de RPC não utiliza da CPU da melhor forma possível, pois, nesse modelo de comunicação, uma aplicação envia parâmetros para uma rotina e em seguida fica ociosa a espera de uma resposta.

Mesmo assim, em muitos casos é preciso executar tarefas remotamente. No Capítulo 3, mostramos como é possível executar uma tarefa remotamente usando troca de mensagens assíncronas, onde o processo que solicitou a execução da tarefa em questão não fica bloqueado a espera de uma resposta.

Devido a sua natureza assíncrona, a programação dirigida a eventos oferece um modelo apropriado para ambiente sujeitos a falhas e retardos, que são freqüentes no contexto de redes geográficas.

Em um sistema distribuído dirigido a eventos, assim como em sistemas de interface gráfica, podemos dizer que o usuário comanda o fluxo de execução da aplicação, ao invés do fluxo de execução ser comandado pela própria aplicação [33]. Mais genericamente, podemos dizer que em um sistema distribuído dirigido a eventos, cada componente do sistema pode ser visto da mesma forma que um usuário gerador de eventos para a aplicação. A chegada de uma mensagem ou de uma notificação em um canal de comunicação pode ser tratada como a ocorrência de um evento. Além disso, o processo que recebe uma mensagem não precisa estar

pronto para executá-la quando a comunicação é iniciada. Por outro lado, o processo que envia a mensagem não precisa aguardar que ela seja tratada no momento da comunicação. Os eventos podem ser guardados em uma fila para serem tratados quando o processo que os recebeu estiver disponível. Um *loop* que executa em cada processo é responsável por tratar cada evento recebido.

De acordo com [34], paradigmas dirigidos a eventos são candidatos naturais para se projetar coordenação em sistemas de componentes, onde a interação entre componentes é complexa e partes da computação podem ser escritas em diferentes linguagens de programação.

Ainda segundo [23], sistemas assíncronos são a questão intelectual dominante numa era de pesquisa em sistemas de computadores, onde surgiram pequenos processadores e "selvas" de computadores densamente conectadas. Diversidade entre os elementos de um sistema computacional – diversidade com relação à linguagem, plataforma de hardware, localização física, e até mesmo ao modelo de computação básico – é normal nessa nova era. Sistemas assíncronos são uma coleção de atividades assíncronas que se comunicam, onde uma atividade é um processo, thread ou qualquer outro agente que seja capaz de simular uma máquina de Turing. Poderia ser uma pessoa, ou até mesmo um outro sistema assíncrono inteiro.

PUC-Rio - Certificação Digital Nº 9824820/CA

2.3 O Mecanismo de Coordenação ALua

Neste trabalho, não construímos uma linguagem de coordenação, mas optamos por adotar uma linguagem de extensão completa que pode ser usada para o propósito de coordenação com fronteiras menos rígidas.

A linguagem Lua [16, 17] possui características especiais que a fazem uma poderosa linguagem de extensão de alto nível. Em Lua, é possível definir e manipular funções como valores de primeira ordem. Lua possui vetores associativos dinâmicos, que são o mecanismo básico de estruturação de dados da linguagem, coleta de lixo, que evita a necessidade de se gerenciar explicitamente a alocação dinâmica de memória, o mecanismo de *tag methods*, que permite a extensão da semântica da linguagem e facilidades reflexivas, permitindo a criação de partes do programa altamente polimórficas.

Lua é oferecida como uma pequena biblioteca de funções C, que pode ser ligada a aplicações hospedeiras. Um exemplo simples de um cliente Lua é um interpretador *stand-alone*, onde o programa principal permanece em um *loop* no qual lê cadeias

de caracteres do teclado, que são trechos de código Lua, e chama o interpretador de Lua para executar esses códigos, podendo definir variáveis e funções, chamar funções, etc.

No próximo capítulo apresentaremos o ALua, um mecanismo de programação multilinguagem onde os componentes computacionais podem ser escritos em uma linguagem estaticamente tipada, como C, e a parte de configuração e coordenação pode ser escrita em uma linguagem interpretada, como Lua. O ALua [35, 36, 15] é um mecanismo de comunicação assíncrona para aplicações paralelas distribuídas. ALua utiliza um modelo *single-threaded*, dirigido a eventos, e baseado na linguagem Lua.

O modelo dirigido a eventos que usamos no ALua segue o estilo de execução reativa de processos, que se baseia no envio e recebimento assíncrono e direto (ponto a ponto) de eventos entre processos. Este modelo é diferente do paradigma de interação por eventos [37], onde um componente pode anunciar um ou mais eventos e outros componentes podem registrar interesse em um evento, associando a ele uma função. Quando o evento é anunciado, o sistema chama implicitamente todas as funções que foram registradas para aquele evento.

O modelo de programação do ALua é similar ao modelo assíncrono descrito em [38], com eventos disparando ações atômicas. No ALua, cada nó da computação possui um ambiente local e independente, e as mensagens enviadas entre dois processos são recebidas na ordem em que foram enviadas, porém não há uma ordem global entre as mensagens de todos os nós.

Com o ALua, é possível replicarmos o modelo tradicional de programação dual para coordenação de aplicações paralelas, onde ALua age como um elemento de ligação, permitindo que partes pré-compiladas do programa sejam executadas em diferentes máquinas. Nesse contexto, as aplicações são divididas em duas partes, *núcleo* e *configuração*, normalmente escritas em linguagens diferentes. O núcleo implementa os componentes básicos do sistema e normalmente é escrito em uma linguagem compilada e estaticamente tipada, como C ou C++. A parte de configuração, que normalmente é escrita em uma linguagem interpretada, conecta esses componentes definindo a forma final da aplicação [27, 39]. Com esse modelo, podemos construir aplicações flexíveis sem comprometer seu desempenho [13].

Neste trabalho queremos mostrar que o papel de um mecanismo de coordenação pode ser muito mais que o de um elemento de ligação de componentes. Através de um modelo de coordenação adequado, é possível interferir na execução de uma aplicação, mesmo após ela ter sido iniciada, redefinindo funções, modificando variáveis ou até mesmo disparando novos processos. Com o ALua, temos não apenas

PUC-Rio - Certificação Digital Nº 9824820/CA

uma ferramenta de programação paralela distribuída, mas um mecanismo de coordenação que envolve o desenvolvimento, configuração, monitoramento e adaptação de aplicações distribuídas. Com esse mecanismo, é possível, por exemplo, escrever uma aplicação com a qual o usuário final pode interagir dinamicamente, através de um console. No próximo capítulo apresentamos esse modelo.

PUC-Rio - Certificação Digital Nº 9824820/CA