6
Conclusão

Neste trabalho propusemos usar um modelo multilinguagem para programação paralela distribuída. Em trabalhos anteriores [13], usamos este modelo no domínio de clusters e redes locais. Reimplementamos em ALua aplicações que já haviam sido implementadas no modelo procedural tradicional, usando a biblioteca PVM e a linguagem C, para verificar a adequação do ALua para desenvolver e configurar aplicações, assim como avaliar questões de desempenho desse mecanismo. O ALua se mostrou adequado para prototipação e exploração de alternativas no desenvolvimento de aplicações paralelas distribuídas.

Neste trabalho, mostramos que além de desenvolver e configurar aplicações, o modelo do ALua pode ser usado no contexto de coordenação de aplicações. A flexibilidade oferecida pelo ALua nos permite usar o conceito de coordenação de uma forma mais ampla, para monitorar uma aplicação e adaptá-la de acordo com alterações em seu ambiente de execução. Mais que isso, com o ALua podemos usar técnicas para monitorar o ambiente de execução e interagir com a aplicação através de uma entidade externa, que pode ser um programa monitor ou até mesmo um usuário que envia comandos (mensagens com código) através de um console interativo.

O mecanismo do ALua é baseado na linguagem de extensão Lua. Linguagens de extensão prevêem a separação da aplicação em duas partes, núcleo e configuração, geralmente escritas em linguagens distintas. Modelos de coordenação também prevêem a separação da aplicação em duas partes, computação (núcleo) e coordenação, geralmente escritas em linguagens diferentes, onde coordenação engloba comunicação e cooperação (configuração) entre os componentes da aplicação.

Podemos dividir uma aplicação distribuída em várias partes: computação, configuração e cooperação, comunicação de dados e comunicação de controle. Acreditamos que cada uma destas partes pode fazer uso de uma linguagem distinta e mais apropriada para sua tarefa.

Neste modelo, não estamos redefinindo o conceito de coordenação, mas o estamos usando num sentido mais amplo, onde a linha de separação entre essas diversas partes da aplicação tornam-se menos rígidas, o que nos permite maior flexibilidade na interação entre elas. Podemos usar linguagens diferentes não só para as diversas partes da aplicação, mas numa mesma parte. Por exemplo, alguns componentes computacionais da aplicação podem ser escritos em C, outros em Lua ou ainda podemos ter componentes que utilizam ambas as linguagens. Além disso, parte da comunicação da aplicação pode ser escrita em ALua e outra em MPI. Cabe ao programador decidir onde ele deseja mais flexibilidade e onde a eficiência é mais importante.

6.1 Principais Contribuições

No contexto de redes geograficamente distribuídas, o ambiente de execução é instável e por isso, as aplicações precisam se adaptar a mudanças em seu ambiente. Neste trabalho, avaliamos as contribuições que o modelo utilizado pelo ALua pode trazer para o desenvolvimento de aplicações paralelas distribuídas, como um modelo de coordenação de aplicações, oferecendo uma ferramenta para desenvolvimento e coordenação de aplicações e monitoramento do próprio ambiente de execução.

Através do uso de uma linguagem de extensão como Lua, trazemos ao domínio de aplicações paralelas distribuídas a possibilidade de se ter o papel de um usuário administrador que pode não só monitorar o ambiente e suas aplicações, mas também interferir na execução de uma aplicação durante sua execução.

No Capítulo 2, falamos da importância de modelos de coordenação no desenvolvimento de aplicações paralelas distribuídas e mostramos como linguagens de extensão se encaixam bem neste modelo. Neste trabalho, optamos por adotar uma linguagem de extensão completa, Lua, para ser usada para o propósito de coordenação com fronteiras menos rígidas.

No Capítulo 3 apresentamos o ALua e mostramos como ele pode ser usado para coordenar uma aplicação trocando mensagens com trechos de código entre os processos que a compõem. Mostramos também que através de um console interativo, o usuário final pode adicionar ou alterar dinamicamente a funcionalidade dos processos ALua envolvidos na aplicação. Este modelo foi empregado com sucesso na construção de diferentes aplicações paralelas e distribuídas [15].

