

Clumpt: Um Sistema *Peer-to-Peer* para Instalação e Manutenção de *Software* em Aglomerados de Computadores*

Diego Kreutz^{1,2}, Marcelo Neves⁴,
Elton Mathias⁴, Tiago Scheid², Andrea Charão², Rafael Righi³

¹Universidade Luterana do Brasil, Campus Santa Maria (ULBRA SM)
Caixa Postal 21834 – 97020-001 – Santa Maria – RS – Brasil

²Laboratório de Sistemas de Computação (LSC)
Curso de Ciência da Computação – Universidade Federal de Santa Maria
Campus UFSM – 97105-900 – Santa Maria – RS – Brasil

³Faculdade de Tecnologia SENAI Florianópolis
Serviço Nacional de Aprendizagem Industrial — SENAI-SC
Rodovia SC 401, n. 3730 — 88.032-005 — Florianópolis - SC - Brasil

⁴Instituto de Informática (II)
Programa de Pós-Graduação em Computação (PPGC)
Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15064 – 91501-970 – Porto Alegre – RS – Brasil

{kreutz, scheid, andrea}@inf.ufsm.br

{mvneves, enmathias}@inf.ufrgs.br, righi@ctai.senai.br

Abstract. *This paper presents the design and implementation of Clumpt, a system to ease software installation and maintenance in computer clusters, which are broadly used for applications requiring high amounts of computational resources. An important characteristic of this system is its peer-to-peer architecture, with data storage and software updates implemented in a decentralized manner. This allows the system to provide scalability and fault tolerance to the software installation process. Such characteristics distinguish Clumpt from the existing solutions.*

Resumo. *Este artigo apresenta o projeto e implementação de Clumpt, um sistema que auxilia na instalação e manutenção de software em clusters de computadores, utilizados para execução de aplicações que demandam grandes quantidades de recursos computacionais. Uma importante característica deste sistema é sua arquitetura peer-to-peer, onde o armazenamento de dados e o processamento de atualizações de software são feitos de forma descentralizada. Com isso, o sistema é capaz de proporcionar escalabilidade e tolerância a falhas ao processo de instalação de software, o que distingue Clumpt das demais soluções existentes.*

*Trabalho parcialmente financiado por FAPERGS e CNPq.

1. Introdução

Aglomerados (*clusters*) de computadores estão hoje em dia amplamente disseminados nas mais diversas áreas de aplicação que demandam alto desempenho e/ou alta disponibilidade. O gerenciamento de software nestas arquiteturas, no entanto, é uma tarefa pouco trivial [Harr and Denault 2002, Baldassari et al. 2005]. De fato, em sistemas formados por um grande número de nós, a instalação, configuração e atualização individual de cada máquina é praticamente inviável. Este cenário torna-se especialmente crítico com a proliferação de *clusters* heterogêneos, caracterizados por uma diversidade de recursos de hardware e/ou software.

Com o objetivo de minimizar estas dificuldades, desenvolveram-se várias soluções automatizadas para auxiliar na administração de software em *clusters*, tais como FAI [Lange 1999], SystemImager [Finley 2005] e SCMS [Uthayopas and Rungsawang 1999]. Algumas dessas ferramentas possibilitam a instalação completa de sistemas, enquanto outras se propõem a efetuar uma instalação parcial, pressupondo que o sistema operacional já se encontra instalado no *cluster*. A instalação completa costuma utilizar imagens de sistemas, que são replicadas nas máquinas do *cluster*. A instalação parcial, por outro lado, consiste geralmente na instalação de pacotes ou atualizações de software de forma paralela.

Neste contexto, este artigo apresenta o projeto e implementação do sistema Clumpt, cujo principal objetivo é auxiliar nas tarefas de atualização e manutenção de software em *clusters* homogêneos ou heterogêneos. A justificativa para o desenvolvimento deste sistema vem da carência de ferramentas que combinem escalabilidade, tolerância a falhas e suporte a *clusters* heterogêneos. Levando em consideração este cenário, o sistema Clumpt foi estruturado segundo um modelo *peer-to-peer* (P2P) híbrido [Schollmeier 2001], de forma a evitar gargalos e tolerar possíveis falhas e interrupções no processo de instalação de software.

A próxima seção fornece uma visão geral do sistema Clumpt, apresentando sua arquitetura e suas funcionalidades. Os principais componentes do sistema são descritos em maiores detalhes nas seções 3, 4, 5 e 6. A seção 7 apresenta uma avaliação do sistema e a seção 8 compara-o com alguns trabalhos relacionados. Por fim, a seção 9 apresenta a conclusão e sugestões de trabalhos futuros.

