1
Introdução

A *mobilidade –ou migração– de computações* é comumente definida como o movimento de uma computação entre diferentes dispositivos. Computação aqui se refere a uma execução, seja na forma de processos, threads, agentes móveis, ou registros de ativação. Durante a migração, a execução é enxergada como sendo dados: a migração de computações consiste basicamente na manipulação e transmissão do estado de execução capturado na origem de modo que possa ser restaurado no destino na forma de uma nova computação equivalente. A implementação deste procedimento envolve tanto os mecanismos internos que permitem executar o movimento, quanto o mapeamento desses mecanismos em construções da linguagem ou do sistema.

A migração permite otimizações baseadas na melhoria da localidade ou na mudança dinâmica da topologia da aplicação. Melhorias da localidade podem ser obtidas, por exemplo, ao executar a computação perto dos dados ou de recursos ou serviços especiais, como bancos de dados volumosos ou software licenciado. Mudanças dinâmicas da topologia ocorrem em aplicações de balanceamento de carga, minimização do *downtime* em serviços contínuos, suporte para operação desconectada, entre outros.

Embora largamente pesquisada [Smith88, Eskicioglu90, Nuttall94, MDPW+00], esta técnica não tem sido extensivamente aplicada na prática. Dentre as possíveis causas estão sua relativa complexidade e as dificuldades em obter bom desempenho, assim como aspectos de segurança e até fatores sociológicos [CHK94, MDPW+00]. Por exemplo, aplicações baseadas em agentes móveis são interessantes do ponto de vista acadêmico mas são difíceis de levar na prática por causa dos problemas de segurança que lhes são inerentes. Porém, existem atualmente várias áreas, incluindo as de grades computacionais e computação ubíqua, em que o desempenho da migração fica em um segundo plano quando comparado às capacidades que acrescenta. Nesses contextos, a migração pode permitir, por exemplo, mover computações que estão executando remotamente para devolver o controle da máquina ao seu dono, ou a transferência de aplicações em execução de/para um dispositivo sem fio. Também, aplicações de tempo real podem ser beneficiadas por este

tipo de procedimento.

Um aspecto comum a essas áreas é a sua potencial heterogeneidade. Quando os componentes podem se mover entre plataformas diferentes a migração é chamada de *heterogênea*. Diferentemente da migração heterogênea, a migração chamada de *homogênea* se executa entre plataformas similares. Ela tipicamente se baseia na cópia integral do conteúdo da memória para o nó destino com no máximo alguns ajustes. A migração heterogênea é mais complexa do que a homogênea. Para que a migração heterogênea possa ser executada, precisa-se conhecer a estrutura dos dados, pois cada dado deve ser extraído de forma que, após as devidas traduções, ele possa ser entendido na máquina destino. Isto implica, por um lado, na necessidade de trabalhar no nível da aplicação: apesar das vantagens de implementar a migração no nível de *kernel* em termos de desempenho e transparência (no sentido de que os detalhes do procedimento estão ocultos para o usuário), implementações de migração no nível do kernel dependem do sistema operacional, o que em geral as invalida em ambientes heterogêneos. A migração heterogênea forte de computações é comumente definida como aquela em que o estado de execução também migra [FPV98]. A maioria das linguagens de programação atuais não oferece o suporte para a captura e restauração do estado de execução que a implementação de migração heterogênea forte de computações precisa [MRS08]. Para superar essa limitação, os programadores são forçados, no nível da aplicação, a usar truques que afetam a portabilidade ou o desempenho, ou ambos.

O suporte de linguagens para a captura e restauração heterogênea do estado de execução de computações pode ser caracterizado como a provisão de construções que permitam ambos os procedimentos em um ambiente heterogêneo. Este suporte deve oferecer um substrato comum para a implementação de diferentes tipos de aplicações que precisam de captura e restauração heterogêneas.

Como exemplo, há uma grande similaridade dos requisitos da migração forte e a persistência de execuções. Entretanto, ambos os procedimentos são diferentes no nível da aplicação. A migração requer suporte para coordenação on-line, assincronismo e comunicação em um ambiente distribuído, e, possivelmente, suporte para a manipulação de exceções remotas. As aplicações de persistência, por outro lado, envolvem problemas relacionados com a restauração atemporal da computação. Ainda no contexto mais restrito da mobilidade, podem existir diferenças substanciais entre as aplicações. Estas diferenças estão relacionadas à granularidade dos valores migrados, quanto do estado de execução será transferido (vital para o desempenho da migração), os métodos de serialização e a forma em que a computação será re-vinculada ao novo

contexto local, entre outros. Por exemplo, uma aplicação de balanceamento de carga deve precisar migrar a computação como um todo; já se o objetivo é migrar uma computação para perto dos dados, só um registro de ativação poderia ser suficiente. Na construção do grafo de dependências, algumas referências podem ser anuladas ou então, re-vinculadas no destino. Esses aspectos, freqüentemente fixados nos *frameworks* de migração, são políticas e devem ser considerados como escolhas de implementação da aplicação do ponto de vista do projeto da linguagem. O ideal é que a linguagem ofereça mecanismos flexíveis, e não políticas específicas. Ao acrescentar funcionalidades a uma linguagem de programação de propósito geral, é necessário manter a simplicidade da linguagem e evitar redundâncias e conflitos, mas ao mesmo tempo oferecer suporte para tantas aplicações quanto for possível, privilegiando a generalidade.