Na Seção 3.4 avaliamos a flexibilidade que o ALua pode trazer para pro-

gramação paralela. Além da facilidade com a qual os diferentes algoritmos propostos foram implementados, um resultado importante desta seção foi mostrar o suporte que o ALua oferece para testes e prototipação. Na implementação do paradigma de programação 'Multiplicação de Tuplas', pudemos construir uma única infra-estrutura baseada no modelo de comunicação escolhido, permitindo a experimentação com diferentes implementações, sem a necessidade de testar e depurar o padrão de comunicação básico (paralelo ou seqüencial) para cada aplicação. Na discussão da aplicação de times de agentes assíncronos, mostramos como o ALua pode ser útil para aplicações com longo tempo de execução. A facilidade de um console interativo que o programador pode usar para monitorar e controlar uma aplicação dinamicamente é uma ferramenta importante neste cenário.

No Capítulo 4, apresentamos o ALua multicanal, uma extensão do modelo do ALua que o torna apropriado para aplicações que precisam transferir grandes *streams* de dados. Mostramos que o ALua permite a uma aplicação *single-threaded* lidar concorrentemente com diferentes fluxos de dados e controle, associando a flexibilidade de se transmitir código Lua através de um canal principal de controle à possibilidade de permitir que o programador defina funções para tratar os fluxos de dados nos canais secundários. Também mostramos que por causa de suas características dinâmicas, o ALua multicanal também se encaixa bem no contexto de aplicações multimídia distribuídas [51].

O modelo de programação multilinguagem do ALua, que permite que o código de comunicação (e coordenação) seja escrito em ALua e o código de computação seja escrito em uma linguagem compilada, nos permite obter flexibilidade sem ter que pagar um preço muito alto pelo desempenho. Os resultados descritos na Seção 4.2, apesar de preliminares, são estimulantes, uma vez que indicam que o custo de usar o ALua para aplicações paralelas pode ser relativamente baixo. Estes custos são bastante consistentes com os custos de se usar uma linguagem interpretada em aplicações seqüenciais [67]. É importante lembrar que, como discutido em [68], ao medir o desempenho de uma aplicação paralela ou ferramenta de programação, é preciso considerar métricas diferentes. A facilidade de construção de uma aplicação e o tempo que o programador leva para implementá-la podem ser mais importantes que o tempo final de execução. Algumas áreas de programação distribuída, como serviços WWW, já estão trocando eficiência por flexibilidade através do uso de linguagens como Perl e Java.

No Capítulo 5, apresentamos um mecanismo de monitoramento e políticas de adaptação mais automáticas para as aplicações. Em seguida, mostramos como esses mecanismos podem ser aplicados a aplicações paralelas para grades, mais especificamente para ambientes de grade que utilizam o Globus [12].

Mostramos ainda que o ALua pode ser usado em diferentes níveis de coordenação (comunicação). É possível, por exemplo, utilizar uma biblioteca de comunicação diferente do ALua para a comunicação da tarefa principal da aplicação e usar o ALua apenas como instrumento de monitoramento e adaptação, de modo que ele fique responsável pelas mensagens que carregam código e que modificam o comportamento do programa, enquanto bibliotecas de comunicação tradicionais, como o MPI, ficam responsáveis pela massa de dados trocada entre os diversos processos da aplicação.

O ALua já foi utilizado no desenvolvimento de vários projetos de pesquisa. Alguns desses trabalhos implementaram abstrações de comunicação assíncrona sobre o modelo flexível, mas relativamente baixo nível, de envio de código oferecido pelo ALua. O LuaTS [26] usou o ALua para implementar um espaço de tuplas reativo, semelhante ao espaço de tuplas de Linda [6]. O ORFEO [33] construiu uma abstração sobre o ALua para dar suporte à comunicação por chamadas a funções remotas, as quais possuem o mesmo status de primeira classe de funções locais. Em outra linha, o LuaSpace [69] usou o ALua como ferramenta para reconfiguração dinâmica de aplicações.