2. Visão Geral do Sistema Clumpt

Conforme mencionado anteriormente, a arquitetura do sistema baseia-se em um modelo P2P híbrido [Schollmeier 2001], onde os *peers* são os nós do *cluster* gerenciado. No contexto deste trabalho, o sistema é dito híbrido pelo fato de que os *peers* armazenam e recuperam seu estado em uma base de dados externa, buscando nessa mesma base informações sobre configuração e *peers* vizinhos¹. Os *peers* interagem entre si sem a interferência de um servidor centralizado, conforme ilustra a figura 1. Cada *peer* mantém informações sobre instalações de software efetuadas via o sistema Clumpt, juntamente com um repositório de pacotes instalados localmente. Dessa forma, cada *peer* atua ao mesmo tempo como cliente e servidor de dados referentes a instalações de pacotes no *cluster*.

¹No contexto deste trabalho, *peers* vizinhos são aqueles conhecidos localmente, endereços armazenados em tabela local, por um determinado *peer*.

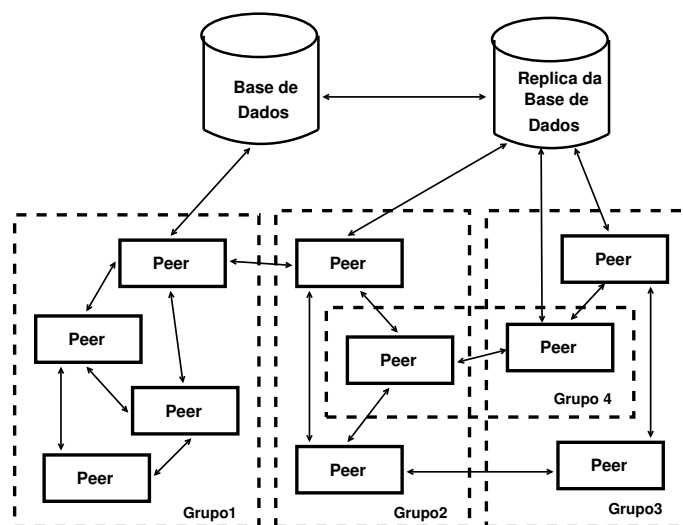


Figura 1. Arquitetura do sistema Clumpt.

Essa estrutura descentralizada cria um ambiente com uma menor probabilidade de formação de gargalos de rede, em comparação à utilização de um repositório de pacotes único. Soluções centralizadas comumente utilizadas em aglomerados de máquinas, como o compartilhamento de pacotes via NFS (*Network File System*), podem congestionar a rede e comprometer a eficiência e escalabilidade do processo de instalação. Além disso, quando existe somente uma rede interligando os nós do *cluster*, um congestionamento devido à manutenção de algum nó poderá interferir no desempenho de aplicações que executam em outros nós.

Além dos benefícios referentes ao desempenho e escalabilidade, uma estrutura descentralizada permite evitar que falhas e interrupções em um determinado *peer* prejudiquem o funcionamento do sistema Clumpt como um todo. Nesses casos de falha, um outro *peer* estará apto a atender a consultas de estado ou satisfazer requisições do mecanismo de atualização de software.

Internamente ao sistema Clumpt, os *peers* são organizados em grupos, de acordo com a estrutura da rede que compõem ou as especificidades de software inerentes a sua finalidade na rede. O agrupamento dos *peers* é realizado, manualmente, pelo administrador do sistema. Uma das principais vantagens desse tipo de organização é a facilidade em gerenciar software de conjuntos de máquinas com sistemas operacionais (ou distribuições) distintos. Além disso, esse tipo de organização permite a separação da rede em sub-redes, definição de grupos de máquinas com hardware heterogêneo, formação de grupos especiais de máquinas destinadas a um fim específico ou mesmo a definição de políticas que definem restrições no uso de determinado software.

Uma base de dados completa a estrutura do sistema Clumpt. Esta base de dados é responsável por armazenar informações globais, como registro e autenticação de *peers*, histórico de instalações/atualizações² submetidas e realizadas, entre outras. Sua principal atribuição é a de manter informações consistentes a respeito do sistema para, em caso de

²O histórico de um *peer* é composto pelo último estado consistente dos dados da tabela de atualizações, local ao *peer*.

falhas, permitir uma restauração do sistema junto aos *peers*. Em casos de falhas normais, os custos de recuperação serão pequenos, visto que praticamente todos os dados de estado e configurações (com exceção dos dados em processamento no instante da falha no *peer* ou no ambiente do *cluster* de computadores) estarão salvos na base de dados e/ou localmente em cada *peer*.