Entretanto, uma abordagem escolhida com freqüência para a implementação de mecanismos de suporte a captura e restauração de estado –que pode ser chamada de *opaca* ou *caixa-preta*– consiste em seguir uma semântica predeterminada pelo sistema. Embora fáceis de usar, essas implementações padecem de uma falta de versatilidade que as faz pouco adaptáveis a diferentes aplicações. Para satisfazer esse requisito é necessário oferecer mais controle sobre os procedimentos. Abordagens baseadas na reflexão do estado de execução permitem manipular o estado de execução no nível da linguagem, na forma de valores portáveis e serializáveis. Já que é o programador quem conhece a semântica de cada aplicação, ele deveria poder decidir como ajustar as variáveis que determinam o comportamento dos procedimentos de captura e restauração (através de chamadas diretas às primitivas da linguagem ou através de bibliotecas que implementam diferentes políticas). Em conseqüência, os requisitos de diferentes aplicações podem ser atendidos, aumentam-se as chances de uma restauração bem sucedida, e o desempenho da captura e restauração pode ser controlado, além de facilitar a depuração.

1.1 A tese proposta

Essa tese partiu do estudo das dificuldades associadas à implementação da migração heterogênea forte de computações em execução. Após estudar os trabalhos nessa área concluímos que há necessidade de que as linguagens de programação ofereçam mecanismos que permitam esta implementação. Resolvemos então estudar estes mecanismos. Linguagens de programação devem suportar a implementação de diferentes tipos de aplicações, como é o caso de migração e persistência heterogêneas. Deste modo, este trabalho se dedica ao

estudo do problema chamado de *suporte flexível para a captura e restauração heterogêneas*, e consiste em como produzir uma nova computação – como conseqüência de procedimentos como migração ou persistência de computações, possivelmente heterogêneos– cujo estado de execução seja equivalente ao original.

Acreditamos que o suporte das linguagens deva ser baseado na manipulação explícita da representação das estruturas básicas que definem o estado do programa. A nossa abordagem se baseia em dois mecanismos que chamaremos de *reificação* e *instalação* das estruturas da linguagem. O suporte para reificação e instalação de execuções foi implementado como uma extensão de Lua 5.1 [Ierusalimschy06] chamada de *LuaNua*. Várias facilidades reflexivas de Lua já oferecem algum suporte para captura e restauração. LuaNua estende o conjunto de funcionalidades reflexivas de Lua através da modificação e extensão da biblioteca de depuração.

Abordagens reflexivas, quando permitem a manipulação de estruturas navegáveis e componíveis, são meios poderosos e expressivos de implementar o suporte para várias aplicações. Eles deixam o programador no controle de cada aspecto do procedimento. Entretanto, esta abordagem afeta a usabilidade, sobrecarregando o programador e levando facilmente a erros [Kennedy04, SLW+07]. Espera-se que a API reflexiva seja usada através de bibliotecas intermediárias ("*bibliotecas de facilitadores*") que permitem uma certa customização para determinados conjuntos de aplicações, enquanto as primitivas oferecidas pela linguagem continuam acessíveis para aplicações ou propósitos mais específicos. Ao nos referirmos ao "programador" neste trabalho, estaremos falando tanto do usuário da API quanto do usuário dessas bibliotecas de mais alto nível.

A captura e restauração do estado de execução é uma operação intrinsecamente relacionada com o sistema em que a computação está sendo executada. Este estado pode ser descrito em termos do estado interno e a informação do ambiente que é a visão que a computação tem do sistema. A solução proposta nesta tese consegue lidar com a necessidade de manter o estado interno consistente ainda depois da restauração. Esta solução também permite manipular a visão que a computação tem do sistema, de forma a fazer a restauração possível. Entretanto, isto pode não ser suficiente nos casos em que o ambiente contém entidades especiais como descritores de arquivo e *sockets* abertos. Tratá-los de forma a permitir sua restauração correta requer a utilização de técnicas adicionais como a implementação de *wrappers*. Este problema é tratado em outros trabalhos ([LTBL97, DO99]) e está fora do escopo desta tese.

1.2

Organização do trabalho

O Capítulo 2 introduz os termos que serão usados ao longo do texto, em particular aqueles relacionados à migração heterogênea de computações. Já que a dificuldade fundamental para a implementação dessa técnica está na captura e restauração do estado de execução, discute-se também a forma como as linguagens de programação atuais oferecem esse suporte. Durante o estudo dos trabalhos relacionados, detectamos características comuns entre os métodos atuais em diferentes granularidades e linguagens. Isto motivou a proposta de uma visão diferente para as classificações dos métodos de implementação de migração heterogênea forte de computações, focando no suporte que as linguagens de computação oferecem para a captura e restauração de estado.

No Capítulo 3 se discutem os aspectos relacionados com o projeto de mecanismos de suporte a captura e restauração e a nossa abordagem para a captura e restauração flexível de computações, que é o foco desta tese. A API proposta é apresentada e a semântica operacional dessas funções é definida formalmente.

No Capítulo 4 é validada a efetividade da proposta através de implementações de diferentes exemplos.

O Capítulo 5 comenta detalhes relevantes da implementação de LuaNua. O último Capítulo destina-se às conclusões.