O mecanismo do ALua também foi utilizado no desenvolvimento de uma aplicação multimídia distribuída, o DynaVideo [51], onde um processo gerente envia código para configurar um processo chamado de servidor secundário. No DynaVideo, a implementação do servidor secundário (Seção 4.3.1) é toda feita em ALua, o que nos permite explorar a flexibilidade que o ALua oferece ao máximo. O gerente pode enviar todo o código que será executado pelo servidor secundário, ou ainda alterar o algoritmo que ele usa mesmo após o início de sua execução. Além disso, todo o modelo de monitoramento e adaptação apresentado no Capítulo 5 não só se adequa a esta aplicação como é muito fácil de se integrar.

6.2 Trabalhos Correlatos

Ousterhout [31] foi um dos primeiros a defender o uso de uma linguagem "dura" com um linguagem de script (chamada de uma linguagem embutida). Este trabalho também usou a idéia de usar trechos de código como conteúdo de mensagens para comunicação entre processos. Entretanto, até onde sabemos, esta idéia não foi aplicada a algoritmos paralelos ou distribuídos (apesar de ter sido usada para comunicação entre componentes no *toolkit* de interface Tk).

Alguns grupos usaram Tcl para programação distribuída. Tcl-DP é uma extensão de Tcl para programação distribuída [70]. Entretanto, seu objetivo é facilitar a programação de aplicações cliente/servidor, baseadas em soquetes, e não oferecer modelos de comunicação de um nível mais alto. Em outra direção, o projeto Agent Tcl [71] usa Tcl como base para um sistema de agentes. Um agente migrando de uma máquina para outra poderia de algumas formas ser comparado a capacidade do ALua de enviar código em suas mensagens. Entretanto, agentes que chegam em uma nova máquina são executados em um novo ambiente, enquanto no ALua, quando um processo recebe uma mensagem ele a executa em seu próprio ambiente, com acesso a variáveis e funções existentes. Novamente, uma vez que agentes podem trocar mensagens, o modelo de comunicação do ALua poderia ser simulado, entretanto, este não é o objetivo do trabalho.

Uma desvantagem de sistemas que usam Tcl é o desempenho. "The Great Computer Language Shootout" [67], um *benchmark* abrangente entre dúzias de linguagens, reporta o desempenho de Lua como sendo tipicamente duas a cinco vezes mais rápido que programas correspondentes em Tcl.

O trabalho [10] classifica modelos de coordenação em duas amplas categorias: *orientada a dados* e *orientada a controle*. A idéia principal desta classificação é que na coordenação orientada a dados, o estado da computação em qualquer instante é definido pelos valores dos dados sendo trocados e pela configuração dos componentes coordenados, enquanto na coordenação orientada a controle o valor dos dados que estão sendo manipulados não estão envolvido na definição do estado da computação. Supostamente, modelos orientados a controle permitiriam uma separação clara entre interesses de programação e de configuração.

De acordo com a definição acima, o ALua seria classificado como uma linguagem de coordenação orientada a dados, uma vez que a informação recebida nas mensagens pode conter código e por isso pode redefinir partes de componentes e padrões de interação. Entretanto, em contraste com exemplos típicos do paradigma orientado a dados, como Linda [6], o modelo do ALua não é baseado na adição de primitivas de comunicação a uma linguagem existente: ele propõe o uso de uma linguagem de programação completa como um elemento de ligação entre componentes compilados e distribuídos. Isto permite ao programador separar completamente a programação básica de componentes da configuração, se ele assim desejar. Entretanto, uma vez que Lua é uma linguagem de programação completa e possui uma interface com C muito flexível, diferentes níveis de integração entre o código de ligação e a computação "dura" podem ser escolhidos pelo programador. Acreditamos que esta escolha é uma facilidade importante para o programador. Como discutido em [72], a separação entre o código de coordenação e computação pode ser muito

difícil de manter quando se requer facilidades de coordenação dinâmicas. A mesma questão é ilustrada pelos mecanismos para instanciação dinâmica em Darwin [21]. Apesar da linguagem de configuração de Darwin, chamada Regis, incluir um mecanismo de instanciação dinâmica elegante para estruturas recursivas, a instanciação dinâmica de módulos arbitrários deve ser programada diretamente no código computacional.