Para aumentar a disponibilidade e segurança de dados referentes ao estado e à configuração do sistema, a base de dados pode ser replicada e distribuída no domínio administrativo do *cluster*. A replicação da base de dados aumenta a tolerância a falhas do sistema, enquanto distribuição proporciona maior disponibilidade, evitando possíveis sobrecargas.

As duas seções seguintes descrevem a estrutura dos *peers* e da base de dados. Mais adiante, apresenta-se maiores detalhes sobre o funcionamento do sistema, para depois descrever-se sua interface com o usuário.

3. Arquitetura dos *Peers*

A arquitetura dos *peers* tem por objetivo principal prover disponibilidade de pacotes e tolerância a falhas. Para tornar isso possível, cada *peer* dispõe de um repositório de pacotes (recebidos de *peers* vizinhos) e uma tabela de atualizações realizadas. Com estes dois componentes, e fazendo uso das informações de configuração e estado armazenadas na base de dados do sistema, um *peer* é cliente e servidor de pacotes para a instalação de software. Além disso, o *peer* pode estar indisponível durante algum período e voltar à ativa, processando todas as atualizações pendentes com base no seu estado e na tabela de pacotes dos vizinhos.

A figura 2 ilustra a arquitetura de cada *peer*. Todos os elementos desta estrutura são encapsulados no programa `clumptd`, que é executado em cada *peer* como um processo *daemon*. Os parágrafos seguintes descrevem em detalhes a composição e o funcionamento destes elementos.

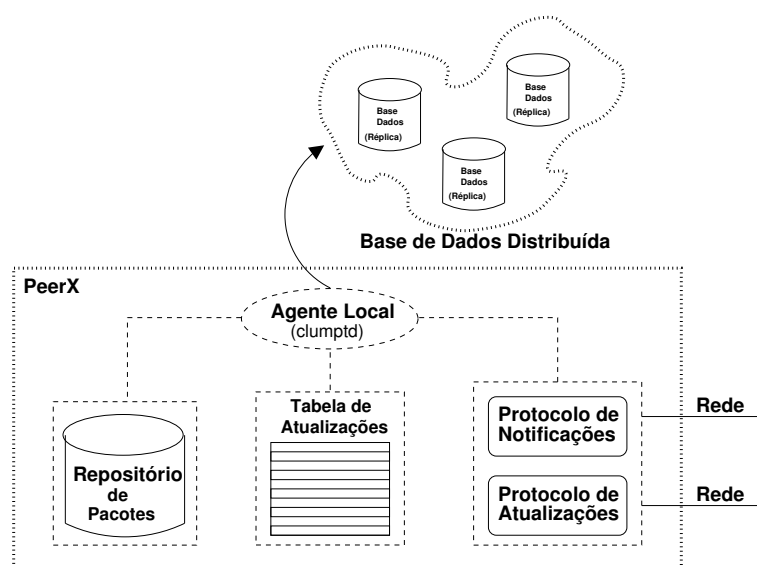


Figura 2. Arquitetura de cada *peer*.

Repositório de Pacotes Diretório responsável por armazenar os pacotes de software recebidos pela rede. Com isso, é possível aos *peers* a obtenção de pacotes junto aos outros membros do sistema, evitando possíveis gargalos ou sobrecargas em nós centrais ou sistemas como o NFS.

Tabela de Atualizações Esta tabela mantém o registro das atualizações submetidas ao sistema Clumpt. Cada entrada dessa tabela contém os seguintes itens: um número seqüencial que identifica o pacote, o caminho aonde o pacote está armazenado, o(s) grupo(s) de máquinas ao qual o pacote se destina e o comando que será utilizado para sua instalação. *Peers* diferentes poderão ter tabelas distintas, devido aos diversos grupos de máquinas aos quais cada um pode pertencer. Mesmo assim, há convergência para um estado de consistência, que é atingido quando todos os *peers* recebem todos os dados das atualizações realizadas pelo sistema. A ordem temporal das atualizações é alcançada graças à geração de identificadores seqüenciais únicos para as atualizações submetidas.

Agente Local Elemento que coordena as atividades dos demais módulos. Cabe ao agente as seguintes tarefas: inicialização do sistema e integração do *peer* à rede P2P, controle do mecanismo de recebimento e envio de notificações e manutenção das tabelas de atualização e repositório de pacotes. Quando um *peer* é inicializado no sistema ele atualiza seu último estado armazenado na base de dados, sincroniza a tabela de pacotes com os *peers* vizinhos e inicia, quando necessário, requisições de dados de atualizações pendentes aos *peers* conhecidos. A manutenção das tabelas de pacotes é feita através da troca de informações entre *peers* da rede. Os pacotes recebidos e processados são armazenados no sistema de arquivos local, servindo de repositório de dados para os demais integrantes do sistema, em especial aos *peers* que fazem parte do mesmo grupo.

Protocolo de Notificações Este protocolo é utilizado para notificar os *peers* vizinhos da existência de novos pacotes para instalação. Um determinado *peer*, ao receber uma notificação de um pacote referente a um grupo ao qual ele pertence, cria um canal de comunicação com seus conhecidos para iniciar a requisição do novo pacote. Esta comunicação é feita através do protocolo de atualizações. Assim, quando os novos pacotes estão disponíveis localmente, são enviadas notificações aos demais *peers* da vizinhança.

Protocolo de Atualizações A função básica deste protocolo é garantir a troca de informações entre os *peers* da rede durante as atualizações de software. A requisição tem início com o pedido de um pacote (através de seu identificador). Neste momento, é consultada a tabela local para saber da disponibilidade das atualizações solicitadas. Em caso positivo, é iniciado o envio das informações referentes ao caminho de armazenamento do pacote, grupo de máquina ao qual a instalação se destina, comando a ser executado para a instalação do pacote e os arquivos de instalação propriamente ditos. A fase final consiste no processo de instalação local das atualizações de software recebidas.

4. Base de Dados

A base de dados do sistema Clumpt contém informações sobre as máquinas, os grupos, configurações e histórico dos pacotes instalados. Todos estes dados são armazenados em um servidor OpenLDAP [Zeilenga 2005]. Esta ferramenta foi escolhida por ser bastante utilizada como base de autenticação em servidores de redes e de *clusters*. Trata-se de uma implementação do protocolo de acesso a diretório de serviço LDAP (*Lightweight*

Directory Access Protocol). Diretório de serviço é um diretório otimizado para leitura, busca e navegação, pois é comum que esses diretórios recebam um grande volume de requisições e operações de busca.

As modificações na base de dados do sistema Clumpton ocorrem no momento de sua configuração e a cada instalação de pacote. Acessos para consultas são realizados toda vez que um *peer* inicia suas atividades e quando há recuperação em caso de falhas.

Para fins de escalabilidade e tolerância a falhas, OpenLDAP permite a replicação da base de dados de forma transparente. Isso pode ser feito através da configuração de um servidor OpenLDAP principal com um ou vários servidores secundários.

5. Funcionamento do Sistema

Durante o processo de inicialização, o *daemon* `clumpton` consulta a base de dados para obter a listagem de nós e a estrutura de grupos que modela a rede. As informações do histórico (último estado consistente) de atualizações processadas anteriormente pelo *peer* também são retiradas da base de dados. De posse dessas informações, o *peer* está apto a interagir com a rede P2P, isto é, conhece os componentes de seus grupos, aos quais ele pode prover serviços.

Após a inicialização, `clumpton` verifica, junto aos *peers* vizinhos, a existência de instalações pendentes. Isso permite que máquinas que ficaram desligadas por algum tempo consigam atingir um estado de homogeneidade de software em relação aos outros membros de seus grupos.

A manutenção das tabelas de atualização e as instalações de pacotes são iniciadas quando uma notificação é recebida, quando o *peer* é iniciado ou após um determinado intervalo de tempo. Este intervalo é definido na configuração dos *peers*, na base de dados.

No caso do processo de manutenção das tabelas por intervalo de tempo, este inicia-se com a eleição de um dos *peers* conhecidos (escolha aleatória dentro de um conjunto conhecido), pertencente a algum dos grupos aos quais o nó faz parte. Depois disso, são requisitadas novas entradas presentes na tabela de atualização do *peer* escolhido. Então, a sincronização das tabelas é processada.

Após cada sincronização, verifica-se a existência de instalações pendentes para a máquina local. No caso de haver pacotes a instalar para algum dos grupos ao qual o *peer* pertence, realiza-se a busca pelo pacote nos repositórios de pacotes dos *peers* do mesmo grupo.

Depois de encontrado o pacote, procede-se com a sua instalação local. Em caso de sucesso, o pacote instalado é salvo no repositório de pacotes e ocorre registro da instalação na base de dados. A partir desse momento, este *peer* é capaz de servir requisições que busquem por esse pacote.

A manutenção das tabelas de atualização no caso de notificação é realizada conforme o funcionamento dos protocolos de notificações e atualizações, descritos anteriormente. No caso da inicialização do *peer*, a sincronização é feita logo após a leitura dos dados de configuração e histórico de atualização, armazenados na base de dados. Esta sincronização é realizada através de um pedido especial, disponível no protocolo de atualizações, que solicita o encaminhamento da lista de identificadores de pacotes dis-

poníveis nos *peers* vizinhos. A partir disso, estes identificadores são comparados com os armazenados nas tabelas locais, procurando identificar pacotes novos. Uma vez identificado um novo pacote, é iniciado um pedido normal de dados através do protocolo de atualizações.