O modelo do ALua pode ser melhor comparado ao paradigma *MESSENGERS* apresentado em [40]. Este paradigma é baseado em objetos *Messenger*, que podem navegar entre os nós de uma rede, realizando ações de navegação e computação expressas com o uso de C. Messengers são compilados para código intermediário que pode ser movido dinamicamente entre as máquinas. A biblioteca *messenger* inclui funções para chamar separadamente, funções C pré-compiladas, permitindo a integração de código desenvolvido de forma independente.

O trabalho [40] descreve o paradigma *MESSENGERS* no contexto de uma discussão sobre algoritmos de troca de mensagens. A expressão *objetos autônomos* é usada para denominar paradigmas onde mensagens foram elevadas de serem simples carregadoras de dados para uma forma mais elevada, de modo que alguma informação de comportamento pode ser carregada por cada mensagem e interpretada pelo receptor. Este trabalho ainda classifica estes modelos em duas linhas: autonomia navegacional e composição dinâmica. Autonomia navegacional se refere ao grau no qual uma mensagem pode incluir decisões sobre seu próprio destino (que não seja seu receptor imediato). O ALua permite ao programador desenvolver sistemas onde mensagens contêm todo o comportamento de um sistema, e os nós são meros interpretadores de mensagens. (A mensagem auto-replicante da Seção 3.4.1 segue esta abordagem.) A composição dinâmica representa o grau com o qual objetos autônomos podem ativar programas independentes e carregar novo comportamento funcional para diferentes nós. O ALua (ou, mas especificamente, Lua) pode chamar outros processos, pode carregar e chamar funções escritas em Lua e pode chamar funções escritas em C previamente ligadas ao interpretador. Além disso, Lua possui uma biblioteca não padrão que a permite carregar bibliotecas C pré-compiladas dinamicamente [66].

Outra linguagem para conectar aplicações distribuídas é Glish [73]. Apesar de não ser tipicamente citada neste contexto, Glish pode ser facilmente vista como uma linguagem de coordenação. Assim como ALua, Glish adota uma abordagem orientada a eventos, e utiliza mais de uma linguagem de programação. Entretanto, diferente de ALua, onde cada agente pode ser multilinguagem (com um núcleo escrito em C e uma camada de comunicação escrita em Lua), em Glish há um único processo *mestre* escrito em Glish, que coordena o trabalho dos escravos escritos inteiramente

em C. Como os escravos são codificados em uma linguagem compilada, eles podem tratar apenas um conjunto fixo de mensagens, com tipo de conteúdo pré-definidos.

A linguagem Piccola [20] é uma linguagem de composição, que assim como o ALua, combina as aspectos de linguagens de script e linguagens de coordenação. Entretanto, Piccola foi especialmente projetada para compor componentes de software e propõe uma separação clara entre implementação e composição de componentes, enquanto o ALua é baseado em uma linguagem de programação completa e propõe uma divisão mais flexível entre elementos computacionais e suas relações.

Na Seção 4.3, mostramos uma aplicação desenvolvida em Aglets e em ALua. Aglets é uma biblioteca para dar suporte a agentes móveis, totalmente escrita em Java. Aglets possui uma arquitetura dirigida a eventos e oferece suporte a migração de código. Apesar do ALua não oferecer suporte direto a migração de código, no sentido de suspender a execução de uma aplicação e continuá-la em outra máquina no mesmo ponto, no caso de aplicações implementadas totalmente em ALua e que não guardam o estado da computação que realizam, é possível utilizar o ALua para este propósito.

O trabalho [74] apresenta um projeto de pesquisa que envolve um processo iterativo de definição, desenvolvimento e aplicação de *Commodity Grid Toolkits* (CoGs): conjuntos de componentes genéricos que mapeam funcionalidades de grades, mais especificamente do Globus [12], em ambientes ou *frameworks* de tecnologias específicas, como Java, Web/CGI, CORBA, DCOM e Python. A palavra 'mapear' é importante: a integração entre grades e outras tecnologias não é apenas um problema de definição de interfaces, mas uma questão de como os conceitos e serviços de grades são melhor expressos em termos de conceitos e serviços de um *framework* de uma tecnologia particular.