6. Interface com o Usuário

A figura 3 apresenta as formas pelas quais o usuário interage com o sistema. Os casos de interação são, basicamente, a instalação de pacotes, a configuração e registro de novos *peers* e a configuração do ambiente. Esta seção descreve a implementação e o funcionamento dos programas `clumplt` e `clumpltconf`, que permitem ao usuário a utilização das funcionalidades oferecidas pelo sistema Clumplt.

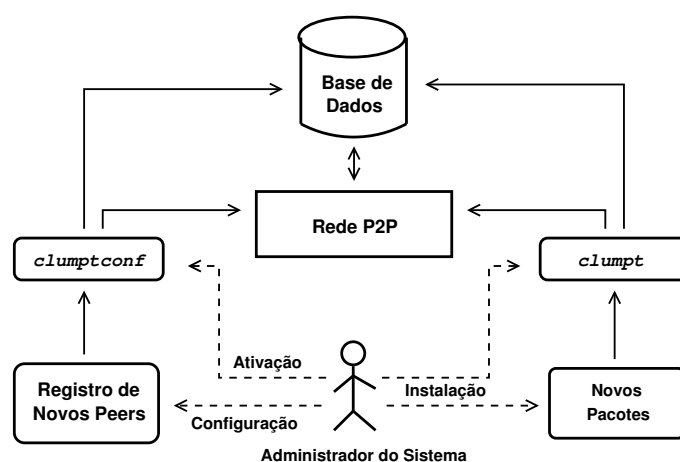


Figura 3. Interação com o sistema Clumplt.

6.1. Programa `clumplt`

Este programa permite a submissão de novas atualizações (instalação de software) ao sistema Clumplt. A interface de `clumplt` é semelhante às interfaces dos gerenciadores de pacotes mais utilizados (`emerge`, `apt-get`, `rpm`, `pkgtool`, etc.). Os mesmos pacotes instalados através destas ferramentas podem ser submetidos ao sistema através do programa `clumplt`. No momento em que o pacote é registrado, o administrador informa o caminho do arquivo do pacote, os grupos de máquinas no qual o pacote deverá ser instalado e o aplicativo que será utilizado para a instalação local do pacote. O programa `clumplt` gera então um novo identificador único (baseado em um número seqüencial disponível na base de dados) e registra o pacote na base de dados.

Na seqüência, o programa notifica os *peers* da existência de uma nova atualização de software. Para isso, `clumplt` consulta a base de dados e identifica os *peers* pertencentes aos grupos do pacote a ser instalado. O programa também verifica suas configurações, definidas pelo administrador do sistema, onde está definida a porcentagem de *peers*, escolhidos aleatoriamente, que irão receber as primeiras notificações de atualização. Ao mesmo tempo que notifica a existência de novos pacotes, `clumplt` está preparado para atender pedidos de requisição de pacotes através do protocolo de atualizações, ou seja, este programa implementa tanto o protocolo de notificações como o de requisição de pacotes. Quando os *peers* escolhidos tiverem recebido as novas atualizações, `clumplt` finaliza suas atividades. A propagação dos novos pacotes é responsabilidade da rede P2P.

6.2. Programa `clumptconf`

Este programa é responsável pela inclusão de novos *peers* no sistema. Quando o administrador do *cluster* de computadores desejar incluir novas máquinas, `clumptconf` poderá ser utilizado para incluir os novos *peers* no sistema e notificar os *peers* que terão novos vizinhos. Isso porque cada *peer* recebe, em sua configuração na base de dados, uma lista dos vizinhos conhecidos. É tarefa de `clumptconf` gerar estas listas e notificar os *peers* ativos, afetados pelas novas modificações, da necessidade de atualização de informações.

Os vizinhos de um *peer* são escolhidos com base nos grupos aos quais ele pertence. Um determinado *peer* terá, ao menos, um vizinho de cada grupo ao qual ele pertence. Estes vizinhos são escolhidos de forma aleatória. O número de vizinhos que um *peer* terá em suas tabelas é definido pelo administrador do sistema. No entanto, supondo que todos os vizinhos estejam indisponíveis, o *peer* irá consultar a base de dados, buscando informações sobre todos os *peers* do sistema. Neste caso, ele tentará conectar-se com todos os *peers* de todos os grupos aos quais ele pertence. Caso nenhum deles esteja acessível, o *peer* é considerado isolado (ou somente ele está ativo ou existe alguma problema de comunicação entre ele e os demais *peers*).

7. Avaliação do Sistema

A seguir é apresentada a descrição do cenário utilizado para avaliação do sistema. Na seqüência são discriminados os testes realizados e os resultados obtidos.

7.1. Apresentação do Cenário de Teste

Foram utilizados 20 *peers* para o teste, representando 2 *clusters* de 10 nós. Cada *peer* do sistema faz parte de dois grupos, um que representa o seu *cluster* e outro que representa todos os *peers* do sistema. Os nomes dos grupos são ilustrados na figura 4. O grupo `grupoIntel` representa os *peers* que trabalham com a arquitetura Intel. O grupo `grupoAMD` representa o grupo de *peers* que cuida da manutenção de software na arquitetura AMD. O grupo `grupoTodos` engloba os *peers* dos dois outros grupos. Com essa arquitetura, pode-se facilmente efetuar atualizações em uma ou outra arquitetura, ou em ambas.

7.2. Inicialização da Rede P2P

No cenário de teste, os 20 *peers* inicializaram suas tabelas de atualização corretamente, buscando as informações de configuração na base de dados e sincronizando as tabelas com os *peers* vizinhos. Para verificar o correto funcionamento da sincronização entre tabelas, durante a inicialização do *peer*, 50% dos *peers*, escolhidos aleatoriamente, foram configurados como se já tivessem processado mais de 100 atualizações de software. No momento em que os demais inicializassem eles deveriam receber todas as informações de pacotes processados pelos *peers* previamente escolhidos. Nos testes realizados, a sincronização entre as tabelas de atualizações ocorreu corretamente. Os *peers* que não faziam parte dos 50% escolhidos tiveram suas tabelas, e conseqüentemente seu estado na base de dados, corretamente atualizados, recebendo informações sobre os pacotes e os respectivos arquivos, quando aplicável (pacote destinado a algum dos grupos ao qual o *peer* pertence), necessários para o processo de instalação local.

A inicialização e sincronização das tabelas entre os *peers*, para o ambiente utilizado, não passou de 7 segundos. Este tempo é a soma do tempo gasto com a carga das

configurações iniciais da base de dados e sincronização entre tabelas dos *peers* vizinhos. Nos casos de teste, cada *peer* tinha, no máximo, 5 vizinhos e as tabelas de atualização continham, no momento das medidas, 223 entradas (registros de atualizações). Nos testes, 50% dos *peers* estavam com as tabelas atualizadas. Os outros 50% continham apenas os dados de 1 pacote, ou seja, precisavam pegar com os seus vizinhos as informações de 222 pacotes. O tempo médio de inicialização e sincronização ficou em 2.4 segundos. Os dados foram coletados através do uso da chamada de sistema *time* dentro do código dos agentes em cada *peer*. O tempo resultante foi obtido através da subtração do tempo de sistema coletado logo após as rotinas de inicialização e sincronização e do tempo de sistema coletado no início da aplicação (anteriormente as rotinas de inicialização e sincronização).

7.3. Submissão de Pacotes de Atualização

A ferramenta `clumpt` é responsável pelo registro do pacote na base de dados e notificação dos *peers* que irão iniciar o processo de distribuição do pacote. No cenário em estudo, foram levados em conta duas situações. A primeira considera que todos os *peers* estão ativos no sistema. Na segunda, alguns *peers* estão desativados, caracterizando a ocorrência de falhas ou interrupções para manutenção física e lógica de equipamentos.

7.3.1. Todos os *Peers* Ativos

A partir da entrada dos dados referentes ao pacote (ver seção 6.1), o `clumpt` registra o na base de dados e escolhe, aleatoriamente, uma porcentagem de *peers* pertencentes aos grupos destinatários do pacote e os notifica sobre o novo pacote. Assim que são notificados, os *peers* (que serão os pontos de partida para a propagação dos pacotes) fazem a requisição do pacote ao programa `clumpt` (emissor da notificação).

O cenário de testes foi configurado para que 20% dos *peers* pertencentes aos grupos dos pacotes submetidos fossem notificados pelo `clumpt`. Nos testes realizados, com

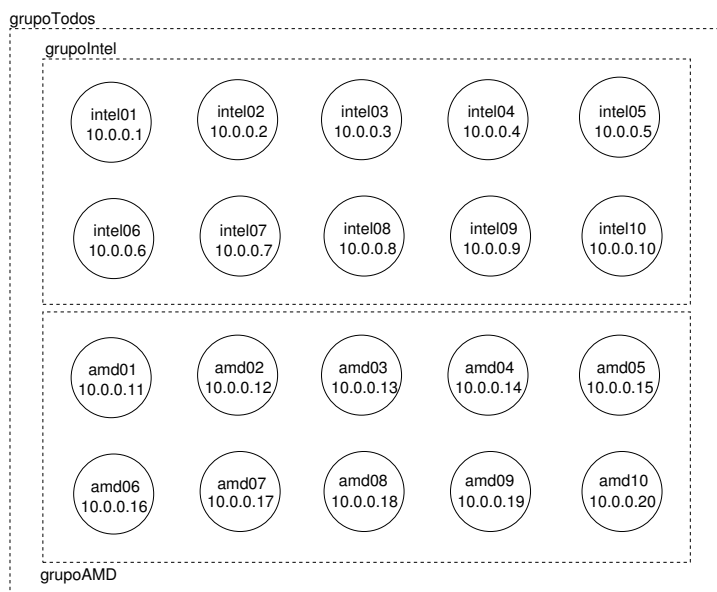


Figura 4. Ilustração da arquitetura utilizada.