O CoG Kit para Java já está sendo desenvolvido e o grupo está investigando o mapeamento dos componentes de serviços do Globus para diversas linguagens, em particular Perl e Python, de modo a oferecer suporte à rápida prototipação de aplicações para grade e programação para Web, baseada em scripts CGI. Além disso, o projeto tem interesse em oferecer acesso a serviços de grade através de *frameworks* de computação distribuída de alto nível. Para isso, consideram integrar ao Globus às ferramentas e ambientes de desenvolvimento mais utilizados, como DCOM e CORBA.

O trabalho desenvolvido para o Java CoG Kit [74] é similar ao que desenvolvemos na Seção 5.2 para Lua, onde criamos um mapeamento entre Lua e serviços oferecidos pelo Globus, como o MDS [61] e GRAM [60]. Podemos dizer que implementamos um Lua CoG Kit, que também oferece suporte à rápida prototipação de

aplicações para grades. O sistema LuaOrb [53] oferece mapeamentos entre a linguagem Lua e componentes CORBA, COM e Java, permitindo a interoperabilidade entre esses componentes. Acreditamos que a integração desses dois trabalhos pode permitir ao desenvolvedor de aplicações explorar os serviços de grades (como gerenciamento de recursos, segurança, descoberta de recursos) enquanto desenvolve componentes de alto-nível utilizando a ferramenta que julgar mais apropriada.

Rana & Walker [75] descrevem o *Agent Grid*, uma abordagem baseada em agentes para integração de serviços e recursos para estabelecer ambientes de solução de problemas (PSE - *Problem Solving Environments*) multi-disciplinares. Segundo eles, o gerenciamento de dados normalmente não aparece na maioria das aplicações de computação de alto desempenho, embora a velocidade com a qual os dados podem ser movidos de e para sistemas de armazenamento secundários e terciários seja uma ordem de magnitude menor que a taxa de processamento.

Por isso, no *Agent Grid*, eles defendem a idéia de estender a noção de agentes colaborativos para oferecer mobilidade de código, de modo que algoritmos numéricos possam migrar através da grade, evitando-se a necessidade de migrar grandes quantidades de dados. No estudo de caso apresentado, um agente carrega o algoritmo, implementado em Java, a ser utilizado na execução de uma tarefa até a fonte dos dados. A facilidade que o ALua oferece de enviar mensagens com trechos de código é similar em vários aspectos ao modelo do Agent Grid.

6.3 Trabalhos Futuros

Como trabalho futuro, pretendemos estender a implementação do Lua CoG Kit e fazer a sua integração com o LuaOrb. Desta forma permitiremos o acesso a facilidades como o serviço de Trader, etc. Ainda nesta linha, pretendemos implementar um mapeamento da biblioteca de comunicação MPI para Lua (*luampi*) de modo a facilitar a integração de aplicações que usam o modelo do ALua com MPI.

Uma outra linha de trabalhos futuros envolve a questão de segurança no modelo do ALua. Neste caso, podemos adotar duas abordagens distintas: usar bibliotecas ssl (*secure sockets layer*) para troca de mensagens ALua e construir um modelo *sandbox* para o interpretador ALua.

Queremos ainda explorar o uso do ALua em aplicações reais, possivelmente através de uma parceria com projetos do Grid-Rio, onde podemos fazer uso do ALua para monitorar e adaptar aplicações. Queremos também, estudar a possibilidade de,

usando o ALua, determinar os serviços que irão compor o ambiente de grade.

Também temos interesse em explorar mais o uso do ALua em aplicações que precisam trocar grandes fluxos de dados, como é o caso de aplicações multimídia, tirando proveito da possibilidade de criar e reconfigurar dinamicamente o comportamento de canais concorrentes de comunicação, através do modelo *single-threaded* do ALua.