mais de 200 pacotes distintos representando novas atualizações, o sistema como um todo, desde a notificação dos *peers* iniciais pelo `clumplt`, não levou mais do que 10 segundos (pior caso de propagação de um pacote) para notificar todos os *peers* do sistema e propagar as novas atualizações até eles. Os pacotes de software submetidos ao sistema variaram de 2 a 20 KB.

7.3.2. Alguns *Peers* Desativados

No cenário de teste com *peers* desativados, o objetivo foi verificar se ao voltarem a fazer parte do sistema as atualizações processadas durante este período seriam recebidas, atualizando o estado local do *peer* desativado por algum tempo. Nos vários testes realizados, foram desativados e re-ativados (após novas submissões de pacotes terem sido feitas no sistema) diferentes números de *peers*. Em todos os casos, ao voltar ao sistema, os *peers* sincronizavam suas tabelas de atualizações com os vizinhos, recebendo todos os dados e pacotes de software que haviam sido processados durante o período de inatividade.

Porém, *peers* desativados têm um impacto negativo no tempo de comunicação do sistema. Quando mais *peers* estiverem desativados, maior é a probabilidade de as atualizações levarem mais tempo até chegar a todas as unidades do sistema. Isso porque *peers* desativados levam a *timeouts* de conexão nos *peers* ativos que os conheciam como vizinhos. Uma solução para este problema pode ser a implementação de uma *thread* para cada *peer* a ser notificado. O ponto negativo dessa abordagem é que a necessidade de recursos da unidade do sistema cresce de acordo com o número de vizinhos que o *peer* conhece, pois, para cada vizinho conhecido ele criaria um novo fluxo de execução para controlar as notificações.

7.4. Inclusão de Novos *Peers*

Algumas configurações de testes iniciaram com apenas poucos (como por exemplo: quatro) *peers* no sistema. A partir desse ponto os demais *peers* começaram a ser incluídos no sistema através do `clumpltconf` (ver seção 6.2).

O usuário informa ao `clumpltconf` qual é o endereço IP do novo *peer* e quais são os grupos aos quais ele pertence. O `clumpltconf` procura garantir que cada novo *peer* contenha, ao menos, um vizinho pertencente a cada grupo. O novo *peer* poderá conhecer N vizinhos e M vizinhos o conhecerem. Estas duas variáveis, utilizadas pelo `clumpltconf`, são ajustáveis pelo administrador do sistema.

Os novos *peers*, assim que entram no sistema, estão imediatamente aptos a receber notificações de atualização (assim que elas ocorrerem) dos *peers* que o conhecem e propagar as atualizações para os *peers* conhecidos, como se fossem membros antigos do sistema. Os testes realizados nesse sentido mostraram um funcionamento dentro do esperado, ou seja, novos *peers* entraram no sistema, sincronizaram as tabelas de atualização com seus vizinhos e começaram a fazer parte ativa do sistema. Essa característica tornando fácil e prática, através do `clumpltconf`, a inclusão de novos *peers* no sistema.

7.5. Tolerância a Falhas

Em relação à tolerância a falhas, foram feitos diferentes testes, com diferentes *peers*, de mudança de estado de atividade para inatividade e novamente, após um certo tempo, ati-

vidade. Em todos os casos de testes os *peers* que foram desativados durante períodos variados de tempo voltaram ao sistema, sincronizaram suas tabelas de atualizações, processaram as atualizações que foram classificadas como pendentes (devido ao estado de inatividade) e chegaram a um estado consistente (em relação aos demais *peers* do sistema) de software.

A característica de tolerância a falhas é especialmente importante, pois é difícil prever-se falhas. Além disso, nem sempre é possível esperar para que todos os elementos do sistema estejam ativos para então serem realizadas atualizações de software. Muitas vezes, os elementos ativos precisam ser atualizados para poderem continuar processando dados, ou até novos tipos de dados, os quais causaram a necessidade de atualização de software. No momento em que os *peers* que falharam voltarem à ativa, eles também precisam receber as atualizações para que o sistema continue com uma imagem única e uniforme.

8. Trabalhos Relacionados

Existem diversas ferramentas que se propõem a realizar instalação automática nos vários nós de um *cluster* ou rede de computadores. Entre elas podem ser citadas:

- FAI [Lange 1999]: baseia-se na utilização de pacotes de software e permite a instalação inicial do sistema em um *cluster*. Sua arquitetura é centralizada, com a necessidade de um servidor de pacotes. FAI permite a instalação de ambientes heterogêneos levando em consideração diferenças de hardware. Quanto a software, este sistema é restrito à instalação da distribuição Debian Linux.
- SystemImager [Finley 2005]: é baseado na geração de imagens, podendo ser utilizado independentemente da distribuição de Linux escolhida. Esta ferramenta realiza instalação desde o início. Também permite a atualização de software através da geração incremental de imagens. Necessita de um servidor responsável pelo armazenamento e distribuição das imagens. Para a instalação de sistemas heterogêneos é necessária a criação de imagens para cada conjunto de máquinas.
- SCMS [Uthayopas and Rungsawang 1999]: baseado em pacotes, serve exclusivamente à instalação da distribuição Red Hat Linux. Tem o objetivo de efetuar instalação e atualização de software em sistemas já operacionais. Sua arquitetura é centralizada (modelo cliente-servidor).

Comparado com essas ferramentas, Clumpt difere basicamente no fato de possuir uma arquitetura P2P descentralizada, a fim de garantir tolerância a falhas e escalabilidade, e também por suportar *clusters* heterogêneos. Assim como FAI e SCMS, Clumpt baseia-se em pacotes. Esta abordagem apresenta a vantagem de não necessitar a criação ou alteração das imagens a cada nova instalação ou atualização de software. Além disso, não é necessária a criação de um servidor de imagens, que geralmente constitui um gargalo e um ponto única de falha.

A utilização do sistema Clumpt, assim como SystemImager, independe da distribuição Linux utilizada. Além disso, Clumpt permite a utilização do próprio sistema de gerenciamento de pacotes que acompanha a distribuição, o que auxilia na resolução de dependências.

9. Considerações Finais

A instalação e manutenção de software em *clusters* de computadores é um problema que tem motivado o desenvolvimento de diversas ferramentas de auxílio a administradores. Este problema aumenta ainda mais com o surgimento e disseminação dos *clusters* heterogêneos, onde diferentes arquiteturas de software e/ou hardware precisam ser gerenciadas.

Neste artigo, apresentou-se a arquitetura, implementação e funcionamento do sistema Clumpt. Este visa automatizar as tarefas administrativas que dizem respeito ao gerenciamento de software em *clusters* de computadores, mantendo níveis de escalabilidade e tolerância a falhas, características pouco presentes nas ferramentas de gerenciamento existentes. Para prover estes recursos o sistema possui uma arquitetura P2P que fornece independência entre *peers* e descentralização do processamento e distribuição das atualizações de software. Além disso, cada instância do Clumpt mantém seu estado atual localmente e em uma base de dados, podendo facilmente recuperá-lo em caso de falhas.

A solução proposta apresenta-se como uma forma de amenizar necessidades gerenciais do dia-a-dia de administradores de máquinas paralelas. Os recursos providos pelo sistema Clumpt fornecem flexibilidade e tolerância a falhas para a manutenção de software tanto em *clusters* homogêneos quanto heterogêneos. Além disso, o sistema é aplicável a ambientes distribuídos e redes de computadores em geral.

Referências

- Baldassari, J. D., Kopec, C. L., Leshay, E. S., Truszkowski, W., and Finkel, D. (2005). Autonomic cluster management system (ACMS): A demonstration of autonomic principles at work. In *ECBS*, pages 512–518.
- Finley, B. E. (2005). *SystemImager v3.2.0 Manual*. <http://www.systemimager.org/doc/systemimager-manual.pdf>.
- Harr, J. and Denault, G. (2002). Issues concerning linux clustering: Cluster management and application porting. In Werner, B., editor, *Proceedings of the 16th International Parallel and Distributing Processing Symposium (IPDPS)*, pages 54–54.
- Lange, T. (1999). Fully automatic installation (FAI). <http://www.informatik.uni-koeln.de/fai>.
- Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing*, pages 101–102.
- Uthayopas, P. and Rungsawang, A. (1999). SCMS: An extensible cluster management tool for beowulf cluster. In *Proceedings of Supercomputing'99 (CD-ROM)*, Portland, OR. ACM SIGARCH and IEEE. Department of Computer Engineering, Kasetsart University.
- Zeilenga, K. (2005). *OpenLDAP Software 2.3 Administrator's Guide*. <http://www.openldap.org/doc/admin23/>.